Dissertation

Contributions To Ontology-Driven Requirements Engineering

bearbeitet von

Dipl.-Medieninf. Katja Siegemund

geboren am 26.05.1981 in Leipzig

vorgelegt an der
Technischen Universität Dresden
Fakultät Informatik
Lehrstuhl Softwaretechnologie

zur Erlangung des akademischen Grades
Doktoringenieur (Dr.-Ing.)

Betreuender Hochschullehrer: Prof. Dr. Uwe Aßmann
(Technische Universität Dresden)

Zweiter Gutachter: Prof. Dr. Gerd Wagner
(Technische Universität Cottbus)

Verteidigt am: 29.04.2014

Abstract

Today, it is well known that missing, incomplete or inconsistent requirements lead to faulty software designs, implementations and tests resulting in software of improper quality or safety risks. Thus, an improved Requirements Engineering contributes to safer and better-quality software, reduces the risk of overrun time and budgets and, most of all, decreases or even eliminates the risk for project failures.
One significant problem requirements engineers have to cope with, are inconsistencies in the Software Requirements Specification. Such inconsistencies result from the acquisition, specification, and evolution of goals and requirements from multiple stakeholders and sources. In order to regain consistency, requirements information are removed from the specification which often leads to incompleteness. Due to this causal relationship between *consistency*, *completeness* and *correctness*, we can formally improve the correctness of requirements knowledge by increasing its completeness and consistency. Furthermore, the poor quality of individual requirements is a primary reason why so many projects continue to fail and needs to be considered in order to improve the Software Requirements Specification.
These flaws in the Software Requirements Specification are hard to identify by current methods and thus, usually remain unrecognised. While the validation of requirements ensures that they are correct, complete, consistent and meet the customer and user intents, the requirements engineer is hardly supported by automated validation methods.

In this thesis, a novel approach to automated validation and measurement of requirements knowledge is presented, which automatically identifies incomplete or inconsistent requirements and quality flaws. Furthermore, the requirements engineer is guided by providing knowledge specific suggestions on how to resolve them. For this purpose, a requirements metamodel, the Requirements Ontology, has been developed that provides the basis for the validation and measurement support. This requirements ontology is suited for Goal-oriented Requirements Engineering and allows for the conceptualisation of requirements knowledge, facilitated by ontologies. It provides a huge set of predefined requirements metadata, requirements artefacts and various relations among them. Thus, the Requirements Ontology enables the documentation of structured, reusable, unambiguous, traceable, complete and consistent requirements as demanded by the IEEE specification for Software Requirement Specifications. We demonstrate our approach with a prototypic implementation called OntoReq. OntoReq allows for the specification of requirements knowledge while keeping the ontology invisible to the requirements engineer and enables the validation of the knowledge captured within.
The validation approach presented in this thesis is capable of being applied to any domain ontology. Therefore, we formulate various guidelines and use a continuous example to demonstrate the transfer to the domain of medical drugs. The Requirements Ontology as well as OntoReq have been evaluated by different methods. The Requirements Ontology has been shown to be capable for capturing requirements knowledge of a real Software Requirements Specification and OntoReq feasible to be used by a requirements engineering tool to highlight inconsistencies, incompleteness and quality flaws during real time requirements modelling.

Acknowledgement

Apart from the efforts of myself, the success of any project depends largely on the encouragement and guidelines of many others. I take this opportunity to express my gratitude to the people who have been instrumental in the successful completion of this project. I would like to show my greatest appreciation to Prof. Uwe Aßmann who introduced me to Ontologies many years ago and offered me the chance for this research. I felt motivated and encouraged every time we have talked. Furthermore, I wish to thank my colleagues and also project partners of the MOST project for their guidance and support and their contribution to this project. In addition, I especially would like to thank my colleague Jan Polowinski for his useful remarks and comments and for proofreading my thesis.
I also want to thank my husband for his love, kindness and support he has shown during the past year it has taken me to finalize this thesis. Furthermore, I would also like to thank my parents for their endless love, support and their never ending trust in me. I would like to thank my friends, who have supported me throughout the entire process by taking care of my children, keeping me harmonious and also sometimes distracting me from work when it was essential. Last but not least, I also want to thank the ESF (Europäische Sozialfonds) for their financial support granted for the past two years and thus, enabling me to finalize my thesis.

This work has partially been funded by the ESF.

Europäische Union

Europa fördert Sachsen.
ESF
Europäischer Sozialfonds

Contents

1 Introduction

Requirements Engineering (RE) [1] refers to the process of eliciting, evaluating, specifying, consolidating, and changing the objectives, functionalities, qualities, and constraints to be achieved by a software-intensive system [131]. This process is the first part of the software engineering lifecycle and is divided into four subprocesses (see Figure 1.1): (1) Elicitation, (2) Analysis, (3) Specification and (4) Validation. Developing software for industries and organizations is becoming a more and more complex task due to increasing demands of customers and new technologies. Concurrently, software developers are expected to build more complex software in less time in order to be competitive. Developing complex software systems raises a large number of needs, wishes, and requirements that are - due to different viewpoints and stakeholders - often conflicting with each other.

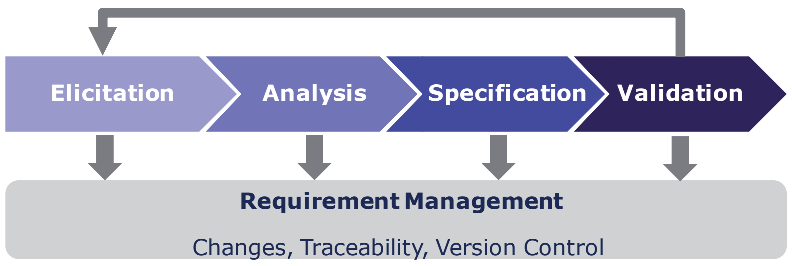


Figure 1.1: Requirements Engineering Process

This chapter is structured as follows: in Section 1.1 we describe the context of this thesis and our motivation for the research questions in this work. We introduce the problem areas we aim to tackle in Section 1.2 and clarify the aims of this work in Section 1.3. In Section 1.4 we explain the research background of this thesis and summarize the contributions and limitations of our approach in Section 1.5.

1.1 Context and Motivation

Today, it is well known that missing, incomplete or inconsistent requirements lead to faulty software designs, implementations and tests resulting in software of improper quality or safety risks. Project cost and time overruns due to missing requirements and underestimated budgets may even cause a project to be aborted. Customers may reject such a system and end-users will be dissatisfied with it [42]. A review of several studies regarding reasons for project failure [5] concluded that "five of the eight major reasons for failure are requirements-based". The importance of Requirements Engineering is already supported in several studies (e.g., [58]). Thus, an improved Requirements Engineering contributes to safer and better-quality software, reduces the risk of overrun time and

[1]The term 'Requirements Engineering' appears in different technical domains. However, in this thesis we will refer to the domain of Software Engineering.

budgets and, most of all, decreases or even eliminates the risk for project failures.

One significant problem requirements engineers have to cope with, are inconsistencies in the Software Requirements Specification (SRS). Such inconsistencies result from the acquisition, specification, and evolution of goals and requirements from multiple stakeholders and sources [132]. It is frequently the case that changes of requirements have a particularly significant impact on the consistency of specifications. In order to regain consistency, requirements information are removed from the specification which often leads to incompleteness. Zowghi in [151] describes this vicious circle as a causal relationship between consistency, completeness and correctness. Due to this causal relationship, we can formally improve the correctness of requirements knowledge by increasing its completeness and consistency. Although it would be desirable to have a formal proof of correctness of a SRS, it may not be practical or may be too costly [151]. Whether applying such formal correctness proofs or not depends on the degree of risk the stakeholders are prepared to take [151]. According to Zowghi et.al., such proofs can be carried out in many cases by informal inspections of the requirements and domain, involving customer and stakeholder. Thus, we adapt the formal point of view for correctness and define correctness as the combination of completeness and consistency. Hence, the correctness of requirements knowledge is improved by improving its completeness and consistency.
In addition to completeness, consistency and correctness, another aspect of the SRS becomes important. Firesmith in [43] states that "poor quality of individual requirements and the requirements specifications that document them is a primary reason why so many projects continue to fail [Standish 1994]". Furthermore, Firesmith states that "the poor quality of the requirements is typically not recognized during Requirements Engineering and the evaluation of requirements specifications". Thus, the quality of individual requirements is another aspect we need to consider.

In the meantime methods for RE are not only increasing in number, but also in complexity. It is fairly hard for an average requirements engineer to cope with all these methods and approaches. Apart from that, RE itself is a complex task that needs the ability to abstract, refine and relate various information. It is far more knowledge intensive than other activities [120]. Although RE tools may provide a valuable User Interface (UI) and technology background, they lack in guidance for the the requirements engineer that is not only based on general guidelines. Thus, guidance must play a crucial role in the RE process.

1.2 Problem Description

Customers and users are usually not familiar with requirements. In most cases, they have a vague image of how the software should look like and what they aim to do with it. These intentions are also important for Requirements Engineering, yet they are not enough. Requirements engineers need to understand these intentions (customer goals) and identify underlying requirements that serve these goals. This very first elicitation and analysis of requirements is called "Early Requirements Engineering" and is a crucial task for upcoming steps in the Requirements Engineering process. Obviously, this is the time where a huge amount of errors is produced. Improving this communication with stakeholder is one of the major challenges in Requirements Engineering and research proposes promising methods and technologies for this part of the process. However, "Late Requirements Engineering",

that is the specification and validation of requirements, is hardly addressed and suffers from mistakes in Early Requirements Engineering.
According to Bahili and Henderson in [124], we define the validation[2] of requirements as follows:

Definition 1 (Requirements Validation:): *"Ensuring that (1) the set of requirements is correct, complete, and consistent, (2) a model can be created that satisfies the requirements, and (3) a real-world solution can be built and tested to prove that it satisfies the requirements."*

It is often the case that requirements engineers do not identify a sufficient set of requirements to deliver a software that meets all user intentions. Furthermore, they provide incomplete requirements information such as missing priorities, risks, costs or test-cases. And most of all, requirements are hardly interrelated with each other. This lack of information leads to inconsistencies such as contradictions or redundancies.
The following main categories of problems in Requirements Engineering, described in [80] are addressed in a number of articles from industry and research in the last decade (e.g., [5, 48, 104, 71]):

1. a requirement is used for the wrong purpose
2. a requirement or important information *about* a requirement is missing
3. a requirement is stated in an ambiguous manner
4. requirements or their rationales are inconsistent

The growing complexity of software and their increasing use in safety-critical environments (e.g., car, aircraft, medical systems) multiplies the importance of sophisticated Requirements Engineering and also leads to new challenges in Requirements Engineering. The following sections briefly introduce the main problem areas addressed in this thesis. Figure 1.2 illustrates these problems and their consequences.

1.2.1 Completeness of Requirements Knowledge

According to [151] when using the term "completeness" we need to distinguish between *internal* and *external* completeness. Internal completeness refers to individual requirements rather than the whole Requirements Specification and implies that no information is left unstated or "to be determined" and information does not contain any undefined objects or entities. [42] refers to these data as *metadata*. External completeness refers to the information in the whole *Requirements Specification* (requirements analysis models, individual requirements, requirements analysis documents, repositories, individual requirement specification documents and the requirements baseline [42]). Thus, an individual requirement is complete if it contains all necessary information to avoid ambiguity and needs no amplification to enable proper implementation and verification. To avoid ambiguity, a requirement must express the entire need and state all conditions and constraints under which it applies [142]. This can be ensured by complete metadata for requirements that serve as a metamodel so that internal completeness regarding this metamodel can be measured and increased.

[2] In contrast to validation, the *verification* of requirements is defined as "Building the system right: ensuring that the system complies with the system requirements and conforms to its design." [124]

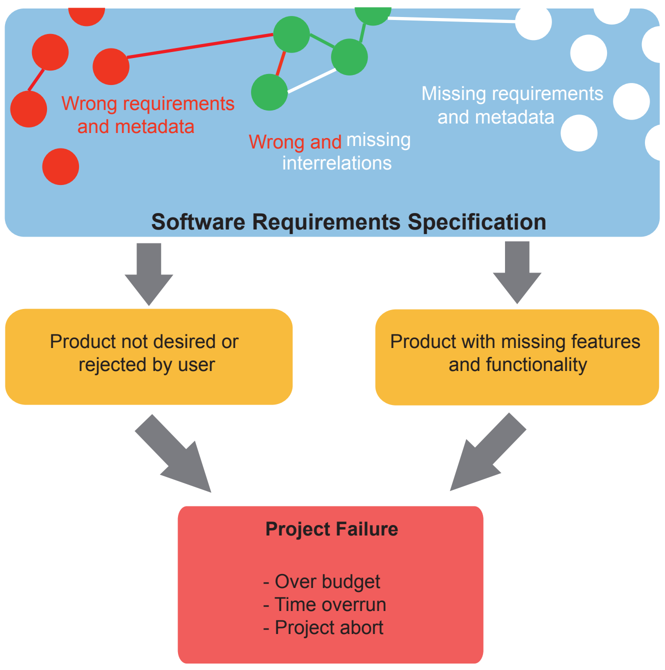

Wrong requirements and metadata
Wrong and missing interrelations
Missing requirements and metadata
Software Requirements Specification
Product not desired or rejected by user
Product with missing features and functionality
Project Failure
- Over budget
- Time overrun
- Project abort

Figure 1.2: Overview of problems and consequences in Requirements Engineering

Davis in [32] names *completeness* to be the most difficult of these (above mentioned) specification attributes to define and *incompleteness* the most difficult violation to detect. In contrast, it is reasonably well known, that requirements will never be *totally* complete, finished or finalized as long as the system is in service and must evolve [42]. However, it is possible to improve the internal completeness of requirements, namely the completeness of the metadata of individual requirements, their interrelations and relations to other requirements artefacts such as use-cases or metrics.
Although some approaches exist that aim to ensure complete requirements (e.g., [20]), up to now there is no absolute way to determine the internal or external completeness of requirements in advance. Though current Requirements Engineering tools provide means for capturing requirements (and some of them even diagrams and other kinds of descriptions), they fail in providing sufficient metadata about requirements and leave it to the requirements engineer to define them. Such missing metadata, interrelations and requirements artefacts may become expensive and time consuming in the Software Engineering process.

1.2.2 Consistency of Requirements Knowledge

Another problem to tackle are inconsistencies among requirements and their metadata. According to completeness, we also need to distinguish between *internal* and *external* consistency. We understand internal consistency as the absence of overlapping, redundant and contradictory requirements and any conflicts between their metadata. A typical example for internal inconsistent requirements are the following two statements: "*The device must reduce energy consumption at night.*" and "*The device must execute maintaining tasks at night.*" Each single requirement is valid, but without any additional information,

they are contradictory and mutual exclusive. Such inconsistencies are often introduced when adding new requirements, changing or deleting existing ones [151]. External consistency refers to the whole Requirements Specification, for example inconsistent links to documents or domain knowledge.
Various approaches have been proposed to handle inconsistencies of requirements with respect to a domain description (e.g., [69, 141]). These approaches aim to ensure that requirements comply with the concepts of a given domain. However, the detection of conflicts between requirements and their metadata is often only addressed regarding refinement relationships or conceptual overlapping. Moreover, most techniques consider only binary requirements conflicts, that are, conflicts among two requirements. But requirements are often crosscutting, scattered through the whole system. The may appear in use-cases or test-cases and are a part of metrics. These various interrelations are not sufficiently considered to improve the consistency of requirements and their metadata.

1.2.3 Quality of Requirements Knowledge

As stated above in Section 1.1, the poor quality of individual requirements and the SRS may lead to unsatisfied customers or even failing projects [43]. In [32], Davis proposes 24 criteria and formulas for measuring the quality of a SRS, namely:

- 1. Unambiguous
- 2. Complete
- 3. Correct
- 4. Understandable
- 5. Verifiable
- 6. Internally Consistent
- 7. Externally Consistent
- 8. Achievable
- 9. Concise
- 10. Design Independent
- 11. Traceable
- 12. Modifiable
- 13. Electronically Stored
- 14. Executable/Interpretable
- 15. Annotated by Relative Importance
- 16. Annotated by Relative Stability
- 17. Annotated by Version
- 18. Not Redundant
- 19. At Right Level of Detail
- 20. Precise
- 21. Reusable
- 22. Traced
- 23. Organized
- 24. Cross-Referenced

As can be seen from the list above, the quality of requirements requires explicitly completeness (2.), consistency (6.) and (7.) and correctness (3.) among other characteristics. Thus, by improving these properties of the SRS, we may already improve its quality. However, we need to consider more aspects of the SRS to enhance its quality. Therefore, we distinguish between improving the quality of the description of individual requirements and the quality of the SRS itself. While the former comprises approaches to improve the linguistic description of a requirement (e.g., precision, ambiguity), the latter comprises ambitions regarding the entire requirements knowledge (e.g., structure, traceability, reusability, completeness). The quality of the linguistic description of individual requirements can already be measured as proposed in various approaches (e.g., [38], [17] and [83]). However, improving and measuring the quality of the SRS is little addressed in literature.
In order to discuss quality aspects of the SRS, we use the term "*quality flaw*" and refer with it to any (missing or existent) information that hinders or decreases any of the above quality attributes defined by Davis or the quality rules we define in Chapter 8.5.

1.2.4 Validation of Requirements Knowledge

The validation of requirements ensures that they are correct, complete, consistent and meet the customer and user intents. Unfortunately, current Requirements Engineering tools lack the possibility of validating the internal completeness and consistency of requirements. If validation is provided at all, it is external and aims for example for identifying corrupt links to documents and models. Thus, the requirements engineer will never notice that important requirements information is missing or inconsistent. There is hardly systematic support for detecting and resolving such faults. One notable exception is [132]. Typical validation methods for Requirements Engineering are mainly human-centred. This means that techniques like inspection, tests, interviews or demonstrations are organized and accomplished by the requirements engineer himself. This kind of validation may identify a small number of incompleteness and inconsistency faults, yet it is not sufficient. Especially complex and huge sets of requirements are hard to validate with such techniques alone. Inspecting a requirements document and identifying inconsistencies in it only by re-reading is a time-consuming and error-prone task. Therefore, it is important to support the requirements engineer in the validation of requirements.

1.2.5 Guidance

All of these above mentioned problems may lead to the question: How can an average requirements engineer perform all these tasks in a correct way? One major problem in Requirements Engineering is the plurality of methods and guidelines to be considered and chosen from. At the same time, requirements engineers often lack sufficient methodology knowledge. Moreover, such methodologies and guidelines are often not specific enough and usually do not consider the current requirements and their structure. Yu in [144] states that "users need help in coming up with initial requirements in the first place". While approaches exist that guide the requirements engineer in the process of Requirements Engineering by displaying upcoming tasks and providing decision support for choosing an appropriate methodology, there is a lack of guidance for the *validation* of specified requirements. This includes suggestions on how to proceed in certain situations in contrast to simply showing possible errors and leaving the requirements engineer alone with their elimination. Additionally, guidance is needed for reporting errors and possible solutions. Thus, as stated in [120], guidance needs to be far more knowledge intensive than in other activities. It is clearly beyond the simple automated control of sequences of activities that ignore the present requirements knowledge and only provide general suggestions. Without knowledge specific guidance, Requirements Engineering may certainly stay a highly error-prone and sometimes puzzling task.

1.3 Thesis Aims and Objectives

The work presented in this thesis is motivated by two main movements: goal-driven approaches in Requirements Engineering and ontologies in general. We base our research on the following three hypotheses:

1. The specification of goals and their relation to requirements artefacts support efforts in improving the completeness, consistency and quality of requirements knowledge.

2. The formalisation of requirements knowledge allows for an automated validation of requirements knowledge.

3. Ontologies provide means to structure and reason about requirements knowledge, facilitate traceability[3] and enable the automated validation of completeness, consistency and quality criteria captured within.

Our approach aims to support the Goal Oriented Requirements Engineering (GORE) paradigm and attaches to the specification and validation phases of the RE process (see Figure 1.1). The objectives of this thesis are to improve the internal completeness and consistency of requirements and, thus, increase the quality and correctness of the entire SRS. Additionally, we aim to develop a method to automatically validate and measure the completeness, consistency and quality of requirements knowledge. Furthermore, we intend to provide guidance support for the validation and elimination of the identified completeness, consistency and quality faults.
We base our concept on an analysis of existing problems and challenges in Requirements Engineering, State-of-the-Art and Requirements Engineering tools. A software demonstration exemplifies our approach. To this end, the objectives of the thesis are:

1. Analysis of existing problems and challenges of Requirements Engineering reported from industry and research

2. Deduction of coherent and general requirements for Requirements Engineering

3. Provide means to capture, structure and analyze requirements and their metadata

4. Automated validation for internal completeness, consistency and quality

5. Automated measuring of quality of requirements knowledge

6. Facilitate traceability of requirements and related requirements artefacts (e.g., goals, test-cases, sources)

7. General guidelines on how to apply this approach for any other domain

8. Software demonstration of the approach

1.4 Research Background

The work presented in this thesis is the confluence of three technologies: *Ontology Modelling*, *Feature Modelling* and *Process Support*. The term “Ontology Modelling” refers to techniques describing knowledge of any kind by means of ontologies. An ontology is defined to be a ”formal, explicit specification of a shared conceptualisation” [54]. An ontology formally represents knowledge by a set of concepts within a domain and the relationships between these concepts. This formal specification allows *reasoning*, that is, deriving facts not explicitly expressed in the ontology.

[3]“Requirements traceability refers to the ability to describe and follow the life of a requirement, in both forwards and backwards direction (i.e. from its origins, through its development and specification, to its subsequent deployment and use, and through all periods of on-going refinement and iteration in any of these phases.)” [49].

"Feature Modelling" originates from Software Product Line[4] Modelling where a Feature Model represents all software products of the Software Product Line. The variability of features captured in a Feature Model facilitates the configuration of different software products. Feature models were first introduced in the Feature-Oriented Domain Analysis (FODA) method by [70]. Since then, Feature Modelling has been widely adopted not only by the software product line community but also by a number of approaches in Software Development and Requirements Engineering (e.g., [149]).

The term "Process Support" within this thesis refers to the process performed by requirements engineers in order to capture, analyze, specify, consolidate, and change the goals, requirements, and constraints to be achieved by a software system. Approaches in this area are based on the process modelling paradigm in the Software Engineering field [25, 91]. Process support mainly focuses on *prescriptive* models which enforce rules and behavioural patterns. Following these rules leads to the desired process performance [73]. Another option to support the user in a certain process is *process enforcement* by trying to ensure a specific sequence of tasks.

1.5 Thesis Contributions and Limitations

The thesis offers two main contributions:

1. Conceptualisation of requirements knowledge, facilitated by ontologies.
2. Automated validation and measurement of requirements knowledge regarding the internal completeness, consistency and quality.

In this work, we developed a Requirements Ontology suited for GORE that provides a huge set of predefined requirements metadata, requirements artefacts and various relations among them. This metamodel as ontology TBox[5] serves as a template for specifying concrete requirements and associated information (e.g., goals, use-cases, priority) in the ontology ABox[6]. The Requirements Ontology allows for the documentation of structured, reusable, unambiguous, traceable, complete and consistent requirements as demanded by the IEEE specification for Software Requirement Specifications [63].
Furthermore, we provide an automated validation of internal completeness and consistency as well as an automated measuring of different quality criteria. This validation is based on the Requirements Ontology and allows for identifying missing metadata, requirements artefacts and interrelations. The validation of the internal consistency detects conflicts such as missing mandatory or coexistent requirements, conflicting or excluding requirements. The requirements engineer is guided by validation results that explicitly point to sources of faulty information and computed solution suggestions to resolve them. The validation and solutions suggestions are always based on the present state of the requirements knowledge. Additionally, the quality of the requirements knowledge can automatically be measured by metrics provided.
We demonstrate the concepts and methods described above with a software demonstrator called "OntoReq". OntoReq also allows for the modification of requirements knowledge by

[4]A Software Product Line is a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way [65].

[5]"TBox statements describe a conceptualization, a set of concepts and properties for these concepts." [138]

[6]"ABox are TBox-compliant statements about that vocabulary [138]. Thus, individuals of the ABox are instances of the TBox."

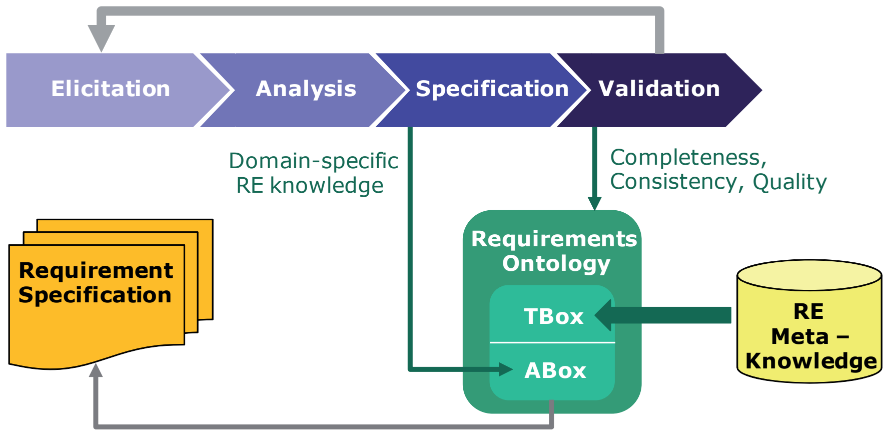


Figure 1.3: Overview of Thesis Approach

a prototypic user interface instead of manipulating the ontology directly in an ontology tool (e.g., Protegé). Thus, the Requirements Ontology is kept in the background and invisible to the user. An exemplar is used to illustrate all steps from the specification of requirements, their metadata and interrelations, their formalisation and validation of this requirements knowledge and its revision. Furthermore, we propose general guidelines that allow for applying the requirements validation services for any domain ontology. Finally, we compare our approach with features of existing RE tools that are currently used in industrial projects and evaluate our work.

We limit our approach to *internal* completeness of individual requirements rather than the *external* completeness of the whole Software Requirements Specification. Thus, we do not identify missing requirements, but missing information about individual requirements and interrelations among the requirements knowledge. Furthermore, we validate the internal consistency of a *requirements configuration* and do not consider to validate the *logical* consistency of individual requirements descriptions (e.g., conflicting data types or redundant information). Although our approach facilitates the traceability of requirements during the Requirements Engineering process, we do not provide an implementation of seamless traceability for design or implementation artefacts. However, the Requirements Ontology can be mapped to appropriate design or implementation models to realise this traceability.

1.6 Organisation of the Thesis

This thesis is organized in four parts as depicted in Figure 1.4. The first part covers the foundations of the presented approach developed in this thesis in Chapters 2, 3, 4 and 5. The part is closed by an overview of related work and state-of-the-art in Chapter 6.
Part II describes the conceptual solution for an ontology-driven automated validation of requirements knowledge. An identification of current challenges in RE and deduction of requirements for a systematic support of RE is provided in Chapter 7. The Requirements Ontology and the automated validation are comprised to the OntoReq approach covered in Chapter 8.
Part III describes the technical solution for OntoReq. First, the Requirements Ontology is

presented in Chapter 9, followed by the requirements validation services in Chapter 10. We proceed with a description of the architecture in Chapter 11 and finally formulate guidelines for transferring our validation approach for arbitrary domain ontologies in Chapter 12.
In Part IV, an exemplar is provided in Chapter 13 that demonstrates the use of OntoReq by a continuous example. This is followed by several evaluations in Chapter 14 and a comparison of OntoReq with other RE tools in Chapter 15. The thesis is closed by a description of future work and a conclusion in Chapter 16.

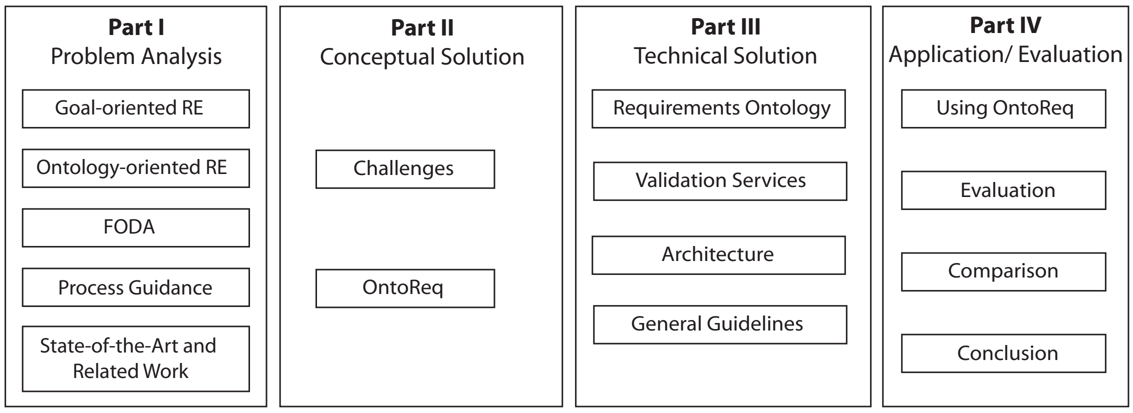


Figure 1.4: Organisation of Thesis

Part I

Problem Analysis

2 Goal-Oriented Requirements Engineering

This chapter investigates the role of goal modelling in Requirements Engineering and reports about current issues and problems covered in literature. Since there is no standardized terminology, this chapter also provides definitions that will be used within this thesis.

This chapter is structured as follows: Section 2.2 introduces a definition for Requirements Engineering and describes its main tasks. A brief explanation of the main terms and activities of Goal Oriented Requirements Engineering Goal Oriented Requirements Engineering (GORE) is given in Section 2.3. Additionally, a definition for Goal-Oriented Requirements Engineering is developed that will be used within this thesis. Important insights of this chapter are summarized in Section 2.4.

2.1 Introduction

Since the 1970's, people have recognized the importance of correct and complete requirements for successful software development [35]. Requirements Engineering increases the understanding of the proposed system. Faults in Requirements Engineering influence all phases of software development. Missing or incorrect requirements are often detected too late. Thus, more time and money is needed to cope with them. In the worst case, the final software does not comply to the customer's wishes or is not accepted by users. These problems have been realized by industry in the last years and RE has emerged in order to improve the development of software in a more systematic way. However, despite of tremendous research in RE, the gathered knowledge and methodologies have hardly been transferred to industry organizations. One main reason is the lack of intelligent tool support for RE [148] in the last decades. The development of such software support has become more and more important in the last few years.

2.2 Requirements Engineering

Requirements Engineering (RE) is concerned with the elicitation, evaluation, specification, consolidation, and change of objectives, requirements, functionalities, qualities, and constraints to be achieved by a software-intensive system [131]. RE has the objective to establish a complete, consistent and unambiguous description of requirements (Requirements Specification) for a given application domain on an abstract conceptual level. This incremental process involves stakeholders from different backgrounds and requirements engineers.

The most cited definition for the term 'requirement' is given by IEEE in [66]:

> "(1) A condition or capability needed by a user to solve a problem or achieve an objective. (2) A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification,

> or other formally imposed documents. (3) A documented representation of a condition or capability as in (1) or (2)."

As already mentioned above, literature does not provide an unique definition of Requirements Engineering nor a dissociation from Requirements Management. Often RE is also referred to as Requirements Analysis, -elicitation, -management, or -specification. [146] provides an intuitive description of RE:

> *"Requirements Engineering is the branch of software engineering concerned with the real-world goals for, functions of, and constraints on software systems. It is also concerned with the relationship of these factors to precise specifications of software behaviour, and to their evolution over time and across software families."*

This definition highlights the importance of "goals" as a motivation for the development of software systems. These goals represent the 'why' and 'what' of a system [106]. Additionally, it includes "precise specifications" which are the basis for requirement analysis, validation and verification. Finally, the definition also addresses the "evolution" of requirements "over time and across software families", reflecting the changes of customer wishes and technologies and the need to deal with them.

Another definition given in [92] mainly focuses on the *process* of RE:

> Requirements Engineering is the "*systematic process of developing requirements through an iterative co-operative process of analysing the problem, documenting the resulting observations in a variety of representation formats, and checking the accuracy of the understanding gained.*"

This definition addresses the activities to be performed in order to accomplish RE. Furthermore, it points out that this process is 'co-operative', thus, incorporating various interdisciplinary stakeholders (customers, users, requirements engineers, software developer, etc.). Additionally, this definition emphasizes the documentation and validation of requirements.
It is the author's intention to focus both on the *process* of RE and the different RE artefacts and their relationships within this process. Since *goals* play an important role we will give a definition for GORE in Section 2.3. We propose the following definition used throughout this thesis, comprising parts of the definitions of [146], [92] and [109]:

Definition 2 (Requirements Engineering): *Requirements Engineering is the branch of Systems Engineering concerned with the development of requirements through a systematic, iterative and co-operative process. This process includes the elicitation, negotiation, specification and validation of requirements. It is also concerned with the relationship of these RE artefacts to precise specifications of software behaviour, their evolution over time and across software families. Requirements and all related artefacts are documented in a Requirements Specification that needs to be validated regarding customer wishes, correct understanding and accuracy.*

Requirements *elicitation* is the practice of understanding the system under consideration and obtaining its requirements, user needs, constraints and problems from users, customers and stakeholders. By requirements negotiation, the various stakeholders involved in the process agree on these requirements. Furthermore, the elicited requirements knowledge may be specified by using various concepts such as (functional and non-functional) requirements, use-cases, priorities, etc. We will refer to all these concepts as requirements artefacts and propose the following definition to be used within this thesis:

Definition 3: *Requirements artefacts are parts of the requirements knowledge that hold specific types of information, e.g., requirements, test-cases, metrics, prioritiess.*

One product of the RE process is the documentation of the requirements, the *Requirements Specification.* It exists a variety of definitions for this term, focussing on different aspects. Although IEEE released 1984 a standard that describes how a SRS should be structured and what it should contain, many approaches provide modified or new definitions due to new appearing methods. However, we will use the following definition for Requirements Specification since it describes the most general and durable facts:

Definition 4 (Requirements Specification): *A Software Requirements Specification (SRS) is a comprehensive description of the intended purpose and environment for software under development. The SRS fully describes what the software will do and how it will be expected to perform.*

Obviously, this definition makes no statements about *how* to structure and specify requirements, e.g., which methods or model of representation to use. It is a fact that the Requirements Specifications differ from RE approach to another. Some methods make use of text documents, some of databases and others of UML diagrams. Whatever technique is used, according to [63] it must be:

- Correct
- Unambiguous
- Complete
- Consistent
- Ranked for importance and/or stability
- Verifiable
- Modifiable
- Traceable

as stated by the IEEE in [63].

Finally, the *validation* task ensures that the derived specification corresponds to the stakeholders needs and to the internal and/or external constraints set by the enterprise [73].
Another important attribute of a good SRS is to describe requirements to the system rather than providing solutions. Often these two things are admixed, for example: *"The software user interface shall have a drop down list of three items to choose a language."* instead of *"The software shall provide means to select different languages."* The first requirement description already gives a solution on how to provide the selection of languages, it already focuses on design and, thus, gives a design solution. But requirements are about purposes and the purpose for the system is found *outside* the system itself. Once we already specified a requirement containing a solution, it is not possible to reuse this requirement for any other product. Therefore, it is important to clearly separate requirements and goals from any solution during RE.

Lin et al. [67] summarizes a need for a representation of requirements that complies to the following main points:

- Provides an unambiguous and precise terminology.
- Allows traceability of requirements including all dependencies and relationships among them.
- Supports the identification of redundant or conflicting requirements.
- Is generic, reusable, and easy to extend.

2.3 Goal-Oriented Requirements Engineering

Conventional RE in its early times concentrated on what the system should do and how it should do it. This lead to fairly low-level requirements on data, operations, etc. [86]. *Goals* have become more and more popular for the early phases of RE (e.g., [7], [103], [90]). Yue showed in [145] that goals in requirements models provide a criterion for requirements completeness.

Goal Oriented Requirements Engineering (GORE) must not be understood as a different or new form of RE. It is rather an already widespread supplement of the RE process and starts earlier than conventional RE[1]. As already manifested in its name, GORE puts much emphasize on *goals* which will be used to identify, describe and correlate requirements. Literature provides many definitions for such goals, we will use the definition from Lamsweerde [129]:

Definition 5 (Goals): *Goals are declarative statements of intent to be achieved by the system under consideration.*

Goals are formulated in terms of prescriptive assertions (as opposed to descriptive ones) [147]; they may refer to functional or non-functional properties and range from high-level concerns to lower-level ones [130]. In most goal-oriented approaches, the concept of goals expresses the same type of information as requirements. The difference is the level of abstraction. Goals explore *why* certain requirements are necessary for the system to be. Thus, they capture stable information and provide means to separate stable from volatile information which enables a better reuse.
There are several reasons to extend RE with the identification and formulation of goals. Lamsweerde describes in [130] the importance of goals for RE. The main benefits of goal-orientation can be summarized as follows:

- Achieving requirements completeness (e.g., [145], [8], [86])
- Specification of pertinent requirements, that are requirements that serve at least one of the identified goals [145].
- The refinement of goals provides means for well structured requirements specifications (e.g., [8], [86]).
- Goals may be satisfied by different alternative requirements. Thus, relationships between goals and requirements can help to choose the best one (e.g., [89], [86], [131]).
- Various stakeholders may have different conflicting requirements to the software. Goals can be used to identify and resolve these conflicts (e.g., [8], [131]).
- Separation of stable information from volatile [86].
- Goals drive the identification of requirements (e.g., [8]).

Goal identification is accomplished prior to requirements identification, requirements may be elaborated at the same time and more goals may be identified when discussing requirements. Sometimes goals are explicitly stated by stakeholders or are documented in already existing material. But in most cases they are implicit and goal elicitation has to be accomplished. More goals can be identified by *refinement* and *abstraction*, that is asking HOW and WHY questions about a preliminary set of goals and requirements. Thus, goal-oriented approaches are not inherently top-down. The relation between goals and requirements is much the same as between programs and design specifications: requirements

[1]In this thesis the basic form of RE will be denoted as 'conventional RE' in contrast to GORE and any other extended form of RE

”implement” goals [130].

In the following, we extend the definition for RE from Def. 2 and provide the following definition for GORE:

Definition 6 (GORE): *Goal-oriented Requirements Engineering is the branch of systems engineering concerned with the development of requirements through a systematic, iterative and co-operative process. This process involves the elicitation, negotiation, specification and validation of real-world goals and requirements for, functions of, and constraints on software systems. It is also concerned with the relationship of these RE artefacts to precise specifications of software behaviour, their evolution over time and across software families. Requirements and all related artefacts are documented in a Requirements Specification that needs to be validated regarding customer wishes, correct understanding and accuracy.*

2.3.1 Modelling Goals

Goals can be of different types. Generally, they can be classified into *functional* goals (describing the kind of service the system shall perform) and non-functional goals (referring to system qualities such as security, safety, performance, usability, flexibility, customizability, interoperability, and so forth [74], [130]. However, these taxonomies follow the style of requirement topologies and differ only on the level of abstraction on information. This basic typology has been specialized in different approaches in various ways.
Goals are linked together in a *goal model.* In order to relate goals with each other and other requirements artefacts, Lamsweerde et. al in [130] proposed *goal links.* Links between goals capture situations where goals positively or negatively contribute to other goals. They are called *support* links [130]. *Refinement* links may be used to relate a goal to a set of subgoals that must be satisfied for satisfying the parent goal (*AND-refinement*) or to relate a goal to an alternative set of goals where it is sufficient to satisfy one of these in order to satisfy the parent goal (*OR-refinement*) [130]. This is much similar to Feature Trees described by Czarnecki in (e.g., [27]). There is a good amount of work on linking goals with requirements artefacts, e.g., [101, 86, 85, 29].

Goal models have been proposed to be used for various tasks of RE. Apart from using goal models in requirements elicitation (e.g., [130], [143]), they have also been used to relate business goals to functional and non-functional system specifications in order to describe organisational change (Kaos [29], GBRAM [7], NFR framework [101], [122]). Finally, requirements validation can profit from goal modelling. Therefore, goal analysis techniques can be used to define stakeholders' criteria the system shall be assessed against [73] (GQM [11], [140]). Beside links between goals, goal models may be linked to other RE models. Goal formulations may be linked to specific objectives (e.g., entities, relationships, agents) in object models [29], GBRAM [8]. Furthermore, design models, business models or process models may be related to goal models as well.

2.3.2 Specifying Goals

Goals must be specified precisely to support RE tasks such as requirement elaboration or conflict management [130]. Literature proposes several kinds of specifications: informal (but precise), semi-formal and formal specifications. Informal specifications are used to describe what the goal name designates [147].
Semi-formal specifications may use a textual or graphical syntax to declare goals in terms of their type attribute, and links [130, 30].

Formal specifications assert the goal formulation in a fully formal system amenable to analysis [130]. Such assertions may for example be written in temporal logic.

2.3.3 Reasoning about Goals

Once goals have been modelled and specified it is possible to support goal-based reasoning for RE tasks. Yue describes in [145] a technique to verify that the requirements satisfy the identified goals and check whether this set of requirements is sufficiently complete for a certain goal set. This verification can be accomplished formally if goal specifications and domain properties are formalized (e.g., [96] or informally by using formal refinement patterns as proposed by Darimont in [31]. Another scenario for reasoning about goals are goal satisfaction methods, which are associated with reasoning about alternative requirements. Evaluating alternatives with respect to goal satisfaction has been addressed by qualitative and quantitative reasoning techniques [89]. The idea of qualitative reasoning techniques is to expose positive or negative influences of different alternatives on goals. Once specified, this information can be used to compare refinement alternatives, that is choosing the set of requirements that (qualitatively) best satisfies a certain goal set (e.g., NFR framework [101], Win Win model [64]). Quantitative techniques use numerical weights instead of qualitative contribution values such as '++' or "–" [89]. These numerical weights can be based on subjective (e.g., [134], [4], [46]) or objective criteria. While subjective criteria are often fairly applicable and verifiable, objective data with some domain-specific physical interpretation (e.g., the percentage of customers attended in 30 minutes) is better suited (e.g., [89]). Even probabilistic models have been proposed to quantitatively assess system security [94].

2.4 Summary

This chapter introduced a definition for RE and GORE. It can be highlighted that especially relationships between requirements artefacts play a crucial role for ensuring consistent requirements. Additionally, the validation of requirements is highly important in order to facilitate the further development of correct software. GORE has been noticed as a method for facilitating requirements completeness, resolving conflicts and separating stable information from volatile.

3 Ontology-Driven Requirements Engineering

Ontologies have long been used in the knowledge engineering community to perform conceptual domain modelling. In this domain, ontologies are interpreted as ”an explicit specification of a shared conceptualisation” [54]. Thus, an ontology is a formal description of objects and their properties, relationships, constraints and rules that govern those relationships. Ontologies contain explicitly defined and generally understood concepts and constraints that are machine understandable.

Requirements Engineering calls for an explicit domain knowledge. This domain knowledge generally resides in different areas, such as experiences, functionality, non-functional requirements, stakeholders and so on. Thus, it is necessary to concentrate this knowledge for the most appropriate application. Knowledge-driven techniques seem promising for this purpose. Kossmann et. al. in [79] define Knowledge-driven Requirements Engineering when Requirements Engineering is guided not only by a process but as well by knowledge about the process and the problem domain. In order to use knowledge-driven techniques, it is necessary to apply knowledge repositories that can be easily updated and utilised. Furthermore, inferencing and decision support must be applicable on such a repository. Ontologies are one possible way for representing, organising and reasoning about the complex knowledge that requirements documents embody and have been proposed to be used in different ways for RE.

This chapter investigates the role of ontologies for RE and summarizes their use as proposed in literature. It is structured as follows: definition for the term ontology, as well as an explanation of upper and domain ontology are given in Section 3.1. Section 3.2 provides background information about ontology components, modelling ontologies, ABox, TBox and reasoning ontologies. In advance, Section 3.3 introduces Ontology-Driven Requirements Engineering, including terms and definitions. Finally, a summary is given in Section 3.4.

3.1 Definitions

Although the term “ontology” has already been ill-defined in numerous papers and books, we need a definition of ontology that is used within this thesis. Therefore, we follow the revised ontology definition of Gruber [55] from 2009, neglecting all the facts about the historical background:

Definition 7 (Ontology): *In the context of computer and information sciences, an ontology defines a set of representational primitives with which to model a domain of knowledge or discourse. The representational primitives are typically classes (or sets), attributes (or properties), and relationships (or relations among class members). [...] Ontologies are typically specified in languages that allow abstraction away from data structures and* *implementation strategies; in practice, the languages of ontologies are closer in expressive power to first-order logic than languages used to model databases. For this reason, ontologies are said to be at the "semantic" level, whereas database schema are models of data at the "logical" or "physical" level. Due to their independence from lower level data models, ontologies are used for integrating heterogeneous databases, enabling interoperability among disparate systems, and specifying interfaces to independent, knowledge-based services.*

Furthermore, one has to distinguish between *upper ontologies* (a.k.a. top-level or foundation ontology) and *domain ontologies.*
Upper ontologies capture common objects that are generally applicable across a wide range of knowledge domains. It usually employs a core glossary that contains the terms and associated object descriptions as they are used in various relevant domain sets. There are several standardized upper ontologies available for use. Some well-known are Dublin Core[1] and the Gene Ontology[2].
Domain ontologies model a specific domain, which represents part of the world. Particular meanings of terms applied to that domain are provided by a domain ontology. For example the word "‘mouse"’ has different meanings. An ontology about biology would model the "‘animal mouse"’ meaning of the word, while an ontology about computers would model it as "‘computer-mouse"’ meaning.

3.2 Ontology Engineering

Ontology Engineering (or Ontology Modelling) is a subfield of Knowledge Engineering and concerned with methods and methodologies for building ontologies. Ontology Engineering aims to capture the *explicit* knowledge (e.g. of a software system or business procedure) for a particular domain. Explicit means that the concepts and relations are explicitly defined. This definition is realised by a formal language that is machine understandable and interpretable. Thus, an ontology is a catalogue of the types of things that are assumed to exist in a domain of interest from the perspective of a person who uses a language for the purpose of talking about this domain [123]. By interpreting the knowledge captured in the ontology, new, *implicit* knowledge can be reasoned.
Sowa in [123] distinguishes ontologies into informal and formal ones. Whereas *informal* ontologies may be specified by a catalogue of types that are either undefined or defined only by statements in a natural language, a *formal* ontology is specified by a collection of names for concept and relation types organized in a partial ordering by the type-subtype relation.
The following Sections describe the main aspects of Ontology Engineering and include components of ontologies, ontology languages and how ontologies are modelled and reasoned.

3.2.1 Ontology Components

Regardless of the language in which ontologies are expressed and the domain they describe, they share structural similarities, which are explained in the following :

- *Classes* describe concepts in the domain, types of objects, or kinds of things. A class defines a group of individuals that belong together because they share some

[1]http://dublincore.org/
[2]http://www.geneontology.org/

properties e.g. the class `Person` represents all Persons. Classes may have subclasses that represent concepts that are more specific than the superclass, e.g. `MalePerson` for the superclass `Person`.

- *Individuals* are specific instances of a class, e.g. `Paul` for the class `MalePerson`.
- *Attributes/ Properties* describe the instances of a class regarding the characteristics or parameters they can have. Properties can be used to state relationships between individuals or from individuals to data values, e.g. `eyeColour` or `address` for `Paul`.
- *Relations* are thy way in which classes and individuals are related to one another, e.g. `Person hasAddress address`.
- *Rules* are if–then statements describing the logical interference that can be drawn from an assertion in a particular form, e.g. `If Paul hasAdress German then Paul = German`.
- *Axioms* are assertions (including rules) in a logical form. They comprise the knowledge described in the ontology for a particular domain

3.2.2 Modelling Ontologies

Ontology Languages

Describing formal knowledge in an ontology requires a formal description language. Within the last decades, many ontology languages have been proposed. While all of them are based on predicate or description logic, they differ in expression potential, decidability and complexity. OWL (Web Ontology Language) [33] is the most famous description language for ontologies and has come to dominate the ontology layer of the semantic web. OWL is a W3C specification based on RDF (Resource Description Framework) which itself builds upon XML (eXtensible Markup Language). RDF is a formal language for representing metadata in the World Wide Web so that this information is suitable to be processed by applications. All resources are identified by URI (Uniform Resource Identifier), and the basic structure of RDF consists of *statements* of the form subject – predicate – object. Statements are resources themselves so that they can link to other resources (Reification). While these statements can be represented as a graph, readable to humans (RDF Model), an XML syntax (serialised for evaluation by machines; this complies to RDF Syntax) is defined for the interchange of RDF. The RDF Framework also consists of RDF Schema, a formalism that allows the definition of classes of resource, properties, relationships etc. Thus, RDF includes three components: RDF-Model, RDF Syntax and RDF-Schema. RDFS can be viewed as an simple ontology language. However, the semantic web stack contains a further layer on top of RDF - the ontology layer [34].

OWL is a RDF language (and therefore XML language, but behind this syntax it is a Description Logic (DL)[3] [10]. OWL has three increasingly-expressive sublanguages:

- *OWL Lite* is used for describing simple taxonomies (classifications hierarchy) with simple constraints or ontologies with lower expressiveness and complexity, but full decidability.

[3]"A logic that focuses on concept descriptions as a means of knowledge representation and has semantics which can be translated to first-order predicate logic." [34]

- *OWL DL* is based on a restricted DL. It provides the maximum expressiveness while retaining computational completeness (all conclusions are guaranteed to be computable) and decidability (all computations will finish in finite time) [98].
- *OWL Full* contains the same language constructs as DL, thus ontologies in OWL Full can use predicate logic expressions but are not computable anymore.

Furthermore, in accordance with DL, the ontological knowledge base is distinguished into *ABox* (data) and *TBox* (schema). The TBox describes a conceptualization, a set of concepts and properties for these concepts (e.g. `Person hasAddress Address`) the ABox holds the facts, which are various data and relationships (Individuals) compliant to the TBox statements (e.g. `All Teachers are Persons`). Together ABox and TBox statements make up the knowledge base.
However, various things cannot be expressed using OWL. Therefore, rule languages such as SWRL (the Semantic Web Rules Language) have been proposed. SWRL also allows arithmetic expressions to be used.

Ontologies can be modelled as *domain ontologies* and *upper ontologies*. Since domain ontologies are very specific, they are often incompatible. Thus, domain ontologies often have to be merged into a more general representation. This process is called *ontology merging*. While ontologies with the same upper ontologies can be merged automatically, ontologies without need to be merged mainly manually.

3.2.3 Ontology Reasoning

Reasoning is perhaps the main motivation to use ontologies instead of relational databases that only "store" data but do not allow for computing new implicit facts from the specified explicit data. (Logical) reasoning can be defined as "the process of drawing conclusions from premises using inference rules" [111]. The conclusion drawn by reasoning is called *inference*. It is distinguished between class and instance inferences. Class inferences infer classes or subclasses, e.g.

- A car driver is a person that drives a car.
- A car is a vehicle.
- A car driver drives a vehicle, so must be a driver. (according to [12])

Instance inferences infer instances to the according class(es).

- A bird is an animal that can fly.
- A blackbird can fly, so it must be a bird.

When modelling an ontology, the engineer must be aware of the distinction *open world* and *closed world*. Closed world assumes that all information that cannot be found by the reasoner does not exist, thus it will be computed as false. Negation as failure is related to the closed world assumption (CWA), as it amounts to believing false every predicate that cannot be proved to be true. In open world, we cannot assume that if we do not know something then it is false, it is just unknown. CWA is typically used in at least two situations: 1. when the knowledge base is known to be complete (e.g., a corporate database containing records for every employee), and 2. when the knowledge base is known to be incomplete but a "best" definite answer must be derived from incomplete information [135]. Ontology reasoning is accomplished with a *reasoner*, a software (also called inference engine) able to infer logical consequences from a set of asserted facts or axioms. Reasoner perform the following main tasks:

- *Consistency checking* finds contradictory facts in the ontology. An ontology is consistent if no contradictions exist. OWL already provides the formal definition of ontology consistency.
- *Concept satisfiability* determined whether the concepts (classes) can have any instances (individuals). If a class is unsatisfiable, an instance of that class causes the ontology to be inconsistent.
- *Classification* computes the subclass relations between every named class to create the complete class hierarchy. It infers individuals to a specific class due to their definition (as explained in the example of instance inference). The class hierarchy can be used to answer queries such as getting all or only the direct subclasses of a class.
- *Realisation* computes the most specific class an individual belongs to, e.g. the class `Bird` for the individual `blackbird`.

There is a huge amount of reasoners available, differing in speed, features (e.g. OWA/CWA, explanation of errors, usability) programming language, user interface and so on. While OWA is supported by almost all reasoners, CWA is not. Some well-known and widely used reasoners for ontologies are FaCT++ [127], Racer [24], TrOwl [3] and Pellet [1].

3.3 Ontology-Driven Requirements Engineering

Ontologies are useful for representing and interrelating various knowledge. Since RE involves knowledge capturing and analysis, there is a clear synergy between the ontological modelling of a domain and the modelling that a requirements engineer will perform during the requirements process [34]. Due to this overlap, numerous works dating back have addressed the use of ontologies in RE, e.g. [50], [102], [29]. All formalisms for RE need a particular conceptualisation, and almost all of them are reducible to first order logic [34]. Thus, they have much in common with ontologies that are constructed by using a formal language.
Since the semantic web[4] emerged, there has been a renewed interest in ontologies. An increasing amount of research is devoted to utilising semantic web technologies in RE (e.g. [97], [69], [79]) and software engineering in general.

However, according to Dobson et. al [34], there is a great potential for using ontologies in RE, including the representation of:

- The requirements model itself, imposing and enabling a particular paradigmatic way of structuring requirements

[4]"An extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation." [14]

- Acquisition structures for domain knowledge
- The application domain
- The environment [34]

Obviously a great deal of approaches propose ontologies to be used for any task of RE, for example for representing other reusable models that are relevant to RE. In contrast to Ontology-Driven Requirements Engineering they do not aim to address a large part of the RE process or do not embed in it. Usually, they are more or less detached from the process, concentrating on a specific problem of RE to be solved by ontological techniques (e.g. domain or upper ontologies for RE, goal satisfaction). Although these approaches make use of ontologies especially for RE there is no definition of ontology-Driven RE. In order to distinguish these approaches, we define ontology-*driven* Requirements Engineering in accordance to the definition 2 in Section 2.2:

Definition 8 (Ontology-Driven Requirements Engineering (ODRE)): *Ontology-driven (or ontology-based) RE describes a RE process or at least a RE method comprehensively aided by ontologies. Therefore, ontologies are involved for some or all tasks of the RE process. ODRE clearly states the method how to integrate a proposed ontological technique into a continuous RE process.*

3.4 Summary

Ontologies model a domain of knowledge. Several formal descriptions languages can be used for describing formal knowledge in an ontology, the most well-known is OWL [98]. Furthermore, the ontological knowledge base is distinguished into *ABox* (data) and *TBox* (schema). Together ABox and TBox statements make up the knowledge base. One has also to distinguish between upper ontologies and domain ontologies. Upper ontologies model general knowledge, applicable to various domains, while domain ontologies capture specific knowledge about one domain.

Since RE calls for an explicit domain knowledge, ontologies have great potential to be used in this area. Reasoning is perhaps the main motivation to use ontologies. Especially in RE it provides means for consistency checking and concept satisfiability of requirements and in RE domains. Ontological techniques have already been widely used in RE for different purposes. However, we distinguish Ontology-driven Requirements Engineering from single ontological methods in the manner of process or process parts to be supported by ontologies. ODRE is characterized by comprehensively involving ontologies for a RE process, a part of it or at least some RE tasks or method. Additionally, ODRE states how to integrate a proposed technique into a continuous RE process.

4 Feature Oriented Domain Analysis

Feature Oriented Domain Analysis (FODA) Feature-oriented domain analysis was first developed by the Software Engineering Institute in 1990. It "introduced the concept of *Feature Models* to Domain Engineering in an effort to represent the standard features within the family of systems (Software Product Line (SPL)) in the domain as well as the relationships between those features.
This chapter introduces Feature Oriented Domain Analysis, explains terms and provides definitions for the main aspects.

4.1 Introduction

Feature Oriented Domain Analysis aims to support the functional and architectural reuse. Therefore, a domain model is developed which represents a SPL which can then be refined into the particular desired system within the domain [136]. Therefore, the scope of the domain must be analysed (known as FODA context analysis) to identify not only the systems in the domain but also the external systems which interact with the domain. The Institute of Electrical and Electronics Engineers defines the term feature in [62] as:

Definition 9 ((Software) Feature): *"A distinguishing characteristic of a software item (e.g., performance, portability, or functionality)" [62]*

FODA comprises three analysis processes: (1) context analysis, (2) domain modelling and (3) architecture modelling. We use the following three definitions according to [70]:

Definition 10 (Application Domain): *"A set of current and future applications which share a set of common capabilities and data." [70]*

Definition 11 (Domain Analysis): *"The process of identifying, collecting, organizing, and representing the relevant information in a domain based on the study of existing systems and their development histories, knowledge captured from domain experts, underlying theory, and emerging technology within the domain." [70]*

The context analysis focuses on defining the bounds of a domain for analysis. During domain modelling, problems within the domain are described that shall be addressed by software and are documented as domain model. The products of this phase provide features of software in the domain, standard vocabulary and generic software requirements [70].

Definition 12 (Domain Model): *"A definition of the functions, objects, data, and relationships in a domain." [70]*

The identified software features are documented in a feature model for Software Product Lines[1] Software Product Line (SPL). Feature models capture the variability of software

[1] "A set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way." [2]

features and their dependencies. A feature *configuration* is “a set of features which describes a member of an SPL: the member contains a feature if and only if the feature is in its configuration. A feature configuration is permitted by a feature model if and only if it does not violate constraints imposed by the model” [125]. The most basic variability mechanism is the notion of optional, mandatory, alternative (xor) and or (at least one of the sub-features must be selected) features. Feature models are a part of the domain model that allow for customers to select from configurable requirements to specify a final system [70]. Thus, FODA ensures that a business can meet customers’ demands efficiently through reuse of technology [136].
Finally, architecture modelling “establishes the structure of implementations of software in the domain. The representations generated provide developers with a set of architectural models for constructing applications and mappings from the domain model to the architectures” [70].

4.2 Synergy between Feature Models and Ontologies

Kim in [76] explores the relationship between ontologies and feature models in three dimensions: notation, modelling philosophy, and role in MDSPL. According to [76], feature models form a notational subset of ontologies and describe concepts more specialized than those described by ontologies. Furthermore, both feature models and ontologies are domain models. Therefore, Kim et. al. suggest that feature models are “*views* on ontologies, namely, projections of the ontologies from different viewpoints” [76]. It is possible to realise a syntactic correspondence between a feature model and an ontology by establishing traceability links between feature model and ontology elements as proposed in [76]. Kim furthermore states that “semantically, the configurations of a feature model represent the set of viewpoint restrictions that can be applied to an ontology. The restricted ontology must represent at least one valid set of ontology individuals” [76].

4.3 Summary

This chapter briefly introduces the feature-oriented domain analysis. It can be stated that feature models form a notational subset of ontologies. Both models are domain models. Thus, feature models may be seen as views on ontologies.

5 Process Guidance

This chapter gives a brief overview of process guidance in general. It illustrates some shortcomings of guidance methods and provides definitions and explanations for the most relevant concepts of computer-based guidance.
The chapter is structured as follows: Section 5.2 provides information on process guidance and lists some of the reported shortcomings and problems. Furthermore, it describes the main parts of computer-based guidance. Section 5.3 gives a summary of this chapter.

5.1 Introduction

In the meantime, methods for RE are not only increasing in number, but also in complexity and understandability. It is fairly hard for an average requirements engineer to cope with all these methods and approaches. Apart from that, RE itself is a complex task that needs the ability to abstract, refine and relate various information. As stated in [120], guidance for RE needs to be far more knowledge intensive than in other activities. It is clearly beyond the simple automated control of sequences of activities detached from the current state of requirements knowledge which is provided by most methods in practice and by process Software Engineering environments. Although RE tools may provide a valuable UI and technology background, they lack for guidance support for the user. Thus, guidance plays a crucial role in the RE process

5.2 Terms and Definitions

Since Requirements Engineering is a process, guidance in this area aims to support this *process*. Process guidance is based on the process support paradigm of Software Engineering [25, 91]. In [108] Pohl defines process guidance as a set of activities in order to provide the process performers with assistance regarding allowed steps at any point during process enactment. Guidance may be provided synchronously or asynchronously with the actual process performance [72]. In contrast to process automation, process guidance aims to support the process performer in choosing which task to complete next instead of automatically accomplishing a certain task without user interaction. Thus, process guidance falls in the category of *prescriptive support* [72]. Rather than dictating which task to perform next, process guidance provides a set of applicable tasks that can be dynamically selected depending on the current state and context of the process [52, 108, 116].

A *process guide* is a reference document for an intended process. It provides assistance for process performers in carrying out that process [75]. According to Kellner [75], benefits of process guides are:

- Facilitation of communication between process performers
- Support for tracking work by capturing process event information

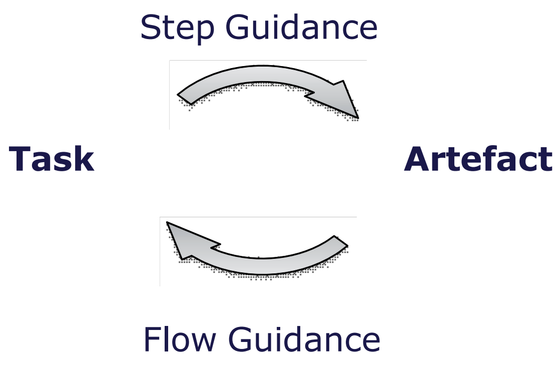


Figure 5.1: Step and flow guidance

- Increase efficiency of process performers
- Allows process performers to return to a partially-complete process

Process guides must not necessarily be computer based. Often printed guidebooks, standards, process and procedure manuals are used in industry [75]. According to [75], some of the most important drawbacks of such documents are:

- Traditional guidebooks, even those made available on intranets, often lack key information.
- Readers of guidebooks can't easily navigate through the pages when their strategy of understanding does not match the document's flow.
- Guidebooks either contain a mixture of information for different audiences, or multiple documents tailored to specific needs require a common process description.
- Guidebooks are not designed to store information about the status of a project.
- Version control, especially of example development documents, is not well supported.
- Distribution of a new version of a process to process participants is unreliable.

Since such documents are frequently not felt to be useful, computer based guidance is becoming more and more attractive. However, the most important and central function of process guides is to facilitate process understanding. Information technology can most effectively provide this function in contrast to paper documents, etc.

Si-Said et. al. [120] describe the process of RE as decision-oriented. Thus, guidance assumes that there is an intention to achieve. In order to fulfil this, support is needed. Si-Said distinguishes between *step guidance* and *flow guidance* as illustrated in Figure 5.1 and explained in the following paragraphs.

5.2.1 Step Guidance

Guidance can always be offered when the current intention of the user has been recognised. Thus, *step guidance* is always related to a *step point* consisting of a situation and an intention. The situation describes the actual artefact (e.g. a goal or requirement), the intention describes which activity shall be executed regarding this artefact, e.g. goal refinement. The achievement of an intention causes a change of that artefact. For this reason, step points are constructed from all reasonable combinations of two sets: the set of all domain artefacts and the set of all intentions. Thus, step guidance provides guidance in means of guidelines. They include instructions for every step point how to execute a certain intention. After executing a certain activity the user is guided in the decision which task to accomplish next. This is realised by the *flow guidance*, explained in the following section.

5.2.2 Flow Guidance

Flow guidance supports the process performer to progress from one guidance point to another, using a certain *strategy*. This strategy is a way to progress in a process and is domain specific. Flow guidance is based on different guidelines and strategies. These guidelines help to select a decision on how to progress further in the process [120].

5.3 Summary

Process guidance provides assistance for process performers in carrying out a process. Since documents such as guidebooks, websites or procedure manuals are not felt to be useful, computer based guidance is becoming more important. Especially the highly creative and complex task of RE can benefit from support for requirements engineers.
Since RE is decision-oriented, any guidance support must acknowledge this. Thus, guidance assumes that their is an intention to achieve in which the process performers is to be assisted in. Step points (or guidance points) consist of a situation and intention. Step guidance provides guidance in the form of guidelines at certain step points. Flow guidance supports the process performer to progress from one guidance point to another.

6 State-of-The-Art and Related Work

This chapter investigates different approaches related to the context of our thesis and provides an overview about ontological techniques that are used for RE tasks.
The chapter is structured as follows: we first give an overview about GORE in Section 6.1. In Section 6.2, we discuss state-of-the-art in ontology-driven RE. Furthermore, we present approaches related to guidance in RE in Section 6.3 and conclude with implications for this thesis in Section 6.4.

6.1 Goal-oriented Requirements Engineering

The following sections discuss current GORE approaches. It starts with well-known approaches and concludes with an introduction of further interesting GORE methods and ideas.

6.1.1 The NFR Framework

The NFR Framework is one of the best known Software Engineering approaches regarding GORE. It was proposed in [101] and further developed in [29], [84] and [23]. The NFR Framework concentrates on representing and modelling organisational goals and their relations to operational system components. Non-functional requirements (NFRs) are put foremost in the developer's mind [86]. Therefore, the framework provides a process-oriented approach for dealing with NFRs. Furthermore, the approach deals with ambiguities, trade-offs, priorities, selecting operationalizations, supporting decisions with design rationale, and the evaluation of the impact of decisions.

The NFR methodology aims to support the process of requirement elicitation and decomposition of non-functional requirements and the identification of possible operationalizations. High-level goals are progressively refined to NFRs until constraints, objects and operations that are assignable to individual agents are obtained [73]. NFRs are systematically modelled and refined to expose positive and negative influences on different requirement alternatives. The main modelling elements in the NFR Framework are *softgoals*. The framework supports three types of them:

- *NFR softgoals* represent non-functional requirements [86]
- *Operationalization softgoals* describe lower-level techniques to satisfy NFR softgoals [86]
- *Claim softgoals* allows the analyst to record design rationale for softgoal refinements, priorities, contributions, etc. [86]

In order to choose the requirement that best satisfies a given goal or set of goals, softgoals may be refined using AND and OR refinements. Positive or negative contributions can be

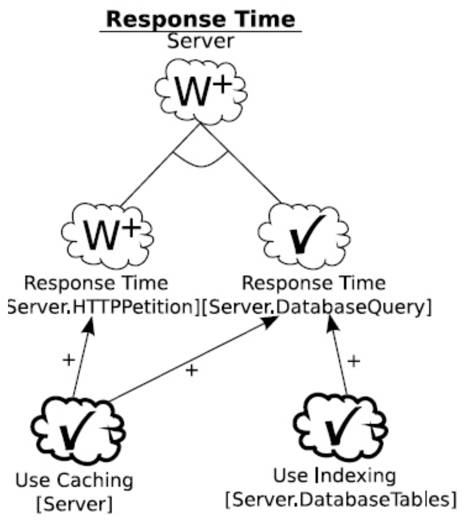


Figure 6.1: Example of a Softgoal Interdependency Graph [117]

captured for softgoal interdependencies.

The NFR framework provides a main graphical modelling tool, the *softgoal interdependency graph* (SIG). SIGs represent softgoals, their interdependencies, softgoal refinements (AND/OR), softgoal contributions (positive/negative), operationalizations and claims. The framework comes with a label propagation algorithm to choose the best alternative for satisfying the high-level non-functional requirements. This algorithm works down-top, receiving contributions from a number of softgoals refinement links. Finally, these alternative operationalization can be analysed by the developer [86].

6.1.2 *i**/Tropos

The agent-oriented modelling framework *i** [144] was developed for modelling and reasoning about organisational environments and their information systems. The framework can be used for several purposes, e.g. Requirements Engineering and Software Process Modelling. Since we are only interested in RE, we only refer to RE-relevant aspects in our description of *i**.

*i** can be used for both early and late phases of RE. Therefore, it supports modelling activities that take place before the system requirements are formulated. During the early phase, the framework is used to model the environment of the system. The resulting *i** models help understanding why a new system is needed [86]. During the late phase of RE, the *i** models are used to check the system configurations regarding their satisfaction of functional and non-functional requirements.
The main concept in *i** is that of an intentional actor. These actors have intentional attributes such as goals, beliefs, abilities, and commitments. Dependencies between these actors allow one actor to achieve goals, perform tasks, and furnish resources he could not achieve on its own [144]. Therefore, each actor can use various opportunities to achieve more by depending on other actors. Actors are seen as strategic in the sense that they are

concerned about opportunities and vulnerabilities and in finding a balance in (re)arranging their environments. The actors are used to represent the system's stakeholders[1] as well as the agents of the new system [86].
Actors can be agents, roles, and positions. Whereas *agents* are concrete actors, systems or humans with specific abilities, a *role* is only an abstract actor with expectations and responsibilities. By separating these actors from each other, the social context of software can be analysed more efficiently. The dependencies between actors are classified into four types based on the subject of dependency: *goal*, *softgoal*, *task*, and *resource*.

The *i** framework consists of two main modelling components. The Strategic Dependency (SD) model and the Strategic Rationale (SR) model. The SD model captures all the dependencies among actors in an organizational context and allows for their analysis. SR models are used to describe stakeholders interests and concerns, and how they might be addressed by various system configurations and environments [144]. This can be explicitly described in terms of process elements, such as goals, softgoals, tasks, and resources, and relationships among them. While the SD model only focuses on the external relationships among actors, the SR model provides the capability to analyse the internal process within each actor in great detail [86].

The SR process elements are related by *decomposition* links (AND/OR) and *means-ends* links. Whereas decomposition links connect a goal/task with its components (softgoals, subtasks, etc.), means-ends links are often used with goals to specify alternative ways to achieve them [86]. Additionally, *contribution* links, similar to the ones in the NFR framework, are used to specify levels of contribution to the softgoals. Softgoals are used as a selection criteria to choose the system configuration that best satisfies the non-functional requirements.

The *i** meta-framework is described in the language Telos [22]. Thus, *i** models can be analysed in various ways (e.g. consistency checking between models). *i** is the basis for the requirements-driven agent-oriented development methodology Tropos [22]. Tropos guides the user through four phases of RE:

- "Early requirements, concerned with the understanding of a problem by studying an organizational setting; the output of this phase is an organizational model which includes relevant actors, their respective goals and their inter-dependencies.
- Late requirements, where the system-to-be is described within its operational environment, along with relevant functions and qualities.
- Architectural Design, where the system's global architecture is defined in terms of subsystems, interconnected through data, control and other dependencies.
- Detailed Design, where behaviour of each architectural component is defined in further detail." [22]

[1]A person or organization that has a (direct or indirect) influence on a system's requirements. Indirect influence also includes situations where a person or organization is impacted by the system [47].

6.1.3 KAOS

The KAOS (Knowledge Acquisition in automated Specification [29] or Keep All Objects Satisfied [133] methodology is a GORE approach. KAOS is described in [133] as a multi-paradigm framework that allows to combine three levels of expression and reasoning: semi-formal for modelling and structuring goals, qualitative for selections among alternatives and formal for more accurate reasoning when needed [86]. A generic ontology forms a metamodel for requirements.

KAOS includes the concepts of *objects*, *operations*, *agents* and *goals*. Objects can be entities, relationships or events in the composite system. Operations are input-output relations over objects with pre-, post- and trigger-conditions. An agent is a kind of object that executes operations. Agents are active components such as humans, devices, software, etc. Goals in KAOS can be functional (referring to services) or non-functional (quality of services). They are organised in the usual AND/OR refinement-abstraction hierarchies. A goal refinement is completed when every subgoal can be realised by some individual agent assigned to it. The *requirement* in KAOS is defined as a ”goal under the responsibility of an agent in the system-to-be” [86] whereas *expectations* are defined as ”a goal under the responsibility of an agent in the environment” [86]. KAOS supports different types of goals [29], for example *satisfaction* goals that are functional goals concerned with satisfying agents requests.

A KAOS specification is a collection of *goal model*, *object model* and *operation model*. In the goal model, goals are represented and assigned to agents. The object model is a UML model that can be derived from formal specifications of goals. The operation model defines various services the software agents provide [86].

6.1.4 GBRAM

The GBRAM method [7], [8] focuses on the initial identification and abstraction of goals from various information sources. It supports the elaboration of goals (*goal analysis*) and *goal refinement* until they are translated into operational requirements for the system specification. GBRAM has been influenced by KAOS. It also uses the concept of *agents* which are ”entities or processes that seek to achieve goals within an organisation or system based on the implicit responsibility that they must assume for the achievement of certain goals” [73]. The method provides goal-identification heuristics and a set of recurring questions to guide the process and support the practitioner in it.

GBRAM distinguishes between *achievement* and *maintenance goals*. Goals are decomposed into subgoals and finally refined to requirements with an operational definition. This process is called *operationalization*. Goal *obstacles* are identified and defined in order to specify which behaviour or other goals may prevent or block the satisfaction of a certain goal. Another activity required by GBRAM during goal refinement is the identification of goal precedence in order to denote that certain goals must be completed before others. The method suggest asking questions like ”What goal(s) must follow this goal?” or searching for agent dependencies [86].
GBRM also integrates scenarios as behavioural descriptions of a system and its environment.
These concepts are consolidated into a set of goal schemas [73]. These schemas provide only a textual representation of the system requirements and their interrelation with the

system goals. A graphical notation is not provided.

GBRAM is partly supported by GBRAT [6], a web-based requirements analysis tool and serves as a medium for collaborative working. However, the tool only provides means to specify, view and order goals gathered beforehand. In order to create a goal in GBRAT, the user has to complete a form with all necessary information. It also supports the specification of precedence relations.

6.2 Ontology-Driven Requirements Engineering

The following sections discuss current approaches for RE aided by ontologies. We first discuss approaches that allow for building a requirements ontology from text documents such as requirements specifications or other business documents. Afterwards, we discuss the application of ontologies that aim to enable reasoning in RE. Finally, we give an overview about approaches that support ontology-driven RE according to the definition in Section 8.

6.2.1 Mining Requirements Ontologies from Text

According to a market research study published by Luisa et al. in [93], the overwhelming majority of requirements are written in natural language (NL). This is reasoned by the fact that requirements are usually formulated by customers in natural language, even if it is only for the initial conception until they are somehow dealt with by software developers. Natural Language has the advantage to be understood by all stakeholders, albeit differently by each due to the inherently ambiguous nature of natural language. However, it is of main importance for the software developer to identify the concepts and relations used by the writer who is an expert of that domain. These concepts and relations are meaningful in order to understand the information exactly as purposed by the writer. This is where text mining techniques can be applied. On the basis of extracted concepts and relations from from NL documents, a domain ontology may be constructed, which is according to [18] itself a valuable RE product. This process is called *ontology learning*.

Kof [78] proposes a method to build a domain ontology from requirement documents provided in natural language. Therefore, terms are extracted from text and clustered, a taxonomy is built. Associations between the extracted terms are identified and make up together with the associated terms the domain model. While the formatting, tagging, parsing and concept cluster building are automatic, the identification of cluster intersection and taxonomy building as well as deciding which associations are sensible remains interactive and, thus, need human interaction.
There is a good amount on literature addressing NL approaches for ontology learning (e.g. [95], [28], [45], [19]). Although they do not explicitly concentrate on ontologies for RE, it makes no difference for which application area or domain the ontology in the approach is built. The content of the ontology always relies on the initial text document and its content. Thus, these approaches can be used for building ontologies in the RE domain as well.

6.2.2 Ontologies in Requirements Engineering

Ontologies in RE can be applied in various ways. They may either serve as domain ontologies, capturing concepts and relations of the software domain and thus become a valuable source for RE. Or they may be used to enable reasoning for different purposes, e.g., goal satisfaction analysis. Furthermore, ontologies may support the RE process in general, coming with additional software support such as process guidance or validation. Castañeda in [21] identifies three different kinds of ontologies in RE:

- *(Application) Domain Ontology* represents the knowledge of the application domain as well as business information required for software building. It also includes the semantic relationships between the concepts. Domain ontologies help to identify dynamic and changing requirements by understanding the domain.
- *Requirements Ontology* captures the core requirements and their dependencies and relationships. This ontology can be used during requirements elicitation in order to reduce ambiguous requirements and avoid incomplete requirements definitions. A requirements ontology can later be used for validation and verification purposes.
- *Requirements Specification Document Ontology* describes the structure of requirements specification documents in order to reduce insufficient requirements specifications.

Jureta propose in [68] a core ontology for requirements (named CORE) for capturing basic stakeholder concerns during RE, namely beliefs, desires, intentions, and evaluations. The ontology grounds on the foundation ontology DOLCE (Linguistic and Cognitive Engineering). Jureta et al. proposes four relationships to relate instances of concepts (speech acts) in CORE: refine, approximate, compare and evaluate. These relationships do not relate requirements and are stored outside the ontology.

6.2.3 Ontologies for Reasoning in RE

Ying et al. [141] present an algorithm for detecting and resolving inconsistencies of domain ontologies for RE. The domain ontology is considered to be a thesaurus containing all the information about domain concepts and their role. Thus, inconsistency of domain knowledge can be found by ontology consistency checking. The algorithm is based on the Tableaux algorithm, consistency rules are formally defined and semantic checking is proposed to resolve detected inconsistencies. However, consistency checking is only performed regarding the logical consistency of the ontology. That is checking whether the ontology is satisfiable, which means that there is no contradicting information in the ontology.
Zhu et al. in [150] propose an ontology-based approach for inconsistency measurement of requirements specifications based on a requirements refinement tree. Therefore, requirements are stepwise decomposed until a requirement can be realized. During this process of requirements refinement, external requirements from the customers are extracted first. In case this requirement is too complex to fit a problem pattern, it is decomposed and smaller systems will cooperate later to realize that requirement. Then, each of this smaller systems has external requirements which are captured and decomposed again, until no further decomposition is possible. This refinement process is represented as AND/OR tree. A domain ontology is used as a infrastructure for the refinement of requirements. If

the subsystem requirements can be described by instances of the domain ontology, then further decomposition is not necessary. The author's goal are requirements specifications that are then comparable due to iterative refinement.

Kassab and Daneva in [71] developed an NFR ontology in order to consider non-functional requirements early in software development. The NFR ontology defines the meaning of a set of concepts for the NFR domain. The ontology allows for capturing relationships of NFRs with functional requirements in the form of association points. NFRs can be further decomposed (AND/OR decomposition) and have operationalizations, that is a refinement of a NFR into a solution in the target system (operations, functions, data representations and architecture design decisions) that will satisfy the NFR. An individual NFR may interact with another NFR and can hinder (negative interaction) or help in (positive interaction) the achievement of other NFRs. This can be used to search for operationalizations that have positive/negative effects on NFRs. The operationalizations that are identified to have a negative effect on other NFRs sharing the same association point with their parent NFRs, are used to identify potential conflicts.

Letier and Lamsweerde in [89] present a method for reasoning about partial goal satisfaction for RE. The aim of the authors is to support decision-making during RE. Therefore, they quantify the impact of alternative system designs on the degree of partial goal satisfaction and non-functional goals. In contrast to quantitative reasoning techniques (low, medium, high satisfaction) that are usually based on subjective criteria, their approach focuses on objective criteria with some domain-specific physical interpretation. Therefore, objective functions and quality variables are used to specify partial goal satisfaction in a semi-formal but precise, application-specific way. Rules are provided to identify the impacts of alternatives requirements options on degrees of goal satisfaction.

6.2.4 Ontology-driven RE

Kossmann et al. developed the OntoREM Metamodel, a comprehensive specification of the ODRE methodology, including the underlying concepts in the RE domain and relationships between them [79]. OntoREM consists of the OntoREM Metamodel ontology and a number of domain ontologies. The concepts *OntoREMGoalHierarchy* and *OntoREMRequirement* with their described relationships define "templates" of goals, soft goals and requirements that are used when creating an instance of a goal, soft goal or requirement and are linked to the relevant areas of available domain ontologies. These requirement instances represent the main outcomes of OntoREM, i.e. the requirements specification for a given domain in a given context [79]. OntoRem comes with a workflow for the RE process. With the newly developed tool OntoRAT, requirements can be analysed regarding status, goal, softgoal, requirements and traceability as part of the OntoREM process. All requirements are managed with DOORS[2] from IBM which has been integrated due to a more intuitive and user-friendly interface.

Lee et al. [87] present an approach to elicit and analyze domain requirements based on an ontology. The metamodel proposed is adopted from [100] and improved. The central model element is a *domain requirement* which is separated into functional and

[2]http://www-01.ibm.com/software/awdtools/doors/

non-functional requirements. Functional requirements can be further refined to primitive requirements which can also be stated as optional. Optional properties are realized as variability in software product lines. Functional requirements can be interrelated and non-functional requirements may influence the functional ones. However, relations are kept quite simple. Common and variant points are also included in the metamodel. The approach furthermore enables to analyze the completeness of requirements. Completeness is reached when concepts and relations of the domain are a subset of the stakeholder's needs. This is detected by inherent relation.

In [69], Kaiya et al. propose a method that allows for requirements analysis of a functional requirements specification (FRS). The method is based on a domain ontology and a mapping to the requirement specification. The mapping between the FRS and the ontology is assumed to be achieved by the requirements analyst. The ontology consists of a thesaurus and inference rules. Three kinds of semantic processing are supported: (1) detecting incompleteness and inconsistency, (2) measuring the quality of a specification w.r.t. its meaning and (3) predicting requirements changes based on semantic analysis on a change history. The completeness of a FRS is based on a previously modelled domain ontology. Thus, everything from the domain must be part of the ontology. Since the domain ontology is assumed to be complete, a complete FRS must be inconsistent, if the ontology contains contradict relations.

Riechert et al. developed the ontology SWORE [115] to support the RE process semantically. SWORE provides a semantic structure for capturing requirements information and linking this information to domain- and application-specific vocabulary. The core of the ontology was adopted from [130]. SWORE contains concepts for goal, scenario and requirement. All of them are defined by stakeholders whose interaction are of main interest. In order to allow the various stakeholder a collaborative elicitation of requirements, SWORE has been integrated into the semantic collaboration platform SoftWiki [57].

6.3 Guidance in Requirements Engineering

Si-Said and Rolland developed MENTOR, a process guidance engine suitable for any process governed by a ”way-of-working” [120]. MENTOR uses point guidance in a certain *situation* where the engineer needs to make a *decision* on which task to accomplish. Therefore, guidance points, consisting of a (meaningful) situation and decision, have been defined. The guidance is governed with guidelines associated to each guidance point. Flow guidance supports the engineer in advancing the RE process. It is also realised by guidelines based on a number of different strategies. These guidelines help in selecting the next guidance point. Although MENTOR was directed towards Requirements Engineering, it is generic and can guide the enactment of any process modelled in terms of the process meta-model proposed by [120].

Kavakli developed the GDC (Goal-driven Change) approach. GDC provides a systematic way of reasoning about the RE process in terms of goal modelling and supports the user by a process guidance framework. G-Map is a web-based software tool that supports the assembling and execution of goal-driven methods. A methodology roadmap is used as a metaphor to visualize alternative goal-driven ways of working. Web-based technology enables the navigation within this roadmap. Guidance is provided through information which shall support project participants in deciding which action to perform next. G-Map leaves the control of process execution to the user and does not impose any constraints on the process [73]. However, G-Map guides the user through different goal-driven methods and thus, supports the application of goal-driven methods.

An interesting approach is described in [39]. Farefelder et al. developed a prototype of a semantic guidance system that assists the requirements engineer in capturing requirements by using semi-formal representation. Their approach aims to prevent specifying and finally resolving incorrect requirements. Instead, the prototype automatically proposes at least parts of the requirements by using information originating from a domain ontology. On these suggestions the requirements engineer can build on to define requirements.

6.4 Implications for the Thesis

In the previous sections, we have highlighted a number of promising approaches from research. The prevalent conclusion from these approaches is that goals can be used to capture stable information and provide means to better structure, refine and manage requirement. Ontologies have been shown to be applicable in many ways throughout RE. Especially reasoning seems to be a powerful technique to address the completeness and consistency of requirements specifications. The main aspects are summarized below:

- Goals pertain stable (in contrast to requirements) and can be reused.
- Relationships between goals and requirements can help to choose the best requirement (goal satisfaction).
- Goals can be used to identify and resolve conflicts resulting from different conflicting requirements brought in by stakeholders.
- Goals may be used to verify that the requirements satisfy the identified goals.
- Goals provide a meaningful criterion for sufficient completeness of a requirement specification.
- Ontologies are useful for representing and interrelating requirements knowledge.
- The requirements model can be represented by an ontology.
- Ontology reasoning can be used to detect incompleteness and inconsistency of requirements specifications.
- Ontological techniques allow for goal satisfaction analysis and further requirements analysis.

Analysing the previous state-of-the-art also exposed some problems and shortcomings that can be summarized as follows:

- The detection of conflicts between requirements is often only addressed regarding refinement relationships or conceptual overlapping.
- Most techniques consider only binary conflicts (conflicts between two requirements).

- Relationships among requirements artefacts are not sufficiently captured and analysed.
- There is a lack of systematic support for detecting and resolving requirements inconsistencies.
- Inconsistencies are not explained to the requirements engineer, options for resolving them are not suggested.
- Current RE tools do hardly cover requirements traceability.
- There is no applicable guidance for requirements engineers during RE.

Part II

The Conceptual Solution

7 Challenges and Requirements for improved RE

In this chapter, we define the problems and challenges we aim to tackle and also state the requirements for our approach. We envision the proposed approach and describe the ideas behind it. All of the concepts for OntoReq may also be used for different design tasks. Therefore, we put much emphasize on design and implementation independence. Thus, different design alternatives and technical solutions are possible. However, the technical solution for OntoReq is explained in detail in Section III.

This chapter is structured as follows: Section 7 gives a summary of challenges for GORE based on our problem analysis in Part I. We deduce requirements from these challenges and give some short examples for their application. The conceptual solution for the OntoReq approach is presented in Section 8. We also describe the scientific rationale and key solution ideas based on the challenges and requirements given in Section 7. Here, we also clarify the scope of our research, the requirements to be realised and the limitations of our approach (Section 8). In Section 8.4, we define the requirement artefacts of OntoReq. Furthermore, the various relationships between requirements and their application is exemplified by short examples. Section 8.4 presents concepts for the knowledge repository. The various requirements validation services provided by OntoReq are explained in depth in Section 8.5. Here, we illustrate the verification of completeness and consistency and propose measures for the quality of a software requirement specification. In addition, we briefly describe the underlying goal satisfaction method and the semi-automatic selection of requirements that best suit a specified purpose (e.g. goals, uncriticality). Section 8.6 provides a concept for providing guidance for GORE within OntoReq. Finally, we give a summary in Section 7.3. As already discussed in Part I of the thesis, experiences from industry, research and analysis in the last two decades brought up a bunch of shortcomings and problems in Requirements Engineering. At this point, we give a brief overview about the most significant challenges in order to derive requirements for an enhanced Requirements Engineering process.

7.1 Challenges for Requirements Engineering

We divide the challenges for RE into the following six categories:

1. **Technological**: problems based on changes or challenges in general technology and/or RE, e.g. tools, IT technology, etc.
2. **Organisation/Management/Communication**: shortcomings due to business (process) organisation, management of resources (e.g. time, money, human resources), communication between stakeholders, customers and requirements engineers
3. **Requirements**: problems originating from the nature of requirements themselves
4. **Requirements Engineering Process**: challenges regarding the process of requirements elicitation, analysis, etc., structure of the process, methods, traceability and guidance
5. **Requirements Documentation**: problems with the type of documentation, documentation structure, traceability, models, reuse, comprehensibility, etc.
6. **Validation/Verification**: challenges regarding checking specified requirements *during* the RE process

We tried to consider representative challenges that have been reported in various papers (referenced at the appropriate headwords) while also concentrating on the most meaningful. However, this categorization and mapping of challenges does not claim to be complete. Sometimes one and the same challenge occurs in different categories on purpose. This is justified by the fact that a challenge affects multiple categories. For example, the complexity of requirements must be treated in management, documentation and verification. Thus, this challenge will later be translated to different requirements.
This categorization has the purpose to exemplify problem areas and increase understandability of the challenges reported. References after the statements give a selection of the most significant sources for the problem described.

Technological

- New types of requirements due to changing technology

Organisation/Management/Communication

- Comprehension of stakeholder's needs [48]
- Requirement engineers may have difficulties in understanding the application domain and business process [56]
- Conflicting or redundant requirements due to different stakeholders [9]
- Conflicting user/client views and requirements
- Inadequate knowledge of stakeholders [56]
- Insufficient information provided by stakeholders/ customers [56, 151]

- Imprecise and vague process descriptions in manual-based documentation of requirements [151]
- Inadequate or not applicable requirements management and configuration in manual documentations
- Confirmation of requirements specifications
- Constantly changing requirements
- Management of complex requirements [15]
- Reusability of requirement information

Requirements

- Poorly specified NFRs (by stakeholders) [40, 151]
- Ambiguous NFRs

RE Process

- Poorly specified NFRs [40, 151]
- Error prone, incomplete, inconsistent and ambiguous requirements specifications [41]
- Constantly changing requirements
- Complexity of validation and changes of requirements [124, 119]
- Difficult verification of requirements
- Need to consider many factors in order to choose the best approach
- No adequate guidance

Requirements Documentation

- The single requirements specification document (manually produced paper documents) is time consuming, expensive to produce and rarely maintained [41]
- Insufficient information provided by stakeholders/ customers [56, 151]
- Conflicting or redundant requirements due to different stakeholders [9]
- Conflicting user/client views and requirements
- Error prone, incomplete, inconsistent and ambiguous requirements specifications[41]
- Requirements Specifications (manually produced paper documents) are hard to read and often not up-to-date [41]
- Classification of extensive data [56]
- No documentation of reasons and decisions behind solutions or changes

- Functional requirements are engaged with difficult explanations and complicated structure models which are difficult to reveal [60, 9]
- Correctness of SRS
- The level of detail required in a requirements specification differs greatly depending on the type of product that is being developed [121]
- Reusability of requirement information
- Little guidance on how to achieve traceability of requirement information [113]

Validation/Verification

- Verification of NFRs is difficult [40]
- Verification of requirements is difficult
- Validation and changes of requirements are complex [124, 119]

These above mentioned challenges and shortcomings can be transferred to a number of requirements for a systematic support in Requirements Engineering, independent of a specific tool, technique or method. In the following we explain these requirements and their deduction.

7.2 Deduction of Requirements for a Systematic Support for RE

We use the categories of RE challenges from the previous section to deduce the following requirements. In order to use these requirements for different purposes (e.g., RE software development, RE methods and techniques, etc.) we keep them as general as possible. Thus, they must be further interpreted and refined for a specific purpose.

Requirements due to Technological Changes
Because of continuously arising new software technology spaces[1], hardware and other technical solutions, software development will always be faced with new or changing requirements and new kinds (or types) of requirements that cannot be found in any classification so far. The most important challenge is to become aware of such new types of requirements. The next step would be to identify the concretely requirements and define them, far better, to include them in existing requirement classifications. Thus, we define the following requirement:

TechReq 1: *New types of requirements must be identified and defined. They should be integrated into existing requirement classifications.*

Requirements Regarding Communication, Organisation and Management in RE
Some major problems when eliciting requirements are communication specific. The main cause lies in the comprehension between stakeholders (users, customers, etc.) and requirements engineers. Requirements from stakeholders are usually not communicated in the way

[1]A technology space is a working context with a set of associated concepts, body of knowledge, tools, required skills, and possibilities. It is often associated to a given user community with shared know-how, educational support, common literature and even conference meetings. [82]

a requirements engineer can use them. Thus, he must interpret, interpolate and correct the stakeholder requests. Here, misunderstanding often arises, inparticular if the requirements engineer is not familiar with the application domain. These facts needs to be acknowledged and supported with any possible method and lead to the following requirements:

OMCReq 1: *The communication between stakeholders and requirements engineers must be supported. For this, appropriate techniques, methods and other resources (e.g. time) must be accessible. The gathered requirements must finally be validated and confirmed by the stakeholders before software development proceeds.*

OMCReq 2: *Requirements engineers must be familiar with the application domain of the software to be developed.*

Further, it cannot be expected that stakeholders provide a complete list of requirements or sufficient information at all. Obviously it is the responsibility of the requirements engineer to identify and gather missing information. However, since this is not an easy task, this activity should be supported by a software solution. Furthermore, the completeness and correctness of requirement information can be improved if requirements information can be reused. Therefore, we deduce the following requirements:

OMCReq 3: *The identification of insufficient and incomplete requirements must be supported.*

OMCReq 4: *Requirements information shall be reused if possible.*

Often several stakeholders are involved during RE. This naturally causes conflicting and redundant requirements that must be treated.

OMCReq 5: *Inconsistency in requirements and information about requirements must be identified and resolved. This must be followed by (perhaps repeated) validation of requirements.*

The following requirements are based on challenges in requirements management. Requirements are complex and often requirement specifications may contain hundreds of requirements. Thus, it is of main importance to handle this complexity in an appropriate way. Therefore, we define:

OMCReq 6: *The management of complex requirements (e.g. data, information, relationships, etc.) must be supported in an appropriate way. Stakeholder decisions and changes must be captured. All available requirement information must be accessible at any time.*

OMCReq 7: *The management of a huge number of requirements must be supported. This includes appropriate support for readability, understandability and documentation of requirements, preparation of requirement specifications, validation and verification. The documentation must be held complete and consistent. The requirement specification must be up-to-date at any time.*

Requirements are naturally changing during software development. New requirements may be identified and must be integrated into the existing requirement specification. This needs to be considered for requirement management:

OMCReq 8: *Change management must be supported. This includes capturing of changes and their justification (reasons, decisions, etc.) and traceability of changes. The requirement specification must be kept consistent after changes.*

Requirements Regarding the Nature of Requirements
In the following, we list requirements resulting from the nature of requirements.
Requirements are usually classified into functional and non-functional (also quality-) requirements. Since functional requirements can be specified more concrete and can be assigned to metrics and test-cases, non-functional requirements (NFRs) are often forgotten or specified insufficiently. Thus, we deduce the following requirements:

StrucReq 1: *The identification of NFRs must be supported.*

StrucReq 2: *NFRs must be specified precisely and connected to soft-metrics and test-cases for verification.*

StrucReq 3: *NFRs must be validated by stakeholders and verified if possible.*

Requirements Regarding the Requirements Engineering Process
As already described in the previous chapter, RE is a highly complex task. Thus, improvement of and support during the process itself will lead to better requirements specifications. Here, we will give a list of requirements regarding the Requirements Engineering process.

Another reason for poorly specified NFRs may also be found in the Requirements Engineering process. Process descriptions and tools often neglect NFRs due to their perceived "vague" nature. Nevertheless, they are important, not without cause they are also called quality requirements. Thus, they must be acknowledged and their identification and specification must become a part of the Requirements Engineering process. Therefore, we define the following requirement:

ProcReq 1: *The Requirements Engineering process must include the identification and specification of NFRs. The requirements engineer shall be supported in doing so.*

Requirements Specifications are said to be error prone, incomplete, inconsistent and ambiguous. Reasons therefore are manifold. Some of them can be treated by improving the Requirements Engineering process. We deduce the following requirements:

ProcReq 2: *The Requirements Engineering process must include activities for validation of requirements. The requirements engineer shall be supported herein.*

ProcReq 3: *The Requirements Engineering process must be flexible enough to handle changing or new requirements. This includes validation and verification activities at different times within the Software Engineering lifecycle.*

The need to consider several factors in order to choose the best configuration of requirements must be treated during the Requirements Engineering process as well. Therefore, it is necessary to capture sufficient information in order to be able to compare different configurations and find alternative solutions. Thus, we deduce this challenge to the following requirement regarding the Requirements Engineering process:

ProcReq 4: *Activities regarding identification of insufficient information, incomplete requirements, etc. must be included during the Requirements Engineering process. The Requirements Engineering process should provide opportunities to find the best suitable configuration of requirements.*

The Requirements Engineering process is a highly complex task. A huge variety of methods and techniques exist but it is still up to the requirements engineer to choose and follow a selected methodology. Therefore, we deduce the following requirement:

ProcReq 5: *The requirements engineer must be guided in accomplishing the Requirements Engineering process. This guidance must be understandable, complete, correct, meaningful and connected to present requirements data.*

Requirements for the Requirements Documentation
Manually produced paper documents as SRS have been proven to be time consuming, expensive and rarely maintained. They can barely be kept up-to-date. Thus, we deduce the first requirement:

DocReq 1: *The creation of the SRS document should be at least semi-automatically and easy to maintain. Requirements and all associated information must be (additionally) electronically specified apart from the final document that is intended for a contract with the customer.*

One major problem of SRS reported in literature is their consistency and completeness. Software Requirement Specifications are said to be error-prone and ambiguous. Functional requirements are often engaged with difficult explanations and complicated structure models which are difficult to reveal. Thus, the SRS will be incorrect. Therefore, we define the following requirements.

DocReq 2: *Requirements and all associated data must be easy to understand for all people involved in the development of the software.*

DocReq 3: *The completeness of requirements must be supported. This involves the identification of incomplete and insufficient information (e.g. requirements, requirement descriptions, relationships, decisions, etc.).*

DocReq 4: *The SRS must be consistent. Inconsistencies must be detected and resolved.*

DocReq 5: *Reasons and decisions for changes, solutions, etc. must be captured in an appropriate, accessible manner.*

One minor challenge is the level of detail in a requirement specification that differs depending on the product to be developed. Therefore, we infer the following requirement:

DocReq 6: *The level of detail in a requirement specification must be adjustable.*

Last, reuse of requirement information is hardly practised in Requirements Engineering. One reason are requirements documentations that are not reusable or just not worth to be reused due to insufficient information, understandability, and so on. The above listed requirements for a requirement documentation already acknowledge this challenge and provide means for a reusable requirement documentation. For reasons of completeness we require:

DocReq 7: *The specified requirements and their associated information must be reusable for any other task in the Software Engineering process, especially for another Requirements Engineering activity.*

A variety of methods and tools propose different traceability mechanisms, focusing on different aspects of traceability, e.g. source, stakeholder or objects [113]. It is often the case that requirements engineers treat traceability concerns in such a focused way, neglecting important aspects. Ramesh in [113] constitutes that “traceability in many organizations is haphazard, standards provide little guidance, and the models and mechanisms vary to a large degree and are often poorly understood”. Thus, we define the following requirements:

DocReq 8: *The life of requirements must be forward and backward traceable. This includes their origin, specification, design, development and their subsequent deployment and use, and “through periods of ongoing refinement and iteration in any of these phases” [49].*

DocReq 9: *The requirements engineer must be equipped with comprehensible traceability guidelines.*

DocReq 10: *Requirements must be associated with relevant data to enable traceability. This includes information for all aspects described in requirement DocReq 8.*

Requirements regarding Validation and Verification
As already explained in Chapters 1 and 2, the validation of requirements is crucial for providing correct requirements. Due to changing and newly arising requirements during software development, validation must accompany the development process. At the end of the Software Engineering process, requirements need to be verified. In contrast to validation, verification is accomplished to check whether the previously defined requirements have been satisfied. Requirements, especially non-functional requirements, are often found hard to validate and verify. Therefore, we define:

VVReq 1: *Requirements need to be validated before software development proceeds.*

VVReq 2: *The validation of requirements must be supported by software solutions.*

VVReq 3: *Requirements (including NFRs) must be assigned to test-castes for later verification. These test-cases must be correct, understandable and complete (regarding all information required for testing, e.g. metrics).*

7.3 Summary

In this chapter, we analyse problems and challenges in RE and classify them into six categories: (1.) Technological, (2.) Organisation/Management/Communication, (3.) Requirements, (4.) Requirements Engineering Process, (5.) Requirements Documentation and (6.) Validation/Verification. These challenges are transferred to general requirements for RE that may be interpreted for various aims (e.g., RE tool development, RE process enhancement). We furthermore present a selection of these requirements we aim to tackle within this thesis. We envision the proposed approach and describe the ideas behind it. All of the concepts for OntoReq may also be used for different design tasks. Therefore, we put much emphasize on design and implementation independence. Thus, different design alternatives and technical solutions are possible.

8 The OntoReq Approach

With OntoReq we aim to address some of the shortcomings reported in the Sections 1 and 2. We demonstrate the application of ontology techniques to eliminate a bunch of meaningful problems. The overall aims of our approach are to ensure the identification of missing, incomplete and inconsistent data in the requirements specification and their elimination as well as an improved quality of the SRS. We base our approach on the GORE process and explicitly allow for the specification of goals and related relationships.

The chapter is structured as follows: we start with a list of requirements and limitations of our approach in Section 8.1 which is followed by a description of how OntoReq supports the RE process in Section 8.3. The metamodel of OntoReq is introduced in detail in Section 8.4 and uses examples to illustrate the main concepts of it. This section is followed by an explanation of the completeness, consistency and quality validations in Section 8.5. Furthermore, in Section 8.6 we conceptually describe an approach for guidance with OntoReq. Finally, we give a summary of this chapter in Section 8.7.

8.1 Requirements

Most of the challenges we want to address with OntoReq are summarized in the categories *Requirements Documentation* and *Validation/Verification*. However, we also picked requirements from the other categories. All of these requirements correspond to the requirements defined in section 7.2.

- OMCReq 3, 5, 6
- StrucReq 1, 2
- ProcReq 1, 2, 4
- DocReq 3, 4, 5, 7
- VVReq 2, 3

8.2 Limitations

OntoReq aims to improve the *internal* completeness, consistency and quality of the requirements knowledge captured by applying ontology techniques. Thus, the appropriate analyses and validations are performed with respect to the ABox of the Requirements Ontology and its metadata described in this chapter. The completeness validation checks the completeness of requirements metadata, instead of making assumptions about whether requirements are missing in the SRS. Additionally, we validate the consistency of the requirements knowledge in a requirements configuration, but not the semantical consistency and, thus, correctness of individual requirements descriptions. Furthermore, it is not our aim to make any assumptions about the *external* completeness, consistency or quality of a SRS (e.g. whether a use-case has been described properly, readability of requirements).

OntoReq supports a part of the RE process where problems and shortcomings have been stated popular and of significant negative effect. However, the Requirements Engineering process needs further inspection and probably support to address other known problems and to include various models, such as use-case or test-case models. Furthermore, we do not aim to provide seamless traceability of requirements for the whole Software Development process, but the Requirements Engineering process. However, we provide sufficient data structures and information to support any effort in realising seamless traceability.

8.3 From Textual Requirements To Formalisation

As already described in Section 1, the Requirements Engineering process can be divided into early and late Requirements Engineering. Figure 8.1 illustrates our modification of the RE diagram presented in Chapter 1. We extend the original RE diagram by attaching three models to the appropriate phases and show how OntoReq supports these phases and utilises these models. The textual, conceptional and formalized model are not explicitly discussed in literature. However, from our investigation of the RE process and related articles, we extracted this informal information and defined the appropriate models to facilitate the understandability of our approach.
In the early phase (elicitation & analysis phases), the requirements are elicited by stakeholder interviews, document inspection and so on. The identified requirements knowledge is documented in natural language in documents before it is analysed and validated by stakeholder and customer for a first time. Subsequently, this knowledge will be conceptualized in some way, which differs from method to method. It is perhaps connected to domain knowledge and inspected further to complete this knowledge. Use-cases and metrics might be as well identified and described as test-cases, priorities and so on. We will refer to this process phase as the *conceptualization* of the requirements knowledge. Usually, the conceptual model is not developed any further. However, with OntoReq we seamlessly connect to this conceptualization phase, the already gathered requirements knowledge is further structured and interrelated and attached to the requirements metamodel, the Requirements Ontology. The resulting formalized model allows for the validation of internal completeness, consistency and quality as depicted in the figure below.
As can be seen in Figure 8.1, there is no hard border between the early textual description, the conceptual model and the formalized model. OntoReq connects to the conceptual modelling phase in the way that the Requirements Engineer may insert the already conceptualized requirements knowledge into OntoReq. It is the basis for the formalisation process of that knowledge. Alternatively, OntoReq may also be used right on from the beginning to record early requirements artefacts. The Requirements Engineer may be supported in all these tasks by a guidance system (see Section 8.6). Finally, the formalized requirements knowledge will be validated by OntoReq.

8.4 Requirements Metamodel

All of the requirements artefacts (concepts, relations, attributes, ...) must be captured in an appropriate way. According to the requirements for OntoReq, this requirements repository must:

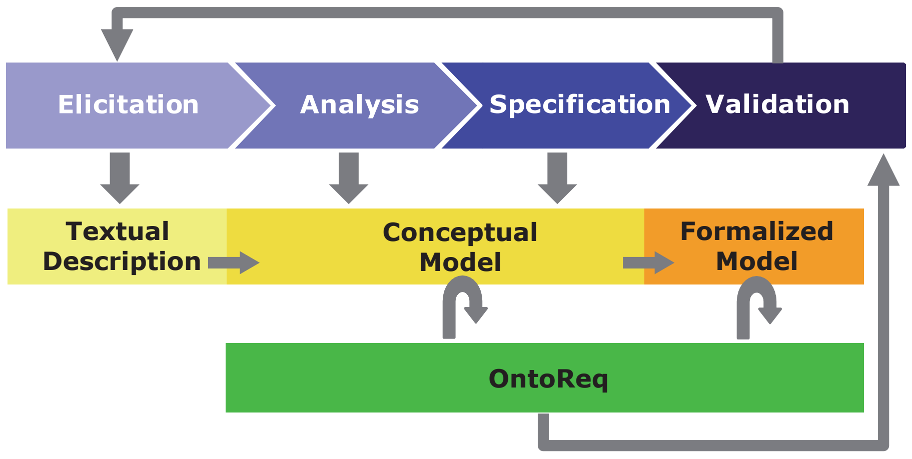


Figure 8.1: OntoReq support in Requirements Engineering process

- enable the management of complex requirements (data, requirements, relations, attributes, decisions, metrics, a.s.o.)
- allow for accessing and manipulating all the information captured within
- allow a consistent specification of relevant information
- enable reusablity of any information captured and
- facilitate the verification of incompleteness and inconsistency

As already discussed in Chapter 3, ontologies can be applied as knowledge repositories and facilitate the realisation of the above requirements. The ontology elements (e.g., classes, properties, instances of classes, relations between instances) can be used to specify requirements artefacts and their relations and are important for the application of ontology reasoning. We use an Ontology as metamodel for the specification of requirements, referred to as *Requirements Ontology*. This Requirements Ontology provides a a terminology in form of a TBox for all requirements artefacts and associated metadata described in Section 8.4. We define *requirements artefact* and *metadata* as follows:

Definition 13 (Requirements Artefact): *Requirements artefacts comprise all concepts related to requirements knowledge (e.g., goal, obstacle, stakeholder, use-case, test-case).*

Definition 14 (Metadata): *Metadata comprise all attributes for requirements artefacts (e.g., priority, state, cost) and the relations between requirements artefacts (e.g., goal contribution, refinements).*

The metamodel is instantiated with concrete requirements knowledge, building the ABox. The Requirements Ontology especially enables to follow GORE. In the following sections, we briefly describe the requirements artefacts, their relations and metadata to be captured in this metamodel.

Extended GORE Model for ONTOREQ

Goal-Oriented Requirements Engineering includes the identification (elicitation), negotiation and specification of requirements and associated meaningful information through a systematic iterative and co-operative process. Besides requirements, goals are of particular interest and importance. Goals capture stable information and provide means to separate stable from unstable information, which enables a better reuse. Furthermore, goals are an excellent way to identify requirements and to finally decide for a requirements configuration that satisfies these goals. Since GORE is the main background of our approach, we will briefly describe and define the main concepts we intend to use in OntoReq.

Requirements Artefacts

Most GORE approaches differ in the concepts and terminology. To be consistent with well-known and suitable GORE concepts, we adopted those that are widely used. However, we also intentionally decided to omit certain concepts or definitions and we will briefly explain the reasons here. We refer to all these concepts as "*requirements artefacts*" due to the fact that all of these concepts are somehow related to requirements and to prevent confusing it with the terminology "concept". Additionally, we describe and define relations between these requirements artefacts.

Definition 15 (Goal): *Goals are declarative statements of intent to be achieved by the system under consideration [129].*

Goals will be used to identify, describe and correlate requirements. As already described in Section 2, goals are formulated in terms of prescriptive assertions (as opposed to descriptive ones) [147]; they may refer to functional or non-functional properties and range from high-level concerns to lower-level ones [130]. In most goal-oriented approaches, the concept of goal expresses the same type of information as requirements. The difference is the level of abstraction. Goals explore *why* certain requirements are necessary for the system to be.

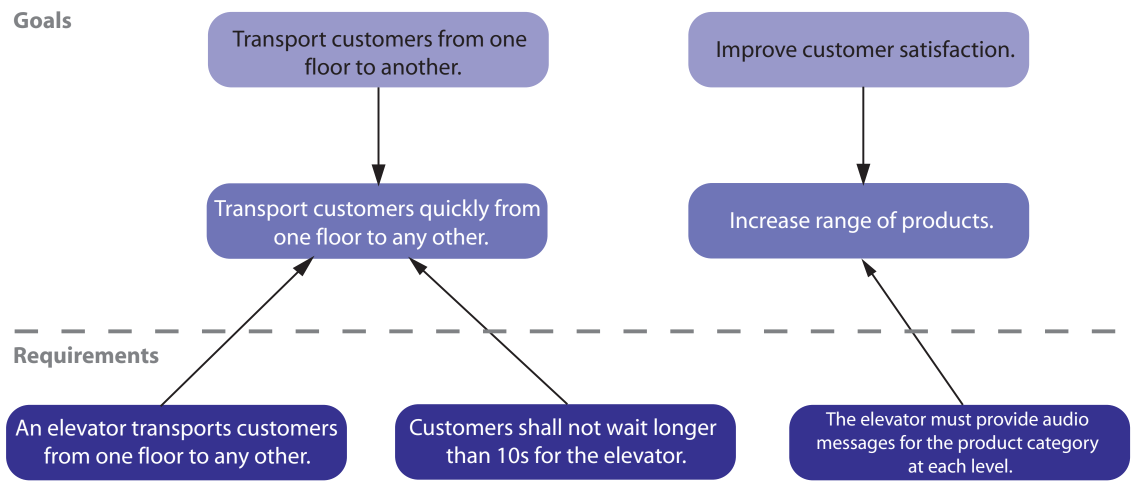

Goals
Transport customers from one floor to another.
Improve customer satisfaction.
Transport customers quickly from one floor to any other.
Increase range of products.
Requirements
An elevator transports customers from one floor to any other.
Customers shall not wait longer than 10s for the elevator.
The elevator must provide audio messages for the product category at each level.

Figure 8.2: Example of Goals and Requirements

In contrast to [130] we will use the term "goal" in a slightly different way. We adapt the separation of stable and unstable information and allow refinements of goals from an abstract level to a more concrete one. However, a goal does not describe the same type of information as requirements. Where requirements are platform and application specific, goals are independent of these and capture the most abstract intentions, e.g., "*Improve customer orientation.*" without specifying whether this shall be a sign posts, information signs or software guidance. These intentions also include business goals, e.g, "*Improve customer satisfaction.*" Obviously a goal that has been refined over several steps provide sufficient means to operationalize (transform) goals down to requirements to meet this goal as illustrated in Figure 8.2. These requirements will be connected to the goals they are intended to satisfy. Thus, we still keep the separation of stable information from unstable and enable the tracing of goals and requirements. However, the main reason for this decision is to add the concept "requirement" since this term is familiar to all Requirements Engineers and additionally introduce "goal". We also believe that a separation of goals and requirements, even though minimal, facilitates validation, verification and traceability of requirements.

Customers often do not only communicate their goals and intentions for a software, but also their apprehension or possible predictable problems in regard of possible software features or design solutions. We refer to this as "*obstacles*".

Definition 16 (Obstacle): *Obstacles are declarative statements of identified behaviour that have a negative effect on the satisfaction of goals or requirements.*

Lamsweerde defines obstacles as "declarative statements of behaviour **not** intended to be achieved by the system under consideration." [129]. In contrast, we define obstacles as behaviour, actions or intentions that hinder the achievement of a goal or the realisation of a requirement. Thus, they describe some kind of barrier for the development of the software product. Usually, obstacles are not acknowledged during Requirements Engineering. Nevertheless, they are important since one can derive requirements also from obstacles. Like goals, obstacles can be refined in several steps. If a system might be hindered by any functionality, it is necessary to transform this "negative goal" to requirements that ensure that this behaviour will not occur as illustrated in Figure 8.3. Apart from that, obstacles are of great help to define test-cases later on. Finally, the validation of the requirements should show that the predefined obstacles will not influence the system. Obstacles will be assigned to requirements.

Definition 17 (Risk): *A risk is an event that threatens the success of an endeavour, e.g., of developing or operating a system. A risk is typically assessed in terms of its probability and potential damage [47].*

Risks can be used in RE for several purposes. They allow for deriving obstacles and verification scenarios. Additionally, they can be a source to develop goals and requirements. Risks may be associated with requirements if appropriate.

Definition 18 (Functional Requirement): *A requirement concerning a result of behaviour that shall be provided by a function of a system, a component or service [47].*

Functional requirements may be specified in different abstraction levels. They can be refined from an abstract requirement to a more precise one.

Definition 19 (Non-functional Requirement): *A non-functional requirement (also quality requirement) is a requirement that pertains to a quality concern that is related to a functional requirement [47].*

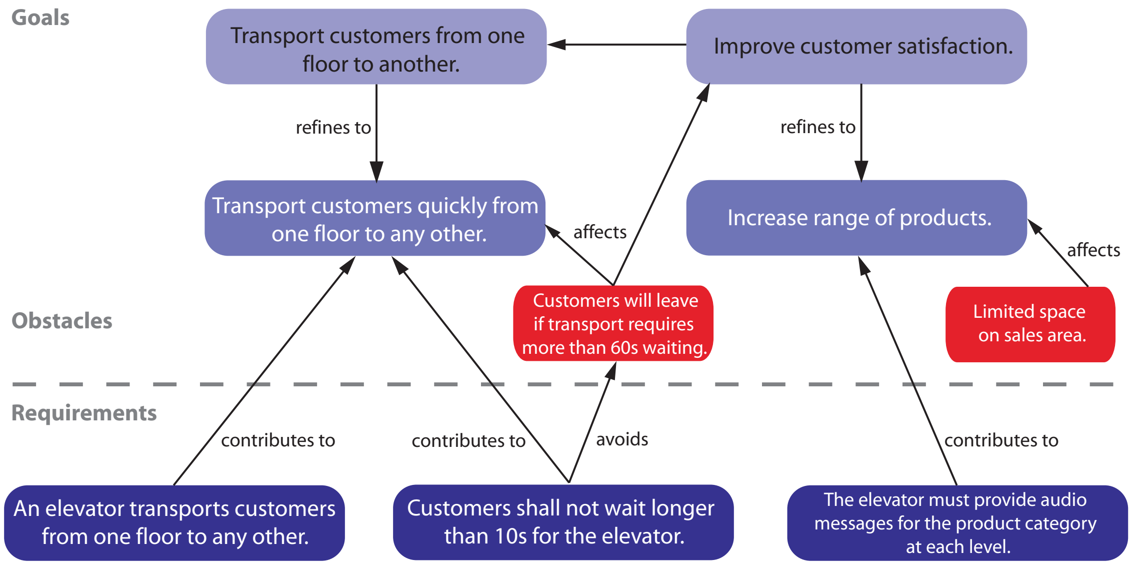

Goals
Transport customers from one floor to another.
Improve customer satisfaction.
refines to
refines to
Transport customers quickly from one floor to any other.
Increase range of products.
affects
affects
Obstacles
Customers will leave if transport requires more than 60s waiting.
Limited space on sales area.
Requirements
contributes to
contributes to
avoids
contributes to
An elevator transports customers from one floor to any other.
Customers shall not wait longer than 10s for the elevator.
The elevator must provide audio messages for the product category at each level.

Figure 8.3: Example of obstacles

Non-functional requirements describe *how* the software shall operate or *how* a function is executed [23]. According to the ISO Standard for RE [63], non-functional requirements include efficiency, functionality, maintainability, portability, reliability and usability.

Definition 20 (Platform Requirement): *Platform Requirements (also System Requirements) refer to requirements regarding the platform of the software, e.g., RAM size, Operating System, Hardware Architecture, etc.*

Platform (or system) requirements may be treated as functional or non-functional requirements. When specifying the speed of memory access, we could also specify this as a functional or non-functional requirement. In order to facilitate the comprehensibility, we decided to use platform requirements as another category of requirements.

Definition 21 (Process Requirement): *Process requirements are constraints placed upon the development process of the system [66].*

Process requirements typically include requirements on development standards, guidelines and methods which must be followed. Additionally, they may specify CASE[1] tools that must be used.

Definition 22 (Use-Case): *A description of the interactions possible between actors and a system that, when executed, provide added value. Use-cases specify a system from a user's (or other external actor's) perspective: every use case describes some functionality that the system must provide for the actors involved in the use case [47].*

Use-cases will be assigned to the appropriate requirements whose functionality is described.

Definition 23 (Scenario): *A textual description of a small part of a Use-Case that leads to a desired (or undesired) result. A scenario describes a sequence of user actions in general terms.*

[1]Computer-aided software engineering

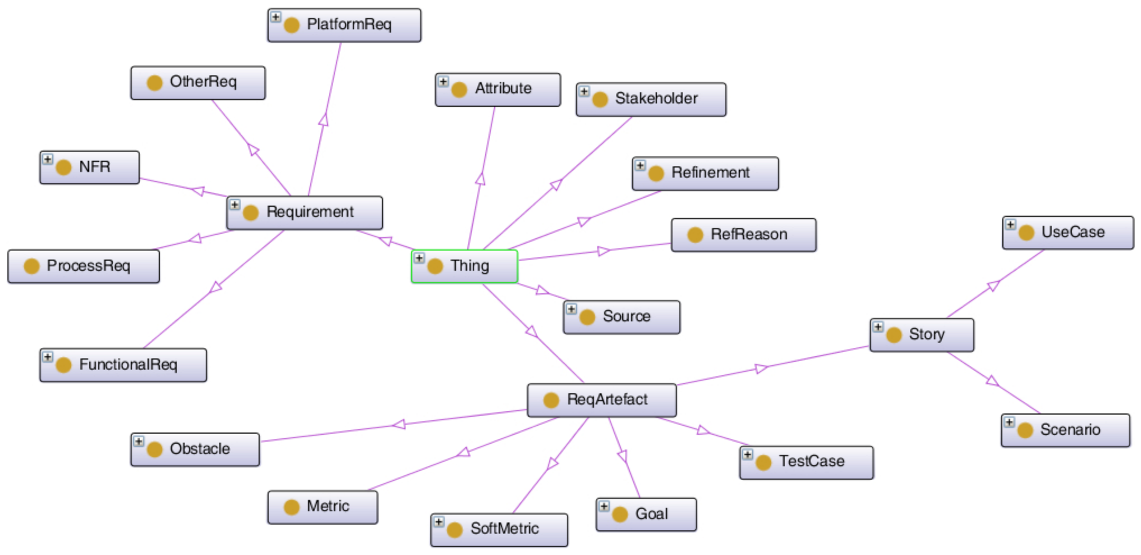


Figure 8.4: Requirements artefacts of the requirements metamodel

Scenarios are parts of use-cases and thus, will be associated to use-cases. They can also be assigned to the relevant requirements.

Definition 24 (Metric): *A functional description of values that can be measured, e.g., response time.*

Definition 25 (Soft-Metric): *A non-functional description of quality values that cannot be easily measured, but still provide important information for the verification of non-functional requirements, e.g., learnability, understandability.*

Definition 26 (Test-Case): *A description of possible interactions between actors and the system to be tested for. Every test-case describes some functionality that the system must (or must not) provide for the actors involved in the test case.*

Definition 27 (Stakeholder): *A person or organization that has a (direct or indirect) influence on a system's requirements. Indirect influence also includes situations where a person or organization is impacted by the system [47].*

Stakeholders can be associated to different requirements artefacts: requirements, goals, use-cases, scenarios, test-cases, risks, obstacles, etc. Thus, they act “responsible author”.

Definition 28 (Source): *The origin of a requirements artefact (e.g., document, guidelines, law).*

A source specifies the origin of any information and allows traceability of information and decisions. It can be assigned to several requirements artefacts, e.g., requirements, goals and use-cases.

Requirements Relations

Interrelations) among requirements are often not considered or not adequately covered. The most well-known relations between (two) requirements are *refinement* and *conflict*. Although these relations are important, they are still not sufficient to document requirements precisely. As introduced in Chapter 4, Feature Models capture the variability of software features and their dependencies for Product Line Engineering. The most basic variability mechanism is the notion of optional, mandatory, alternative (XOR) and OR (at least one of the sub-features must be selected) features. Based on the benefits and shortcomings of Feature Models for Requirements Engineering and the synergy with ontologies discussed in Chapter 4, we adapt these variability mechanisms for the requirements relations and map them to ontological elements. Additionally, we extend these relations to express further dependencies between requirements and to allow scattered dependencies in spite of hierarchical connections in Feature Models. This way, we maintain consistency with Feature Models and enable future transformations of requirements configurations into Feature Models and vice versa. Furthermore, we adopt the definition of a "(feature) configuration" for requirements and define:

Definition 29 (Requirements Configuration, RC): *A requirements configuration is a set of requirements to be implemented. A requirements configuration is valid if and only if it does not violate constraints imposed by the Requirements metamodel.*

We propose seven relations between an arbitrary number of requirements, referred to as "requirements relations" as well as two properties feasible for a single requirement, referred to as "requirements attributes". OntoReq provides more attributes which also relate to other requirements artefacts (e.g., to use-case or goal). These are described at the beginning of this section. Based on these relations and attributes, later decisions, changes, manipulations, etc. can be inferred from this knowledge. It also facilitates the semi-automatic suggestion of requirements that best contribute to a certain goal or to provide specific suggestions for eliminating identified inconsistencies. In the following, we will define these requirements relations and attributes and use simple examples for their explaination. We use the notation r_i to denote requirements and big letters to define relations among requirements, e.g., $S(r_1, r_2)$. The examples and their semantics are summarized in Table 8.4.

Definition 30 (Mandatory Attribute): *A requirement is mandatory if it is definitely required by the system and must be realised.*

Most of the requirements in a requirement specification will be mandatory, this means they will definitely be chosen for implementation. This attribute is also important for the validation of requirements regarding their consistency. For example, all mandatory requirements must be included in the requirements configuration.

Definition 31 (Optional Attribute): *A requirement is optional if it is in scope but not necessarily required by the system. It is not mandatory.*

Optional requirements are requirements that have finally been considered as not mandatory after stakeholder or customer discussions. Requirements that are optional should nevertheless be included in the SRS. They might be considered necessary later and the optional attribute might be changed to "mandatory". Optional requirements may also be suitable as alternative requirements where appropriate. Finally, they can still be chosen to be implemented. We distinguish three forms of alternative requirements:

Definition 32 (Optional-Alternative Relationship, S): *One requirement may be replaced by another;* $S(r_1, r_2), r_1 \neq r_2$.

Providing such optional-alternative requirements increases the expressiveness of solution suggestions and the computation of goal satisfactions.

Definition 33 (Or-Alternative Relationship, A): *A choice limited to at least one requirement of a set of requirements;* $A(r_1, r_2), r_1 \neq r_2$.

Or-alternative requirements are adapted from Feature Models. Sometimes it is necessary to provide a set of alternative requirements where at least one must be realised and the others are optional, e.g., different options to allow for moving people from one floor to another (elevator, staircase, escalator). It will often be the case that one requirement refines to multiple alternative requirements. The example illustrates the latter case.

Definition 34 (X-Alternative Relationship (XOR), X): *A choice limited to exactly one requirement of a set of requirements;* $X(r_1, r_2), r_1 \neq r_2$.

X-alternative requirements are also adapted from Feature Models[2]. Often there is only exactly one requirement from a set of alternatives that can be realised, e.g., the place where the customer guidance system shall be placed (front, middle, back). Similar to or-alternative, x-alternatives must not but may relate to a super-requirement and span a hierarchy.

Definition 35 (Coexistence (Implication) Relationship, I): *A requirement implies (requires) one or more other requirements;* $I(r_1, r_2), r_1 \neq r_2$.

This binary implication relationship is adapted from Feature Models as well and eases the effort to ensure consistency among requirements. It is often the case that some requirements require others, e.g., an elevator requires space in the area where it shall be located. Thus, elevator *requires* space.

Definition 36 (Conflict Relationship, N): *A requirement* r_1 *conflicts with another requirement* r_2 *if the fulfilment of* r_1 *excludes the fulfilment of R2 and vice versa. A conflicting relationship must also specify a conflict reason (p). This relationship is symmetric;* $N(r_1, r_2, p)$.

This triple relationship is an extension to the variability provided in Feature Models. Although it is identical to the the exclusion relationship in Feature Models, the semantics differs in OntoReq. The Requirements Engineer can specify any reason for a conflict, e.g., conflicting stakeholder intentions, violation of company rules, etc. by using the attribute *conflict reason.* In contrast to Feature Models, OntoReq is able to capture much more of this background knowledge which facilitates the traceability of requirements, decisions and problems.

Definition 37 (Exclusion Relationship, E): *The exclusion of a requirement by another requirement. In contrast to the conflict relationship, this exclusion is non-symmetric and may define an exclusion reason (p);* $E(r_1, r_2, p)$.

[2]In Feature Models they are simply called “alternative”. To enhance distinction we use the prefix “x-” which refers to “xor”.

This relationship can be understood as a specialisation of the conflict relationship with the difference that it is already known that this conflict cannot and shall not be solved. In contrast to Feature Models, it is non-symmetric.

Definition 38 (Refinement Relationship): *A requirement r_1 refines a requirement r_2 if r_1 is derived from r_2 by adding more details to it. r_1 can be seen as an abstraction of the detailed requirement R2. The triple relationship consists of the refinement source, the refinement target and the refinement reason (p); $R(r, r_1, p)$.*

The refinement of requirements is one of the main tasks in Requirements Engineering and thus, needs additional support. Usually, the reason for the refinement is not documented and thus, difficult to regain during the realisation phase. Often, the requirement is revised by simply adding a new requirement and perhaps deleting the previous one which was the source of the refinement. This way, important knowledge about the history of requirements gets lost and traceability decreases significantly. But with refinement relations, traceability is enabled.

Example of Requirements Relations and their Semantics

Below we extend the previous examplar by additional requirements and dependencies.

r_0: The user shall be guided in product selection.
r_1: The user shall be guided by voice.
r_2: The user shall be guided by text.
r_3: The user shall be guided by pictures.
r_4: Each product must be listed with its location in the physical store.
r_5: Customers can choose between product categories.
r_6: Software maintenance tasks must be executed at night.
r_7: Software must reduce energy consumption at night.

The table below provides single examples for each of the above described requirements relations. However, it is not our aim to give a complete or consistent set of dependencies between these requirements.

Requirements Metadata

In [42], Firesmith et. al propose a list of requirements metadata to ensure the completeness of individual requirements. Metadata is data about that requirement rather than data listed in the requirements specification. Firesmith identified the following metadata as the most important and most likely to be mandatory:

- Project-Unique Identifier (PUID)
- Prioritization
- Rationale
- Source
- Status (may include more than one kind of status)
- Verification Method

Additionally, according to Firesmith et. al. the following metadata are usually missing:

Requirements Relationship or Attribute	Corresponding Formula Fragment
r is mandatory.	$\wedge r$
r_1 is optional.	$\vee r_1$
r_1, r_2, r_3 x-alternative requirements of r	$(r_1 \vee r_2 \vee r_3 \iff r) \wedge \bigwedge_{i<j} \neg(r_i \wedge r_j)$
r_1, r_2, r_3 or-alternative requirements of r	$(r_1 \vee r_2 \vee r_3 \iff r)$
r_2 is optional-alternative of r_1	$\vee r_1 \vee r_2$
r_6 is in conflict with r_7	$\neg(r_6 \wedge r_7)$
r_1 excludes r_2	$\neg(r_1 \wedge r_2)$
r_5 is coexistent with (requires) r_4	$r_5 \Rightarrow r_4$
r_1 refines r	$r_1 \wedge \neg r$

Table 8.1: Requirements relation and corresponding formula fragments

- Categorization
- Criticality to Customer
- Criticality to Users
- Estimated Cost Range
- Frequency of Execution
- Implementation Status
- Owners (e.g., owning producer for implementation)
- Prioritization
- Probability of Defects (in the implementation)
- Project-unique identifier for requirements identification and traceability
- Rationale
- Risk (associated with implementation)
- Source (e.g., requirements trace to document, goal, stakeholder)
- Status
- Stakeholders
- Verification Method
- Verification Status
- Validation Status
- Volatility (e.g., high, medium, low)

For OntoReq, we adapt most of the proposed metadata published in [42] and added three additional metadata (obstacle, goal contribution, soft-metric) as shown below:

- Categorization (e.g., functional requirement, quality requirement, constraint)
- Mandate (specifies whether a requirement is optional or mandatory)
- Source (e.g., requirements trace to document, goal, stakeholder)
- Status (e.g., validated, rejected)
- Priority Range (e.g., low, medium, high)
- Obstacle (defines identified obstacles for a requirement)
- Estimated Risk Range (e.g., low, medium, high)
- Estimated Cost Range (e.g., low, medium, high)
- Refinement (specifies the reason for a requirement refinement)

- Contribution (specifies whether a requirement has a positive or negative contribution to another requirenment)
- (Soft-)Metric (specifies a metric that can be used for verification, soft-metrics are used for non-functional requirements)
- Verification Method (e.g analysis, demonstration, inspection)

Since the SRS consists of several different requirements artefacts, we can use an appropriate subset of these metadata not only for functional or non-functional requirements, but also for other requirements artefacts. For example, a goal should also specify its stakeholder and a priority.

8.5 Requirements Validation Services

From a formal point of view, correctness is usually meant to be the combination of consistency and completeness. However, from a practical point of view, correctness is often more pragmatically defined as satisfaction of certain business or customer goals [151]. Zowghi in [151] defines completeness “with respect to an *external* body of knowledge” and consistency as an “internal property of a certain body of knowledge”. This external body of knowledge can be understood as a metamodel, in our case the Requirements Ontology. Thus, we can validate completeness regarding this metamodel (external body) and measure consistency regarding the instantiation of the Requirements Ontology with concrete requirements knowledge (internal property). Due to this causal relationship between completeness, consistency and correctness, we can formally improve the correctness of requirements knowledge by increasing its completeness and consistency. Although it would be desirable to have a formal proof of correctness of a SRS, it may not be practical or may be too costly [151]. Whether applying such formal correctness proofs or not depends on the degree of risk the stakeholders are prepared to take [151]. According to Zowghi et. al., such proofs can be carried out in many cases by informal inspections of the requirements and domain, involving customer and stakeholders (stakeholder validation). Thus, we adapt the formal point of view for correctness and define correctness as the combination of completeness and consistency. This way, the main advantage of our approach lies in the capability of early identifying those changes in the requirements knowledge that might introduce errors in the specification, thus achieving more precise validation and verification of the requirements later on.

In addition to completeness, consistency and correctness, another aspect of the SRS becomes important. Firesmith in [43] states that “poor quality of individual requirements and the requirements specifications that document them is a primary reason why so many projects continue to fail [Standish 1994]”. Furthermore, Firesmith states that “the poor quality of the requirements is typically not recognized during Requirements Engineering and the evaluation of requirements specifications”. Thus, the quality of individual requirements is another aspect we need to consider.
In [32], Davis proposes 24 criteria and formulas for measuring the quality of a SRS, namely:

- 1. Unambiguous
- 2. Complete
- 3. Correct
- 4. Understandable
- 5. Verifiable
- 6. Internally Consistent
- 7. Externally Consistent
- 8. Achievable
- 9. Concise
- 10. Design Independent

- 11. Traceable
- 12. Modifiable
- 13. Electronically Stored
- 14. Executable/Interpretable
- 15. Annotated by Relative Importance
- 16. Annotated by Relative Stability
- 17. Annotated by Version
- 18. Not Redundant
- 19. At Right Level of Detail
- 20. Precise
- 21. Reusable
- 22. Traced
- 23. Organized
- 24. Cross-Referenced

As can be seen from the list above, the quality of requirements requires explicitly completeness (2.), consistency (6.) and (7.) and correctness (3.) among others. However, some other criteria refer to completeness and consistency as well. Therefore„ we will divide Davis' criteria into the following four categories and refer to them as Davis' criteria:

Completeness

- 2. Complete
- 15. Annotated by Relative Importance
- 16. Annotated by Relative Stability
- 17. Annotated by Version
- 24. Cross-Referenced

Consistency

- 6. Internally Consistent
- 7. Externally Consistent
- 18. Not Redundant
- 19. At Right Level of Detail

Quality

- 1. Unambiguous
- 4. Understandable
- 5. Verifiable
- 8. Achievable
- 9. Concise
- 10. Design Independent
- 11. Traceable
- 12. Modifiable
- 14. Executable/Interpretable
- 20. Precise
- 21. Reusable
- 22. Traced
- 23. Organized

Other

- 3. Correct
- 13. Electronically Stored

Since correctness (3.) is already acknowledged by the categories completeness and consistency, we do not need to take any more action therefore. Due to the fact that we store the requirements as instantiation of the Requirements Ontology, we also satisfy the criteria of electronically storing requirements (13.). We discuss the remaining criteria and their realisation in the following appropriate sections and describe our approach for validating the completeness, consistency and quality of requirements knowledge. Therefore, we introduce completeness, consistency and quality rules (Sections 8.5.1 to 8.5.3) and subsequently explain their application in OntoReq. Furthermore, we conceptually show how these requirements validation services satisfy the requirements OMCReq3, OMCReq5, StrucReq2, DocReq2, DocReq3, VVReq2 and VVReq3 documented in Section 8.1.

8.5.1 Completeness Validation

Zowghi et. al. distinguish between internal and external completeness [151]. *Internal* completeness implies that no information is left unstated or "to be determined" and information does not contain any undefined objects or entities [64]. In [42], Firesmith refers to these data as metadata. *External* completeness is characterized by exhaustive information in the SRS [151]. Thus, completeness is a relative measure and may be determined only in relation to an external reference [151] that is provided by the Requirements Ontology. According to [42], we use the following definition for the internal completeness of an individual requirement:

Definition 39 (Internal Completeness (of Individual Requirements)): *An individual requirement is complete if it contains all necessary information to avoid ambiguity and needs no amplification to enable proper implementation [42].*

Definition 40 (External Completeness (of Requirements Specification)): *External Completeness is stated with regard to the whole SRS, which may include various documents and models [42].*

In this thesis, we concentrate on internal completeness and provide means to identify and correct incomplete metadata of individual requirements and requirements artefacts. Davis' criteria demand that requirements must be annotated by relative importance (15.), stability (16.) and version (17.). Furthermore, we already acknowledge (24.) and facilitate the cross-referencing of requirements by appropriate requirements relationship in the metamodel, explained in Section 8.4. The remaining criteria are realised by providing the corresponding requirements metadata (e.g. priority) to be completed during instantiation of the metamodel. In addition to Davis' criteria, Firesmith defined a huge number of metadata that is important for the completeness of requirements (see Section 8.4).
We transform these metadata to a set of 50 completeness rules[3]. Since we aim to check our SRS against these rules, we also provide warning and error message and, most notably, specific suggestions for the elimination of these completeness problems. We define completeness rules as follows:

Definition 41 (Completeness Rules): *Completeness rules comprise three parts: rule definition, fault message and solution suggestion. The definition states all the requirement artefacts that need to be specified in the SRS and their associated metadata that must (or should) be specified. The fault message provides additional information of the concrete problem for each rule that fails and the solution suggestion proposes knowledge-specific opportunities for each incompleteness problem to be eliminated.*

For readability reasons we list only two completeness rules as examples. The complete list can be found in the appendix in Chapter A.2.

- **AT LEAST ONE Goal must be specified.** *(rule description)*
  Error: "You did not specify any Goal. *(fault message)*
  Please specify at least one Goal." *(solution suggestion)*

[3]The term "rule" is not used in any technological context, but rather intends to describe requirements for requirements completeness. Since this would be hard to read we use the term "completeness rules" instead.

- **ALL Functional Requirements (FR) should have a priority**.
  Warning: “You did not define a priority for the Functional Requirements [...]. Please choose a priority for the following Functional Requirements: [...].”

While the first rule ensures that all requirements artefacts are specified, the second rule refers to the completeness of the metadata. As can be seen, we distinguish between warnings and errors which is also indicated by the use of the terms “should” and “must”. This categorization into rules resulting in warnings and those resulting in errors can be used for different purposes, e.g., to realise different levels of completeness validation (e.g., weak and strong completeness) as part of a certain quality level to be achieved for the SRS.

We use the following extract of requirements knowledge illustrated in Figure 8.5. We use FR as abbreviation for functional requirement.

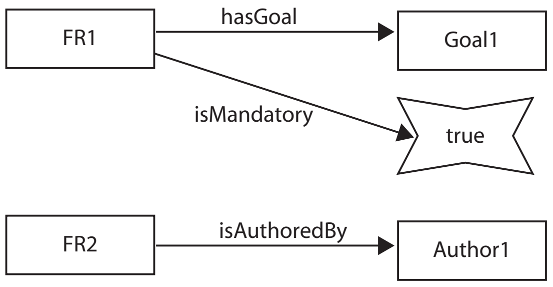


Figure 8.5: Example Requirements Information for Completeness Rules.

Completeness Rules to be checked:

1. AT LEAST ONE Goal must be specified.
2. ALL FR should have a priority.
3. ALL FR must have an author.
4. ALL FR must state their mandate.

Expected Result:

- Requirements with missing priority: FR1, FR2
- Requirements with missing author: FR1
- Requirements with missing mandate: FR2

The above examples illustrates the specification of the metadata (goal, author and mandate). While rectangles denote requirements artefacts that are interrelated with binary relations (here *hasGoal*, *isAuthoredBy*), stars hold a boolean value for an unary relation (here “true” for the property *isMandatory*). As we can see, the functional Requirement FR2 is not related to a goal and does not specify its mandate. FR1 is not related to an author. This information is missing, thus the above completeness rules are violated.

8.5.2 Consistency Validation

Inconsistencies in SRS are one of the most important problems Requirements Engineers are facing (see Section 1.2). Reasons for inconsistencies are manifold: incomplete or faulty requirements acquisition and specification, the evolution of goals, requirements, use-cases, etc. from multiple stakeholders or sources, the refinement of requirements, and so on [132]. According to [64], [32] and [151], consistency is considered as the absence of conflicts in the SRS. Furthermore, Davis' criteria require the absence of redundancy (18.) and the right level of detail (19.). While this definition may be adequate for an entire SRS, it is too weak for a requirements configuration as defined in Section 8.4. When excluding requirements from such a configuration, new inconsistencies may occur due to neglected relationships between requirement artefacts, e.g., mandatory requirements that are not included in the requirements configuration or requirements that imply others. Thus, we need to define consistency in much stronger terms.
The consistency of a requirements configuration requires complete metadata for all requirements artefacts in the SRS. Only if this can be assured, consistency checking can be realised. Therefore, we need to check all requirements relationships and identify those which are corrupted. For OntoReq and its proposed requirements artefacts, relationships, metadata, etc. (see Section 8.4) these are in detail:

- Mandatory requirements in the SRS which are not included in the requirements configuration
- Disregarded coexistent requirements (missing requirements that are required by others in the requirements configuration)
- Excluding requirements in the requirements configuration (requirements that exclude each other)
- Conflicting requirements in the requirements configuration (requirements that have already been identified as conflicting)
- Compliance with or- and x-alternative relationships (minimum number of requirements included as specified by relationship)
- Requirements with negative contributions to others that may not be included
- Incomplete refined requirements in the requirements configuration (requirements that have been refined, but instead of the refinement, the more abstract requirement was included)

Hence, we define internal consistency as follows:

Definition 42 (Internal Consistency of a Requirements Configuration): *A requirements configuration is internally consistent if it is free of conflicting and excluding requirements. All mandatory requirements and coexistent requirements must be included. The requirements configuration must contain the most refined requirement of each particular requirement refinement and comply to the alternative relationships (or- and ex-alternative).*

Our approach is limited to the detection of conflicts regarding the use of the proposed requirements relationship "conflict relationship". Such conflicts are usually detected during an early stakeholder validation and can then be incorporated in the SRS. It is not possible to automatically detect semantic conflicts between requirements by automatically investigating

the requirements description. However, there might be some text-classification approaches that can become meaningful in time.
From the above list, the reason for requiring completeness becomes obvious. If requirements relationships are incomplete or faultily specified, the consistency checking might indeed prove consistency, but under a wrong premise. Thus, it is also necessary to validate consistency after any manipulation in the requirements configuration (e.g., adding or deleting the requirements, changing metadata, etc.) since these changes are the most significant reasons for introducing (new) inconsistencies [151].
Similar to the completeness rules we define rules for consistency.

Definition 43 (Consistency Rule): *Consistency rules comprise three parts: rule definition, fault message and solution suggestion. The definition states a condition of the requirements configuration to ensure consistency. The fault message provides additional information of the concrete problem for each rule that fails and the solution suggestion proposes options for each inconsistency problem to be eliminated, based on the specified requirements knowledge. A requirements configuration is consistent if all consistency rules are satisfied.*

We propose seven consistency rules (A.3), listing the following two as example:

1. **ALL mandatory requirements must be included in the requirements configuration.**
   Error: "The following requirements are mandatory and should be included as well: [...].
   Please include these requirements in your requirements configuration or revise their mandate."

2. **Conflicting requirements must not be included in the requirements configuration.**
   Error: "The following requirements have conflicts [...]."
   "Please choose one of the following options:

   Solve the conflict between the following requirements [r_1 and r_2, ...],
   Revise the requirements relationship of [[r_1 and r_2],...],
   Choose one of the alternative requirements instead of [r_1]: [r_y],
   Choose one of the alternative requirements instead of [r_2]: [r_y],
   Exclude the following requirements from the requirements configuration: [...]."

In contrast to the completeness rules, the solution suggestions of consistency rules need various considerations to allow for recommending precise options for resolving detected errors as illustrated in the example below.

We use the following extract of requirements knowledge illustrated in Figure 8.6. We use FR as abbreviation for functional requirement.

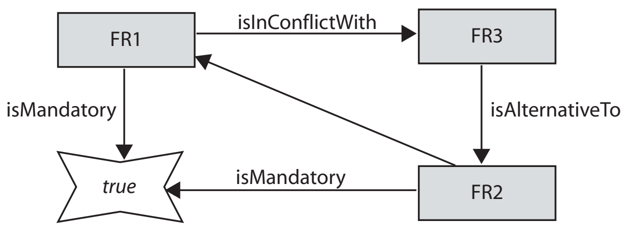


Figure 8.6: Example Requirements Information for Consistency Rules.

Chosen Requirements Configuration: FR1, FR3
Consistency Rules to be checked:

1. All mandatory requirements must be included in the requirements configuration.
2. Conflicting requirements must not be included in the requirements configuration.

Expected Result:

- FR2 is mandatory and must be included in the requirements configuration.
- FR1 is in conflict with FR3. Please choose from the following options:
  - Solve the conflict between the following requirements: FR1, FR3.
  - Revise the requirements relationship of FR1 and FR1.
  - Choose one of the alternative requirements instead of FR3: FR2
  - Exclude the following requirements from the requirements configuration: FR3

As can be seen in the example above, we can recommend several options to solve inconsistencies. Due to the specified alternative relationships we can advise to exchange FR3 with FR2 and thus, solve the conflict. Furthermore, we can suggest to exclude FR3 since it is not mandatory. These decisions must be considered by background checks, that is, testing

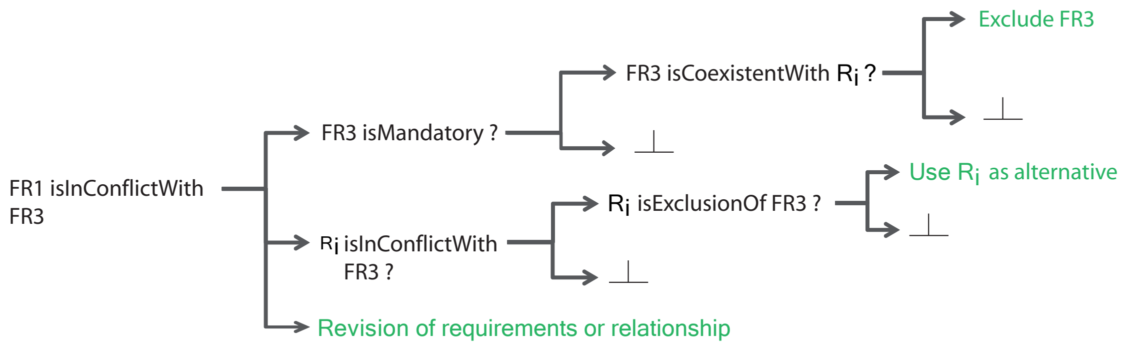


Figure 8.7: Decision tree for solution suggestion in the consistency example.

whether an option indeed eliminates a problem and does not introduce new inconsistency. The decisions are depicted in Figure 8.7.
The simplified decision tree in Figure 8.7 shows the background tests for alternative requirements. Alternatives are not restricted to be within the requirements configuration. If a requirements outside the requirements configuration is suitable, it may be added to the requirements configuration. The background tests allow for guaranteeing that an alternative requirement is not in any conflicting state or is excluded by the original requirement. Furthermore, a requirement may only be replaced with an alternative if it is not mandatory or coexistent with any other requirement in the requirements configuration. If a requirement is mandatory but may be replaced with an alternative (as one of the solution options), the Requirements Engineer will be informed and further guided (e.g. reminded to change the mandate of the original requirement to “optional”)

8.5.3 Quality Improvement of SRS

Due to the extensive requirements knowledge that can be captured in the Requirements Ontology, it is possible to improve the quality of the final SRS. Although we cannot guarantee the quality of external documents, we provide meaningful mechanisms to check the internal quality and to support the Requirements Engineer in improving quality. Therefore, we firstly propose quality validation criteria for requirements artefacts and secondly, quality rules, similar to the completeness and consistency rules described above.

Quality Validation

Davis’ criteria about quality include a number of characteristics for the description (semantics) of requirements (unambiguous (1.), understandable (4.), concise (9.), design independent (10.) and precise (20.)). Since these criteria refer to the semantics of requirements, we can hardly provide mechanisms for improvement. There may be methods to enhance the description of requirements (e.g. structured language), but since such techniques add up a lot of time for learning and require an extensive modification of accomplishing Requirements Engineering, we decided not to use such techniques. However, we discuss possible extensions of our approach that also include mechanisms to improve these quality criteria.
As agreed on in most literature concerning the quality of SRS (e.g., [32]), requirements must be validated after specification in order to ensure for example their satisfiability of

customer wishes, understandability and realisability. Thus, we introduce the attribute "*valid*" in the Requirements Ontology for the above criteria that we cannot automatically measure. This attribute may be assigned to challenges, goals, requirements, use-cases, scenarios and test-cases. It is the task of the Requirements Engineer and stakeholders to validate these artefacts manually and finally assign "*true*" or "*false*" for the attribute "*valid*". To enhance an overview about already validated requirements and those that still require a validation, we additionally provide an attribute "*validated*" whose value is "*false*" as long as it is not selected to be valid or invalid.
Additionally, some of these quality criteria are covered by the facts that ontologies are reusable (21.), modifiable (12.) and provide means to organize and structure information (23.).
Based on the available knowledge specified in the Requirements metamodel we extend Davis' criteria by the following:

Definition 44 (Goal Satisfaction): *The satisfaction of goals comprises the avoidance of negative goal contributions and the consideration of positive goal contributions.*

Definition 45 (Low-risk): *The realisation of low-risk requirements comprises efforts to avoid requirements with high risk, high cost or obstacles.*

Similar to the completeness and consistency rules, we define nine quality rules. These rules are partly based on the remaining quality criteria described by Davis in [32]. Furthermore, we define:

Definition 46 (Quality Flaw): *A quality flaw is an undesired characteristic in the requirements knowledge that decreases its quality.*

Definition 47 (Quality Rules): *Quality rules comprise three parts: rule definition, fault message and solution suggestion. The definition states a condition of the requirements configuration to improve quality. The fault message provides additional information of the concrete problem for each rule that fails and the solution suggestion proposes options for each quality flaw detected, based on the specified requirements knowledge.*

In the following, we list the quality rules.

1. **AT LEAST EACH most refined requirement must be described by a use-case.**
   Error: The following requirements have no use-case assigned: [...]
   Please choose one of the following options:
   - Assign or extend an existing use-case for these requirements,
   - Specify a use-case for the requirements and assign them to the appropriate requirements.

2. **EVERY most refined requirement must have a test-case or metric assigned.**
   Error: The following requirements are not assigned to any test-case or metric: [...]
   Please choose one of the following options:
   - Assign or extend an existing test-case for these requirements,
   - Assign an existing metric to the requirements,
   - Specify a test-case and assign it to the appropriate requirement,
   - Specify a metric and assign it to the appropriate requirement.

3. **There should be no requirement that is a negative contribution to a goal to be achieved.**
   WARNING: The following requirements are a negative contribution to a goal: [r_1 on the goal g_1, ...]
   Please choose one of the following options:
   - Exclude the optional requirements [...] from the requirements configuration,
   - Choose one of the alternative requirements instead of [r_1]: [r_x],
   - Choose one of the alternative requirements instead of [r_2]: [r_y],
   - Revise the goal satisfaction relationship.

4. **There should be no optional requirement with a high risk or high cost.**
   WARNING: The following optional requirements have a high risk and/or high cost [...]
   Please choose one of the following options:
   - Exclude the following optional requirements from the requirements configuration [...]
   - Choose one of the alternative requirements instead of [r_1]: [r_x], ...
   - Revise the cost and/or risk of these requirements,
   - Specify the requirements as mandatory (if reasonable).

5. **ALL requirements must state their priority.**
   Error: The following requirements do not state their level of priority [...]
   Please add the level of priority to these requirements.

6. **ALL requirements must state their mandate (optional or mandatory).**
   Error: The following requirements do not specify their mandate [...]
   Please add a mandate to these requirements.

7. **Requirements with a positive contribution to a goal should be included in the requirements configuration.**
   Warning: The following requirements are a positive contribution to a challenge or goal: [r_1 to g_1, ...]
   Please consider to include them in the requirements configuration.

8. **Requirements must be complete.**
   Error: The following requirements miss relevant information [...]
   Please execute the completeness validation and add missing information.

9. **The requirements configuration must be consistent.**
   Error: The requirements configuration is inconsistent.
   Please execute the consistency validation and resolve inconsistency.

We use the following extract of requirements knowledge illustrated in Figure 8.8. We use FR as abbreviation for functional requirement.

Chosen Requirements Configuration: FR3
Quality Rule to be checked:

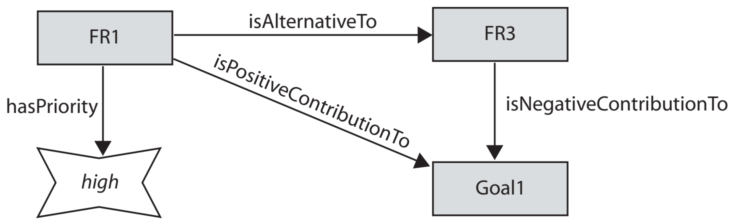


Figure 8.8: Example Requirements Information for Quality Rules

1. There must be no requirement that is a negative contribution on a goal to be achieved.

Expected Result:

- FR3 is a negative contribution to: Goal1. Please consider any of the following options:
  - Exclude the optional requirement FR3 from the requirements configuration,
  - Replace FR3 with one of the alternative requirements: [FR2]
  - Revise the goal contribution relationship.

Quality Measurement

The above quality rules involve the Requirements Engineer in improving the quality of the SRS. Although these rules help to exactly identify where improvements are necessary, the Requirements Engineer will not exactly know how important the modifications were for the overall quality of the SRS. Therefore, we provide a measurement for the quality of the SRS, based on Davis' criteria and the additional criteria "uncritical" defined in Section 8.5.3

We decided to measure[4] the criteria "internally complete", "correct", "verifiable", "internally consistent" and "traceability". Additionally, we measure the newly defined criteria, "uncritical".

In the following, we give a list of these quality criteria, their definitions and describe how to measure and weight them in OntoReq. We define a quality metric Q_i for each quality attribute and a weight W_i that is used to weight each single quality attribute according to its importance for the overall quality Q, explained at the end of this section. The weights for the quality functions have been adapted from [32] where applicable. Davis justifies the different weights due to the fact that some quality attributes are essential for the SRS and others are generally less important. These weights might be modified for each project to meet project-specific quality criteria.

Definition 48 (Internal Completeness of Requirements Knowledge): *The requirements knowledge in the Requirements Ontology is internally complete, if all necessary requirements artefacts, metadata for requirements and requirements relationships have been specified. Internal completeness is reached if the requirements configuration complies to all completeness rules (see Section 8.5.1).*

This definition follows up the Definition 25 of internal completeness for requirements in Section 8.5.1. Quality requires the completeness of requirements knowledge. As already

[4]Some of the definitions and most of the computations given in [32] have been modified by us to keep them up-to-date with today's Requirements Engineering knowledge and to make them applicable for practical use and ontology techniques.

mentioned, completeness is a relative measure and may be determined only in relation to an external reference [151]. This external reference is the Requirements Ontology. Thus, we define a target state (completeness of all metadata and requirements relationships) of the ABox in relation to the TBox and also determine the actual state of completeness in the ABox. We use this information to measure the internal completeness of the requirements knowledge in the requirements configuration as follows:

$$Q_1 = \frac{m_c}{m_n}$$

This formula measures the percentage of complete metadata for all requirements in the Requirements Ontology. Therefore, we compute all incomplete requirements (requirements with missing metadata) as explained in more detail in Section 10 and divide it by the overall number of specified requirements. We use (m_n) for the metadata that must be complete (target state) and (m_c) for the metadata that is indeed complete (actual state). Values range from 0 (100% incomplete) to 1 (100% complete). Since internal completeness is of main importance for reaching a high quality, we use a weight of 1 for internal completeness, $W_1 = 1$.

Definition 49 (Validity of Requirements Knowledge): *The requirements configuration is valid, if all individual requirements are valid (regarding their context and domain).*

In Section 1 we defined correctness formally as the combination of completeness and consistency which is also measured by the quality rules. However, we also discussed that it is hard to validate the correctness of requirements semantics. This can only be accomplished by stakeholder validations. Therefore, we acknowledge this aspect by the above definition and measure the requirements that have been validated and set "*valid*" by the Requirements Engineer. We use the following quality metric:

$$Q_2 = \frac{r_c}{r_n}$$

where r_c is the number of valid requirements and r_n the number of all requirements in the Requirement Ontology. Values range from 0 (100% incorrect) to 1 (100% correct). Because requirements validity is so critical to project success, we use a weight of 1, $W_2 = 1$ [32].

Definition 50 (Verifiable Requirements Knowledge): *Requirements knowledge can be verified if a test-case or a metric is specified for each requirement.*

Requirements Knowledge can only be verified if it provides means to verify the individual requirements specified within. Thus, it is necessary to provide options for each requirement to be verified. For OntoReq, these are test-cases and metrics. We use the following function to compute the percentage of verifiable requirements:

$$Q_3 = \frac{r_v}{r_n}$$

where r_v is the number of requirements with a test-case and/or metric and r_n the number of all requirements in the Requirements Ontology. Values range from 0 (100% unverifiable) to 1 (100% verifiable). Since we agree that the verifiability of requirements is not as important as completeness and consistency and is usually not a source of errors in the SRS, we adopt the weight of 0.7 as recommended by Davis in [32], $W_3 = 0.7$.

Definition 51 (Internal Consistency of Requirements Knowledge): *The requirements configuration is consistent if it is free of conflicts [32, 63].*

The consistency of the requirements knowledge is the percentage of conflict-free requirements. The above definition is not as strong as the definition for consistency of the requirements configuration given in Section 8.5.2. This is due to the fact that a requirements configuration is a selection of requirements from the requirements specification and thus, can be checked against more consistency rules (e.g., whether mandatory requirements have been included) than the overall requirements knowledge. It must be noted that conflicting requirements in the Requirements Ontology are actually no problem as long as they are not included in the final requirements configuration (or eliminated beforehand). So we propose the following function for measuring the internal consistency of the requirements knowledge in the Requirements Ontology:

$$Q_4 = \frac{r_c}{r_n}$$

where r_c is the number of requirements without a conflict and r_n the number of all requirements in the Requirements Ontology. Values range from 0 (100% inconsistent) to 1 (100% consistent). In accordance to Davis, we weight this function with $W_4 = 1$ due to its importance for the overall quality of the SRS.

Definition 52 (Traceability of Requirements Knowledge): *Requirements knowledge is traceable if it facilitates the traceability of requirements artefacts, especially of each requirement. A requirement(s) (artefact) is traceable if its life from analysis to implementation can be followed. This requires at least the specification of its source, an author responsible for the documentation, the goal it must satisfy, a use-case where it is described and a test-case or metric for verification.*

In order to measure the traceability of requirements knowledge in the Requirements Ontology, we need to be able to trace each single requirement. If we want to follow the life of a requirement, we need to start from its very beginning of life, its origin. Thus, we need to document the source of a requirement (e.g., a customer wish, a document, law, etc.). For possible callbacks we need to know who was responsible for the documentation of the requirement, so we need to specify the author. Since every requirement should satisfy a goal, this must also be traceable. A use-case is important to describe the requirement in a specific situation or environment. This is crucial for comprehensibility and, later on, for design. Finally, by using test-cases and metrics it is possible to verify the requirement after its implementation. Via this chain it is possible to trace the requirement, for example from a verification result up to the goal it was meant to contribute to. However, this chain is only a minimum of knowledge that must be available for traceability. Of course it can be extended, e.g., by adding the responsible person for design, realisation, test, etc. The more information about a requirement's "life" is provided and accessible, the better it can be traced. For measuring traceability of the requirements knowledge, we check the presence of these relations to requirements artefacts for each requirement. We use the following function:

$$Q_5 = \frac{r_t}{r_n}$$

where r_t denotes the requirements that are traceable and r_n is the number of all requirements in the Requirements Ontology. Values range from 0 (100% untraceable) to 1 (100%

traceable). We use the weight $W_5 = 0.6$ since traceability is not of main importance for the quality of the SRS.

Definition 53 (Uncriticality of Requirements Knowledge): *Uncritical requirements are requirements with low or medium risk or cost and without any obstacles.*

If a SRS includes many requirements with a high risk, high costs or obstacles assigned to them, it might be more critical than a SRS with low costs and few risks. Of course, this measure alone is not sufficient to make assumptions of the final criticality of a requirements configuration since the specified risks, costs and obstacles are only assumptions. Nevertheless, we are certain that this value enhances actions to reduce or eliminate risks and obstacles, allows for reviewing the costs of requirements and to reconsider and discuss their benefit compared to their criticality. Thus, we measure the percentage of critical requirements. It must be noted that risky and cost-intesive requirements only may become a problem if they are included in the requirements configuration, which is checked by the quality rules. Nevertheless, we will use the following function to give the Requirements Engineer a feeling for the whole requirements specification:

$$Q_6 = \frac{r_r}{r_n}$$

where r_r is the number uncritical requirements and r_n the number of all requirements in the SRS. Values range from 0 (100% critical) to 1 (100% uncritical). We recommend to weight this function with $W_6 = 0.5$ since critical requirements do not affect the consistency of the requirements knowledge.

These six quality functions can be used as single metrics for the above quality criteria as well as for computing the overall quality of the SRS. For the latter, we propose the following function:

$$Q = \frac{\sum_{i=1}^{6} W_i Q_i}{\sum_{i=1}^{6} W_i}$$

Although we propose these quality measurements for OntoReq, they may be easily adapted for any other knowledge that is based on any computable knowledge repository (e.g., a database). Moreover, these quality measurements may be modified and applied in different contexts and domains where quality plays a major role (e.g., safety-critical environments).

8.5.4 Goal Satisfaction

Lamsweerde in [131] discusses the problem of choosing alternative options for requirements by executing qualitative reasoning about them. Requirements Engineers have to explore the desired effects on the software on its surrounding environment and need to make assumptions about this environment [131]. When operationalizing a system goal by combining requirements, constraints, etc., different combinations might be envisioned and we need to select the "best" one [131]. Hence, these tasks involve trade-offs among a variety of (alternative and optional) requirements to find a requirements configuration that best satisfies the specified goals. We use reasoning techniques to support this decision process. Several approaches concentrate on (partial) goal satisfaction for RE. However, they differ a lot from each other. Some of them propose goal satisfaction on a very basic level, using two or three subjective levels of goal contribution (positive/negative, low/medium/high)

to specify the relation between requirements and goals (e.g., [46, 4]). Others come up with complex mathematical formulas and objective criteria for specifying this relationship (e.g., [89]).
Although the use of contribution levels is strongly subjective, we decided to use this method for OntoReq since it bears several advantages: (1) it is easy to understand by the Requirements Engineer, (2) it requires little effort to specify contribution levels and (3) it can be easily reasoned about. Thus, we intentionally decided against complex goal satisfaction algorithms that require difficult additional data to be specified by the Requirements Engineer. Certainly, using objective criteria for goal satisfaction and complex formulas as in [89] would allow goal satisfaction on a quite high level, but this would take much time for the Requirements Engineer to become acquainted with this technique and to finally use it.
Given these facts, we propose two requirements relationships: positive and negative goal contribution. A requirement can either be a positive or a negative contribution to a goal (or none, if not specified). This allows for selecting the best alternative among requirements regarding their goal satisfaction. If a requirement is, for example, optional and is also a negative contribution to a certain goal, it should be reconsidered whether it must indeed be included in a requirements configuration. Another case is a requirement which could be substituted by an alternative requirement with a positive goal contribution. Given a set of goals and a sufficient specification of goal contribution, OntoReq allows to generate a requirements configuration that best suits this set of goals.

8.5.5 Automated Generation of Requirements Configuration

The idea of the goal satisfaction method described in Section 8.5.4 may be applied to any other requirements artefact and relationship, even to a *set* of requirements artefacts. So it is also possible to compute a requirements configuration that satisfies a predefined set of attributes, e.g., lowest cost, lowest risk, highest priority, etc. By applying all of the previously described consistency and quality rules, such a requirements configuration can be guaranteed to reach the currently highest level of consistency and quality (regarding the specified requirements knowledge). Due to the variety of requirements knowledge captured in the Requirements Ontology, the Requirements Engineer can choose from a set of the most important attributes and will be acquainted with an automated suggestion of a requirements configuration that fits his needs. This configuration may be used as a basis and further modified (adding or excluding requirements) by the Requirements Engineer.
We allow for a selection of an arbitrary set of the following constraints for the automated generation of a requirements configuration :

- Level of Priority (High, Medium, Low)
- Level of Cost (High, Medium, Low)
- Level of Risk (High, Medium, Low)
- Best Achievable Goal Satisfaction
- Best Verifiability (Test-Cases, (Soft-)Metrics)
- Best Requirements Description (Use-Case, Scenario)

8.5.6 SPARQL Queries

Finally, in order to allow generic queries about the instantiated knowledge in the Requirements Ontology (ABox), we provide a set of useful queries to the Requirements Engineer. We propose the following queries whose result will be displayed:

- All functional requirements
- All non-functional requirements
- All platform requirements
- Requirements with a specific missing metadata (e.g., goal, use-case, etc.)
- Incomplete requirements
- Optional requirements
- Important requirements (high priority or mandatory)
- Risky requirements (requirements with a risk defined or high-cost or a conflict)
- Cost-intensive requirements (high cost)
- Validated requirements
- Not (yet) validated requirements
- Verifiable requirements (requirements with test-case or (soft-)metric)
- Requirements with missing verification method or metric
- Conflicting requirements
- Requirements with negative goal contribution

Most of the knowledge is explicitly captured in the ontology (e.g., validated requirements). However, we provide some convenient queries that use reasoning, e.g., for risky or important requirements. This list of queries is not meant to be complete. It can easily be extended or modified if necessary.

8.6 Guidance Support

We distinguish two concepts of guidance support for the Requirements Engineer in OntoReq: (1) support for the specification of requirements knowledge and (2) validation and error elimination support. For reasons of completeness, the latter one has already been described in Section 8.5. In this section, we conceptually describe how to provide Requirements Specification Guidance (RSG) and describe the modelling of a Guidance Ontology.

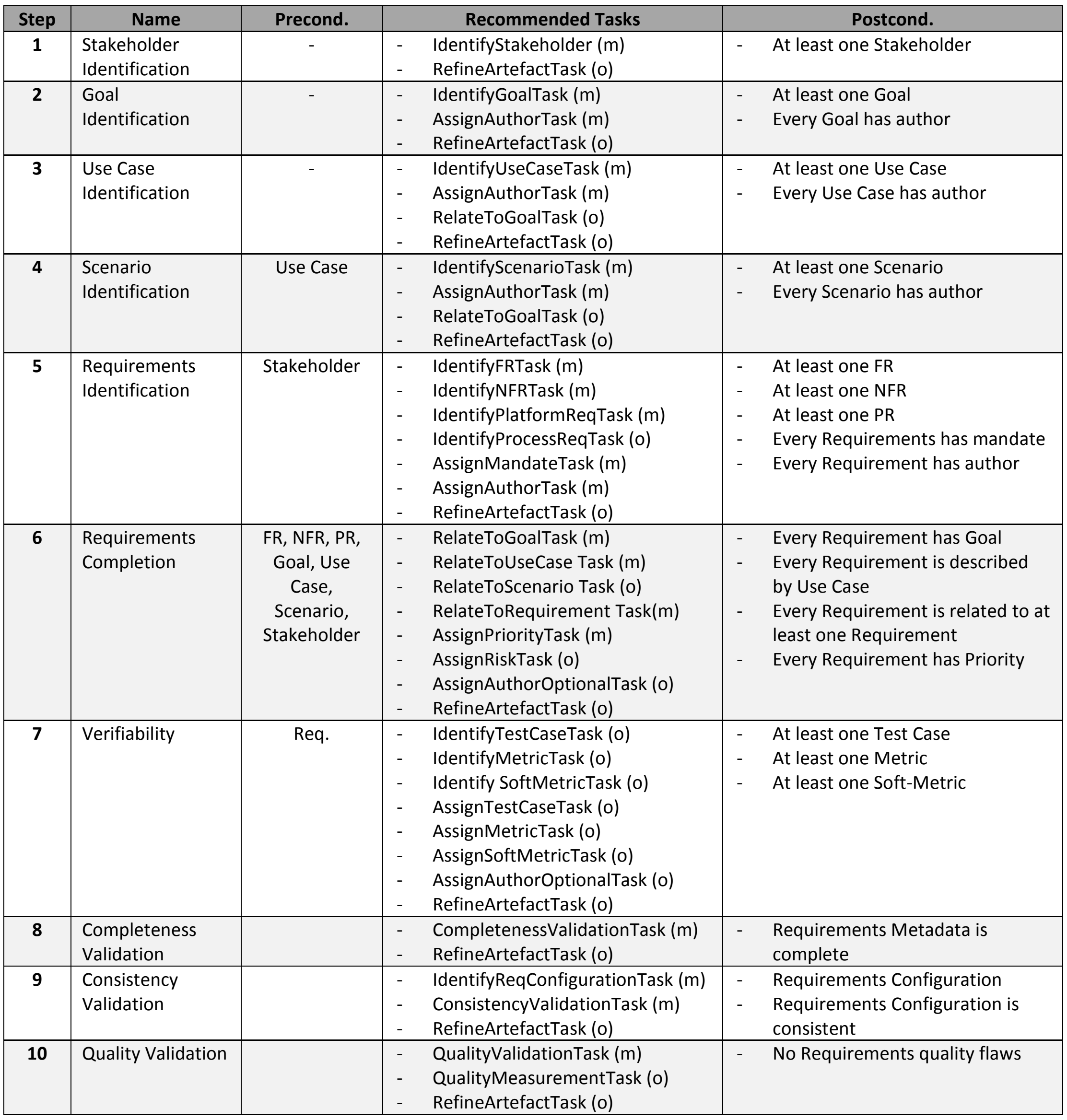
8.6.1 Definition of a GORE process

In order to guide the Requirements Engineer with OntoReq, we first need to define the tasks to be accomplished and the order of their execution. These tasks and their order are depicted in Table 8.9. To allow reasoning about the current state of the requirements knowledge and to identify completed and incompleted process steps, we define pre- and post-conditions for each task. Thus, a task is completed if all the post-conditions are satisfied. Since Requirements Engineering is an ongoing process, the use of these post-conditions allows for setting an already completed task back to the status “incomplete” if one or more of the conditions are not satisfied any more.
As can be seen in Table 8.9, we use six different types of tasks: (1) IdentificationTasks, (2) AssignTasks, (3) RelationTasks, (4) ValidationTasks, (5) MeasurementTasks and (6) RefineArtefactTasks. *IdentificationTasks* are needed to specify requirements and requirements artefacts (e.g., goal). Without any requirement or requirements artefact, the RE

process cannot be further executed. *AssignTasks* are used to complete the metadata about requirements and requirements artefacts (e.g., adding a priority), whereas relation tasks ask to interrelate the already existing requirements knowledge. *ValidationTasks* allow for the validation of completeness, consistency and quality and the *MeasurementTask* for the quality measurement of the requirements knowledge. Finally, *RefineArtefactTasks* enable the refinement of any requirements artefact.

Structured chronological vs. Flexible Guidance

One major problem arising during Requirements Engineering is the order of tasks to achieve. A variety of methods propose different guidelines and suggestions for this problem. Suggestions are as manifold as methods and their aims are. So it is one of our aims to guide the Requirements Engineer through the requirements specification process by displaying a chronological order of tasks to be completed. The (correct) execution of all these tasks guarantees internal completeness. Therefore, the completeness of the actual requirements knowledge must be validated at several times in the background to generate the tasks accordingly. However, we do not want to neglect the fact that a good number of Requirements Engineers may have already found a Requirements Engineering process capable for their project, company or situation. Therefore, we propose the guidance also in such a way that it is possible to leave the recommended *chronological* path suggested by the RSG and to accomplish instead all the displayed open tasks in the order of choice. Nevertheless, the Requirements Engineer will still be supplied with all recommended tasks and point guidance that is continuously refreshed and, thus, can return to chronological guidance at any time. This may be realised by a complete list of open tasks hold by the Guidance Ontology in the background. This list can be filtered and results in the recommended tasks that follow the structured guidance. Thus, the Requirements Engineer may always switch between accomplishing any of the possible open tasks in the order of his choice or the recommended ones in chronological order.



Step	Name	Precond.	Recommended Tasks	Postcond.
1	Stakeholder Identification	-	- IdentifyStakeholder (m) - RefineArtefactTask (o)	- At least one Stakeholder
2	Goal Identification	-	- IdentifyGoalTask (m) - AssignAuthorTask (m) - RefineArtefactTask (o)	- At least one Goal - Every Goal has author
3	Use Case Identification	-	- IdentifyUseCaseTask (m) - AssignAuthorTask (m) - RelateToGoalTask (o) - RefineArtefactTask (o)	- At least one Use Case - Every Use Case has author
4	Scenario Identification	Use Case	- IdentifyScenarioTask (m) - AssignAuthorTask (m) - RelateToGoalTask (o) - RefineArtefactTask (o)	- At least one Scenario - Every Scenario has author
5	Requirements Identification	Stakeholder	- IdentifyFRTask (m) - IdentifyNFRTask (m) - IdentifyPlatformReqTask (m) - IdentifyProcessReqTask (o) - AssignMandateTask (m) - AssignAuthorTask (m) - RefineArtefactTask (o)	- At least one FR - At least one NFR - At least one PR - Every Requirements has mandate - Every Requirement has author
6	Requirements Completion	FR, NFR, PR, Goal, Use Case, Scenario, Stakeholder	- RelateToGoalTask (m) - RelateToUseCase Task (m) - RelateToScenario Task (o) - RelateToRequirement Task(m) - AssignPriorityTask (m) - AssignRiskTask (o) - AssignAuthorOptionalTask (o) - RefineArtefactTask (o)	- Every Requirement has Goal - Every Requirement is described by Use Case - Every Requirement is related to at least one Requirement - Every Requirement has Priority
7	Verifiability	Req.	- IdentifyTestCaseTask (o) - IdentifyMetricTask (o) - Identify SoftMetricTask (o) - AssignTestCaseTask (o) - AssignMetricTask (o) - AssignSoftMetricTask (o) - AssignAuthorOptionalTask (o) - RefineArtefactTask (o)	- At least one Test Case - At least one Metric - At least one Soft-Metric
8	Completeness Validation		- CompletenessValidationTask (m) - RefineArtefactTask (o)	- Requirements Metadata is complete
9	Consistency Validation		- IdentifyReqConfigurationTask (m) - ConsistencyValidationTask (m) - RefineArtefactTask (o)	- Requirements Configuration - Requirements Configuration is consistent
10	Quality Validation		- QualityValidationTask (m) - QualityMeasurementTask (o) - RefineArtefactTask (o)	- No Requirements quality flaws

Figure 8.9: GORE process steps (in collaboration with Tittel [126])

Step Guidance

Guidance can always be offered when the current intention of the user has been recognised. Thus, *step guidance* is always related to a *guidance point* consisting of a situation and an intention that form together a context [120]. Therefore, we define *guidance points* as a tuple of a requirements artefacts (abbreviated as artefact) and a task: GP = <artefact;task>. Thus, the situation describes the actual artefact, in our case for example a goal or requirement. The intention describes which activity shall be executed regarding this artefact, e.g., goal refinement. The achievement of an intention causes a change of that artefact. For this reason, guidance points are constructed from all reasonable combinations of two sets: the set of all requirements artefacts and the set of all intentions. Point guidance

provides guidelines. They include instructions for every guidance point on how to execute a certain action[5] [120].
The steps for these step guidance are listed in Table 8.9. For these steps, the requirements engineer will be equipped with guidelines for each task to be accomplished.

Flow Guidance

The proposed flow guidance for OntoReq aims to support the requirements engineer in choosing the most applicable task from a set of possible tasks. Additionally, we do not want to provide support only for one specific GORE method, but enable the requirements engineer to maintain his approach for GORE he is familiar with. At the same time, we aim to support those requirements engineers that wish a chronological process for the specification and validation of requirements knowledge which can be followed easily. Thus, we decided to extract main GORE tasks that are common to all GORE methods. These tasks can be ordered chronological and summarized to ten *process steps*, illustrated in Table 8.9. These tasks are modelled according to the point guidance. The execution of a task is either optional or mandatory. Each guidance point is connected with textual guidelines, describing how to complete this task. The execution of a task leads to a new guidance point in the guidance process. While the point guidance is visible to the user by a represented list of artefacts and actions, the flow guidance remains invisible and is only recognizable by a refreshed list of tasks. An example for flow guidance is given in Figure 8.10.

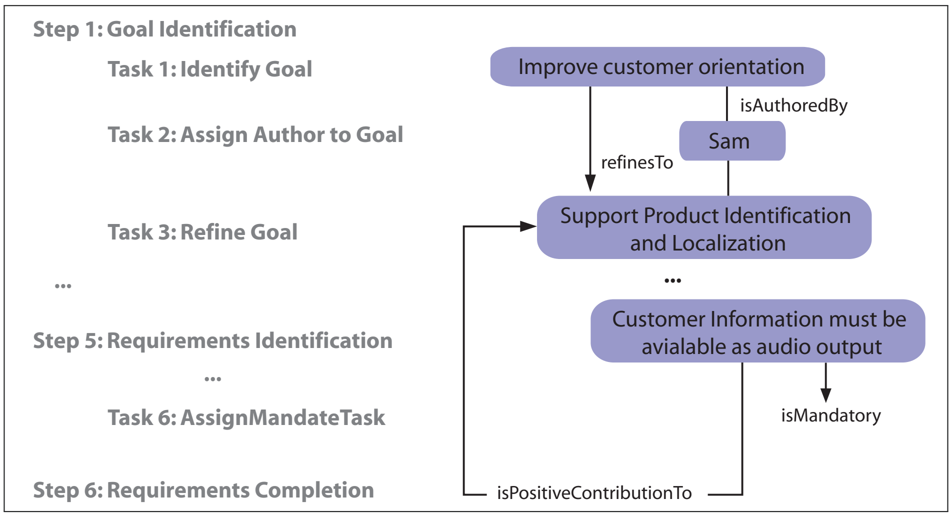

Step 1: Goal Identification
Task 1: Identify Goal
Task 2: Assign Author to Goal
Task 3: Refine Goal
...
Step 5: Requirements Identification
...
Task 6: AssignMandateTask
Step 6: Requirements Completion
Improve customer orientation
isAuthoredBy
Sam
refinesTo
Support Product Identification and Localization
...
Customer Information must be avialable as audio output
isMandatory
isPositiveContributionTo

Figure 8.10: Example for Flow Guidance

Generally, we distinguish between optional and mandatory tasks. This distinction follows the completeness and consistency rules described in Sections 8.5.1 and 8.5.2. Since these rules distinguish optional from mandatory information to be captured, we address this by suggesting optional and mandatory tasks. Figure 8.10 shows an example for flow guidance where the process steps and related tasks are listed on the left side and the modification of the requirements knowledge on the right side.

[5]Originally, Si-Said1997 in [120] uses the term “intention” instead of action. However, we consider “action” as more appropriate due to the fact that intentions cannot be executed.

8.6.2 Requirements Guidance Ontology

Many guidance approaches propose general guidelines but do not consider the actual state of knowledge. Thus, they are rather informal instead of precisely defining a process. Thus, in order to support the requirements engineer, we continuously need to update the information about the actual requirements knowledge. Only with this information, it is possible to generate specific guidelines for each guidance task to be achieved.
Therefore, we propose a Requirements *Guidance* Ontology (abbreviated as Guidance Ontology) that links to the Requirements Ontology to allow for ontology-driven step- and flow guidance for the Requirements Specification Process. This Guidance Ontology must contain a copy of the requirements knowledge from the Requirements Ontology to provide guidance based on the concrete state of the knowledge. Furthermore, it comprises concepts that are necessary for guidance in general, e.g., defining pre- and postconditions of guidance tasks. These concepts are explained in the following.

Ontology Structure

To enable point guidance for OntoReq, we need, additionally to the requirements knowledge, the following information in the Guidance Ontology:

- Task Type: the type of action to be executed
- Artefact: the requirements artefact that the task should be performed upon
- ArtefactInState: holds instances of requirements artefacts (goals, requirements, use-cases) that are in a certain state (complete, incomplete, valid, invalid)
- Identifier: individuals of this class are placeholders for tasks that are not connected to any requirements artefact (this is the case for all identification tasks)
- Instruction: detailed instruction for the particular task and artefact
- Guideline: additional information on how to execute a particular type of task
- Task priority: importance of a particular task, predefined in the Guidance Ontology

These concepts are modelled as classes in the Guidance Ontology that can be instantiated by individuals, e.g., a specific guideline or type of task. The following example illustrates a possible instantiation for the TaskType `RefineGoalTask`.

- TaskType: RelateToGoalTask
- Artefact: Goal1
- Instruction: Relate goal Goal1 to any requirement.
- Guideline: Relate this goal to one or several requirements that contribute to the satisfaction of this goal or hinder the satisfaction of this goal.
- Priority: 1

A complete list of the ontology classes is given in Figure 8.11. Furthermore, we propose the following object properties additional to the object properties copied from the Requirements Ontology:

- isPreConditionOf: links requirements artefacts to possible tasks executed on a specific requirements artefact
- hasMandatoryTask/ hasOptionalTask: subproperties of *isPreConditionOf*; relate requirements artefacts to mandatory or optional tasks
- hasPostCondition: links requirements artefacts to the succeeding artefact
- hasIdentified: chain property (identifyArtefact $\circ$ hasPostcondition $\rightarrow$ hasIdentified) that states that a requirements artefact has been identified

Table 8.9 summarizes all process steps, the associated tasks, pre- and postconditions. All postconditions of one particular step must be fulfilled before the next step of the chronological guidance will be displayed. Additionally, all open tasks shall be permanently displayed. Optional tasks have an optional postcondition, which is neglected in the table for comprehensibility.

Connecting the Requirements Ontology with the Guidance Ontology

When modifying the requirements knowledge (e.g., adding a new requirement), only the information in the Requirements Ontology is changed. In order to generate meaningful guidance steps and tasks, we need to connect the concepts of the Requirements Ontology with concepts of the Guidance Ontology. There are two options: Mapping (also known as Alignment or Matching) and Merging.
Ontology mapping is the process of determining correspondences (semantic equivalence) between concepts. When mapping two ontologies, each of them remains independent. Mapping functions define which classes of one ontology shall be mapped to another ontology.
Ontology merging is similar to database merging (schema matching) and brings together two conceptually divergent ontologies or the instance data associated to two ontologies [137].
Both, mapping and merging, is possible for connecting the Requirements Ontology with the Guidance Ontology. However, both techniques have their advantages and disadvantages, summarized in Table 8.2:

	Mapping	**Merging**
+	Ontologies can differ in their structure Requirements Ontology and Guidance Ontology remain independent from each other and can be further used	No mapping functions neccesarry Information additional to the guidance process can be specified
−	Implementation depends on Mapping functions and thus, depends on the Requirements and Guidance Ontology No additional knowledge about guidance process is neccesary	Equivalent concepts in both ontologies must be identical in their structure The result of merging is a complex ontology

Table 8.2: Advantages and disadvantages of mapping and merging, according to [126]

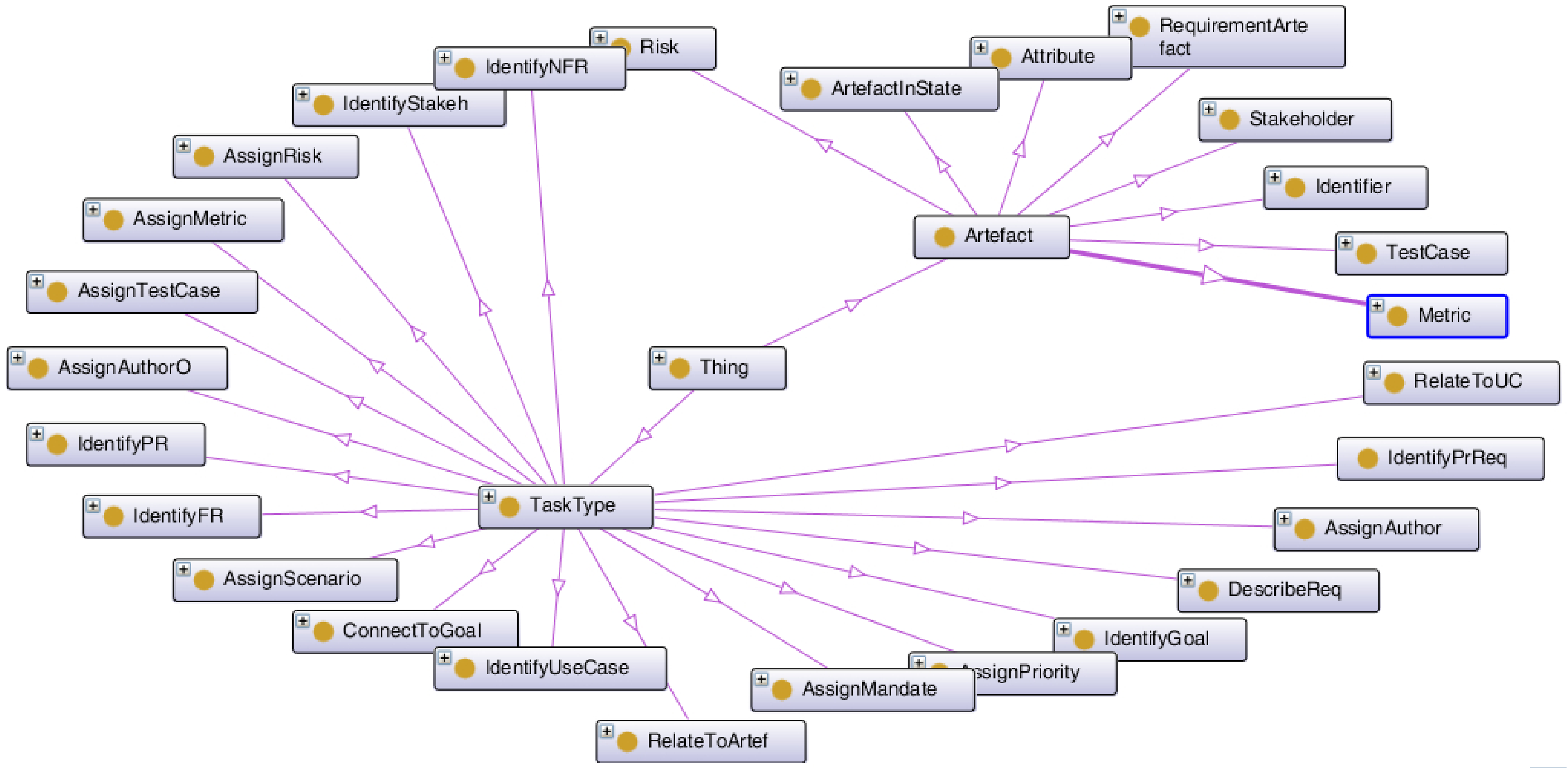


Figure 8.11: Concepts of Guidance Ontology (Task names abbreviated)

8.6.3 Guidance Engine

Since we need additional software to manage and utilize the Requirements Ontology and the Guidance Ontology and the knowledge captured within, we suggest a Java application as Guidance Engine. This Guidance Engine either realises the merging or mapping of both ontologies and provides all the functionality (step and flow guidance) described above. In summary, this Guidance Engine must realise the following requirements:

- Suitable tasks must be displayed for each state of the Guidance Ontology to allow for following a structured GORE process
- Information on how to successfully accomplish a task must be provided for each task
- The guidance process must define states of the Guidance Ontology where a set of tasks is applicable
- The guidance tasks must be distinguished into optional and mandatory tasks
- Tasks must be prioritised
- Requirements and Guidance Ontology and the instantiated knowledge within must be connected in a way that allows for reasoning process steps and specific tasks for different states of the requirements knowledge.

8.7 Summary

This chapter provides a list of identified shortcomings and problems requirement engineers and companies are facing. These challenges are captured in a taxonomy established on basis of problem categories identified by us in various publications from industry and research in the last two decades. We deduce these challenges into general requirements for an improved RE process, that is a RE process without the identified flaws. Furthermore, we use examples to exemplify their application for different purposes (RE process improvement, development of guidance software, development of a RE tool). We also use a selection of these requirements to specify the aim of our approach.
The main part of this chapter describes our concept for the application of ontology techniques to improve and verify the internal completeness, consistency and quality of requirements knowledge, referred to as **OntoReq**. Therefore we state the relevant requirements and limitations. OntoReq is based on an ontology knowledge repository. This knowledge repository serves as meta model (Requirements Ontology), providing a sufficient structure to capture all relevant requirements artefacts and predefines a set of meaningful metadata to formalize requirements knowledge and finally allow the validation of internal completeness, consistency and quality. Additionally, we describe several requirements services based on this Requirements Ontology. These services include the validation of internal completeness, consistency and quality of requirements knowledge as well as the quality measurement of the requirements knowledge based on established metrics. Furthermore, we conceptually describe a method to automatically generate a requirements configuration regarding a set of predefined attributes the Requirements Engineer can select from. We also describe the application of ontology techniques to enable point and flow guidance for GORE.

Part III

The Technical Solution

9 Requirements Ontology

In this chapter, we describe the realisation of the Requirements Ontology TBox conforming to the conceptual model and requirements stated in Part II. We introduce the various ontology classes and properties and illustrate them in tables with metamodel fragments. Ontology classes are represented in `typewriter font` with a capitalized letter at the beginning (e.g., `Requirement`) and properties in italic font (e.g., *isRelatedTo*). For comprehensibility we neglect namespaces such as “rdf:” or “owl:” before ontology components.

The chapter is structured as follows: we first introduce the different ontology classes and their relations in Section 9.1. Subsequently, we describe the concept of equivalentClasses and their use in the Requirements Ontology in Section 9.2. This is follwed by the object and data properties listed in Section 9.3. A summary is provided in Section 9.4.

9.1 Ontology Classes (Tbox)

The most important classes of the Requirements Ontology are: `Requirement`, `RequirementsArtefact` and `Attribute`. Table 9.1 depicts these main classes and their interrelations. Requirements are interrelated with each other and with other requirements artefacts by various objectProperty relations described in Section 9.7. Additionally, they may have several attributes, e.g., an author or priority. These attributes are a part of the important metadata and are necessary to improve the internal completeness of requirements knowledge.

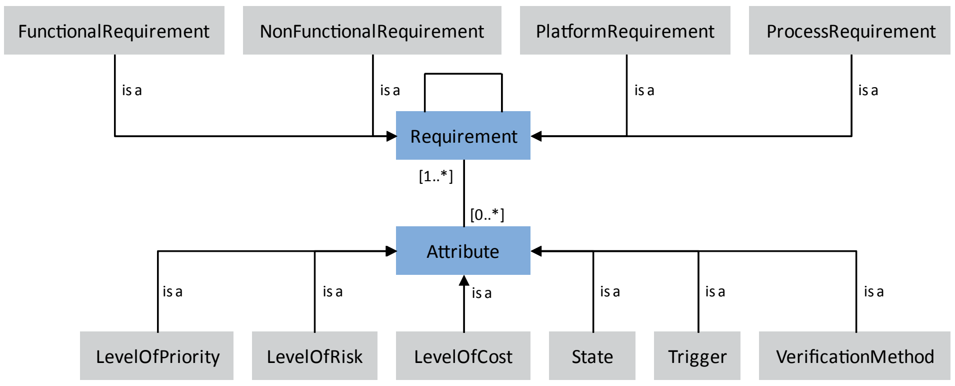


Figure 9.1: Requirements Ontology (`Requirement` and `Attribute`) and their Subclasses

The class `Requirement` splits into several types of requirements (see Figure 9.1). According to the IEEE standard [63], we split the class `NonFunctionalRequirement` additionally into `AccessibilityRequirement`, `EfficiencyRequirement`, `PerformanceRequirement`, `SecurityRequirement` and `UsabilityRequirement`. Further ontology classes in the Requirements Ontology are summarized in the following tables. Since owl:Thing is the superClass of every ontology class, we neglect this fact in the tables and only concentrate on real superclasses.

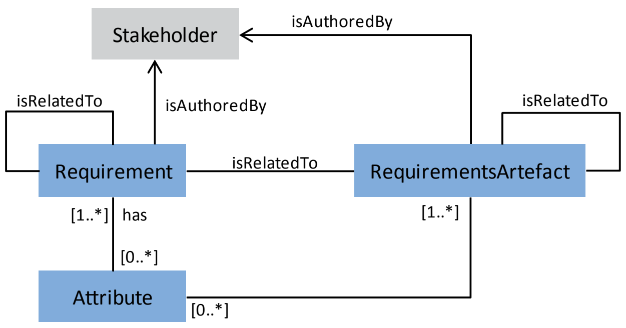
`Requirement`	A `Requirement` captures individual requirements and their description.
Superclasses	–
Subclasses	`FunctionalRequirement`, `NonFunctionalRequirement`, `ProcessRequirement` and `PlatformRequirement`
Meta Model fragment	 Stakeholder isAuthoredBy isRelatedTo isAuthoredBy isRelatedTo Requirement isRelatedTo RequirementsArtefact [1..*] has [1..*] [0..*] Attribute [0..*]
Restrictions	*isAuthoredBy* `some Stakeholder` *hasSource* `some Source` *hasObstacle* `some Obstacle` *hasScenario* `some Scenario` *hasUseCase* `some UseCase` *hasTestCase* `some TestCase` *hasSoftMetric* `some SoftMetric` *isNegativeContributionToGoal* `some Goal` *isPositiveContributionTo* `some Goal` *refinesTo* `some Requirement` *isMandatory* `some boolean` *isValid* `some boolean`

Table 9.1: Requirements Ontology (`Requirement`)

As can be seen in Table 9.1, `Requirements` are related to many other ontology concepts. For reasons of space, we summarize all relations to requirements and requirements artefacts with the objectProperty "isRelatedTo". A detailed description of the summarized relationships (object and data properties) is given in Section 9.3.
The class `RequirementsArtefact` is further divided (see Table 9.2). These requirements artefacts are related to requirements and capture additional knowledge that facilitates the validation and traceability of requirements and decisions.
The class `Attribute` comprises further metadata, such as priorities, risks, state, triggers and verification methods. These attributes are important for the completeness and quality validations (see Table 9.3).
`Goals` and `Obstacles` can be refined to a goal (or obstacle respectively) or operationalised to any requirements artefact, e.g., a requirement or use-case. This operationalisation can

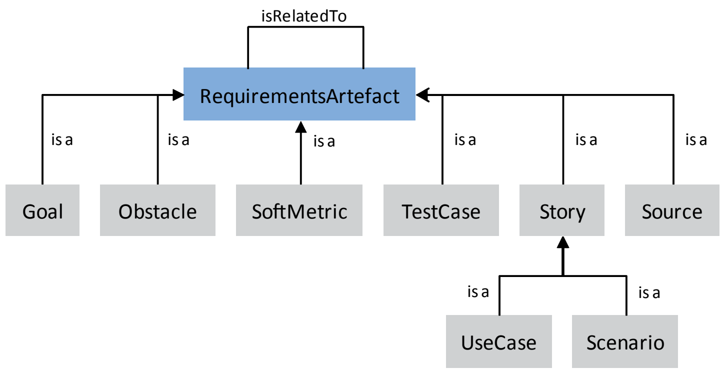
`Requirements-Artefact`	A `RequirementsArtefact` captures additional requirements knowledge that can be related to a `Requirement`.
Superclasses	–
Subclasses	`Goal`, `Obstacle`, `Story`, `SoftMetric` and `TestCase`
Meta Model fragment	 isRelatedTo RequirementsArtefact is a Goal Obstacle SoftMetric TestCase Story Source UseCase Scenario
Restrictions	*isAuthoredBy* `some Stakeholder` *refinesTo* `some RequirementsArtefact`

Table 9.2: Requirements Ontology (`Requirements Artefact`)

be understood as a further development of a goal's or obstacle's statement. In contrast to a refinement, it thereby changes the type of the requirements artefact. It is, for example, often the case that requirements are operationalised from goals. This operationalisation allows for tracing the development of requirements and requirements knowledge.

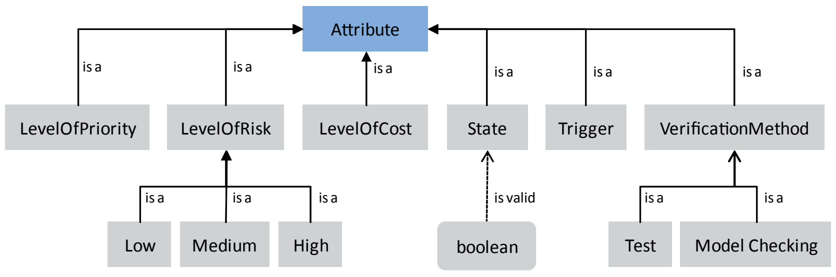
`Attribute`	An `Attribute` captures additional metadata about individual `Requirements` and `RequirementsArtefacts`.
Superclasses	–
Subclasses	`LevelOfPriority`, `LevelOfRisk`, `LevelOfCost`, `State`, `Trigger`, and `VerificationMethod`
Meta Model fragment	 Attribute is a LevelOfPriority LevelOfRisk LevelOfCost State Trigger VerificationMethod Low Medium High is valid boolean Test Model Checking
Restrictions	*isAuthoredBy* `some Stakeholder`

Table 9.3: Requirements Ontology (`Attribute`)

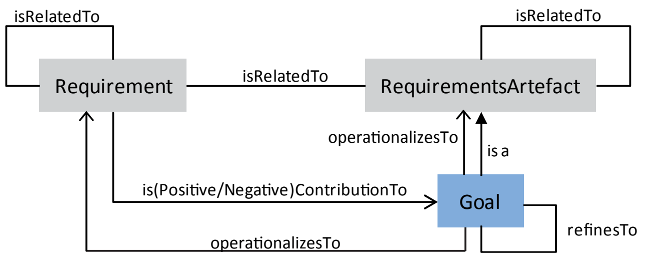
`Goal`	A `Goal` captures a declarative statement of intent to be achieved.
Superclasses	`RequirementsArtefact`
Subclasses	`BusinessGoal`, `ProcessGoal`, `SystemGoal`
Meta Model fragment	 isRelatedTo isRelatedTo Requirement isRelatedTo RequirementsArtefact operationalizesTo is a is(Positive/Negative)ContributionTo Goal refinesTo operationalizesTo
Restrictions	*isAuthoredBy* `some Stakeholder` *refinesTo* `some Goal` *operationalisesTo* `some RequirementsArtefact` *operationalisesTo* `some Requirement`

Table 9.4: Requirements Ontology (`Goal`)

The main types of requirement descriptions are use-cases and scenarios (see Table 9.6, which define special interactions between actors and the system. Usually, they are modelled as use-case diagrams, sequence diagrams and so on. The Unified Modeling Language (UML) provides different types of diagrams for these purposes. However, we cannot store such

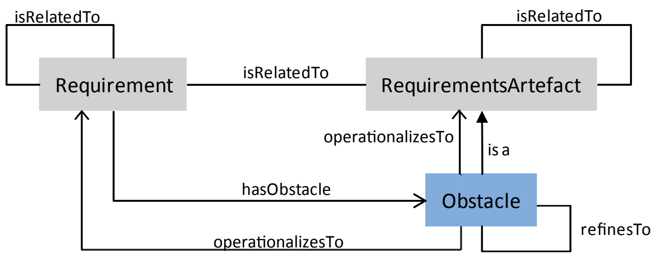
`Obstacle`	An `Obstacle` captures a declarative statement of identified behaviour that hinders the satisfaction of `Goals` or `Requirements`.
Superclasses	`RequirementsArtefact`
Subclasses	–
Meta Model fragment	 isRelatedTo isRelatedTo Requirement isRelatedTo RequirementsArtefact operationalizesTo is a hasObstacle Obstacle refinesTo operationalizesTo
Restrictions	*isAuthoredBy* `some Stakeholder` *refinesTo* `some Obstacle` *operationalisesTo* `some RequirementsArtefact` *operationalisesTo* `some Requirement`

Table 9.5: Requirements Ontology (`Obstacle`)

models in an ontology. Nevertheless they are important for Requirements Engineering. Hence, we store the *textual description* of use-cases and scenarios in the Requirements Ontology as annotation property of use-case instances and scenario instances respectively. The concrete models remain outside in the external part of the Requirements Specification and we link instances of `Requirement` and `RequirementsArtefact` to these models via the *hasSource* object property. This way, we still provide sufficient means for a complete specification of requirements and the validation purposes.

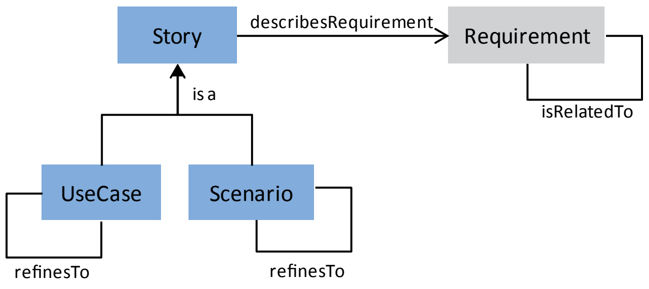
`Story`	A `Story` captures descriptions (`UseCase`s and `Scenario`s) of the interactions possible between actors and a system. A `Story` describes some functionality that the system must provide for the actors involved in the `UseCase`s and `Scenario`s.
Superclasses	`RequirementsArtefact`
Subclasses	`UseCase`, `Scenario`
Meta Model fragment	 Story describesRequirement Requirement isRelatedTo is a UseCase Scenario refinesTo refinesTo
Restrictions	*isAuthoredBy* `some Stakeholder` *refinesTo* `some Story` *describesRequirement* `some Requirement` *hasSource* `some Source`

Table 9.6: Requirements Ontology (`Story`)

9.2 Equivalent Classes

OWL allows for specifying equivalent classes in ontologies, that are classes with specific restrictions on properties or other classes. Although, it is possible to get the same query results without equivalent classes, it is more convenient to define equivalent classes. Especially if an equivalent class contains several restrictions, it is faster to query for individuals of that class instead of querying for individuals of several classes with certain properties. Furthermore, the ontology becomes far more intuitive and understandable for humans. Listing 9.1 below illustrates two of these equivalent classes in the Requirements Ontology.

Listing 9.1: Equivalent Classes

```
Class: InconsistentRequirement

    EquivalentTo:
        (isMandatory value ‘‘true’’^^xsd:boolean)
        and (isOptional value ‘‘true’’^^xsd:boolean)

    SubClassOf:
        Requirement

Class: TraceableRequirement

    EquivalentTo:
        (hasSource some Source)
        and (isAuthoredBy some Stakeholder)
        and (isConnectedWithTestCase some TestCase)
        and (isConnectedWithUseCase some UseCase)

    SubClassOf:
        Requirement
```

Instead of querying for individuals of the class `InconsistentRequirement`, it is also possible to query for individuals of the class `Requirement` that have the data properties *isMandatory* and *isOptional* with the value "true". In this case, individuals of the class `InconsistentRequirement` are wrongly specified since no requirement can be mandatory and optional at the same time. Similar to that, individuals of the class `TraceableRequirement` can be queried easier than querying for requirements with the above object property restrictions.

9.3 Properties

OWL distinguishes between two main categories of properties: object properties that link individuals to individuals and datatype properties that link individuals to data values [53]. "An object property is defined as an instance of the built-in OWL class owl:ObjectProperty. A datatype property is defined as an instance of the built-in OWL class" [53]. For a property, one can define a *domain* and a *range*. A domain axiom "asserts that the subjects of such property statements must belong to the class extension of the indicated class description" [53]. The range axiom "asserts that the values of this property must belong to the class extension of the class description or to data values in the specified data range" [53]. Furthermore, properties have a direction, from domain to range. However, sometimes it is useful to define relations in both directions, e.g., "persons own cars", "cars are owned by persons" (inverse property) [53].
In the following, we briefly introduce the object and data properties of the Requirements Ontology, summarized in Table 9.7.

9.3.1 Object Properties

"An object property is a binary predicate used to state facts of the form subject predicate object, where both subject and object are entities (i.e. individuals)" [59]. We use the object properties in Table 9.7 to interrelate the various classes in the Requirements Ontology. These object properties allow for interrelating the requirements knowledge and thus, facilitate the derivation of solution suggestions, e.g., alternative requirements. This is explained in more detail in Chapter 10. The following table lists the object properties of the Requirements Ontology and their domains and ranges.

9.3.2 Data Properties

The requirements Ontology currently provides two data properties to specify the mandate and validation status of requirements as shown in table 9.8.

Domain	**Object Property/ Inverse Property**	**Range**
Requirement	hasObstacle	Obstacle
Refinement	hasRefinementReason	RefinementReason
Refinement	hasRefinementSource	Requirement
Refinement	hasRefinementTarget	Requirement
Requirement	excludesReq/ isExcludedByReq	Requirement
Requirement	isAlternativeTo/ isAlternativeOf (symm.)	Requirement
Requirement	impliesReq/ isImpliedByReq	Requirement
Requirement	isInConflictWith (symm.)	Requirement
	refinesTo/ isRefinementOf	
Requirements-Artefact	operationalisesTo	RequirementsArtefact
Requirement	hasScenario	Scenario
Requirement	hasUseCase	UseCase
Story	describesRequirement	Requirement
Requirement	hasSoftMetric	SoftMetric
Requirement	hasSource	Source
	isAuthoredBy	Stakeholder
Requirement	hasTestCase	TestCase
Requirement	isNegativeContributionToGoal	Goal
Requirement	isPositiveContributionToGoal	Goal
	throwsErrorException	ErrorException

Table 9.7: Object Properties in the Requirements Ontology

Domain	**Data Property**	**Range**
Requirement	isValid	boolean
Requirement	isMandatory	boolean
Requirement	isOptional	boolean

Table 9.8: Data Properties in the Requirements Ontology

9.3.3 Property Chains for automatic completion of requirements knowledge

OWL 2 allows for deriving a new object property fact by defining a property as composition of two or more object properties in a chain that connect resources [16], [59]. Thus, the entity in the object position of one fact (other than the last fact) is also the subject of the following fact [59]. We use these object property chains in the Requirements Ontology to support the requirements engineer in automatically completing the requirements knowledge where it is semantically feasible. This is the case for requirements that are related to a Use-Case or Test-Case and are refined to a more concrete requirement. Actually, this refined requirement must be related to the same Use-Case or Test-Case. However, this is often forgotten for refinements. Thus, we automatically add Use-Cases and Test-Cases to refined requirements if there was such a relation for the more abstract requirement.

Therefore, we use object property chains and inverse properties, as illustrated below in Listing 9.2.

Listing 9.2: Property Chains for automatic completion of requirements knowledge

```
ObjectProperty: isConnectedWithUseCase

    Domain:
        Requirement

    Range:
        UseCase

    SubPropertyChain:
        isRefinementOf o isConnectedWithUseCase
```

Listing 9.2 demonstrates the application of the inverse object property *isRefinementOf* of the property *refinesTo* that is composed with the object property *isConnectedWithUseCase*. This way, each requirement that is a refinement of a more abstract requirement which is connected with a Use-Case will automatically be connected with the same Use-Case as well.

9.4 Summary

This chapter describes the realisation of the Requirements Ontology metamodel as ontology TBox conforming to the conceptual model and requirements described in Section 8.4. We introduce the different ontology classes and their relations. The most important classes of the Requirements Ontology are: `Requirement`, `RequirementsArtefact` and `Attribute`. The class `RequirementsArtefact` comprises amongst others the subclasses `Goal`, `Obstacle`, `Requirement`, `Use-Case` and `Test-Case`. Subsequently, the object and data properties are listed that are needed to interrelate requirements artefacts. Metamodel fragments illustrate the different ontology components and their usage.

10 Implementation Patterns for Requirements Validation Services

While the TBox of the Requirements Ontology forms the Meta Model for the requirements knowledge, the ABox holds the concrete requirements knowledge in form of individuals (instances) of this Meta Model. In order to provide requirements validation services, we need a knowledge base. This knowledge base is generated by reasoning about the Requirements Ontology and builds the basis of the validation services. The OWL API[1] [61] is a Java Framework that allows for manipulating ontologies (loading, changing, saving, reasoning, etc.). Thus, we need additional software to manage and utilize the Requirements Ontology and the knowledge captured within. Therefore, we implement the Java application OntoReq that demonstrates the application of the concepts described in Part II. OntoReq uses the OWL API and mainly includes methods that enable the validation of the requirements knowledge. A protypic user interface illustrates that the Requirements Ontology can be kept in the background, invisible to the user. The input, manipulation and validation of requirements knowledge can be handled via this user interface.

The following sections describe the technical solution for the requirements validation services we suggested in Chapter 8.5. The chapter is structured as follows: first, we briefly introduce the TrOWL reasoner in Section 10.1 that is used for our approach. Section 10.2 proposes two options to identify incomplete requirements knowledge. We first explain a method to avoid closed world reasoning by finding empty property sets and secondly illustrate how to use closed world reasoning with negation as failure. Section 10.3 describes how to validate for consistency and Section 10.4 explains the validation and measurement of the quality of requirements knowledge. Finally, Section 10.5 summarizes the chapter.

10.1 Reasoning

The OWL Web Ontology Language describes a language for ontologies, equipped with a formal semantics. These semantics enable inferences about ontologies and their individuals. Semantic reasoners are able to infer such logical consequences from a set of asserted facts or axioms.
"OWL 2 is an extension and revision of the OWL Web Ontology Language developed by the W3C Web Ontology Working Group and published in 2004. Like OWL 1, OWL 2 is designed to facilitate ontology development and sharing via the Web, with the ultimate goal of making Web content more accessible to machines" [53]. The Requirements Ontology uses OWL 2 to describe the formal semantics of requirements knowledge.
"The OWL language provides three increasingly expressive sublanguages [(OWL Lite, OWL

[1]"A high level Application Programming Interface (API) for working with OWL ontologies, closely aligned with the OWL 2 structural specification. The OWL API has widespread usage in a variety of tools and applications." [61]

DL and OWL Full)] designed for use by specific communities of implementers and users" [98]. However, the Requirements Ontology is classified to be in OWL DL and thus, provide computational completeness and ensure decidability (all computations will finish in finite time) of reasoning systems.

Currently, there are more than 20 reasoners available, differing in the reasoning algorithms, the expressivity supported, rule support, reasoning speed and much more. However, as explained in the following sections, the reasoners used for OntoReq must be capable to do reasoning with OWL DL and OWL 2 and must support the Closed World Assumption and NBox Reasoning. The TrOWL reasoner from the University of Aberdeen [3] complies to these conditions and was chosen for OntoReq.

10.2 Completeness Validation

As already described in Sections 8.5 and 9, the Requirements Ontology as Meta Model (TBox) is the fundament for our approach. It provides the necessary structure to specify and organize requirements artefacts and to interrelate them.
When checking for the existence of specific information captured in the Requirements Ontology, we are facing one major problem. Information that is not explicitly existing in an ontology is only assumed as "unknown" and thus, true, rather than "not existing" and false. This is due to the open world assumption (OWA) where a deductive reasoner will not infer that the statement is false [139]. Hence, in order to identify missing information we have to use the closed world assumption (CWA) which holds that any statement that is not known to be true is false. But Requirements Engineering is a continuing process. During the whole software development process it is possible that new requirements are identified or existing ones have to be modified in any way. Thus, the requirement specification will always be more or less incomplete and inconsistent, depending on the progress of the project. So CWA alone neither seems to be a solution.
Summarized, we need OWA for the process of specifying requirements and CWA for accomplishing completeness, consistency and quality validations of the Requirements Ontology. Thus, we must switch between OWA and CWA. This way, we can for example specify a number of requirements (OWA), then check for their completeness (CWA), identify that we forgot to specify something (CWA), add some data (OWA) and check again for completeness (CWA). Since we are interested in identifying information that is not existent, we will use NBox reasoning (also called Local Closed World Reasoning) for this purpose as introduced in [114]. A NBox (Negation As Failure Box) is a set of classes and object properties, whose extensions are closed. If an individual is not inferred to be an instance of a closed class, then it is regarded as an instance of the negation of that closed class [3]. However, we will describe another way to retrieve missing information without the necessity to close the ontology or any concepts within. We will refer to this method as "bypassing CWA".
In the following, we describe these two implementation patterns to support reasoning for incomplete information and explain their advantages and disadvantages. Therefore, we use the following brief example knowledge (ABox) fragment of the Requirements Ontology (identical to the example in Section 8.5.1 and an extract of four completeness rules. We use FR as abbreviation for functional requirement.

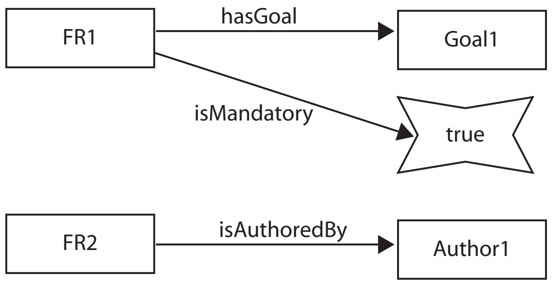


Figure 10.1: Example Ontology Knowledge (ABox) Fragment for Completeness Rules

Completeness Rules to be checked:

1. AT LEAST ONE Goal must be specified.
2. Every FR should have a priority.
3. Every FR must have an author.
4. Every FR must state its mandate.

We use the following prefix for each of the SPARQL queries listed below:

```
ro:<http://www.semanticweb.org/ontologies/2012/4/ro.owl#>
```

10.2.1 Bypassing CWA Pattern by Finding Empty Property Sets

We can bypass closed world reasoning by using the OWL API for accessing the ontology and a reasoner of choice (e.g., Jena, Pellet) to reason about the Requirements Ontology and to identify missing information. One possible solution for bypassing closed world reasoning is based on the following strategy exemplarily described by the identification of missing requirements information. Therefore, we retrieve all requirements and for each requirement we retrieve the relation of a specific owl:ObjectProperty. The quantity of these relations for one individual and one kind of OWLObjectProperty can be accessed via the size() method provided by the OWL API. It returns a set of individuals that are the values of this property. Thus, an empty set is the evidence for missing information. The following code snippet illustrates bypassing.

Listing 10.1: Bypassing Closed World Reasoning

```

String author = ‘‘author’’;
...
public static String identifyMissingInf(Set<OWLNamedIndividual>
    individualSet, OWLObjectProperty property, String rA){
 Set<String> resultList = new TreeSet<String>();
 if (individualSet.isEmpty() != true) {
  for (OWLNamedIndividual i : individualSet)
   if(reasoner.getObjectPropertyValues(i,property.getNamedProperty()).
       getFlattened().size() == 0)
   {
    resultList.add(i.getIRI().getFragment());
   }
12  }
13  String results = resultList.toString();
14  System.out.println("Error: The following requirements do not specify any "
        + rA + ":" + results);
15  return results;
16 }
17 ...
18 identifyMissingInf(requirementInd, hasGoal, goal);
19 identifyMissingInf(requirementInd, hasPriority, priority);
20 identifyMissingInf(requirementInd, isAuthoredBy, author);
```

The method `identifyMissingInf` returns the values of an owl:ObjectProperty `property` for each requirement in a set of owl:NamedIndividual (`individualSet`). If the returned set of individuals is empty, we know that there is no requirement in the ontology with the tested object property. These requirements are added to the `resultList` in line 10 which holds all requirements with the missing `property`. In our example, we use this method to test for the existence of goal, priority and author for each requirement. The String rA is used to individualize the error message in line 14. We get the following result:

```
Error: The following requirements do not specify any goal:[FR2]
Error: The following requirements do not specify any priority:[FR1, FR2]
Error: The following requirements do not specify any author:[FR1]
```

10.2.2 NBox Reasoning Pattern

Far more intuitive and convenient than bypassing closed world reasoning is the use of SPARQL queries with negation as failure and CWA. This enables to explicitly ask for information that is *not* existent in the Requirements Ontology. NBox reasoning allows for closing single classes and properties in the ontology. Thus, “if an individual is not inferred to be an instance of a closed class, then it is regarded as an instance of the negation of that closed class” [3]. NBox reasoning has been proposed by [114]. Therefore, we need the following prerequisites to identify missing requirements metadata: a reasoner enabling LCWR (e.g., TrOWL [3]), OWL API and SPARQL 1.1. The strategy for identifying missing information with CWA and negation is intuitive. We close all concepts in the ontology we want to reason about and use SPARQL to build queries that extract all requirements without a specific information.

Listing 10.2: Closing Axioms in the Requirements Ontology

```
1  File newFile = new File ("C:/.../new_ro.owl");
2  OWLOntology newOnto = manager.createOntology(IRI.create(newFile));
3  for(OWLLogicalAxiom axiom:localRO.getLogicalAxioms())
4   manager.addAxiom(newOnto, axiom);
5
6  OWLAnnotationProperty property = factory.getOWLAnnotationProperty(IRI.
       create("http://TrOWL.eu/REL#NBox"));
7  OWLAnnotation annotation = factory.getOWLAnnotation(property, factory.
       getOWLLiteral("close", "en"));
8  OWLAxiom axiom = factory.getOWLAnnotationAssertionAxiom(requirement.getIRI
       (), annotation);
9
10 manager.applyChange(new AddAxiom(localRO, axiom));
11 manager.saveOntology(newOnto, IRI.create(newFile.toURI()));
```

Lines 1 to 4 illustrate how to create a new ontology and copy all axioms from the Requirements Ontology. To specify which class/property names to be closed in an OWL ontology,

TrOWL [114] uses the specified annotation property in line 6 and 7. Since we want to retrieve all individuals of the class `requirement` with no author, we have to close this owl:Class (line 8). We add this new axiom to `newOnto` (localRO is the owl:OntologyObject of the original Requirements Ontology) and save the ontology in line 11).
The code snippet below finally illustrates the identification of missing requirements information by utilizing Negation as Failure provided by SPARQL1.1.

Listing 10.3: SPARQL Query for Missing Requirements Information

```
SELECT ?r  WHERE {?r a ro:Requirement . FILTER NOT EXISTS {?r ro:
    isAuthoredBy ?a}};
```

This SPARQL query returns all requirements without the owl:ObjectProperty *isAuthoredBy*.

Listing 10.4: Identification of Missing Information

```
public Set<String> performQuery(String query, String var, String artefact){
 Set<String> resultNames = new TreeSet<String>();

 Query q = QueryFactory.create(query);
 QueryExecution qe = QueryExecutionFactory.create(q, ontModel) ;
 ResultSet rs = qe.execSelect();
 while (rs.hasNext()) {
  QuerySolution solution = rs.next();
  String name = solution.getResource(var).getLocalName();
  resultList.add(name);
 }
 for (String item : resultList){
System.out.println(item + "has no" + artefact +".");
 }
 return resultList;
  System.out.println(The following requirements do not specify any " +
      artefact +": " + resultList);
}
...
performQuery(queryReqHasGoal, "g", "goal");
performQuery(queryReqHasPriority, "p", "priority");
performQuery(queryReqHasAuthor, "r", "author");
```

This code snippet demonstrates the execution of the SPARQL query and the output of incomplete requirements. The method `performQuery` executes a query `query` and binds all results for the variable `var`. The variable `artefact` is used for the output of the appropriate requirement or requirements artefact. The final result is identical to the one in listing 10.2.1.

10.3 Consistency Validation

Consistency checking and solution suggestions for inconsistent information is far more complicated than the identification of missing information. Contrary to the completeness validation, we need to consider various aspects for a consistency rule, e.g., a conjunction of conditions that leads to inconsistencies. The same applies for the derivation of solution suggestions that need to be inspected properly in order to avoid new inconsistencies. Consistency validation is based on the requirements configuration. This is due to the fact that usually not all specified requirements are finally chosen to be implemented. There might, for example, be optional requirements or alternative requirements that will not be considered

or treated much later in the project if resources are available. Choosing an (arbitrary) subset of requirements is an error-prone task that often introduces new inconsistency problems as a consequence of not considered or overseen requirements relationships. Thus, we need to validate consistency regarding the requirements configuration.
We generate the requirements configuration by adding all the requirements chosen by the requirements engineer to a set of requirements as shown in listing 10.5. Here, we demonstrate a simplification of this task. In OntoReq we use appropriate functionality to ask the requirements engineer to choose the relevant requirements and save them as a Set of individuals.

Listing 10.5: Generating the Requirements Configuration

```
OWLNamedIndividual fr1 = factory.getOWLNamedIndividual(IRI.create(prefix+"
    FR1"));
OWLNamedIndividual fr2 = factory.getOWLNamedIndividual(IRI.create(prefix+"
    FR2"));

Set<OWLNamedIndividual> chosenRequirements = new HashSet<OWLNamedIndividual
    >();
chosenRequirements.add(fr1);
chosenRequirements.add(fr2);
```

We use the following example knowledge (ABox) in the Requirements Ontology:

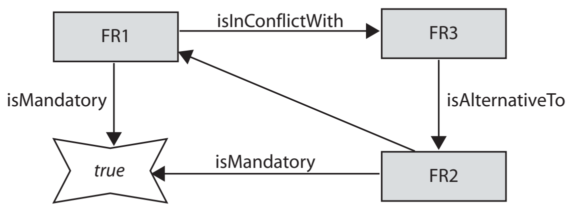


Figure 10.2: Example Ontology Knowledge (ABox) Fragment for Consistency Rules

Consistency Rules to be checked:

1. All mandatory requirements must be included in the requirements configuration.
2. Conflicting requirements should be avoided.

Listing 10.6 demonstrates a check whether all mandatory requirements in `requirementSet` have been included in the requirements configuration `reqConf` (lines 2-6). Those, which are not included are added to `mandatoryResultNames` (line 7).

Listing 10.6: Check for Inclusion of All Mandatory Requirements

```
1 public void isMandatoryInSubset(Set<OWLNamedIndividual> requirementSet, Set
      <OWLNamedIndividual> reqConf){
2     if (requirementSet.isEmpty() != true) {
3       for (OWLNamedIndividual i : individualSet1)
4         if(reasoner.getObjectPropertyValues(i, isMandatory.getNamedProperty
              ()).getFlattened().size() != 0)
5         {
6           if(reqConf.contains(i)== false){
```

```
            mandatoryResultNames.add(i.getIRI().getFragment());
          consistencyCounter++;
          }
    }}
    if (mandatoryResultNames.size() != 0)
     System.out.println("- The following requirements are mandatory and
         should be included in the requirements configuration: " +
         mandatoryResultNames);
}
```

The following listings illustrate how to identify conflicting requirements in the requirements configuration and how to calculate solution suggestions to handle possible conflicts.

Listing 10.7: Identification of Conflicting Requirements

```
1
2  conflRequirementsList = performQuery(queryConflReq, "r", "conflReq");
3
4  String chooseAltReqSugg = "- Choose one of the following alternative
       requirements instead of ";
5
6  public void isInConflictWithInSubset2(Set<OWLNamedIndividual>
       requirementSet, Set<OWLNamedIndividual> reqConf){
7   if (requirementSet.isEmpty() != true) {
8    for(OWLNamedIndividual a : requirementSet){
9     Set<OWLNamedIndividual> referencedIndividuals = new HashSet<
         OWLNamedIndividual>();
10    referencedIndividuals = reasoner.getObjectPropertyValues(a,
         isInConflictWith.getNamedProperty()).getFlattened();
11    for (OWLNamedIndividual b : referencedIndividuals){
12     if(reqConf.contains(b)== true){
13      System.out.println("- Error: " + a.getIRI().getFragment()+" and " + b.
           getIRI().getFragment() + " are specified as conflicting.");
14      System.out.println("  You have the following options: ");
15      System.out.println(" - Revise the requirements " +a.getIRI().
           getFragment()+ " or " + b.getIRI().getFragment() + " to solve the
           conflict");
16      System.out.println(" - Revise the requirements relationship (conflict)
            between " +a.getIRI().getFragment()+ " and " +        b.getIRI().
           getFragment());
17      if (isMandatory(a) == false && isCoexistent(a) == false)
18       System.out.println("  - You may choose to delete the optional
            requirement: " + a.getIRI().getFragment());
19       consistencyCounter++;
20     }
21     if (getAlternatives(a, reqConf, individualSet1).isEmpty()== false &&
          impliesReq(a) == false)
22      System.out.println(chooseAltReqSugg + a.getIRI().getFragment() + ": "
           + getAlternatives(a, reqConf, individualSet1));
23 }}}}
```

Listing 10.7 demonstrates the identification of requirements with a conflict relationship in the requirements configuration `reqConf`. In line 8 - 10 we reason for all conflicting requirements. In line 11 - 12 we check for each requirement with a conflict specified whether it is contained in the requirements specification `reqConf`. If this is the case, an error message is displayed with concrete information. Afterwards solution suggestions are shown that are based on the decision tree in Figure 8.7. Besides the always applicable solution suggestions in lines 13 - 16, we check in line 17 - 19 for each faulty requirement whether it is mandatory or part of an *implies* relationship. If not, we can suggest to exclude it from the requirements configuration and the appropriate suggestion message is displayed (line 16). Additionally, we check in line 21 - 22 for alternative requirements that might be replaced with the faulty ones. If one or more alternatives are identified, then each of them must be free from conflicts, shall not exclude the requirement to be replaced with and is not allowed to be negative contribution to a goal. This is assured by the methode `getAlternative` in Listing 10.8 which checks for these conditions.

Listing 10.8: Computation of Alternative Requirements

```
1
2  public Set<String> getAlternatives(OWLNamedIndividual org, Set<
       OWLNamedIndividual> reqConf, Set<OWLNamedIndividual> requirementsSet){
3   Set<OWLNamedIndividual> alternatives = new HashSet<OWLNamedIndividual>();
4   alternatives = reasoner.getObjectPropertyValues(org, isAlternativeTo.
        getNamedProperty()).getFlattened();
5   Set<String> realAlt = new HashSet<String>();
6
7   for (OWLNamedIndividual alt : alternatives){
8    if (excludesReq(alt) == false && isExcludedInRC(alt, reqConf) == false &&
          isNegGoalContr(alt, requirementsSet) == false && isInConflictInRC(
         alt, reqConf) == false){
9     realAlt.add(alt.getIRI().getFragment());
10   }
11  }
12  return realAlt;
13 }
```

Line 4 in Listing 10.8 computes all alternative requirements for a requirement `org`. These alternatives are not restricted to those from the requirements configuration, since it might be better to add another requirement to the requirements configuration instead of tolerating inconsistencies. As can be seen in line 8, we check whether an alternative requirement does not exclude any other requirement, is not excluded by any other requirement, is no negative contribution to a goal and is not in conflict with any other requirement. Only if these conditions are completely satisfied, a requirement is added to a list of possible alternative requirements `realAlt` (line 9).
The method below in Listing 10.9 shows an example on how to check whether an alternative requirement is excluded by any other requirement in the requirements configuration.

Listing 10.9: Check Excluded Requirements in Requirements Configuration

```
public boolean isExcludedInRC(OWLNamedIndividual alt, Set<
    OWLNamedIndividual> reqConf) {
 boolean isExcluded = false;
 Set<OWLNamedIndividual> excluded = new HashSet<OWLNamedIndividual>();
 Set<OWLNamedIndividual> excludedInRC = new HashSet<OWLNamedIndividual>();
  for (OWLNamedIndividual r : reqConf){
   if (reasoner.getObjectPropertyValues(r, isExcludedBy.getNamedProperty())
       .getFlattened().size() != 0)
        excludedInRC.add(r);
  }
  if (excludedInRC.contains(alt)== false)
   isExcluded = false;
  else isExcluded = true;
  return isExcluded;
}
```

10.4 Quality Improvement

Quality Improvement approaches fall into two categories: quality validation and quality measurements as described in Section 8.5. Implementation patterns for quality validation are a combination of completeness and consistency validation patterns. We either check for additional information that must be provided for a high quality specification or define particular quality conditions, such as avoiding negative goal contributions. We will therefore only give a brief example for the demonstration of quality validation and describe implementation patterns for quality measurements in more detail.

10.4.1 Quality Validation

Similar to the implementation patterns for consistency, we validate quality regarding a requirements configuration. The example below demonstrates a query for requirements in the requirements configuration with a negative goal contribution and the calculation of concrete solution suggestions based on the decision tree and decision process described in Section 8.5.

Example We use the following extract of requirements knowledge illustrated in Figure 10.1. We use FR as abbreviation for functional requirement.

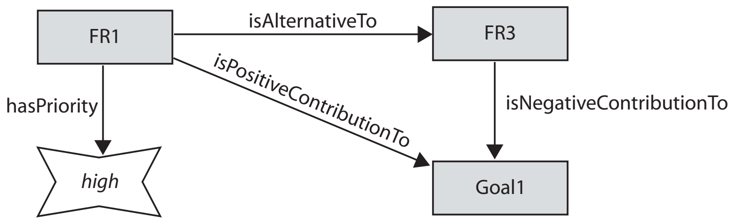


Figure 10.3: Example Requirements Ontology (ABox) Fragment for Quality Rules

Chosen Requirements Configuration: FR3
Quality Rule to be checked:

1. There must be no requirement that is a negative contribution on a goal to be achieved.

Expected Result:

- FR3 is a negative contribution to: Goal1. Please consider any of the following options:
  - Exclude the optional requirement FR3 from the requirements configuration
  - Choose one of the alternative requirements instead of FR3: FR2
  - Revise the goal contribution relationship.

The following code snippet shows how we identify requirements with a negative contribution to a goal and provide solution suggestions. In lines 7 - 12 we check for each requirement with a negative goal contribution whether it is contained in the requirements configuration. If this is the case, an error message is displayed with concrete information and the requirement is added to the list `negReqInConf` for further use. Afterwards solution suggestions are shown (lines 12 - 21) that are based on the decision tree in Figure 8.7 in Section 8.5.

Additionally, we check in lines 20 - 21 for alternative requirements that might be replaced with the faulty ones. These alternative requirements are identified with the method `getAlternative` described in Listing 10.8.

Listing 10.10: Identification of and Solution Suggestions for Negative Goal Contribution

```
1  String chooseAltReqSugg = "- Choose one of the following alternative
       requirements instead of ";
2  String reviseGoalContrSugg = "- Revise the above goal contribution.";
3  String modifyReqSugg = "- Modify the requirement to ";
4
5  public boolean isNegContr(Set<OWLNamedIndividual> requirementSet, Set<
       OWLNamedIndividual> reqConf){
6   if (requirementSet.isEmpty() != true) {
7    for(OWLNamedIndividual a : requirementSet){
8     Set<OWLNamedIndividual> referencedGoals = new HashSet<OWLNamedIndividual
          >();
9     referencedGoals = reasoner.getObjectPropertyValues(a,
          isNegativeContributionToGoal.getNamedProperty()).getFlattened();
10    for (OWLNamedIndividual b : referencedGoals){
11     if(reqConf.contains(b)== true){
12      System.out.println("- Warning: " + a.getIRI().getFragment()+" is a
            negative contribution to goal " + b.getIRI().getFragment() + ".\n
             You have the following options:");
13      System.out.println(reviseGoalContrSugg);
14      System.out.println(modifyReqSugg + "meet the goal");
15      hasNegContr = true;
16      referencedNegGoalRequirements.add(a.getIRI().getFragment());
17     }
18     if (isMandatory(a) == false && isCoexistent(a) == false)
19      System.out.println("    - Delete this optional requirement from your
            configuration");
20     if (getAlternatives(a, reqConf, individualSet1).isEmpty()== false){
21      System.out.println(chooseAltReqSugg + a + ": " + getAlternatives(a,
            reqConf, requirementSet));
22     }
23    }
24   }
25  }
26     return hasNegContr;
27 }
```

10.4.2 Implementation Patterns for Quality Measurements

As we base our quality measures on quality rules and the completeness and consistency of requirements knowledge, we demonstrate in the following how to apply NBox Reasoning for quality calculations. Due to space limitations we will only demonstrate these measurements for one of these metrics, namely internal correctness, as described in detail in Section 8.5.2. The remaining four metrics are implemented in a similar way.

Internal Correctness Measurement Pattern

In Section 10.4.1, we defined a metric to measure the validation status of individual requirements to make assumptions about the internal correctness of a requirements configuration. Listing 10.11 demonstrates this computation of the formula

$$Q_2 = \frac{r_c}{r_n}$$

.

Listing 10.11: SPARQL Queries for Measuring Internal Correctness

```
1  public static int computeCorrectness(){
2
3   Query qV = QueryFactory.create(getQueryAccessor().getValidComputation());
4   QueryExecution qeV = QueryExecutionFactory.create(qV, ontModel) ;
5   ResultSet rsV = qeV.execSelect();
6   ResultSetRewindable rsrwV = ResultSetFactory.copyResults(rsV);
7   double numberOfValid = rsrwV.size();
8
9   Query qI = QueryFactory.create(getQueryAccessor().getInvalidComputation())
       ;
10  QueryExecution qeI = QueryExecutionFactory.create(qI, ontModel) ;
11  ResultSet rsI = qeI.execSelect();
12  ResultSetRewindable rsrwI = ResultSetFactory.copyResults(rsI);
13  double numberOfInvalid = rsrwI.size();
14
15  correctness =((numberOfValid + numberOfInvalid) / getNumberOfRequirements
       ());
16  int result = new Double(Math.round(correctness*100)).intValue();
17  return result;
18 }
19
20 System.out.println("Correctness: " + QueryHandler.computeCorrectness() + "
      \%");
```

Lines 4 to 5 execute the SPARQL query `getValidComputation`. The results (all valid requirements) are returned as a ResultSet. The method `getValidComputation()` only returns the SPARQL query as String. Since we only need to know the number, we use the `size()` method of ResultSetRewindable to count all returned individuals (lines 6 and 7). The percentage of completeness is measured in lines 15 and 16. We divide the number of VALID requirements `numberOfValid` by the number of all requirements in the SRS (`getNumberOfRequirements()`) and multiply it with 100.

Cumulative Quality of SRS

Finally, the code below demonstrates our approach for measuring the cumulative quality. Therefore, we use the single quality metrics and multiply them with the appropriate weight (line 3).

Listing 10.12: Measuring the Cumulative Quality of the SRS

```
1  public static double computeQuality(){
2   double quality =
3   (((1.0*correctness)+(0.7*verifiability)+(1*consistency)+(0.6*traceability
       )+(0.5*uncriticality)+(1*completeness)) / (1.0+0.7+1+0.6+0.5+1));
4   int result = new Double(Math.round(quality*100)).intValue();
5   System.out.println("The quality of the SRS measures: " + result + "\%");
6   return result; }
```

It is more convenient to specify the weights as constants, this way, they can be changed much easier if necessary. For reasons of readability, we put the value of each weight directly in the function above.

10.5 Summary

This chapter describes various implementation patterns that allow for the validation of completeness, consistency and quality of the requirements knowledge. This includes code snippets that demonstrate how to identify missing information in an ontology ABox by

applying Local Closed World Reasoning with Negation as failure (NBox Reasoning) as well as a workaround for closed world reasoning. We furthermore present implementation patterns that show how to proof the satisfaction of the consistency rules defined in Section 8.5 and for the realisation of the quality measurements.
OntoReq is implemented in Java and uses the OWL API to access the Requirements Ontology. Reasoning is facilitated by the Jena Reasoner and TrOWL for NBox Reasoning. We use SPARQL to query for the inferred requirements knowledge.

11 Architecture

Our approach consists of two main contributions: the Requirements Ontology and the Java application OntoReq. The Requirements Ontology forms the knowledge base that is used by OntoReq to execute the requirements validation services and generate the output. In the following two sections (Section 11.1 and 11.2), we explain the components of the demonstrator OntoReq and describe the data flow between them. We conclude with a summary in Section 11.4.

11.1 OntoReq Components

OntoReq comprises three main components depicted in Figure 11.1: the graphical user interface (GUI), validation component and output components. The GUI allows for editing the ABox of the Requirements Ontology in order to support the specification of requirements knowledge, such as adding, deleting or modifying requirements artefacts and their interrelations. Therefore, the GUI invokes the Reasoner component that calls the Requirements Ontology and provides means to realise a manipulation of the ABox. Additionally, the GUI calls the validation component to execute the validation of completeness, consistency and quality of the requirements knowledge. For this purpose, the TrOWL reasoner is invoked to infer the knowledge model from the Requirements Ontology. The validation component finally invokes the (command line) output to display the validation results and solution suggestions.

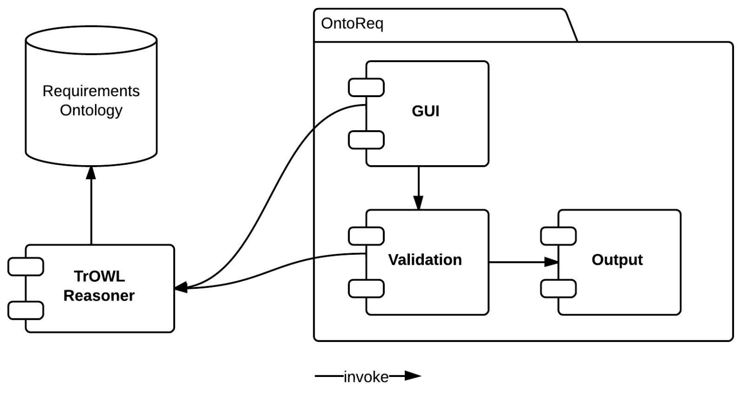


Figure 11.1: Component Diagram

11.2 OntoReq Data Flow

Our approach defines two main tasks that are denoted as process in the UML data flow diagram in Figure 11.2: specification of requirements knowledge (manual input) and validation. In order to capture the requirements knowledge, the data input of the user via the GUI is stored in the ABox of the Requirements Ontology. The validation process retrieves and computes the data from the Requirements Ontology as knowledge base, including TBox and ABox. The results of the validation are passed to the command line output (display).

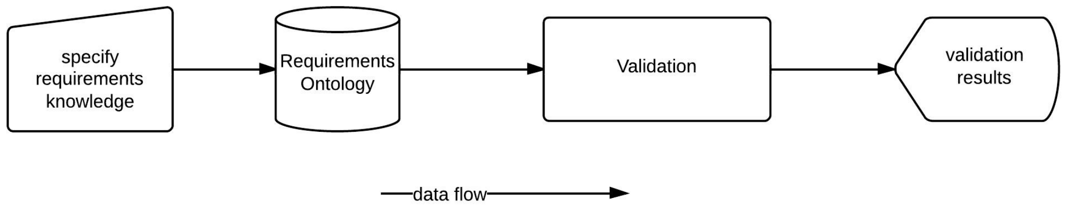


Figure 11.2: Data Flow Diagram

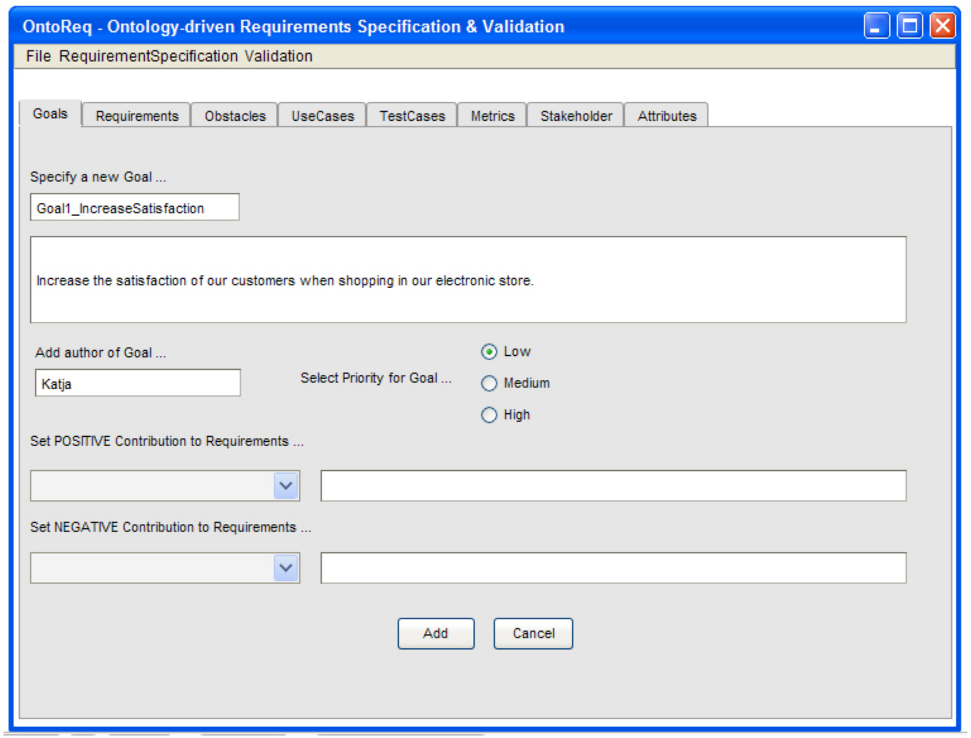
11.3 Graphical User Interface

Ontology editors (e.g. Protégé, NeOn, TopBraid Composer) allow for modelling and manipulating ontologies. However, for Requirements Engineering it is of course not feasible to force the requirements engineer to use any of these tools to specify requirements knowledge. The reasons therefore are obvious: ontology editors are not intended to be used for Requirements Engineering and thus, do not provide appropriate means to specify requirements knowledge. Additionally, the requirements engineer would need to learn quite a good amount about ontology modelling before he could start requirements specification. And finally, editing the Requirements Ontology directly in an ontology editor would bear too many risks to change the metamodel of the ontology itself or to introduce various errors related to ontology modelling. Thus, we decided to keep the Requirements Ontology ABox and TBox in the background, invisible to the user. This way, the requirements engineer is only aware of the data stored within and the manipulation of concrete requirements knowledge. The realisation of any data manipulation and the concrete ontology manipulation remains invisible. However, in order to specify requirements knowledge, we must provide a Graphical User Interface that enables typical tasks, such as adding, deleting and modifying requirements, validating the requirements knowledge, etc. OntoReq provides a simple prototype for such a GUI.

The following sections introduce the GUI prototype for OntoReq. The chapter is structured according to the main tasks: Section 11.3.1 shows how requirements knowledge may be added and Section 11.3.2 demonstrates the validation of requirements knowledge. Besides this brief illustration of the GUI prototype, Chapter 13 demonstrates how to use OntoReq with an exemplary scenario, also demonstrating the GUI with specific data.

11.3.1 Adding Requirements Knowledge

Figures 11.3 and 11.4 exemplify how requirements knowledge can be specified. Figure 11.3 demonstrates the specification of goals and Figure 11.4 the specification of requirements. The different tabs allow for switching between the requirements artefacts. Appropriate fields are presented to specify all related metadata (e.g. priority, author. Goals may get a priority and can be linked to requirements to specify a positive or negative contribution. Requirements may specify their level of priority, cost and risk. They can be interrelated as shown in the lower part of Figure 11.4. Here, the right side shows the requirements description of the requirement to be related with in grey.
The completeness of the requirements knowledge is increased by displaying these fields and check boxes directly and thus, preventing the user to forget the specification of relevant (meta)data.


OntoReq - Ontology-driven Requirements Specification & Validation
File RequirementSpecification Validation
Goals
Requirements
Obstacles
UseCases
TestCases
Metrics
Stakeholder
Attributes
Add
Cancel

Specify a new Goal ...
Goal1_IncreaseSatisfaction

Increase the satisfaction of our customers when shopping in our electronic store.

Add author of Goal ...
Katja

Select Priority for Goal ... [✓ Low] Medium High

Set POSITIVE Contribution to Requirements ...

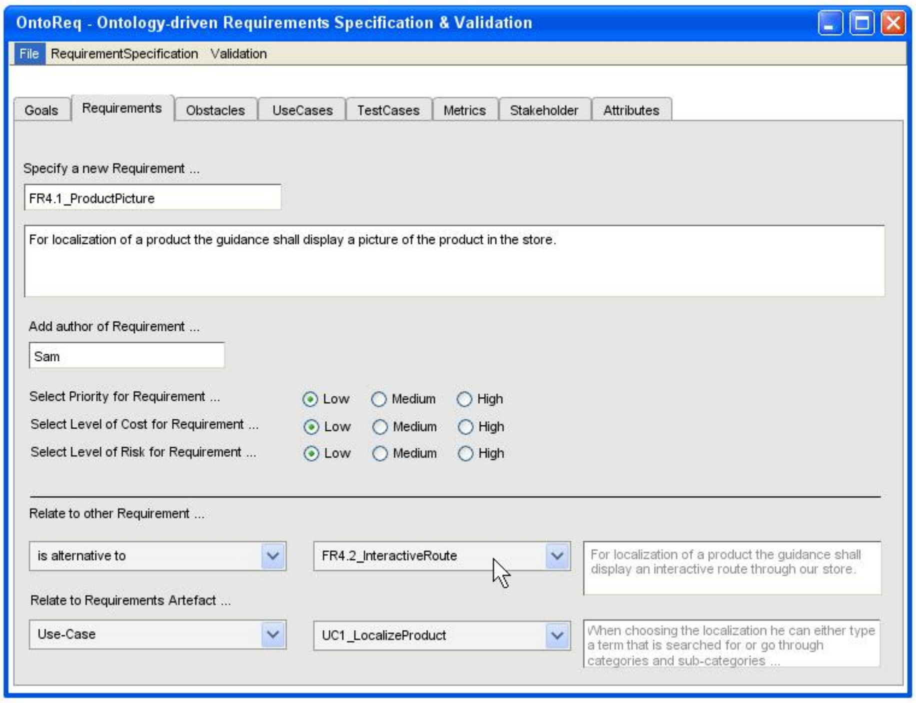
Set NEGATIVE Contribution to Requirements ...

Figure 11.3: Adding goals and associated metadata

11.3.2 Validation

Figures 11.5, 11.6 and 13.7 illustrate the three kinds of validation. In each case, the upper part of the window shows statistics such as the number of certain requirements artefacts and the lower part displays the faults that have been identified by OntoReq. In contrast to the completeness and quality validation, the window for consistency validation (Figure 11.6 provides checkboxes to select requirements for the requirements configuration.

Figure 11.8 demonstrates the quality measurement of the requirements knowledge. The lower part of the window displays the different quality criteria and their values.



Specify a new Requirement ...

FR4.1_ProductPicture

For localization of a product the guidance shall display a picture of the product in the store.

Add author of Requirement ...

Sam

Select Priority for Requirement ... [✓ Low] Medium High

Select Level of Cost for Requirement ... [✓ Low] Medium High

Select Level of Risk for Requirement ... [✓ Low] Medium High

Relate to other Requirement ...

is alternative to | FR4.2_InteractiveRoute | For localization of a product the guidance shall display an interactive route through our store.

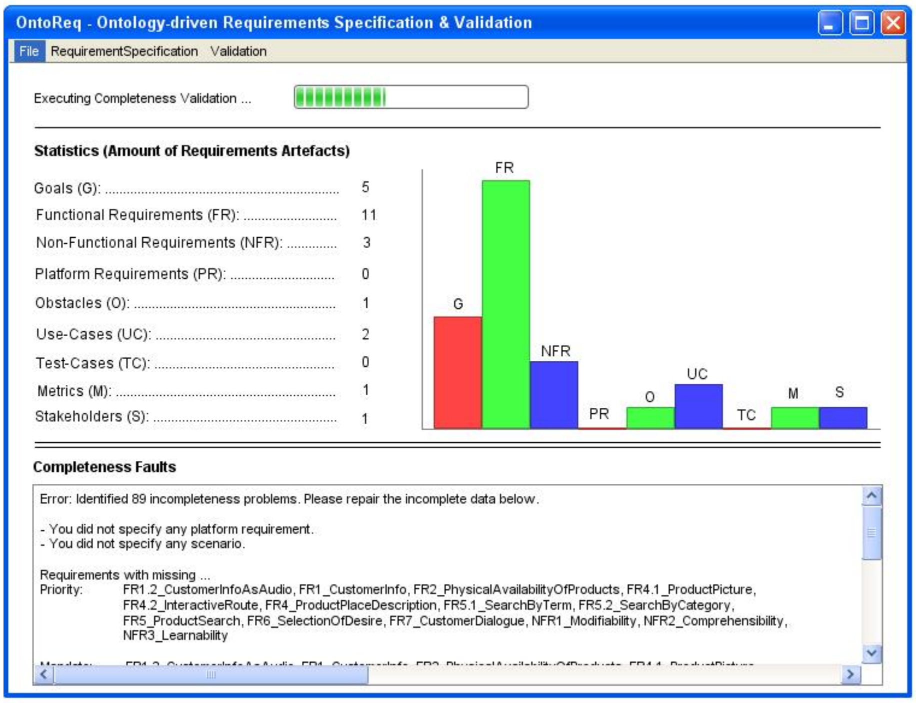
Relate to Requirements Artefact ...

Use-Case | UC1_LocalizeProduct | When choosing the localization he can either type a term that is searched for or go through categories and sub-categories ...

Figure 11.4: Adding requirements and associated metadata

11.4 Summary

The demonstration of our approach consists of the Requirements Ontology as knowledge base and OntoReq, a Java Application that works on this knowledge base. OntoReq consists of three components: GUI, validation component, measurement component and output. The GUI supports the input and manipulation of requirements knowledge and invokes all other components to either allow for the specification of requirements knowledge or the validation and measurement of it. The TrOWL reasoner component is used to generate an inferred knowledge model from the Requirements Ontology that is validated by the validation component. Finally, the validation results are passed to the output component and are displayed together with the solution suggestions.
The GUI is a prototypic demonstration that shows how requirements engineers may specify and modify requirements knowledge captured in the Requirements Ontology without technological background of ontologies or ontology modelling tools. Furthermore, the GUI facilitates completeness, consistency and quality of the requirements knowledge during its specification by providing appropriate input fields for relevant metadata.


OntoReq - Ontology-driven Requirements Specification & Validation
File
RequirementSpecification
Validation
G
FR
NFR
PR
O
UC
TC
M
S

Executing Completeness Validation ...

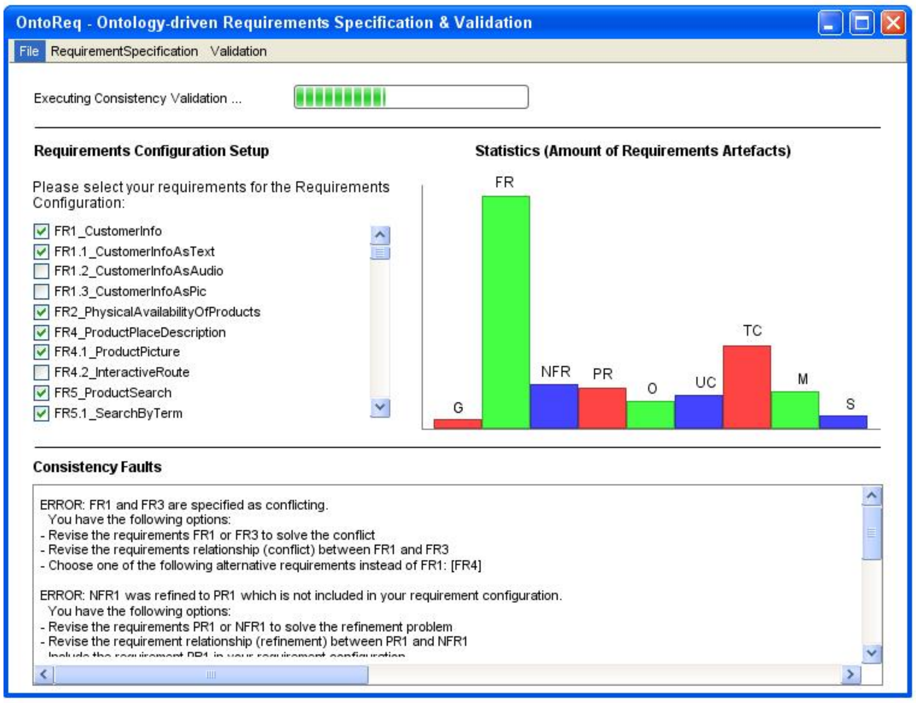
Statistics (Amount of Requirements Artefacts)

Goals (G):	5
Functional Requirements (FR):	11
Non-Functional Requirements (NFR):	3
Platform Requirements (PR):	0
Obstacles (O):	1
Use-Cases (UC):	2
Test-Cases (TC):	0
Metrics (M):	1
Stakeholders (S):	1

Completeness Faults

```
Error: Identified 89 incompleteness problems. Please repair the incomplete data below.

- You did not specify any platform requirement.
- You did not specify any scenario.

Requirements with missing ...
Priority:     FR1.2_CustomerInfoAsAudio, FR1_CustomerInfo, FR2_PhysicalAvailabilityOfProducts, FR4.1_ProductPicture,
              FR4.2_InteractiveRoute, FR4_ProductPlaceDescription, FR5.1_SearchByTerm, FR5.2_SearchByCategory,
              FR5_ProductSearch, FR6_SelectionOfDesire, FR7_CustomerDialogue, NFR1_Modifiability, NFR2_Comprehensibility,
              NFR3_Learnability
```

Figure 11.5: Completeness validation of requirements knowledge


OntoReq - Ontology-driven Requirements Specification & Validation
File
RequirementSpecification
Validation
FR
G
NFR
PR
O
UC
TC
M
S

Executing Consistency Validation ...

Requirements Configuration Setup

Please select your requirements for the Requirements Configuration:

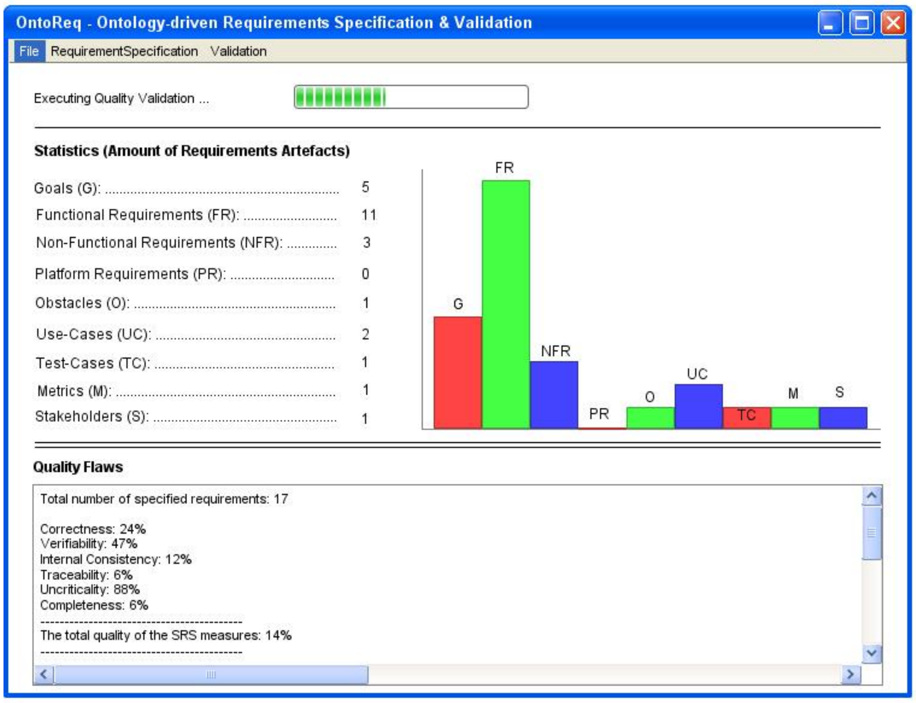
- [✓] FR1_CustomerInfo
- [✓] FR1.1_CustomerInfoAsText
- [ ] FR1.2_CustomerInfoAsAudio
- [ ] FR1.3_CustomerInfoAsPic
- [✓] FR2_PhysicalAvailabilityOfProducts
- [✓] FR4_ProductPlaceDescription
- [✓] FR4.1_ProductPicture
- [ ] FR4.2_InteractiveRoute
- [✓] FR5_ProductSearch
- [✓] FR5.1_SearchByTerm

Statistics (Amount of Requirements Artefacts)

Consistency Faults

```
ERROR: FR1 and FR3 are specified as conflicting.
  You have the following options:
- Revise the requirements FR1 or FR3 to solve the conflict
- Revise the requirements relationship (conflict) between FR1 and FR3
- Choose one of the following alternative requirements instead of FR1: [FR4]

ERROR: NFR1 was refined to PR1 which is not included in your requirement configuration.
  You have the following options:
- Revise the requirements PR1 or NFR1 to solve the refinement problem
- Revise the requirement relationship (refinement) between PR1 and NFR1
```

Figure 11.6: Consistency validation of requirements knowledge



OntoReq - Ontology-driven Requirements Specification & Validation
File
RequirementSpecification
Validation
G
FR
NFR
PR
O
UC
TC
M
S

Executing Quality Validation ...

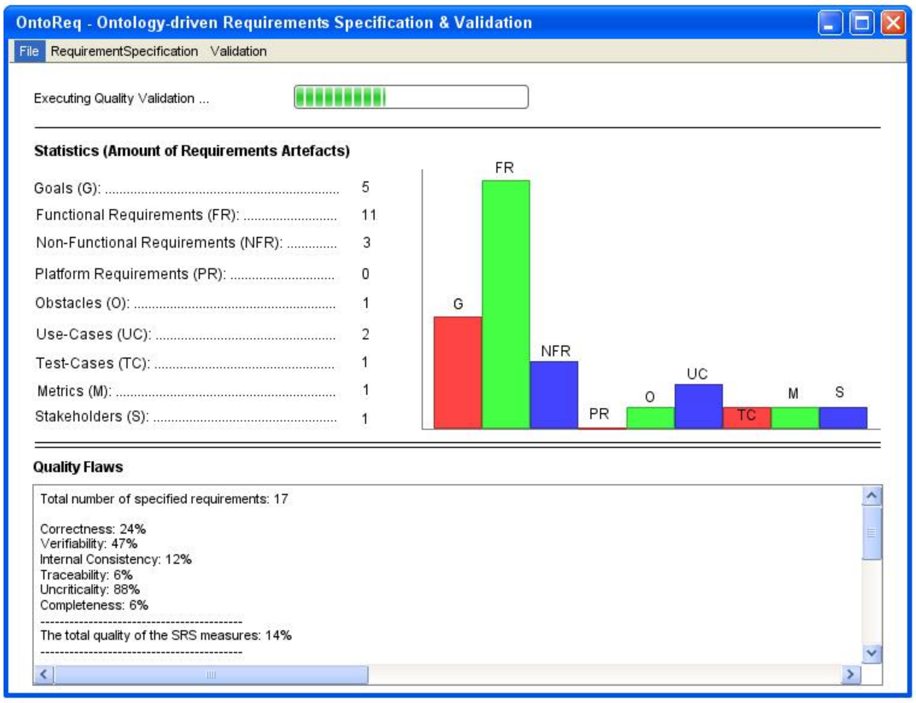
Statistics (Amount of Requirements Artefacts)

Goals (G):	5
Functional Requirements (FR):	11
Non-Functional Requirements (NFR):	3
Platform Requirements (PR):	0
Obstacles (O):	1
Use-Cases (UC):	2
Test-Cases (TC):	1
Metrics (M):	1
Stakeholders (S):	1

Quality Flaws

```
Total number of specified requirements: 17

Correctness: 24%
Verifiability: 47%
Internal Consistency: 12%
Traceability: 6%
Uncriticality: 88%
Completeness: 6%
-------------------------------------------
The total quality of the SRS measures: 14%
-------------------------------------------
```

Figure 11.7: Quality validation of requirements knowledge



OntoReq - Ontology-driven Requirements Specification & Validation

Executing Quality Validation ...

Statistics (Amount of Requirements Artefacts)

Goals (G):	5
Functional Requirements (FR):	11
Non-Functional Requirements (NFR):	3
Platform Requirements (PR):	0
Obstacles (O):	1
Use-Cases (UC):	2
Test-Cases (TC):	1
Metrics (M):	1
Stakeholders (S):	1

Quality Flaws

```
Total number of specified requirements: 17

Correctness: 24%
Verifiability: 47%
Internal Consistency: 12%
Traceability: 6%
Uncriticality: 88%
Completeness: 6%
-------------------------------------------
The total quality of the SRS measures: 14%
-------------------------------------------
```

Figure 11.8: Quality measurement of requirements knowledge

12 General Guidelines for Improving Completeness, Consistency and Quality of Knowledge in Domain Ontologies

In this thesis, we provide an approach to improve and validate the completeness, consistency and quality of requirements knowledge by applying ontological techniques. However, increasing the above mentioned factors is not only crucial to Requirements Engineering but to any kind of knowledge repository. Therefore, we generalize our approach and describe the main steps to validate these criteria for any knowledge domain (e.g., medicine, architecture) whose data is captured in an ontology. We use domain-independent guidelines to sketch fundamental tasks and briefly explain the technology to be applied. The code snippets in Chapter 10 give a more detailed view on how to realize some of these guidelines.
These guidelines and their associated tasks require at least basic knowledge in ontology modelling to follow the guidelines and explanations. We illustrate the general approach with an exemplar in the domain of medical drugs where we want to store information about drugs, their active pharmaceutical components, treatments and various relationships among these concepts.

12.1 Building the Ontology Knowledge Repository

Usually, there are several correct (and many incorrect) ways to build an ontology. The one and important question is, how it shall be used. Modelling an ontology with the only purpose to serve as some kind of taxonomy or glossary differs a lot from modelling an ontology that will be used for reasoning later. However, for enabling validation techniques, we need to invoke a reasoner and, thus, model the ontology in a way that reasoning leads to meaningful and usable results. Therefore, we have to construct a TBox model that serves as Metamodel for a considered domain, that provides sufficient pre-defined knowledge about this domain and that defines the interrelations between the domain concepts. This TBox will be instantiated later, so that the ABox captures concrete information.

Ontology modelling is an iterative process that starts with a rough first abstract model that is revised and refined until it meets the previously defined domain and scope. The following guidelines sketch the way to build such a domain-specific knowledge Metamodel that shall finally enable (at least) reasoning for completeness, consistency and quality of the knowledge specified within.

Preliminary Considerations

The first step before modelling an ontology is to completely understand the domain that is to be described, otherwise the derived model will be incorrect right on from the beginning and thus, lead to undesired reasoning and validation results.

Guideline 54: *Identify and clarify the main concepts of the domain and the knowledge it describes.*

This guideline may be facilitated by existing documents, domain descriptions, glossaries, etc. It might be useful to list all terms about the domain we would like either to make statements about or to explain to a user. Additionally, we have to clarify what properties those terms have and what we would like to say about those terms [105]. For example, important drug-related terms will include `drug`, `treatment`, `components`, `brand`, a `drugs name`, etc; subtypes of drug such as `painkiller` and so on. Possible statements about the domain knowledge in our example are:

- Drugs consist of several active pharmaceutical components and define a number of diseases where they can be used against.
- Drugs or any of their components may be contraindicated to others.
- There may be several possible drugs that can be used against one and the same disease.
- Drugs with the same components may be available from different brands and with different names.
- Active pharmaceutical components may be available in different dosage forms (e.g., tablets, syrup).
- Each drug has a specific intake description and dosage.
- A drug specifies who may take it (children, teens, adults, etc.)

Guideline 55: *Define a domain and scope for the ontology.*

According to [105], answering the following questions helps to define a domain and scope for an ontology:

1. What is the domain that the ontology will cover?
2. For what are we going to use the ontology?
3. For what types of questions the information in the ontology should provide answers?
4. Who will use and maintain the ontology?

These question are related to guideline (1). The identified domain and its knowledge must now be screened regarding the information we want to capture in the ontology. Therefore, we need to figure out the goal of its usage which we can best identify if we answer question (3) above as detailed as possible. One way to do this, is to sketch a list of questions that a knowledge base based on the ontology should be able to answer. Grüninger and Fox [51] refer to these questions as *competency questions*. At the end of the ontology modelling process they can also be used to test whether the ontology contains enough information to answer these types of questions. Competency questions are just a sketch and do not need to be exhaustive [105]. In the medical drugs domain, the following are the possible competency questions:

- Which drug should I take if I want to treat a headache?
- Is Ibuprofen a painkiller or antidepressant?
- Does Ibuprofen go well with Clarithromycin (antibiotic drug)?
- Does Ceterizin treat allergies?
- Which characteristics of a drug affects its appropriateness for a disease?
- Which drugs treat the Alzheimer's disease?
- Can I use Penicillin if I suffer from allergies?
- Which side effects has Ibuprofen?

These competency questions will also show whether it is reasonable to use an ontology or whether another design solution (e.g., a simple database) might be more appropriate. According to [105], the application of an ontology is advisable if we want to:

- make domain assumptions explicit
- separate domain knowledge from operational knowledge
- enable reuse of domain knowledge
- analyse domain knowledge in order to compute implicit knowledge by invoking reasoning
- develop/ use additional software that uses the ontology as basis for problem-solving methods, domain-independent applications, software agents, etc.

The more precisely we answer these questions, the easier it will be to create the ontology so that it serves our purpose. Deciding what the ontology is going to be used for, and how detailed or general the ontology is going to be, will guide many of the following modelling decisions [105]. Due to reasoning technology, we can compute explicit knowledge and derive implicit knowledge that has not been specified. Thus, we may gain new information or validate operational knowledge regarding the defined domain assumptions. Obviously, an ontology itself can hardly be used as standalone program. We need additional software or applications to *make use of the ontology as basis for problem-solving strategies, validations, computations* and so on. This ontology will then be used to build a knowledge model that can be computed for several aspects by any additional software.

Building the Ontology's Foundation

The following guidelines facilitate the modelling of the ontology's foundation. Therefore, we need to concretise several aspects regarding the validation purposes and the ontology hierarchy.

Guideline 56: *Define completeness, consistency and quality rules and approaches for solution suggestions.*

Since we already know that we want to use the ontology to accomplish completeness, consistency and quality validations, we need to specify what we understand about these aspects related to a specific domain knowledge. Thus, we need to define when the knowledge is complete, how consistent knowledge can be characterized and what we understand about quality regarding the knowledge within the ontology. In an ontology about medical drugs, we may for example define a completeness rule that requires to `cover all possible treatments for each drug and all its active components`. One consistency rule may state that `a drug is not allowed to be combined with another drug that is contraindication regarding their active components`. Furthermore, we have to define several options a solution should provide. For the consistency rule in this example we could state that we want to `identify an alternative drug that is applicable for the same disease but no contraindication`.
The measurement of the quality of information provided by an ontology (quality metric) is also domain-specific and needs to be considered thoroughly.

Guideline 57: *Identify and define how to measure the quality (quality metrics) as described in section 8.5.3. Define weights for the quality metrics.*

Just as specifying the quality rules, one has to decide what exactly influences the quality of information contained in the ontology besides the completeness, consistency and quality rules. In our example, we might define a quality metric "`correctness`" similar to the RE domain, that measures all drugs regarding the execution of a user-validation (e.g., verifying that components, dosage, etc. are correctly specified).

12.1.1 Modelling the Ontology

The accomplishment of the previous guidelines builds the basis for the following tasks in setting up the ontology model.

Guideline 58: *Model the domain concepts and class hierarchy.*

Concepts in the ontology should be close to objects (physical or logical) and relationships in the domain of interest. These are most likely to be nouns (objects) or verbs (relationships) in sentences that describe the domain [105]. The completion of the previous guidelines enables the identification of necessary concepts not only regarding the domain knowledge but also for the validation purposes. If we analyse the previously defined domain knowledge in our example, we can derive the following ontology concepts:

- `Drugs` consist of several `active pharmaceutical components` and define a number of `diseases` where they can be used against.
- Drugs with the same components may be available from different `brands` and with different `names`.
- Active pharmaceutical components may be available in different `dosage forms` (e.g., `tablets`, `syrup`).
- Each drug has a specific `intake description` and `dosage`.
- Drugs may have several `side effects` .

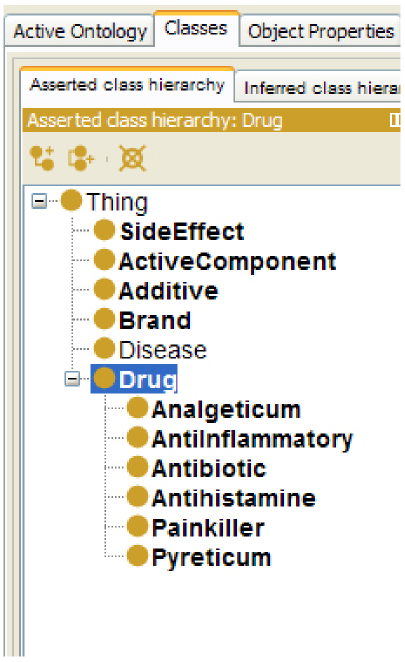
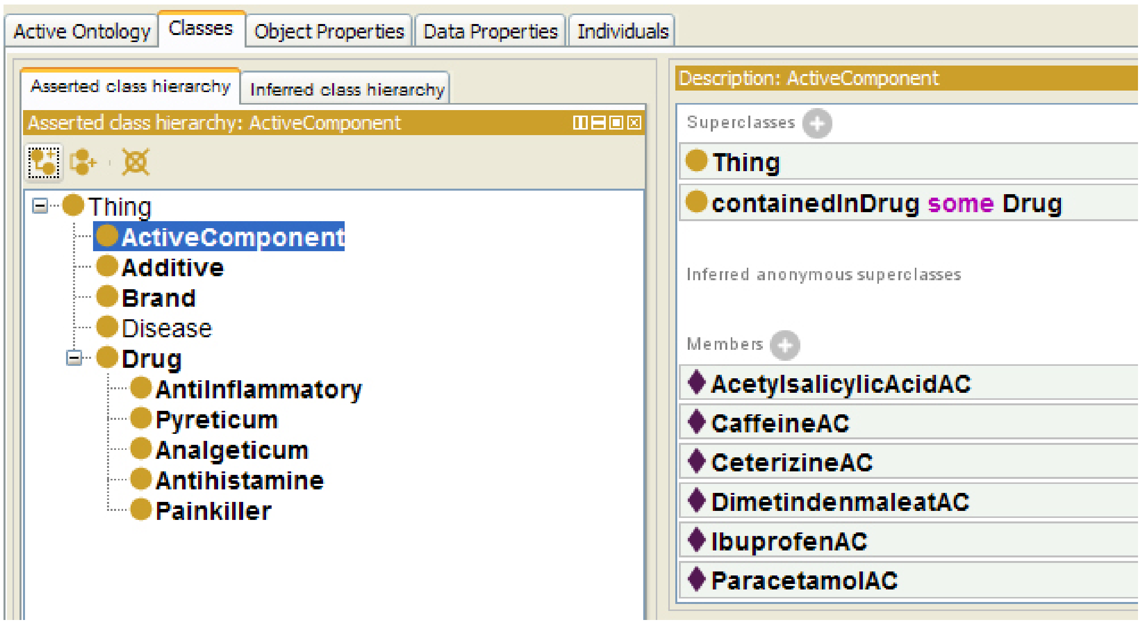


Figure 12.1: Drugs Ontology - Class Hierarchy

Figures 12.1 shows a possible class hierarchy for the concepts above. There are several possible approaches in developing a class hierarchy. However, for reasons of space we will not consider these details but refer to Chapter 8.5 that illustrates one possible approach. Additionally, [128] provides more information on ontology modelling approaches.

Guideline 59: *Model the relationships among the domain knowledge.*

The most important part of the ontology are object and data properties since the classes alone do not provide enough information to answer the competency questions. At this point, we must describe the internal structure of the domain concepts. To model these properties, we use the previously identified knowledge relationships and competency questions to specify relevant object and data properties. These properties become slots attached to classes. A slot should be attached at the most general class that can have that property [105] since all subclasses of a class will inherit the slot. When modelling the slots, we have to define the facets of these slots. They can have different facets describing the value type, allowed values, the number of the values (cardinality), and other features of the values the slot can take [105].
For example, the value of a *hasName* slot (as the name of a drug) is a String. In contrast, the value of a slot *treatsDisease* can have multiple values and these values are instances of the class `Disease`. A property has a domain (the source of an property) and a range (the target of a property). We may, for example, consider the object property *treatsDisease* with the class `Drug` as domain and `Disease` as range or the object property *isContradictionTo* with domain `Drug` and range Drug.
Furthermore, properties may have different characteristics (e.g., transitive, symmetric) that might need to be used to answer the competency questions correctly or to make knowledge-acquisition more convenient. In our example, it is sufficient to have the object property *treatsDisease*, since it allows to query for drugs with specific treatments. It is not necessary to define an inverse property *treatedBy* with the domain `Disease` and the range `Drug` since this information is redundant and an application using the knowledge base can always infer the value for the inverse relation. However, it is convenient to have both pieces of information explicitly available. This approach allows users to fill in the disease in one case and the drug in another.



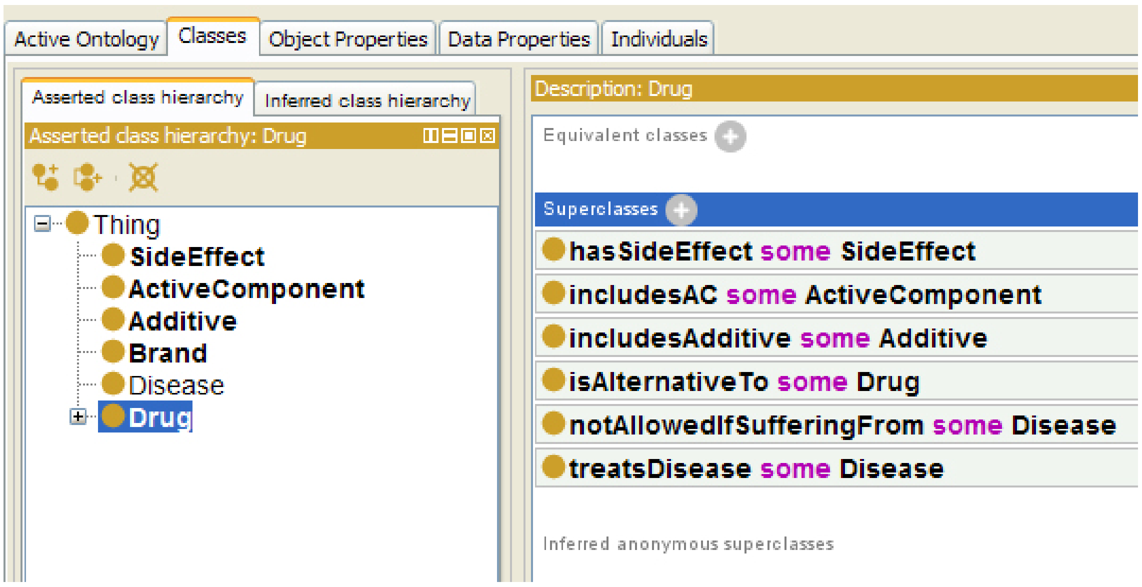
Thing
- ActiveComponent
- Additive
- Brand
- Disease
- Drug
  - AntiInflammatory
  - Pyreticum
  - Analgeticum
  - Antihistamine
  - Painkiller

Superclasses: Thing; containedInDrug some Drug

Members: AcetylsalicylicAcidAC, CaffeineAC, CeterizineAC, DimetindenmaleatAC, IbuprofenAC, ParacetamolAC

Figure 12.2: Drugs ontology - class hierarchy (Active Component selected)

Some of the relationships are illustrated in Figures 12.2 and 12.3. Figure 12.2 shows the superclass relationship *containedInDrug* some `Drug`. This objectProperty has the domain `ActiveComponent` and the range `Drug`. It is inverse to the object property *includesAC* with the domain `Drug` and the range `ActiveComponent`. These object properties exemplifies how to attach and interrelate individuals of the classes `Drug` and `ActiveComponent`. Since these properties are inverse, it does not matter whether to specify a drug to contain a certain active compoment or to say that an active component is contained in a drug. Reasoning will infer this implicit information.

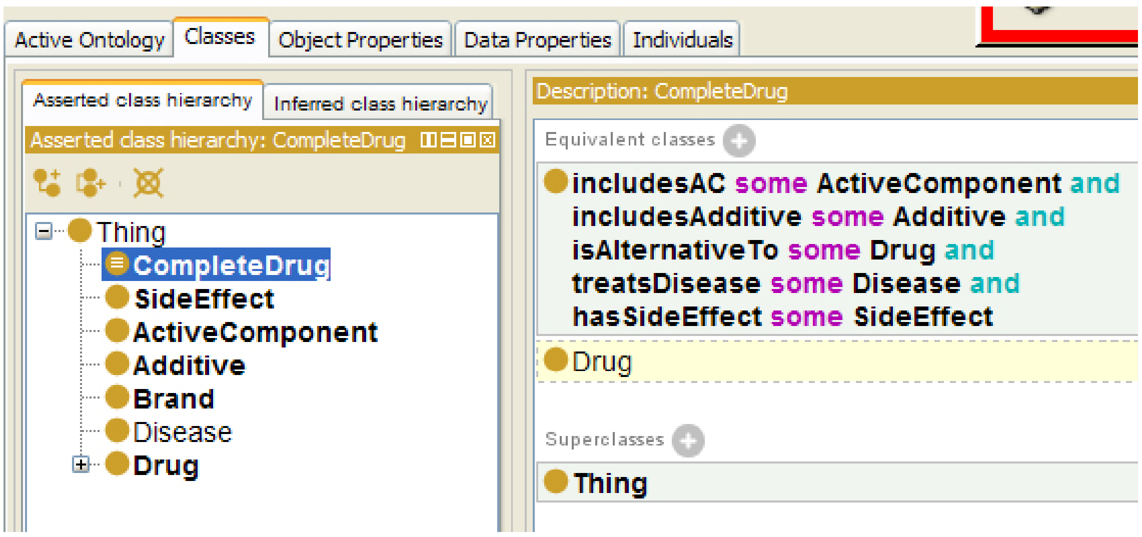


hasSideEffect some SideEffect
includesAC some ActiveComponent
includesAdditive some Additive
isAlternativeTo some Drug
notAllowedIfSufferingFrom some Disease
treatsDisease some Disease

Figure 12.3: Drugs ontology - class hierarchy (Drug selected)

Guideline 60: *The Ontology Metamodel must provide appropriate relations (object and data properties) to validate the completeness, consistency and quality of the instantiated domain knowledge.*

Since one of our scopes for the ontology is to validate the instances regarding their completeness, consistency and quality, we have to extend the ontology appropriately. Here again, we use the competency questions and the validation definitions to identify such relevant properties and their associated ontology classes. At this point, we may also need to add or modify additional classes and existing properties.
In order to validate the consistency of an instantiated domain ontology, we need appropriate relations that allow for specifying relationships between ontology instances. Such relations may, for example, refer to *alternative*, *optional*, *mandatory*, *conflicting*, *excluding* and *coexisting* individuals. Depending on the domain we need to define constraints that allows for deciding when an individual is, for example, in conflict or excludes another.
In our example, the object properties *isContraIndicationTo* and *notAllowedIfSufferingFrom* already allow to specify two kinds of conflicting individuals. If one drug is contraindicated with another, the drug configuration (a combination of several drugs) will be inconsistent and lead to an error. The same is true if we are looking for a specific drug while suffering from a disease that is not allowed to be treated with a specific drug. Figure 12.6 shows an



includesAC some ActiveComponent and
includesAdditive some Additive and
isAlternativeTo some Drug and
treatsDisease some Disease and
hasSideEffect some SideEffect

Drug

Figure 12.4: Drugs ontology - Equivalent class `CompleteDrug`

example for `Penicillin` which is known not be used when suffering from `Allergy`. Thus, *isContraIndicationTo* and *notAllowedIfSufferingFrom* are options to address conflicts in the example domain. How these properties are named is not important as long as the underlying semantics comply to the above guideline. Additionally, we may specify an alternative drug as one that consists of exactly the same active pharmaceutical components. Figure 12.3 shows some more superclass relationships with object properties, most of all the *isAlternativeTo* object property that allows for specifying a drug to be an alternative to another which is necessary for the solution suggestion during the consistency validation.

Guideline 61: *The Ontology Metamodel must provide means to measure the quality of the knowledge specified within.*

Based on Guideline 4, we need to make sure that the ontology provides all the data that is needed to compute the quality measurements of the domain knowledge. Therefore, we may need to add further ontology components to facilitate the reasoning about and the computation of the data.
In our example we add, for example, the data property *isValid* (boolean) the that can be used for drugs whose information have been (or have not been) validated by a domain expert. Furthermore, we may add an equivalent class `CompleteDrug` (see Figure 12.4) that specifies when a drug has been specified completely. This way, (closed world) reasoning will enable us to measure the completeness of drug's information.

12.1.2 Intermediate Evaluation of Ontology Model

The accomplishment of the previous guidelines should be followed by an evaluation to ensure the applicability of the ontology model for the scope defined before.

Guideline 62: *Evaluate and modify the ontology model.*

As already described above, ontology development is an iterative process. By evaluating the current ontology, we can identify modelling errors and revise it accordingly. Therefore, we use the competency questions and check whether the ontology allows for answering them as expected. Certainly, the ontology must be revised to some extent. This task requires evaluation and revision in probably several cycles.

12.2 Implementing the Application

As already described above, developing an ontology is akin to defining a set of data and their structure for other programs to use [105]. Thus, we need to review the answers from the preliminary considerations, where we stated what the ontology shall be used for and how. This functionality must be realised as a software application, software agent, etc. In this example, we will concentrate on the validation purpose and sketch the main tasks to finally enable this functionality. Since there is no general approach in how to allow for answering the competency questions, we will not exemplify this guideline but concentrate on the validation.

Guideline 63: *Implement functionality for the validation and measurement of completeness, consistency and quality.*

The validation of completeness, consistency and quality requires to reason about the domain ontology and the instances within. It is merely a design decision, whether to build a Java Application, a web application or a software agent. Each design solution requires at least the following:

- implementation of validation rules
- queries about the knowledge
- invoking a reasoner that builds a knowledge base of the domain ontology
- output of errors and solution suggestions

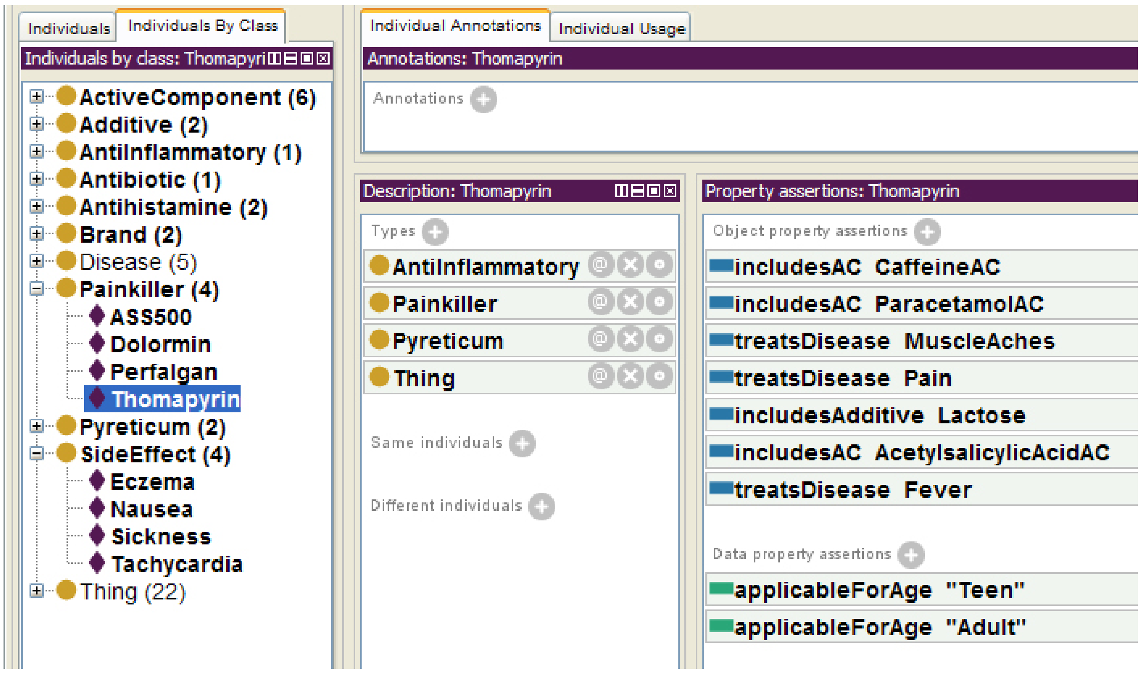
When choosing a reasoner, we need to make sure, that the reasoner allows for closed world reasoning since we need closed world assumption to reason about incompleteness (see Chapter 10 for more details). The previously defined validation rules must now be realised by the software. This means that we either need to use queries (e.g., SPARQL) to compute the relevant implicit knowledge or perhaps use the owlapi. However, we have to make sure that we can check each validation rule and compute solution suggestions. One possible approach that explains closed world reasoning and SPARQL queries is described with code snippets in detail in Section 10.2.
Although it is advisable to model as many aspects as possible within the ontology by using various ontology components (e.g., classes, object and data properties, equivalent classes, etc.), we may need to realise further restrictions and constraints in the software applications since not everything can be modelled in an ontology. The previously defined validation rules and the output of validation results, for example, must obviously be implemented in the software application. Thus, we need to implement queries that reason about the ontology data and allow for its further computation for validation purposes.
In our example, we define among others the following consistency rule:

```
A drug with allergic side effects is not allowed when suffering from
allergies.
```

Since this information is specific to the ontology's ABox, this rule cannot be modelled in the ontology itself. Thus, we need to implement the functionality in the software application and define a method that checks the satisfaction of this rule, e.g. by querying for all drugs without allergy as side effect or, when suggesting alternative drugs, by checking whether one of the known diseases of a patient is an allergy and if so, neglecting all alternative drugs with allergy as side effect.
Additionally we need to implement methods that execute the computation of the various measurements.

Instantiation with Concrete Data

Finally, we can use the domain ontology to instantiate it with concrete knowledge, that are instances of the various classes. In our drugs ontology we need to, for example, specify several individuals for the class `Disease`, e.g., *`Fever`*, *`Pain`*, *`Allergy`*. And we have to use the various object and data properties to interrelate these instances. We specify for example that *`Dolormin`* *treatsDisease* *`Pain`* and *isAlternativeTo* *`Thomapyrin`*.



```
ActiveComponent (6)
Additive (2)
AntiInflammatory (1)
Antibiotic (1)
Antihistamine (2)
Brand (2)
Disease (5)
Painkiller (4)
    ASS500
    Dolormin
    Perfalgan
    Thomapyrin
Pyreticum (2)
SideEffect (4)
    Eczema
    Nausea
    Sickness
    Tachycardia
Thing (22)
```

```
Types
AntiInflammatory
Painkiller
Pyreticum
Thing

Same individuals

Different individuals
```

```
Object property assertions
includesAC  CaffeineAC
includesAC  ParacetamolAC
treatsDisease  MuscleAches
treatsDisease  Pain
includesAdditive  Lactose
includesAC  AcetylsalicylicAcidAC
treatsDisease  Fever

Data property assertions
applicableForAge  "Teen"
applicableForAge  "Adult"
```

Figure 12.5: Instantiation of drugs ontology

Figures 12.5 and 12.6 show an example instantiation of the drugs ontology. As we can see, the reasoner will automatically infer that *`Thomapyrin`* is alternative to *`Dolormin`*, since the object property isAlternativeTo is symmetric (Figure 12.5). Thus, it is sufficient to specify possible alternative only once for one individual, the remaining drugs will automatically be inferred as alternative.

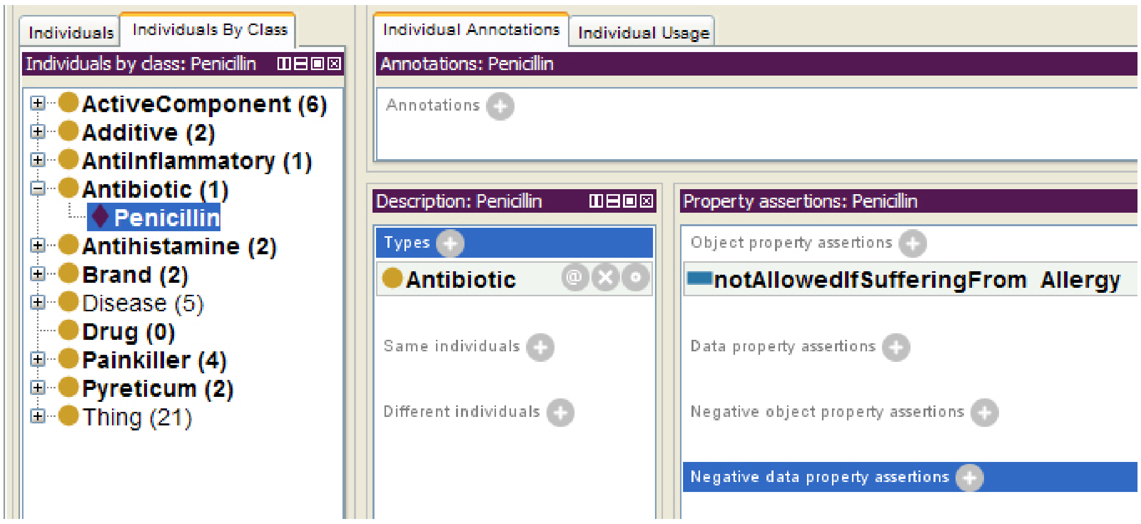


Figure 12.6: Drugs ontology - Consistency property

12.3 Final Evaluation

Now that we have completed the ontology model and the application, we need to accomplish another evaluation. This time, we must ensure that all competency questions are correctly answered and that the validation results and solution suggestions are as expected.

Guideline 64: *Define test-cases for evaluation.*

According to the validation rules, we need to define several test-cases that include incomplete and inconsistent information and aspects of low quality. These test-cases are derived from the competency questions and validation rules. Additionally, we describe the solutions expected for each unsatisfied rule. For our example this may look like the following test-cases:

Validation rules:
1. Each drug must specify its active pharmaceutical ingredients.
2. No drug is allowed to be combined with a drug that is a contraindication.
3. No drug is allowed to be used if a disease hinders its usage.

Test-cases:
Instantiate `Drug` with individuals A,B and C. Make B a contraindication to A and let C have similar diseases specified as A.
Instantiate `Drug` with individual A but no active pharmaceutical ingredients.
Instantiate `Drug` with individual A and `Disease` with individual D. Specify A not be used if suffering from disease D.

Expected results:
Inconsistency error: A and B are contraindicated. Choose one of the alternative drugs: [C].

Inconsistency error: A must not be used when suffering from D. Choose one of the alternative drugs: [X, Y].
Incompleteness error: A does not specify its active pharmaceutical ingredients. Add this information.

12.4 Summary

This chapter provided general guidelines for improving the completeness, consistency and quality of knowledge in domain ontologies. In order to model an appropriate domain ontology, it is necessary to define the domain and scope of the ontology as well as defining competency questions that guide the modelling process and may later be used to define test-cases for the evaluation of the ontology. Based on these preliminary considerations, the ontology class hierarchy can bet set up and the relationships between the domain knowledge can be modelled. This first sketch of the domain ontology must be extended by appropriate ontology components to enable the reasoning for completeness, consistency and quality. Additionally, a software application must be developed to allow for accessing and processing the data stored in the ontology. Finally, the ontology must be evaluated by pre-defined test-cases.

Part IV

Application and Evaluation

13 Using OntoReq for Requirements Engineering - An Exemplar

Cysneiros et.al. state in [26] that a suitable example problem is one way to help clarify strengths and weaknesses of methodologies. This also applies for software. Thus, we defined a suitable example problem for OntoReq that can be used as a common example providing a stable and coherent base for discussion and exchange of ideas and results. This type of example is commonly referred to as an "exemplar" [26]. The examplar given here primarily aims to illustrate and explain the single steps for carrying out several RE tasks with OntoReq. Important steps are visualized by screenshots.

This chapter is structured as follows: in Section 13.1 we describe our examplar. Section 13.2 introduces the specification of requirements, Section 13.3 explains how we validate completeness. The selection of a requirements configuration is demonstrated in Section 13.4.1. This is followed by a demonstration of a consistency validation in Section 13.4 and quality check of requirements in Section 13.6. This example is illustrated with screenshots. Finally, Section 13.8 discusses how to proceed in the software engineering lifecycle with emphasis on making the most benefit of the results and features of OntoReq.

13.1 Exemplar Description

The exemplary scenario is described by a virtual conversation between customer and requirements engineer (see below) as it is usual for a software project to get a first impression about goals and main functionality desired by the customer. This conversation contains most of the requirements artefacts and attributes that will be used for this exemplar. We additionally added further information where appropriate in order to demonstrate most of OntoReqs features. The customer conversation below already highlights all contained requirements artefacts in bold and italic font. In Section 13.2 we summarize them in Table 13.1 and assign the respective type (e.g. goal, functional requirement, priority).

Customer:
"We want to ***increase the satisfaction of our customers when shopping in our electronic store***. We recently conducted a ***survey*** that showed that many of them wish a broader offer of our products. At the same time, it seems that they have already difficulties in identifying which product fits their specific needs and in locating a product. ***We'd like to have some customer guidance for our products*** which ***supports them in identifying the concrete product of their need*** and in ***localizing this product***, if physically available."

Requirements Engineer (RE):
"We can provide a software solution for this. Additionally, you could also ***display further***

information, such as advertisements, announcements and so on."

Customer:
"That sounds good, but it ***must be easy to modify and update***. And it should ***provide information about the physical availability of that product in our store***. I'm just afraid that ***customers will need to learn how to use this guidance system*** which we can only compensate by more personal for a short time periode. Anyway, it ***must be easy to understand and learn***. Let's say ***an average customer (between 18 and 45 years) should not need longer than 5min. to understand how he can use the guidance***."

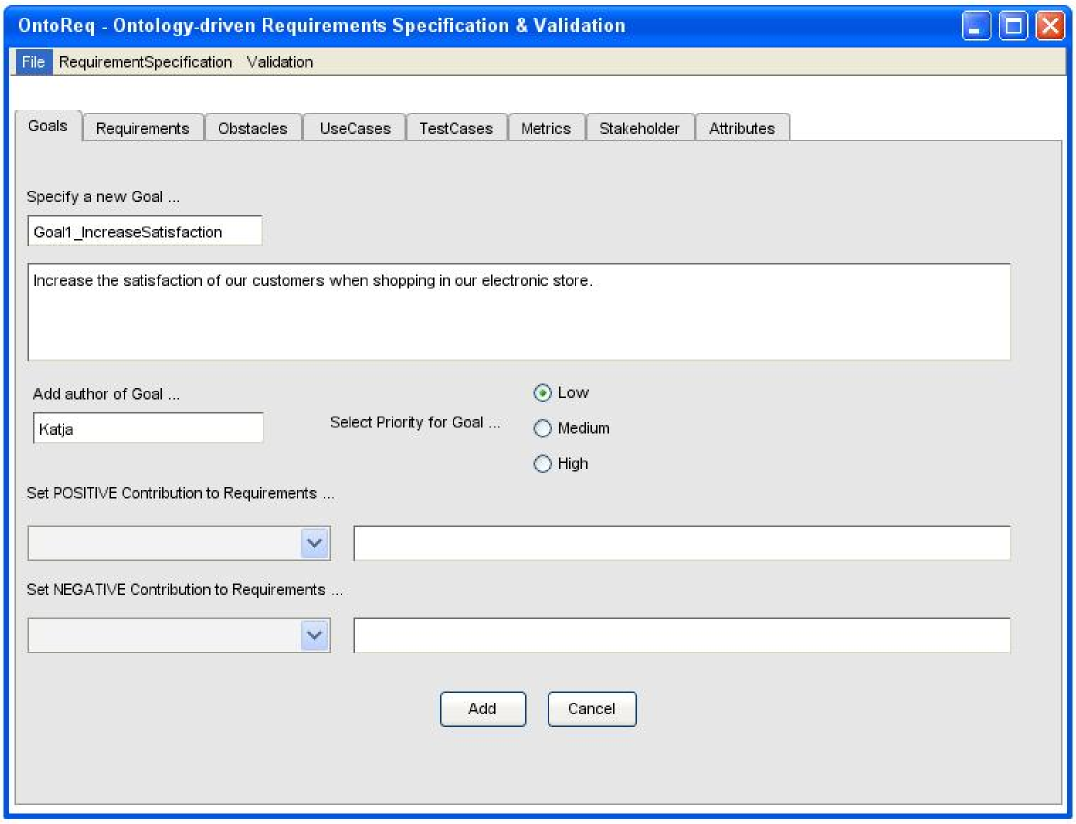
RE:
"OK, could you please try to summarize the usage of this customer product guidance from your perspective in an example (***use-case***)?"

Customer:
"If the customer does not already know where he can find a specific product or even which product indeed fits his needs, he should use the customer product guidance. There he can ***choose between the localization of a product and the identification of a product***. When choosing the localization he can either ***type a term that is searched for*** or ***go through categories and sub-categories to find the product of his choice***. Then he will ***get information about its place in the store and current availability***."

"If the customer is not quite sure what he is actually looking for, he can use the product identification. After a ***pre-selection of categories such as computer, camera, TV***, he ***will be asked several questions to filter the associated products until a set of products is displayed that fits his needs (or he gets a message that the search could not find such a product***). Advertisements, customer information and ***other information can always be displayed when the terminal is currently not used by a customer***. Alternatively, such ***messages could be displayed as audio***. For localization of a product, we ***might*** also ***provide pictures from the place where the product can be found*** or even an ***interactive route through our store***."

13.2 Specifying Requirements

The first step in the RE process is to identify the requirements artefacts and their metadata from the above customer conversation. At this stage, OntoReq is not applied. To facilitate the traceability of requirements in this exemplar and to follow up reasons for these requirements, we provide Table 13.1 below and relate phrases from the text with the requirements that can be identified in it. We use the following abbreviations: Functional Requirement (FR), Non-Functional Requirement (NFR) and Stakeholder (SH). The requirements knowledge illustrated in the table is directly added into the Requirements Ontology. Thereore, OntoReq provides a GUI prototype that allows for adding and modifying requirements knowledge without using an ontology modelling tool such as Protége (see Figure 13.1). We use the IDs from Table 13.1 and add a description for each requirements artefact corresponding to the phrases in the table to enable tracing of text phrases to the requirements.


OntoReq - Ontology-driven Requirements Specification & Validation
File RequirementSpecification Validation
Goals Requirements Obstacles UseCases TestCases Metrics Stakeholder Attributes
Add
Cancel

Specify a new Goal ...

Goal1_IncreaseSatisfaction

Increase the satisfaction of our customers when shopping in our electronic store.

Add author of Goal ...

Katja

Select Priority for Goal ... [✓ Low] Medium High

Set POSITIVE Contribution to Requirements ...

Set NEGATIVE Contribution to Requirements ...

Figure 13.1: Adding requirements artefacts in OntoReq

ID	**Req. Artefacts in RO**	**Phrase**
Goal1_IncreaseSatisfaction	Goal	increase the satisfaction of our customers when shopping in our electronic store
Source1_Survey	Source	survey
Goal2_CustomerGuidance	Goal	We'd like to have some customer guidance for our products
Goal3_ImproveOrientation	Goal	
Goal3.1_SupportProdIdentif	Goal	supports them in identifying the concrete product of their need
Goal3.2_SupportProdLocaliz	Goal	supports them in localizing this product
FR1_CustomerInfo	FR	display further information, such as advertisements, announcements; other information can always be displayed when the terminal is currently not used by a customer
NFR1_Modifiability	NFR	must be easy to modify and update
FR2_PhysicalAvailabOfProducts	FR	give information about the physical availability of that product in our store; get information about product's place in the store
O1	Obstacle	customers will need to learn how to use this guidance system
NFR2_Comprehensibility	NFR	must be easy to understand
NFR3_Learnability	NFR	must be easy to learn
SM1	Soft-Metric	an average customer (between 18 and 45 years) should not need longer than 5min. to understand how he can use the guidance
UC1_LocalizeProduct	Use-Case	The user shall be able to choose between the localization of a product and the identification of a product. When choosing [...] about its place in the store and current availability.
FR6_SelectionOfDesire	FR	choose between the localization of a product and the identification of a product
FR5_ProductSearch	FR	provide search for products in the store
FR5.1_SearchByTerm	FR	type a term that is searched for
FR5.2_SearchByCategory	FR	go through categories and sub-categories to find the product of his choice; preselection of categories such as computer, camera, tv
FR7_CustomerDialogue	FR	will be asked several questions to filter the associated products until a set of products is displayed that fits his needs (or he gets a message that the search could not find such a product)
FR1.2_CustomerInfoAsAudio	FR	messages could be displayed as audio

UC2_IdentifyProduct	Use-Case	After a pre-selection of categories such as computer, camera, tv, he will be asked several questions to filter the associated products until a set of products is displayed that fits his needs (or he gets a message that the search could not find such a product).
UC3_Advertisement	Use-Case	Advertisements, customer information and other information can always be displayed when the terminal is currently not used by a customer. Alternatively, such messages could be displayed as audio.
FR4_ProductPlaceDescription	FR	
FR4.1_ProductPicture	FR	we might also provide pictures from the place where the product can be found
FR4.2_InteractiveRoute	FR	interactive route through our store
S1	SH	Katja

Table 13.1: Identified requirements from customer conversation

13.3 Completeness Validation

When this step is completed, we execute the completeness validation (Figure 13.2) to get a first impression what might be missing and to improve the completeness quite early in the RE process. The complete results of the completeness validation are additionally represented below in Listing 13.1.

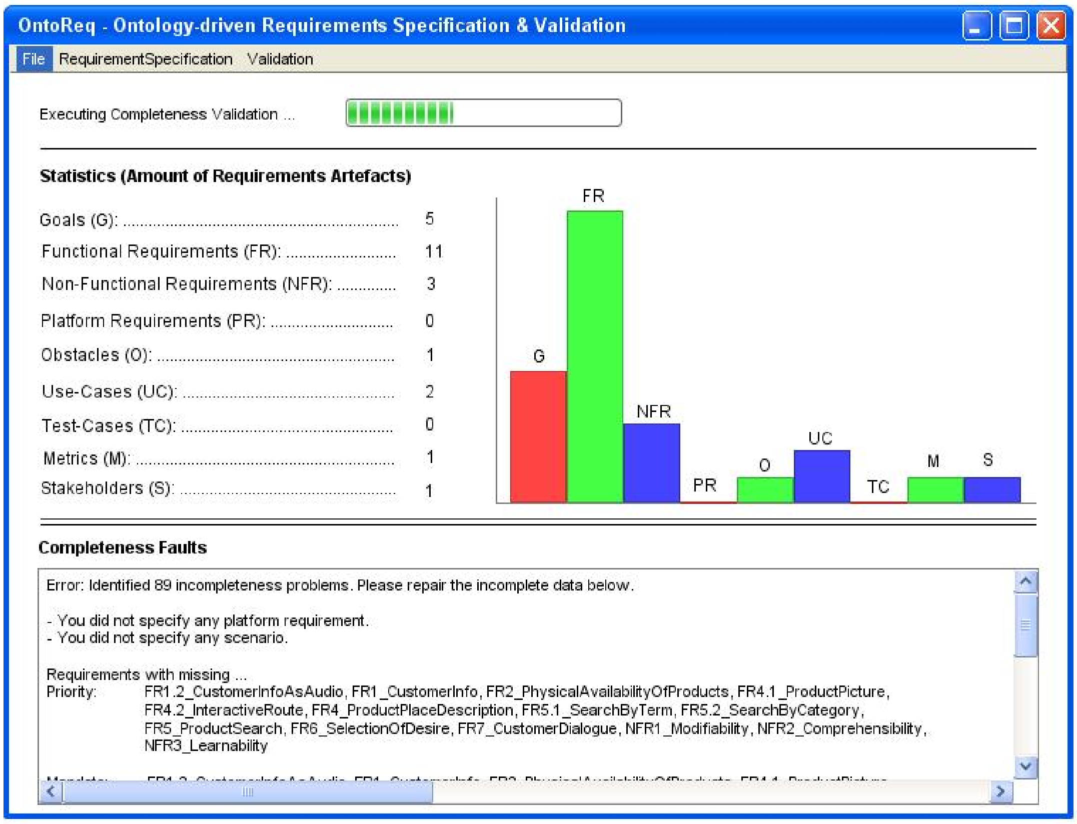
Listing 13.1: Completeness Validation (1)

```
ERROR: Identified 69  incompleteness problems. Please repair the incomplete
    data below:

- You did not specify any platform requirement
- You did not specify any scenario

- The following requirements do not specify any priority:

  [FR1.2_CustomerInfoAsAudio, FR1_CustomerInfo,
  FR2_PhysicalAvailabilityOfProducts, FR4.1_ProductPicture,
  FR4.2_InteractiveRoute, FR4_ProductPlaceDescription,
  FR5.1_SearchByTerm, FR5.2_SearchByCategory, FR5_ProductSearch,
  FR6_SelectionOfDesire, FR7_CustomerDialogue, NFR1_Modifiability,
  NFR2_Comprehensibility, NFR3_Learnability]

- The following requirements do not specify any mandate:

  [FR1.2_CustomerInfoAsAudio, FR1_CustomerInfo,
  FR2_PhysicalAvailabilityOfProducts, FR4.1_ProductPicture,
  FR4.2_InteractiveRoute, FR4_ProductPlaceDescription,
  FR5.1_SearchByTerm, FR5.2_SearchByCategory, FR5_ProductSearch,
  FR6_SelectionOfDesire, FR7_CustomerDialogue, NFR1_Modifiability,
  NFR2_Comprehensibility, NFR3_Learnability]

- The following requirements do not specify any author:

  [FR1.2_CustomerInfoAsAudio, FR1_CustomerInfo,
  FR2_PhysicalAvailabilityOfProducts, FR4.1_ProductPicture,
  FR4.2_InteractiveRoute, FR4_ProductPlaceDescription,
  FR5.1_SearchByTerm, FR5.2_SearchByCategory, FR5_ProductSearch,
  FR6_SelectionOfDesire, FR7_CustomerDialogue, NFR1_Modifiability,
  NFR2_Comprehensibility, NFR3_Learnability]

- The following requirements do not specify any requirement relationship:

  [FR1.2_CustomerInfoAsAudio, FR1_CustomerInfo,
  FR2_PhysicalAvailabilityOfProducts, FR4.1_ProductPicture,
  FR4.2_InteractiveRoute, FR4_ProductPlaceDescription,
  FR5.1_SearchByTerm, FR5.2_SearchByCategory, FR5_ProductSearch,
  FR6_SelectionOfDesire, FR7_CustomerDialogue, NFR1_Modifiability,
  NFR2_Comprehensibility, NFR3_Learnability]

- The following requirements do not specify any soft-metric:

  [NFR1_Modifiability, NFR2_Comprehensibility, NFR3_Learnability]

- The following requirements artefacts do not relate to any use-case:

  [Goal1_IncreaseSatisfaction, Goal2_CustomerGuidance,
  Goal3.1_SupportProdIdentification, Goal3.2_SupportProdLocalization,
  Goal3_ImproveOrientation]

- The following requirements artefacts do not specify any scenario:

  [UC1_LocalizeProduct, UC2_IdentifyProduct, UC3_Advertisement]
```

Obviously, the requirements knowledge lacks a lot of information. The guidance of the completeness validation suggests to add priorities to the specified requirements and to define whether a requirement is mandatory or optional (mandate). Authors are also missing and, most of all, we did not interrelate the requirements with each other and other requirements artefacts (e.g. relate requirements to a use-case). Thus, we need to formalize the requirements knowledge by interrelating the requirements. Therefore, we analyse the previously defined requirements and try to identify conflicts, alternative, coexisting or excluding requirements, refinements, and so on. Additionally, we relate the requirements to requirements artefacts, e.g., specifying for the functional requirement FR1.1_CustomerInfoAsText the use-case UC3_Advertisement. Such relationships are specified in OntoReq (see Figure 13.3) and are important for the following validations. Most of the priorities and requirements relationships can be taken from the customer conversation. However, we will add further information that would have been identified in another customer conversation or requirements analysis, which we will not simulate here. The resulting requirements knowledge is depicted in the Appendix.
After this step, we execute another completeness validation. The results are shown below in Listing 13.2.

Listing 13.2: Completeness Validation (2)

```
ERROR: Identified 8 incompleteness problems. Please repair the incomplete
    data below:

- You did not specify any platform requirement
- You did not specify any scenario
- The following requirements do not specify any soft-metric:
  [NFR1_Modifiability, NFR2_Comprehensibility]

- The following requirements artefactss do not specify any goal:
  [UC3_Advertisement]
```


OntoReq - Ontology-driven Requirements Specification & Validation
File
RequirementSpecification
Validation
Executing Completeness Validation ...
G
FR
NFR
PR
O
UC
TC
M
S

Statistics (Amount of Requirements Artefacts)

Goals (G):	5
Functional Requirements (FR):	11
Non-Functional Requirements (NFR):	3
Platform Requirements (PR):	0
Obstacles (O):	1
Use-Cases (UC):	2
Test-Cases (TC):	0
Metrics (M):	1
Stakeholders (S):	1

Completeness Faults

```
Error: Identified 89 incompleteness problems. Please repair the incomplete data below.

- You did not specify any platform requirement.
- You did not specify any scenario.

Requirements with missing ...
Priority:     FR1.2_CustomerInfoAsAudio, FR1_CustomerInfo, FR2_PhysicalAvailabilityOfProducts, FR4.1_ProductPicture,
              FR4.2_InteractiveRoute, FR4_ProductPlaceDescription, FR5.1_SearchByTerm, FR5.2_SearchByCategory,
              FR5_ProductSearch, FR6_SelectionOfDesire, FR7_CustomerDialogue, NFR1_Modifiability, NFR2_Comprehensibility,
              NFR3_Learnability
```

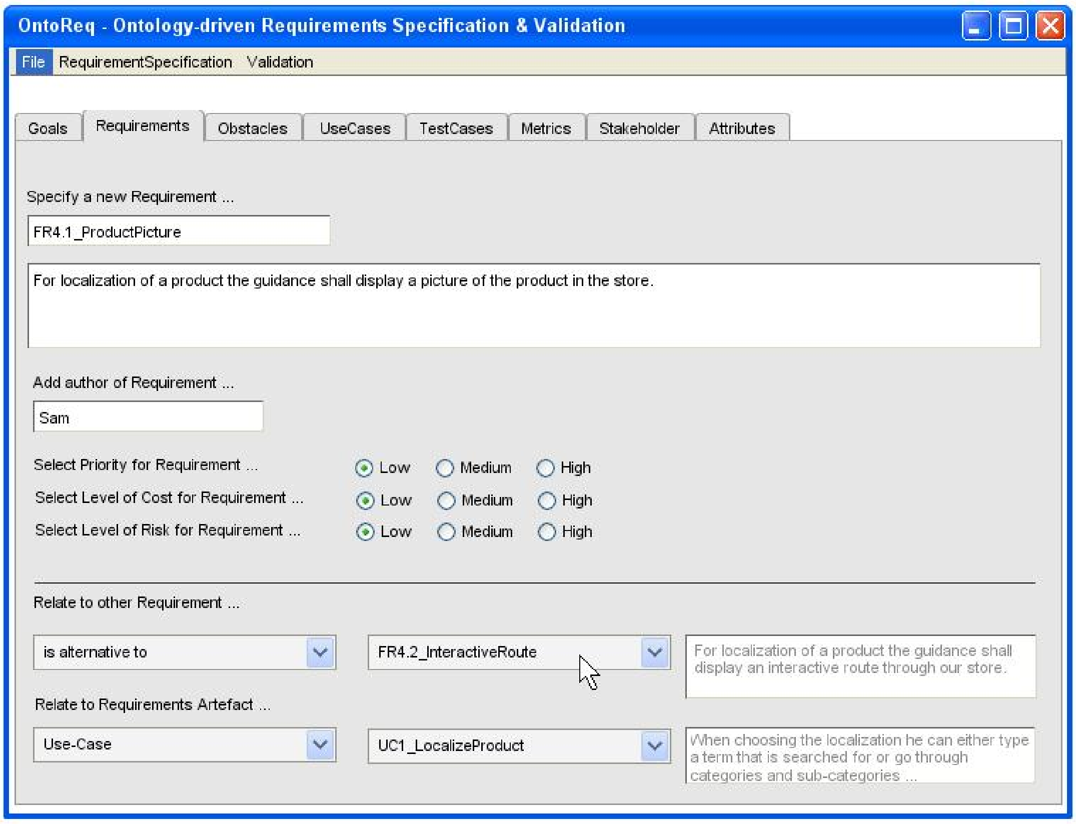
Figure 13.2: Completeness Validation in OntoReq

```

- The following requirements artefactss do not specify any scenario:
  [UC1_LocalizeProduct , UC2_IdentifyProduct , UC3_Advertisement]
```

Listing 13.2 shows that we decreased the amount of incomplete data. The still existing completeness errors indicate that we should add some further information. However, this is only a suggestion to improve the completeness. As a matter of course, it is not always possible to comply with all completeness rules. There may, for example, indeed be requirements we cannot relate to any goal or for which we cannot estimate a Level of Cost or describe a scenario. It is the decision of the requirements engineer which information to add and which suggestions to ignore. Nevertheless, the completeness of the specified knowledge has improved significantly.

13.4 Consistency Validation of Requirements Configuration

Since we cannot add more information by now, we decide to execute the consistency validation (see Figure 13.4). Please note, that it is most important to complete the requirements knowledge as much as possible before executing the consistency validation in order to get meaningful solution suggestions. To execute the consistency validation, we need to choose whether we want to implement all specified requirements or only a selection of them. This is the definition of the requirements configuration. Therefore, we


OntoReq - Ontology-driven Requirements Specification & Validation

File RequirementSpecification Validation

Goals | Requirements | Obstacles | UseCases | TestCases | Metrics | Stakeholder | Attributes

Specify a new Requirement ...

FR4.1_ProductPicture

For localization of a product the guidance shall display a picture of the product in the store.

Add author of Requirement ...

Sam

Select Priority for Requirement ... [✓ Low] Medium High

Select Level of Cost for Requirement ... [✓ Low] Medium High

Select Level of Risk for Requirement ... [✓ Low] Medium High

Relate to other Requirement ...

is alternative to | FR4.2_InteractiveRoute | For localization of a product the guidance shall display an interactive route through our store.

Relate to Requirements Artefact ...

Use-Case | UC1_LocalizeProduct | When choosing the localization he can either type a term that is searched for or go through categories and sub-categories ...

Figure 13.3: Formalisation of requirements knowledge

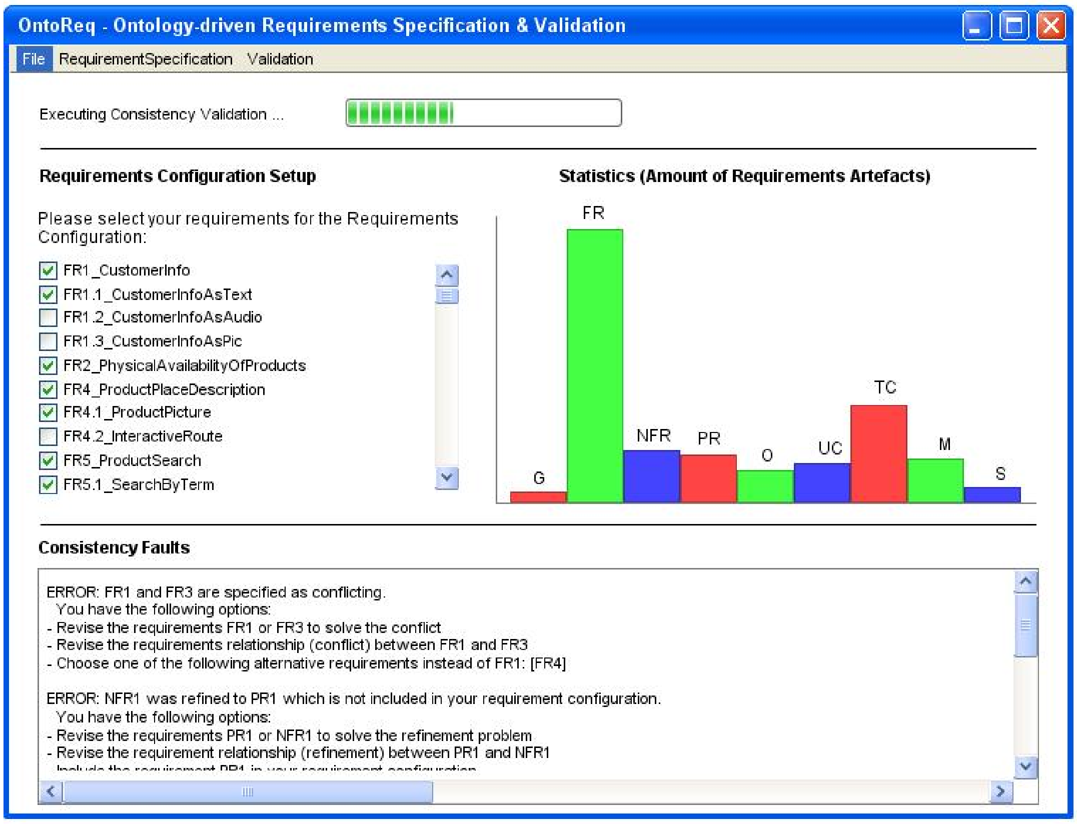
can choose between an automated suggestion of a requirements configuration or a manual configuration. The automated configuration is guaranteed to be consistent and can be used as a basis for manually modifying this selection of requirements. Here, we demonstrate both approaches and start with the manual configuration.

13.4.1 Manual Configuration of Requirements

First, we choose to manually select the following requirements, which form together the requirements configuration RC1. This selection is exceptionally designed in a way that we can illustrate all opportunities of the consistency validation. To allow the reader to follow the consistency errors and solution suggestions, we refer to the Tables in Appendix A.5 in the appendix, which capture the specified requirements knowledge.

Requirements Configuration

- FR1_CustomerInfo
- FR1.1_CustomerInfoAsText
- FR2_PhysicalAvailabilityOfProducts
- FR4_ProductPlaceDescription
- FR5_ProductSearch
- FR6_SelectionOfDesire
- FR7_CustomerDialogue
- NFR1_Modifiability
- NFR2_Comprehensibility


OntoReq - Ontology-driven Requirements Specification & Validation
File
RequirementSpecification
Validation
Executing Consistency Validation ...
Requirements Configuration Setup
Statistics (Amount of Requirements Artefacts)
FR
NFR
PR
O
UC
TC
M
S
G
Consistency Faults

Please select your requirements for the Requirements Configuration:

- ✓ FR1_CustomerInfo
- ✓ FR1.1_CustomerInfoAsText
- ☐ FR1.2_CustomerInfoAsAudio
- ☐ FR1.3_CustomerInfoAsPic
- ✓ FR2_PhysicalAvailabilityOfProducts
- ✓ FR4_ProductPlaceDescription
- ✓ FR4.1_ProductPicture
- ☐ FR4.2_InteractiveRoute
- ✓ FR5_ProductSearch
- ✓ FR5.1_SearchByTerm

```
ERROR: FR1 and FR3 are specified as conflicting.
  You have the following options:
- Revise the requirements FR1 or FR3 to solve the conflict
- Revise the requirements relationship (conflict) between FR1 and FR3
- Choose one of the following alternative requirements instead of FR1: [FR4]

ERROR: NFR1 was refined to PR1 which is not included in your requirement configuration.
  You have the following options:
- Revise the requirements PR1 or NFR1 to solve the refinement problem
- Revise the requirement relationship (refinement) between PR1 and NFR1
```

Figure 13.4: Consistency Validation

The consistency validation shows the following extract of results (the complete results can be found in the appendix):

Listing 13.3: Consistency Validation (1)

```
13 inconsistency errors detected.

- The following requirements are mandatory and should be included in the
    requirements configuration:
[FR1.3_CustomerInfoAsPic, FR4.1_ProductPicture, FR5.1_SearchByTerm, FR5.2
    _SearchByCategory, FR8_AudioOutput, NFR3_Learnability]

----------------------------------------
- Error: FR8_AudioOutput and FR7_CustomerDialogue are specified as
    conflicting.
  You have the following options:
  - Revise the requirements FR8_AudioOutput or FR7_CustomerDialogue to
      solve the conflict
  - Revise the requirements relationship (conflict) between FR8_AudioOutput
       and FR7_CustomerDialogue
  - Choose one of the following alternative requirements instead of FR4.2
      _InteractiveRoute: [FR4.1_ProductPicture]

----------------------------------------
- Error: FR5_ProductSearch was refined to FR5.2_SearchByCategory which is
    not included in your requirements configuration.
  You have the following options:
  - Revise the requirements FR5.2_SearchByCategory or FR5_ProductSearch to
      solve the refinement problem
  - Revise the requirement relationship (refinement) between FR5.2
      _SearchByCategory and FR5_ProductSearch
  - Include the requirement FR5.2_SearchByCategory in your requirements
      configuration.

[...]
```

Now it is the task of the requirements engineer to eliminate these errors. Therefore, he can choose between several options for each error. We demonstrate a possible approach with these errors. First, we will add all mandatory requirements to our requirements configuration. They are either mandatory because they have been specified as mandatory or they have been inferred by OntoReq as mandatory due to the fact that they specify a high level of priority (and we defined that all requirements with a high priority automatically become mandatory). Since we added a number of new requirements to the requirements configuration, we must execute another consistency check before trying to eliminate the remaining errors. The results are the following:

Listing 13.4: Consistency Validation (2)

```
3 inconsistency errors detected.

 Error: FR7_CustomerDialogue and FR8_AudioOutput are specified as
     conflicting.
  You have the following options:
  - Revise the requirements FR7_CustomerDialogue or FR8_AudioOutput to
      solve the conflict
  - Revise the requirements relationship (conflict) between
      FR7_CustomerDialogue and FR8_AudioOutput
```

```

----------------------------------------
- Error: FR1_CustomerInfo was refined to FR1.2_CustomerInfoAsAudio which is
    not included in your requirements configuration.
  You have the following options:
  - Revise the requirements FR1.2_CustomerInfoAsAudio or FR1_CustomerInfo
      to solve the refinement problem
  - Revise the requirement relationship (refinement) between FR1.2
      _CustomerInfoAsAudio and FR1_CustomerInfo
  - Include the requirement FR1.2_CustomerInfoAsAudio in your requirements
      configuration.

- Error: FR4_ProductPlaceDescription was refined to FR4.2_InteractiveRoute
    which is not included in your requirements configuration.
  You have the following options:
  - Revise the requirements FR4.2_InteractiveRoute or
      FR4_ProductPlaceDescription to solve the refinement problem
  - Revise the requirement relationship (refinement) between FR4.2
      _InteractiveRoute and FR4_ProductPlaceDescription
  - Include the requirement FR4.2_InteractiveRoute in your requirements
      configuration
```

As we can see in Listing 13.4, we resolved some of the inconsistency errors. The remaining errors are handled as follows. We revise the requirements description of FR8_AudioOutput so that it is no longer in conflict with FR7_CustomerDialogue. Additionally, we must delete the requirements relationship *isInConflictWith* in Ontoreq. Furthermore, we add all the refined requirements to the requirements configuration as suggested by OntoReq and thus, resolved all inconsistency errors. We execute another consistency validation (see Listing 13.5) after these modifications.

Listing 13.5: Consistency Validation (3)

```

- Error: FR1.2_CustomerInfoAsAudio excludes FR1.3_CustomerInfoAsPic.
  You have the following options:
  - Revise the requirements FR1.2_CustomerInfoAsAudio or FR1.3
      _CustomerInfoAsPic to solve the exclusion problem
  - Revise the requirements relationship (exclusion) between FR1.2
      _CustomerInfoAsAudio and FR1.3_CustomerInfoAsPic
  - You may choose to delete the optional requirement: FR1.2
      _CustomerInfoAsAudio
  - Choose one of the following alternative requirements instead of FR1.2
      _CustomerInfoAsAudio: [FR1.1_CustomerInfoAsText]

----------------------------------------

- Error: FR4.2_InteractiveRoute and NFR1_Modifiability are specified as
    conflicting.
  You have the following options:
  - Revise the requirements FR4.2_InteractiveRoute or NFR1_Modifiability to
      solve the conflict
  - Revise the requirements relationship (conflict) between FR4.2
      _InteractiveRoute and NFR1_Modifiability
  - You may choose to delete the optional requirement: FR4.2
      _InteractiveRoute
  - Choose one of the following alternative requirements instead of FR4.2
      _InteractiveRoute: [FR4.1_ProductPicture]
```

As we can see in Listing 13.5, we resolved the previous inconsistency errors but introduced new inconsistencies in the requirements configuration. This demonstrates the problems of adding requirements to complete the requirements knowledge while introducing inconsistency. According to the solution suggestions we will now delete the optional requirements FR1.2_CustomerInfoAsAudio and FR4.2_InteractiveRoute from the requirements configuration as suggested by OntoReq. Another consistency validation finally shows no errors, thus we have a consistent requirements configuration.

13.4.2 Automated Requirements Configuration

As explained in Section 8.5, OntoReq also provides the option to generate a consistent requirements configuration automatically. This configuration is guaranteed to include all mandatory and their coexistent requirements. Furthermore, refined requirements are considered instead of the more abstract requirements and the configuration does not include any conflicting or excluded requirements. Thus, the requirements configuration complies to the consistency rules. Besides that, it also acknowledges the quality rules and thus, includes for example requirements with a positive contribution to a goal even if those are optional. In contrast, requirements with a negative contribution to a goal are not selected if they are optional. For our exemplar, OntoReq computes the following requirements configuration RC2 in Listing 13.6:

Listing 13.6: Automated Requirements Configuration RC2

```
FR2_PhysicalAvailabilityOfProducts,
FR6_SelectionOfDesire,
FR5.2_SearchByCategory,
FR1.1_CustomerInfoAsText,
FR7_CustomerDialogue,
FR5.1_SearchByTerm,
FR5.2_SearchByCategory,
NFR1_Modifiability,
NFR2_Comprehensibility,
FR4_ProductPlaceDescription,
FR1.3_CustomerInfoAsPic,
NFR3_Learnability,
FR5_ProductSearch,
FR4.1_ProductPicture
```

This requirements configuration can be used as a basis and be complemented by additional requirements. However, any modifications of this automated requirements configuration must be followed by a consistency validation.

13.5 Stakeholder Review of Requirements Knowledge

In addition to the different validations provided by OntoReq, it is necessary to review the requirements artefacts regarding their description and correctness of relationships. Obviously, this a task that cannot be automated due to fact that the semantics of requirements is not machine comprehensible. Thus, stakeholders from the project need to review and revise the requirements, their metadata and the specified requirements relationships. However, OntoReq provides the data property *isValid* to mark those requirements that have been reviewed and accepted. This information is important for the quality measurements described in Section 13.7.
We now assume a review of the requirements knowledge and exemplarily set the following requirements to be valid:

- FR1_CustomerInfo
- FR1.1_CustomerInfoAsText
- FR1.2_CustomerInfoAsAudio
- FR1.3_CustomerInfoAsPic

13.6 Quality Validation of Requirements Configuration

Now that we completed the requirements knowledge as much as possible and eliminated inconsistencies, we can execute a quality validation of our requirements knowledge. The quality validation checks again for incomplete or inconsistent information and suggests another execution of the appropriate validations if such errors have been recognized. Furthermore, it aims to improve goal satisfaction, that is, identifying requirements with a positive goal contribution that have not been included in the requirements configuration and, at the same time, preventing negative goal contributions by searching for possible alternative requirements. Additionally, OntoReq tries to decrease the amount of requirements with a high risk level, by suggesting possible alternatives. The execution of the quality validation with the requirements configuration RC1 leads to the following results in Listing 13.7:

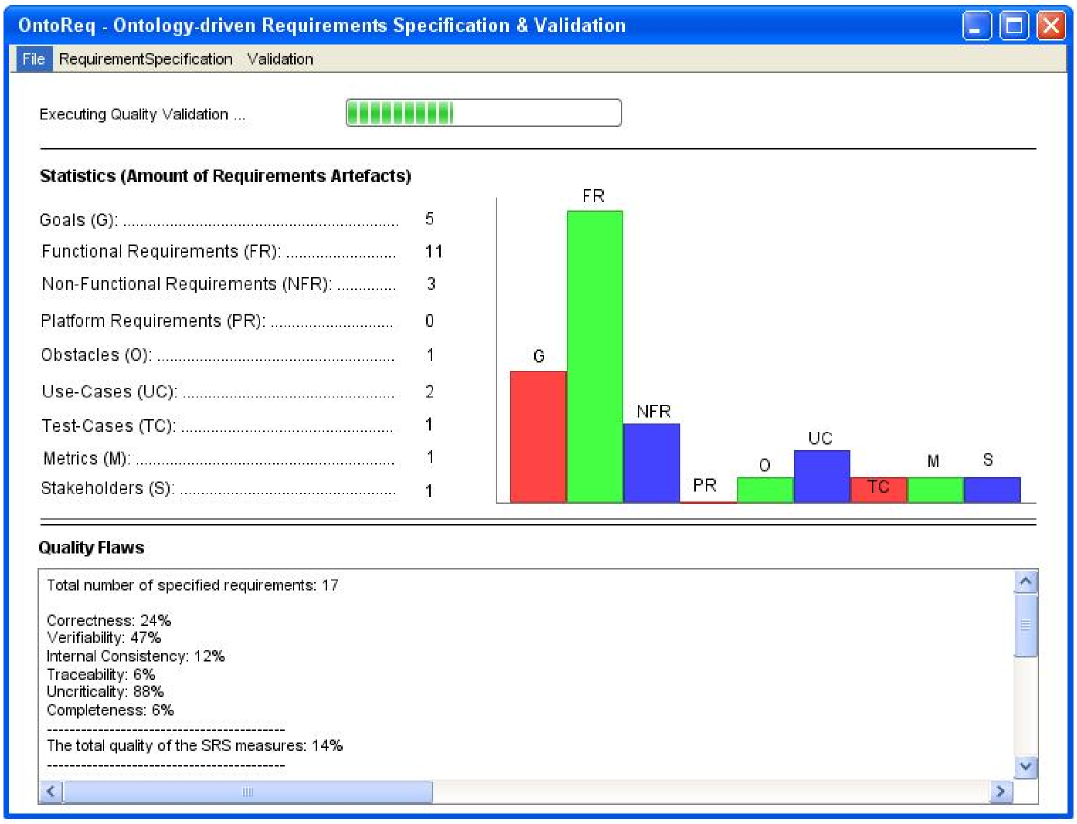
Listing 13.7: Quality Validation

```

- Error: Identified 8 incompleteness problems that decrease the quality of
    your Requirement Configuration.
  Please execute the completeness check and follow the instructions to
      repair incomplete information.

-----------------------------------------
- Warning: FR1.1_CustomerInfoAsText is a negative contribution to goal
    Goal3_ImproveOrientation.
  You have the following options:
  - Revise the above goal contribution,
  - Modify the requirement to meet the goal
  - Choose one of the following alternative requirements instead of FR1.1
      _CustomerInfoAsText: [FR2_PhysicalAvailabilityOfProducts]

-----------------------------------------
- Warning: FR5.2_SearchByCategory is a positive contribution to goal Goal3
    .1_SupportProdIdentification but was not considered in your requirement
     configuration.
You may consider to add this requirement to improve goal satisfaction.

- Warning: FR8_AudioOutput is a positive contribution to goal
    Goal2_CustomerGuidance but was not considered in your requirement
    configuration.
  You may consider to add this requirement to improve goal satisfaction.

-----------------------------------------
- Error: FR4.2_InteractiveRoute has a high level of risk.
    You have the following options:
    - Delete this optional requirement from your configuration
    - Choose one of the following alternative requirements instead of FR4.2
        _InteractiveRoute: [FR4.1_ProductPicture]
```

As shown in Listing 13.7, we can improve the quality by adding the requirement FR5.2_SearchByCategory since it has a positive influcence on a goal and thus, improves goal satisfaction. The opposite is true for FR1.1_CustomerInfoAsText that would be a negative contribution to a goal, although it is optional. It might therefore be a good idea to exclude this requirement from the requirements configuration. Furthermore, in order to reduce the risk of the specified requirements, we may consider to exchange FR4.2_InteractiveRoute with FR4.1_ProductPicture. After these changes, we should run another consistency validation. For reasons of space we omit this at this point and continue with the quality measurements.

13.7 Quality Measurements of Requirements Knowledge

As explained in Section 10.4, OntoReq provide means to make assumptions about the quality of the specified requirements knowledge. In contrast to the consistency and quality validations, these measurements cover not only the requirements configuration but the complete specified requirements knowledge in OntoReq. If the value of a metric is lower than 50%, OntoReq provides suggestions on how to improve a specific quality metric. The execution of the quality measurements (see Figure 13.5) shows the results summarized in Listing 13.8.


OntoReq - Ontology-driven Requirements Specification & Validation
File
RequirementSpecification
Validation
Executing Quality Validation ...
FR
G
NFR
PR
O
UC
TC
M
S

Statistics (Amount of Requirements Artefacts)

Goals (G):	5
Functional Requirements (FR):	11
Non-Functional Requirements (NFR):	3
Platform Requirements (PR):	0
Obstacles (O):	1
Use-Cases (UC):	2
Test-Cases (TC):	1
Metrics (M):	1
Stakeholders (S):	1

Quality Flaws

```
Total number of specified requirements: 17

Correctness: 24%
Verifiability: 47%
Internal Consistency: 12%
Traceability: 6%
Uncriticality: 88%
Completeness: 6%
------------------------------------------
The total quality of the SRS measures: 14%
------------------------------------------
```

Figure 13.5: Quality Measurement

The complete results are shown in Listing 13.8.

Listing 13.8: Quality Measurement (1)

```
Total number of specified requirements: 17

Correctness: 24%
Verifiability: 47%
Internal Consistency: 12%
Traceability: 6%
Uncriticality: 88%
Completeness: 6%
------------------------------------------
The total quality of the SRS measures: 14%
------------------------------------------

You can improve the traceability of the requirements knowledge by
    specifying a use-case, test-case, author and source for each
    requirement.

You can improve the verifiability of the requirements knowledge by
    specifying a test-case or soft-metric for a single requirement.

You can improve the correctness of the requirements knowledge by reviewing
    the requirements and their relations and - if correct - setting the
    value 'isValid' for each validated requirement.
```

These quality measurements can now be used to improve single quality criteria, e.g. correctness. In this exemplar, we follow the suggestion in OntoReq and aim to improve traceability and correctness. Therefore, we review our requirements once more and add the *isValid* relation where appropriate. This will improve the correctness of the requirements knowledge. Furthermore, we relate as much requirements as possible with a test-case or soft-metric to achieve a better value for verifiability. In detail, we modify the requirements knowledge in OntoReq as follows:

- add *isValid* relation to each requirement
- add a new test-case TC3_CustomerDialogue and relate it to appropriate requirements (FR1 - FR1.3, FR5 - R5.2, FR6 - FR8)

Another quality measurement leads to the following results in Listing 13.9:

Listing 13.9: Quality Measurement (2)

```
Total number of specified requirements: 17

Correctness: 100%
Verifiability: 82%
Internal Consistency: 12%
Traceability: 29%
Uncriticality: 88%
Completeness: 29%
------------------------------------------
The quality of the SRS measures: 35%
------------------------------------------

You can improve the traceability of the requirements knowledge by
    specifying a use-case, test-case, author and source for each
    requirement.
```

As we can see in Listing 13.9, the correctness value reached 100% and the verifiability improved as well. Since we added a test-case to some requirements, we also improved the completeness and traceability of the requirements knowledge. Thus, the overall quality of the requirements configuration increased by more than 50%.

13.8 Further Steps

All of the previously described steps may be accomplished as often as appropriate. Especially the validation tasks needs to be executed after any modifications to ensure that completeness, consistency and quality have not been corrupted by the changes or, if so, can be resolved. Any modifications of the requirements configuration (adding or deleting one or more requirements from the configuration) must be followed by a another consistency validation, since this is the time when new inconsistencies are introduced.

14 Evaluation

Basili et. al. in [11] describes and classifies several methods for software experimentation (evaluation). Such evaluation involves a thesis and test process. Since there is no general evaluation method that can be applied for any purpose, software, project, etc., one has to choose which method best fits to the purpose of and resources for evaluation. As illustrated in Figure 14.1, Basili distinguishes software-evaluation studies in terms of a two-dimensional (team size and project size) classification scheme (illustrated in Figure 14.1). This classification scheme has been extended by Kitchenham et. al. in [77] regarding the formality of experimental design. In the following, we present the classification scheme from [11] with additional information for extension from [77] (in italic font):

- **Blocked subject-project studies** = *formal experiment or survey*
  examine one or more objects across a set of teams and a set of projects
- **Replicated projects** = *case study*
  examine object(s) across a set of teams and a single project
- **Multi-project variation** = *case study or formal experiment*
  examine object(s) across a single team and a set of projects
- **Single-project study** = *case study*
  examine object(s) across a single team and a single project.
  It is not possible to have a formal experiment without replication.

Here, Basili defines *teams* as "(possibly single-person) groups that work separately". They may be characterized by experience, size, organization, etc. *Projects* are defined as "separate programs or problems on which teams work". They may be characterized by size, complexity, application, etc. [11]. "A *formal experiment* requires appropriate levels of replication, and experimental subjects and objects that are chosen at random within the constraints of experimental design" [77]. According to [77] any investigation can therefore

Number of teams	**Number of projects**	
	One	**More than one**
One	Single-project	Multi-project variation
More than one	Replicated project	Block subject-project

Figure 14.1: Classification of Evaluation scopes (edited according [11])

be considered as *case study*, *formal experiment* or *survey.*

The Requirements Ontology and the validation of completeness and consistency (Evaluation 1) has primarily been evaluated within the MOST Project[1]. During development, we have simulated its usage by instantiating it against a few case studies of the project, i.e., semantic modelling of network physical devices ([99]) and validating Component-Based Implementations of Business Processes [88]. This allowed for a continuous revision of the Requirements Ontology during the project.
The evaluations of OntoReq followed the formal criteria of a use case as experiment. We examined three projects (or problems) with one and the same team. According to Basilis classification scheme it can be classified as *multi-project variation.* In this chapter, we present one of these evaluations and its results. Additionally, we decided to evaluate the underlying concepts of OntoReq with a broader set of people and defined two universal problems in Requirements Engineering which OntoReq aims to eliminate or at least to reduce (Evaluation 2). This evaluation is targeted to allow for a comparison of solving RE problems without any support and with support by OntoReq.

This chapter is structured as follows: we firstly describe the evaluation of the Requirements Ontology on the basis of a requirements specification of the Names Project [110] in Section 14.1. The second evaluation is described in Section 14.2, where we used two control groups to make assumptions about the impact of guidance support provided by OntoReq. For each evaluation, we describe the motivation, evaluation object and procedure. We also give background information about the case study and the domain and scope of the evaluations. Finally, Section 14.3 summarizes this chapter.

14.1 Requirements Ontology and Requirements Validation Services

The following evaluation investigated the Requirements Ontology TBox for its completeness and consistency and analysed whether the ontology provides sufficient metadata to capture common requirements knowledge.

14.1.1 Description of the Case Study

The case study of the Names project [110] aims to develop a software that solves the problem of uniquely identifying authors. “Without having a means of uniquely and unambiguously identifying those involved in the creation of materials in repositories, it becomes difficult to be sure whether all the materials related to a particular person will be correctly associated with that individual” [110]. Names of authors may be entered in more than one way (e.g., only last name, first name and last name, as initials, etc.), or more than one author may have exactly the same name. The Names project started in July 2007 and ended in February 2009.

The Names project investigated the requirements of the UK’s repository community for a name authority service and developed a demonstrator. The team determined the types of metadata which were required for the service.

[1]Marrying Ontologies with Software Technology, http://www.most-project.eu/

The second phase of the project established the software requirements for the prototype. This involved reviewing the stakeholder requirements and data analysis documents and drawing up a specification based on these.
This resulting requirements specification is used in this thesis for the evaluation of our Requirements Ontology.

14.1.2 Motivation

With this first evaluation we aimed to check the TBox of the Requirements Ontology, the provided metadata and applicability to instantiate requirements knowledge appropriately. Our goal was to answer the following questions:

- Is the structure of the Requirements Ontology reasonable?
- Are the ontology concepts of the TBox correct and complete?
- Are the requirements metadata of the TBox sufficient and applicable?
- Can all data from the Case Study be captured within the Requirements Ontology?
- Are there any difficulties in using the Requirements Ontology?
- Can completeness and consistency faults be detected in the ABox?
- Is the performance of completeness and consistency validations sufficient to support real time modelling actions?

14.1.3 Evaluation Object

The Requirements Specification of the Names Project consists of 37 functional requirements, 5 non-functional requirements, and 95 relationships between these requirements. We regard this as a medium size representation of a Requirements Specification

14.1.4 Evaluation Procedure

In order to evaluate the Requirements Ontology, we first collected all available information about the Case Study that could be of importance (e.g., Project description). Further on, we identified relevant data (e.g., stakeholder, requirements, goals) and instantiated the Requirements Ontology with the set of requirements artefacts extracted from the Names Project. We identified and instantiated 37 functional requirements and five non-functional requirements. Each of these requirements could be assigned a priority level, a source (kind of documentation), an ID and a responsible stakeholder. The identified requirements relationships were refinement and coexistence relations. Furthermore, eight Use-Cases were instantiated and given an ID.
All problems or questions that appeared during this procedure were documented. Finally, we checked the consistency of the Requirements Ontology and validated the completeness and consistency.

14.1.5 Results

We identified two faulty ontology concepts and one missing requirement metadata in the Requirements Ontology. Apart from that, all of the available information in the Case Study could be instantiated in the Requirements Ontology. Thus, the evaluation proved the ontology structure to be reasonable and appropriate to capture requirements knowledge. However, one problem emerged during evaluation: the data provided in the requirements specification of the Names project was not sufficient to allow an instantiation of each requirements artefact or requirements relationship in our Requirements Ontology. Due to the predefined requirements metadata in the Requirements Ontology, it is possible to capture much more requirements knowledge than usual requirements specification provide. But this is a vicious cycle since the Requirements Ontology allows for specifying more requirements knowledge than is currently captured, but at the same time, this assumption can hardly be proven due to the lack of data in available requirements specification.

The evaluation was performed on a 2GHz dual core MacBook Pro. The evaluation was an excellent test of the ability of our approach to deal with incomplete and inconsistent specifications. The Case Study ontology failed 14 of the 37 tests and a total of 132 different individual problems were identified. Thus, the evaluation proved the detection of incomplete and inconsistent requirements artefacts, which was one of our main goals. The total time required to test the completeness constraints was 808ms. The performance of the system is therefore good enough to be used as part of an ongoing cycle of testing and revision of the Requirements Ontology ABox.

The evaluation results can be summarized as positive regarding the ontology structure, completeness and correctness of concepts and requirement metadata. The tests for requirements completeness and consistency proved to be working and showed the expected results. The performance of these tests is such that they can be executed regularly, and used by a requirements engineering tool to highlight inconsistencies or incompleteness during real time requirement modelling.

14.2 Comparison of Universal RE Tasks without and with support by OntoReq

The second evaluation investigates the applicability of the guidance support provided by OntoReq and aims to compare the accomplishement of general RE tasks with and without support by OntoReq.

14.2.1 Motivation

General tasks during Requirements Engineering are the extraction of requirements from text or from customer conversations and the specification of these requirements. Furthermore, it is necessary to finally check them for their completeness and consistency. Since OntoReq aims to support requirements engineers in these tasks, we want to compare the accomplishment of these tasks, supported with OntoReq and without any support.

14.2.2 Description of Evaluation

The evaluation consists of two single evaluation forms (A and B) with two tasks per forms. Forms A asks the evaluation subject[2] in the first task to gather all requirements and requirements relationships of an invented recorded customer conversation by copying and pasting appropriate keywords from the text in prepared empty tables and fields. We will refer to this task as *A1*. The second task depicts a requirements configuration and additional requirements knowledge (e.g., relationships, priorities) and a set of consistency and completeness rules the subject is asked to validate the requirements configuration against. By considering these rules, he needs to modify the previously defined requirements configuration and note the resulting modified requirements configuration that should now be complete and consistent. We refer to it as task *A2*. The subject is not supported in any of these tasks (A1 and A2).
Form B of the evaluation asks the subject to accomplish exactly the same tasks as in A1 and A2 with the only difference that he is supported by OntoReq. We refer to these tasks as task *B1* and task *B2*. Instead of asking the subject to install OntoReq and get familiar with it, we decided instead to use only the real outputs and guidance messages of OntoReq without involving the application itself in the evaluation process. This has the advantage to compare the results of these tasks independently from technical barriers, software competency and increases the acceptance of the evaluation due its simplicity of accomplishement.
Both evaluation forms are designed as pdf documents. The accomplishment of each task is restricted to 20 minutes. A subject is intended to either complete part A or part B. In order to randomly assign both parts, we give form A to subjectťs with the month of birth in the first half of the year and form B to the others.

14.2.3 Evaluation Object

The above mentioned customer conversation is identical with the one already provided in Chapter 13.

14.2.4 Target Group

The evaluation is targeted at persons of any age with at least little knowledge in Requirements Engineering. The evaluation subjects are familiar with software requirements and their extraction from text. Additionally, they must be able to understand German and English. Further background knowledge is not necessary.

14.2.5 Expectation

The evaluation aims to come to a statement whether usual RE tasks such as requirements elicitation and the complete and consistent specification of requirements knowledge can be improved by OntoReq. This statement is based on several metrics measured in the evaluation:

1. the number of correctly identified requirements artefacts (functional, non-functional requirement, goal, priority)

[2]An evaluation subject is the person accomplishing the evaluation tasks. We will refer to it as “subject”.

2. the number of correct identified requirements relationships
3. the correct consideration of completeness and consistency rules

The results of these metrics can be compared between A1 and B1 and A2 and B2.
We expect that the results from tasks A1 and A2 will be less complete and correct (regarding the validation against the defined rules to be acknowledged). It might be the case, that tasks A1 and A2 will not have been completed in time. A1 and B1 should differ in the completeness of requirements artefacts, e.g., goals or non-functional requirements might not be identified in A1. The final requirements configuration in A2 might not be correct while it should in B2.

14.2.6 Results

Unfortunately the number of returned (and complete) evaluation forms was too low (three returns) to allow for meaningful and scientifically interpretable results. However, in the following we will list and analyse the existing results.

Below we list the results of the study persons for each type of requirements artefact and the identified requirements relationships (task 1). The first value denotes the number of identified instances of a particular requirements artefact and the second value states the correct value for this particular requirements artefact. If appropriate, more information on each single result is given after the value of identified requirements artefacts.

Person1, Evaluation Form A, Task 1

- Goals: 3 (4)
- Functional Requirements: 13 (11), 6 correct, 1 comprised 2 FRs, 3 are UseCases, 1 is metric, 2 are NFRs
- Non-Functional Requirements: 0 (2)
- Sources: 0 (2)
- Obstacle: 2 (1), 1 not related to text
- Use-Cases: 0 (3)
- Metrics: 0 (1)
- Stakeholders: 0 (2)

Person1 specified 37% (10 of 27) requirements artefacts correctly. The most obvious mistake was the wrong selection of requirements artefacts (functional requirements instead of use-case, metric and non-functional requirements). Person1 identified 50% (4 of 8) of the different types of requirements artefacts. Stakeholders, non-functional requirements, use-cases, metric and the source were not been identified.
13.8% (5 of 36) of the requirements relationships were identified correctly. Priorities, refinements, coexisting requirements, goal contributions and relations to obstacle and source were not been identified. Requirements were not been specified as mandatory or optional.

Person2, Evaluation Form B, Task 1

- Goals: 5 (4), 4 correct, 1 is FR
- Functional Requirements: 6 (11), 5 correct, 1 comprised 3 FRs

- Non-Functional Requirements: 5 (2), 2 correct, 2 are FRs, 1 is metric
- Sources: 2 (2), 0 correct
- Obstacle: 1 (1), 1 correct
- Use-Cases: 2 (3), 2 correct
- Metrics: 1 (1), 1 correct
- Stakeholders: 1 (2), 1 correct

Person2 specified 59% (16 of 27) requirements artefacts correctly. 100% (8 of 8) different types of requirements artefacts were identified.
The requirements relationships were correctly identified at 75% (27 of 36).

Person3, Evaluation Form B, Task 1

- Goals: 3 (4), 3 correct
- Functional Requirements: 11 (11), 11 correct
- Non-Functional Requirements: 2 (2), 1 correct, 1 is FR
- Sources: 2 (2), 2 correct
- Obstacle: 0 (1)
- Use-Cases: 0 (3)
- Metric: 0 (1)
- Stakeholders: 0 (2)

Person3 specified 62.3% (17 of 27) requirements artefacts correctly and 50% (4 of 8) of the different types of requirements artefacts.
The requirements relationships were correctly identified at 13.8% (5 of 36).

Evaluation Form A and B, Task 2

There are two correct solutions for a consistent requirements configuration. The results of person1 and person2 were correct. Person3 neglected one requirement in the requirements configuration.

14.2.7 Conclusion

Due to the insufficient number of returned evaluation forms, no scientific conclusions can be drawn and no generalization of the results is possible. The comparison of the results only shows that Person1 who was not supported by OntoReq achieved worse results in task 1 than the remaining two persons who were guided by OntoReq.

14.3 Summary

In this chapter we describe our evaluation approaches. The Requirements Ontology and the validation of completeness and consistency has primarily been evaluated within the MOST Project[3]. Therefore, we have simulated its usage by instantiating it against a few case studies of the project. The evaluation results can be summarized as positive regarding the ontology structure, completeness and correctness of concepts and requirement metadata.

Furthermore, we examined three projects (or problems) with one and the same team. According to Basilis classification scheme it can be classified as *multi-project variation*.

[3]Marrying Ontologies with Software Technology, http://www.most-project.eu/

The tests for requirements completeness and consistency of the metamodel proved to be working and showed the expected results. The performance of these tests is such that they can be executed regularly, and used by a Requirements Engineering tool to highlight inconsistencies or incompleteness during real time requirement modelling.
Additionally, we describe our evaluation approach for the underlying concepts of OntoReq and defined two universal problems in Requirements Engineering which OntoReq aims to eliminate or at least to reduce. This evaluation is targeted to allow for a comparison of solving RE problems without any support and with support by OntoReq. However, due to the insufficient number of returned evaluation forms, no scientific conclusions could be drawn and no generalization of the results is possible.

15 Comparison of OntoReq with other RE Tools

An increasing number of software support for Requirements Engineering has been poured out on the market. The aims, strengthens and weaknesses of these tools differ enormously. While some tools aim to support the documentation of requirements, others improve the specification of requirements or support the RE process with a certain methodology. However, there are some major tools that are mainly used within industry (in the years 2006-2008) and are well-known. Our goal is to give a review of a set of these tools, describe their benefits, main features and weaknesses. Therefore, Beger in [13] developed a classification scheme, based on a number of tool requirements that RE software may realize. We use this classification scheme to compare OntoReq with RE tools previously evaluated by Beger. Although OntoReq has the scope of a scientific demonstration prototype instead of being a tool ready for industry, we use this schema to compare several characteristics of RE tools with OntoReq. The selection of tools presented here is not meant to be complete. However, the classification scheme can easily be used for any other tool and thus, provide means for comparing RE tools regarding a set of criteria.

This chapter is structured as follows: we first introduce the classification scheme in Section 15.1 and proceed with a description of the evaluated RE tools in Section 15.2. Afterwards in Section 15.3, we use the classification scheme to compare the features offered by this tools with OntoReq. Finally, a summary is given in Section 15.4.

15.1 Classification Scheme

The structure of the classification scheme is based on an evaluation framework for RE tools developed by Prof. Dr. Klaus Pohl and presented in [118]. This classification scheme lists seven views (criteria groups) that allows for the classification and evaluation of RE tools from different point of views. In the following, we briefly introduce these several criteria according to [118].

Product View

The product view comprises all aspects of the specification, organisation and management of information, thus, the complete functionality of the RE software. This view includes document types (predefined types, structuring, views, constraints), traceability (decisions, sources of change, relation types), reports (screen-, paper, or internet-based), code generation (programming languages, sources) and configuration management (revisions, version control).

User view

The user view considers facets of tools from the user perspective: user management (single- or multi-position system, role-based or project-based administration, user authorization, user profiles), support for groups (parallel working, notifications, reviews) and the user interface (adaptability, clarity, user guidance).

Project View

The project view states how the a tool may support the project management. This includes the preparation (definition of project specific documents and information types), planning (milestones, work- and resource planning), and controlling (automatisms, project control, quality assurance) of projects.

Process View

The process view addresses the method support of the software, e.g., the definition of project-specific procedures or traceability (granularity, creation, usage).

Technical View

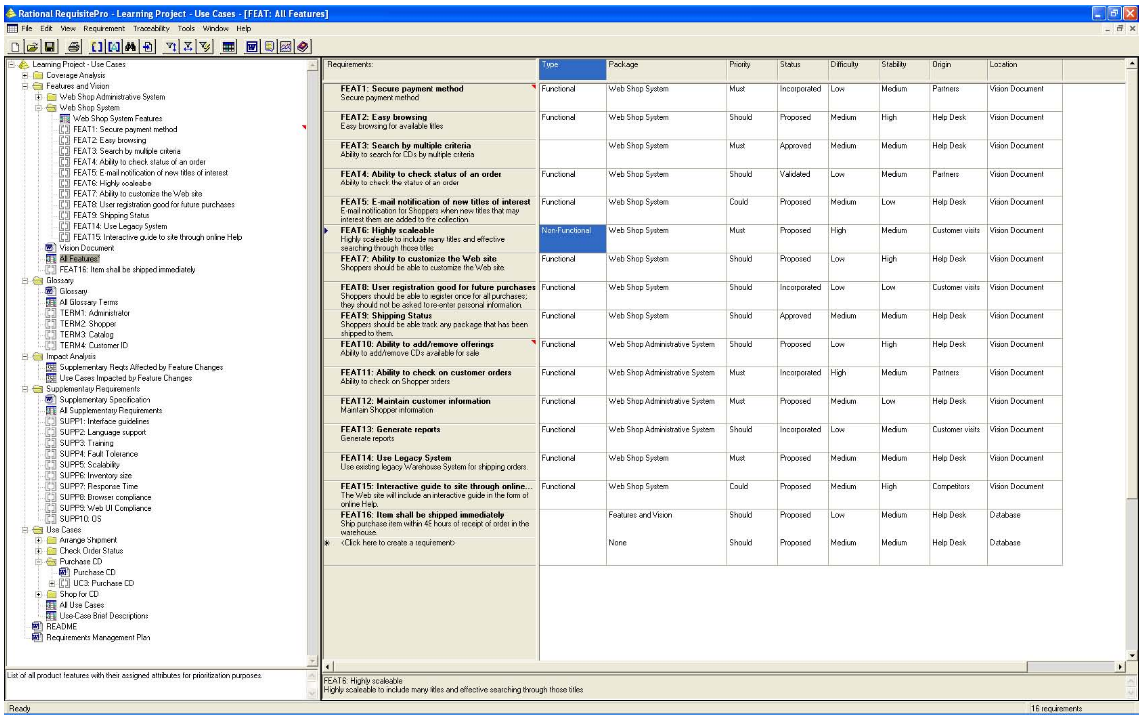
The technical view comprises the technical constraints of the software context the hard- and software (memory, network, local installation, operating system, additionally required software), the integration potential (APIs, process-, data-, control-integration), the data management in a repository (safety concept, data import and export, query interface, supported standards) and the scalability of the software (maximum number of users, objects, active projects).

Supplier View

The supplier view describes the market position and service of the tool developer. This includes among other criteria the consulting (hotline, software documentation), training, company data (market relevance, stability of the the company) as well as release and licensing policy.

Economical View

The economical view comprises potential costs to buy the product. This includes initial costs, costs for infrastructure (additional hard- or software) as well as costs for training, operational- and maintainability costs.
The evaluation of RE tools depends on several also subjective criteria (e.g., the budget, experience, scope of use, optional or mandatory criteria). Thus, the classification scheme includes only basic criteria that allow for a general evaluation of RE tools based on criteria that can be evaluated.


Rational RequisitePro - Learning Project - Use Cases - [FEAT: All Features]
File
Edit
View
Requirement
Traceability
Tools
Window
Help

Learning Project - Use Cases
- Coverage Analysis
- Features and Vision
  - Web Shop Administrative System
  - Web Shop System
    - Web Shop System Features
    - FEAT1: Secure payment method
    - FEAT2: Easy browsing
    - FEAT3: Search by multiple criteria
    - FEAT4: Ability to check status of an order
    - FEAT5: E-mail notification of new titles of interest
    - FEAT6: Highly scaleabe
    - FEAT7: Ability to customize the Web site
    - FEAT8: User registration good for future purchases
    - FEAT9: Shipping Status
    - FEAT14: Use Legacy System
    - FEAT15: Interactive guide to site through online Help
  - Vision Document
  - All Features*
  - FEAT16: Item shall be shipped immediately
- Glossary
  - Glossary
  - All Glossary Terms
  - TERM1: Administrator
  - TERM2: Shopper
  - TERM3: Catalog
  - TERM4: Customer ID
- Impact Analysis
  - Supplementary Reqts Affected by Feature Changes
  - Use Cases Impacted by Feature Changes
- Supplementary Requirements
  - Supplementary Specification
  - All Supplementary Requirements
  - SUPP1: Interface guidelines
  - SUPP2: Language support
  - SUPP3: Training
  - SUPP4: Fault Tolerance
  - SUPP5: Scalability
  - SUPP6: Inventory size
  - SUPP7: Response Time
  - SUPP8: Browser compliance
  - SUPP9: Web UI Compliance
  - SUPP10: OS
- Use Cases
  - Arrange Shipment
  - Check Order Status
  - Purchase CD
    - Purchase CD
    - UC3: Purchase CD
  - Shop for CD
  - All Use Cases
  - Use-Case Brief Descriptions
- README
- Requirements Management Plan

List of all product features with their assigned attributes for prioritization purposes.

Requirements:	Type	Package	Priority	Status	Difficulty	Stability	Origin	Location
FEAT1: Secure payment method Secure payment method	Functional	Web Shop System	Must	Incorporated	Low	Medium	Partners	Vision Document
FEAT2: Easy browsing Easy browsing for available titles	Functional	Web Shop System	Should	Proposed	Medium	High	Help Desk	Vision Document
FEAT3: Search by multiple criteria Ability to search for CDs by multiple criteria		Web Shop System	Must	Approved	Medium	Medium	Help Desk	Vision Document
FEAT4: Ability to check status of an order Ability to check the status of an order	Functional	Web Shop System	Should	Validated	Low	Medium	Partners	Vision Document
FEAT5: E-mail notification of new titles of interest E-mail notification for Shoppers when new titles that may interest them are added to the collection.	Functional	Web Shop System	Could	Proposed	Medium	Low	Help Desk	Vision Document
FEAT6: Highly scaleable Highly scaleable to include many titles and effective searching through those titles	Non-Functional	Web Shop System	Must	Proposed	High	Medium	Customer visits	Vision Document
FEAT7: Ability to customize the Web site Shoppers should be able to customize the Web site.	Functional	Web Shop System	Should	Proposed	Low	High	Help Desk	Vision Document
FEAT8: User registration good for future purchases Shoppers should be able to register once for all purchases; they should not be asked to re-enter personal information.	Functional	Web Shop System	Should	Incorporated	Low	Low	Customer visits	Vision Document
FEAT9: Shipping Status Shoppers should be able track any package that has been shipped to them.	Functional	Web Shop System	Should	Approved	Medium	Medium	Help Desk	Vision Document
FEAT10: Ability to add/remove offerings Ability to add/remove CDs available for sale	Functional	Web Shop Administrative System	Should	Proposed	Low	High	Help Desk	Vision Document
FEAT11: Ability to check on customer orders Ability to check on Shopper orders	Functional	Web Shop Administrative System	Must	Incorporated	High	Medium	Partners	Vision Document
FEAT12: Maintain customer information Maintain Shopper information	Functional	Web Shop Administrative System	Must	Proposed	Medium	Low	Help Desk	Vision Document
FEAT13: Generate reports Generate reports	Functional	Web Shop Administrative System	Should	Incorporated	Low	Medium	Customer visits	Vision Document
FEAT14: Use Legacy System Use existing legacy Warehouse System for shipping orders.	Functional	Web Shop System	Must	Proposed	Medium	Medium	Help Desk	Vision Document
FEAT15: Interactive guide to site through online... The Web site will include an interactive guide in the form of online Help.	Functional	Web Shop System	Could	Proposed	Medium	High	Competitors	Vision Document
FEAT16: Item shall be shipped immediately Ship purchase item within 48 hours of receipt of order in the warehouse.		Features and Vision	Should	Proposed	Low	Medium	Help Desk	Database
* <Click here to create a requirement>		None	Should	Proposed	Medium	Medium	Help Desk	Database

FEAT6: Highly scaleable
Highly scaleable to include many titles and effective searching through those titles

Ready

16 requirements

Figure 15.1: User Interface of RequisitePro (2007)

15.2 Description of Software Tools and Evaluation Results

15.2.1 IBM® Rational® RequisitePro®

In 2003, IBM adopted the company Rational Software, and thus the development of IBM Rational RequisitePro. It counts alongside DOORS to the best known and most widely used development tools in Requirements Engineering/Management.

RequisitePro is based on a document-centric approach and links to the compulsory Microsoft Office Word. It supports the creation of Word documents based on different document types (Word templates), e.g., *Business Rules*, *Requirements Management Plan*, *Stakeholder Request* or *Vision and Features*. Requirements are also typed ("Feature", "Risk", "Stakeholder Request", "Test-Case", "Use-Case", etc.) and can be created as text sections by a special toolbar within Microsoft Office Word. Requirements that have been generated that way, are automatically included in the database of RequisitePro where they can be accessed within. They are hierarchically structured and contain a number of predefined attributes. The linking of requirements as well as the specification of attributes and the requirements analysis are managed via user dialogues or views. Attributes for requirements may be customized and used for filtering.

The organization of requirements knowledge is realised by projects folder ("packages"). The design of RequistePro is similar to the Windows Explorer from Microsoft®(see Figure 15.1). In this work we evaluate version 7.1 in conjunction with Microsoft Office Word 2007. Detailed information about RequisitePro can be found in [36, 118].

15.2.2 Conclusion

RequisitePro with its clear, but visually outdated user interface looks pretty unimpressive, but supports typical RE tasks.
The functionality and thus the learning curve is somewhat lower than in DOORS and Polarion Requirements. A large number of different attributes, requirements and document types is predefined. The predefined templates are similar to metamodels and can easily be created or modified. Requirements types can be defined by the user. However, validation and measurements regarding completeness, consistency and quality is not provided.
Especially characteristic (but also restrictive) for RequisitePro is its close connection with Microsoft Office Word and the document-centric approach. Cross-project Requirements Engineering becomes difficult due to the fact that projects are managed independently from each other. Additional functions, such as UML modelling are not supported. We summarize the following positive and negative aspects of RequisitePro:

Positive

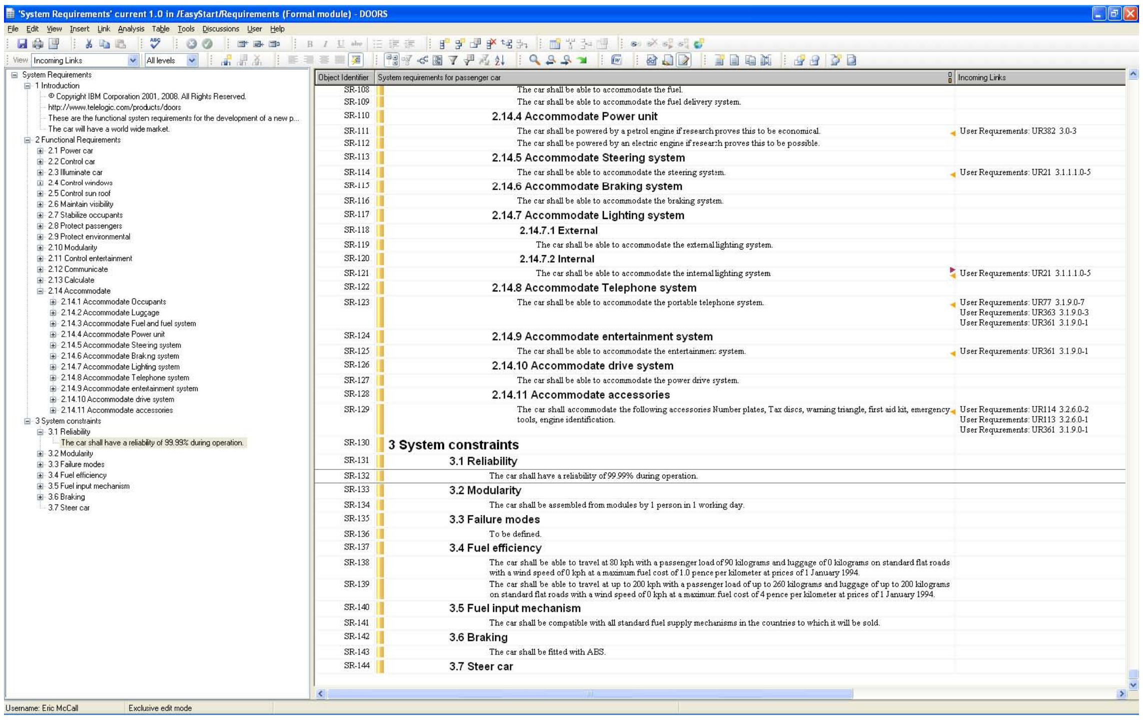
- Relatively low learning curve
- Many predefined attributes/ request types/ document types
- Users can choose auto-numbering of requirements
- Assignment of permissions when selecting attributes for individual attribute values
- User-definable requirement types
- Customizable and filterable requirements attributes

Negative

- No validation support for completeness, consistency or quality of captured requirements
- Cumbersome installation
- Optical outdated user interface
- Requirements that are created in the database cannot be incorporated in a Word document
- Poor integration of external documents
- No simultaneous access to multiple projects
- Dependency to Microsoft Office Word

15.2.3 IBM® Rational® DOORS®

IBM Rational DOORS (Dynamic Object Oriented Requirements System) was developed by Telelogic AB until 2008. In the same year, IBM took over the company and their products and integrated them into the IBM Rational Software portfolio. DOORS counted as the technology and market leader in Requirements Engineering/Management for the last few years. It provides a rich feature set and is used by companies of all sizes worldwide.



'System Requirements' current 1.0 in /EasyStart/Requirements (Formal module) - DOORS

File Edit View Insert Link Analysis Table Tools Discussions User Help

View: Incoming Links | All levels

- System Requirements
  - 1 Introduction
    - © Copyright IBM Corporation 2001, 2008. All Rights Reserved.
    - http://www.telelogic.com/products/doors
    - These are the functional systen requirements for the development of a new p...
    - The car will have a world wide market.
  - 2 Functional Requirements
    - 2.1 Power car
    - 2.2 Control car
    - 2.3 Illuminate car
    - 2.4 Control windows
    - 2.5 Control sun roof
    - 2.6 Maintain visibility
    - 2.7 Stabilize occupants
    - 2.8 Protect passengers
    - 2.9 Protect environmental
    - 2.10 Modularity
    - 2.11 Control entertainment
    - 2.12 Communicate
    - 2.13 Calculate
    - 2.14 Accommodate
      - 2.14.1 Accommodate Occupants
      - 2.14.2 Accommodate Lugcage
      - 2.14.3 Accommodate Fuel and fuel system
      - 2.14.4 Accommodate Power unit
      - 2.14.5 Accommodate Steering system
      - 2.14.6 Accommodate Brakng system
      - 2.14.7 Accommodate Lighting system
      - 2.14.8 Accommodate Telephone system
      - 2.14.9 Accommodate enteitainment system
      - 2.14.10 Accommodate drive system
      - 2.14.11 Accommodate accessories
  - 3 System constraints
    - 3.1 Reliability
      - The car shall have a reliability of 99.99% during operation.
    - 3.2 Modularity
    - 3.3 Failure modes
    - 3.4 Fuel efficiency
    - 3.5 Fuel input mechanism
    - 3.6 Braking
    - 3.7 Steer car

Object Identifier	System requirements for passenger car	Incoming Links
SR-108	The car shall be able to accommodate the fuel.	
SR-109	The car shall be able to accommodate the fuel delivery system.	
SR-110	**2.14.4 Accommodate Power unit**	
SR-111	The car shall be powered by a petrol engine if research proves this to be economical.	User Requrements: UR382 3.0-3
SR-112	The car shall be powered by an electric engine if research proves this to be possible.	
SR-113	**2.14.5 Accommodate Steering system**	
SR-114	The car shall be able to accommodate the steering system.	User Requrements: UR21 3.1.1.1.0-5
SR-115	**2.14.6 Accommodate Braking system**	
SR-116	The car shall be able to accommodate the braking system.	
SR-117	**2.14.7 Accommodate Lighting system**	
SR-118	**2.14.7.1 External**	
SR-119	The car shall be able to accommodate the external lighting system.	
SR-120	**2.14.7.2 Internal**	
SR-121	The car shall be able to accommodate the internal lighting system	User Requrements: UR21 3.1.1.1.0-5
SR-122	**2.14.8 Accommodate Telephone system**	
SR-123	The car shall be able to accommodate the portable telephone system.	User Requrements: UR77 3.1.9.0-7 User Requrements: UR363 3.1.9.0-3 User Requrements: UR361 3.1.9.0-1
SR-124	**2.14.9 Accommodate entertainment system**	
SR-125	The car shall be able to accommodate the entertainment system.	User Requrements: UR361 3.1.9.0-1
SR-126	**2.14.10 Accommodate drive system**	
SR-127	The car shall be able to accommodate the power drive system.	
SR-128	**2.14.11 Accommodate accessories**	
SR-129	The car shall accommodate the following accessories Number plates, Tax discs, warning triangle, first aid kit, emergency tools, engine identification.	User Requrements: UR114 3.2.6.0-2 User Requrements: UR113 3.2.6.0-1 User Requrements: UR361 3.1.9.0-1
SR-130	**3 System constraints**	
SR-131	**3.1 Reliability**	
SR-132	The car shall have a reliability of 99.99% during operation.	
SR-133	**3.2 Modularity**	
SR-134	The car shall be assembled from modules by 1 person in 1 working day.	
SR-135	**3.3 Failure modes**	
SR-136	To be defined.	
SR-137	**3.4 Fuel efficiency**	
SR-138	The car shall be able to travel at 80 kph with a passenger load of 90 kilograms and luggage of 0 kilograms on standard flat roads with a wind speed of 0 kph at a maximum fuel cost of 1.0 pence per kilometer at prices of 1 January 1994.	
SR-139	The car shall be able to travel at up to 200 kph with a passenger load of up to 260 kilograms and luggage of up to 200 kilograms on standard flat roads with a wind speed of 0 kph at a maximum fuel cost of 4 pence per kilometer at prices of 1 January 1994.	
SR-140	**3.5 Fuel input mechanism**	
SR-141	The car shall be compatible with all standard fuel supply mechanisms in the countries to which it will be sold.	
SR-142	**3.6 Braking**	
SR-143	The car shall be fitted with ABS.	
SR-144	**3.7 Steer car**	

Username: Eric McCall   Exclusive edit mode

Figure 15.2: IBM Rational DOORS (2007)

The user interface of DOORS is similar to RequisitePro and the Windows Explorer from Microsoft (see Figure 15.2. The organization of projects is realised by folder modules (collection of objects), whereas requirements are mainly specified as hierarchically structured, textual descriptions (properties). Requirement descriptions are basically comprised of a title ("Object Heading"), a text ("Object Text") and a short text ("Object Short Text"). Attributes are here displayed in a dialogue box or in table form. Particularly in terms of attributes, the dependency relationships or the views, DOORS offers great flexibility. Rational DOORS is integrated with Rational solutions for change management to provide support for a change control process and to allow for automatic notifications of such changesŮby graphically indicating those links to changed objects that require investigation. Furthermore, DOORS allows for traceability views across requirement specifications, designs and tests. Additionally, it provides its own scripting language DXL's (DOORS Extension Language) for specifying automatic routines, calculations, triggers, or user-defined menu options. In this work, we evaluated version 9.1. For detailed information about DOORS refer to [36] and [118].

15.2.4 Conclusion

DOORS is a universal RE development tool. Typical RE Tasks are fundamentally supported. Due to its clearly structured user interface, working with DOORS is much easier, but requires due to its extensive functionality some training. Particularly noteworthy is the simple and flexible creating of attributes. In addition, it provides an extensive DXL programming interface whose application cannot be completely estimated in this work. DOORS supports traceability across multiple requirement documents as well as to designs,

tests and work items managed in other tools. Additionally, it allows for an automatic notification of changes by graphically indicating those links to changed objects that require investigation.
However, DOORS does not support types of requirements and only few features (e.g., attributes) are already predefined. Thus, there is no sufficient metamodel provided. Additional functions, such as UML modelling are not supported. DOORS does not provide validation or measurements for completeness, consistency or quality of requirements a priority. But it might be possible to use the DXL to allow for the definition of certain basic measurements. We summarize the following positive and negative aspects of Rational DOORS:

Positive

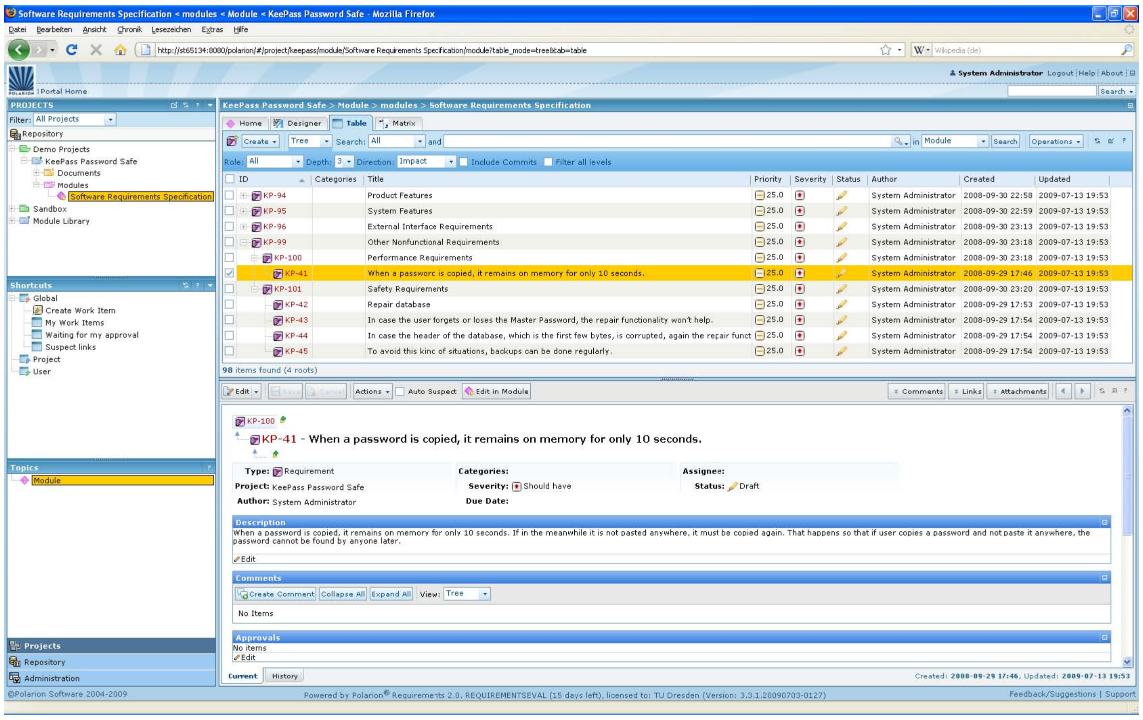
- Traceability across multiple requirement documents as well as to designs, tests and work items
- Automatic notification of changes
- Clear user interface
- DOORS Extension Language (DXL)
- Simple and flexible creation of attributes

Negative

- No predefined metamodel
- Requirement types are not supported cannot be defined by the user
- No validation support for completeness, consistency or quality of captured requirements
- Cumbersome installation
- High learning curve
- Poor integration of external documents
- Generation of reports
- Additional programs needed

15.2.5 Polarion® Requirements 2.0™

Polarion Requirements 2.0 was developed by the Swiss company Polarion Software, which operates on the market since 2005 and specialized on business applications for the Web 2.0. Polarion is a web-based RE development tool (Portal37), which is mainly based on the Java technology, open source applications and Apache Subversion (SVN) touches. The data to be managed is stored in form of XML documents within the repository. The user-interface of Polarion Requirements is divided into a navigation pane with different perspectives ("*Projects*", "*Repository*", "*Administration*") and a work area (see Figure 15.3) and is similar to the appearance of Microsoft Office Outlook. In Polarion Requirements, objects



KeePass Password Safe > Module > modules > Software Requirements Specification

ID	Categories	Title	Priority	Severity	Status	Author	Created	Updated
KP-94		Product Features	25.0			System Administrator	2008-09-30 22:58	2009-07-13 19:53
KP-95		System Features	25.0			System Administrator	2008-09-30 22:59	2009-07-13 19:53
KP-96		External Interface Requirements	25.0			System Administrator	2008-09-30 23:13	2009-07-13 19:53
KP-99		Other Nonfunctional Requirements	25.0			System Administrator	2008-09-30 23:18	2009-07-13 19:53
KP-100		Performance Requirements	25.0			System Administrator	2008-09-30 23:18	2009-07-13 19:53
KP-41		When a passworc is copied, it remains on memory for only 10 seconds.	25.0			System Administrator	2008-09-29 17:46	2009-07-13 19:53
KP-101		Safety Requirements	25.0			System Administrator	2008-09-30 23:20	2009-07-13 19:53
KP-42		Repair database	25.0			System Administrator	2008-09-29 17:53	2009-07-13 19:53
KP-43		In case the user forgets or loses the Master Password, the repair functionality won't help.	25.0			System Administrator	2008-09-29 17:54	2009-07-13 19:53
KP-44		In case the header of the database, which is the first few bytes, is corrupted, again the repair funct	25.0			System Administrator	2008-09-29 17:54	2009-07-13 19:53
KP-45		To avoid this kinc of situations, backups can be done regularly.	25.0			System Administrator	2008-09-29 17:54	2009-07-13 19:53

98 items found (4 roots)

KP-100

KP-41 - When a password is copied, it remains on memory for only 10 seconds.

Type: Requirement
Project: KeePass Password Safe
Author: System Administrator
Categories:
Severity: Should have
Due Date:
Assignee:
Status: Draft

Description
When a password is copied, it remains on memory for only 10 seconds. If in the meanwhile it is not pasted anywhere, it must be copied again. That happens so that if user copies a password and not paste it anywhere, the password cannot be found by anyone later.

Comments
No Items

Approvals
No items

Created: 2008-09-29 17:46, Updated: 2009-07-13 19:53

Powered by Polarion® Requirements 2.0, REQUIREMENTSEVAL (15 days left), licensed to: TU Dresden (Version: 3.3.1.20090703-0127)

Figure 15.3: Polarion (2007)

are referred to as *units of work* ("work items"), which different groups are assigned to (e.g., "Time Points", "Categories", "Queries", "Linking"). Folders and modules (collection of work units) are used for the organisation of projects. Different types ("Work Item Types") combine certain units of work and classify them in their function, such as a *Requirement*, *Task* or *Test Case*. Requirements are hierarchically structured, and may contain many predefined attributes and may be displayed in different ways, for instance as a table or in a Wiki page. Here, the Wiki refers to the repository, so that the Wiki pages always represent the current status of the project. Polarion powerfully supports change management and traceability. Polario allows for the definition of any requirements artefact (e.g. goal) and attributes and links between requirements artefacts. Thus, it is possible to define an own metamodel.
In this work, we evaluated version 3.3.1 in conjunction with the Mozilla browsers Firefox 3.0.13 and Microsoft Internet Explorer 8.0 and Microsoft Office 2007. It shall be noted, that Polarion Requirements is available both as individual tool and also as an integrated part of the larger software product PolarionALM.

15.2.6 Conclusion

Polarion Requirements is also a universal RE-development tool. It is very comprehensive in functionality and supports the typical RE tasks. As a web application the distributed work is completely supported by default, no additional software is necessary. Especially usable is working with Wiki pages. They allow for the simple and collaborative specification of many requirements at the beginning of the RE process. Later on, individual work units may be extracted from the Wiki pages and managed by Polarion Requirements. Additionally, due to the embedded query they may also be used as progress control. Furthermore, Polarion Requirements predefines several characteristica (e.g., attributes) that may be seen as a simple metamodel. However, this metamodel may be sufficiently extended by the user in various ways. Thus, it offers a lot of individual settings via XML configuration files, whereas the documentation however could be better. Also the navigation to the various perspectives, levels (module, project, repository) and views requires some time. However, some useful features are only available with a license for PolarionALM, which is highly recommended. Polarion does not provide any validation or measurement regarding the completeness, consistency or quality of requirements a priori. However, due to the underlying XML technology and the generation of Wiki pages, it might be possible to implement user-defined queries and measurements. In the following we briefly summarize some positive and negative characteristics of Polarion:

Positive

- Simple installation
- Many predefined characteristics (work units, attributes, traceability types)
- Configurability of attributes, work units and processes
- User definable requirements artefacts (e.g., requirement types, goals, relations, attributes)
- Wiki pages with embedded repository queries
- Storage of data in XML format
- Report generation using XSL templates
- Reuse of work units and modules
- Web-based collaboration support

Negative

- No validation support for completeness, consistency or quality of captured requirements a priori.
- High learning curve
- Lots of (administrative) settings through adjustment of XML configuration files
- Frequent switching between views, levels (module, project, repository) necessary
- Document and folder management partially only possible via repository
- Update of the display partially manually

15.3 Comparison of Features of RE Tools and OntoReq

The following tables 15.1 to 15.4 compare the most important classification criteria among three industrial tools (IBM Rational RequisitePro, IBM Rational DOORS and Polarion Requirements) and the OntoReq demonstrator. We use the scale positive (+), neutral (0), negative (-) and without evaluation (x).

Feature	RequisitePro	DOORS	Polarion Requirements	OntoReq
Requirements capture				
... through natural language text	+	+	+	+
... by tables	+	+	+	-
... by diagrams	+	0	0	-
... by graphics	+	-	+	-
... by models, notation languages	-	-	-	+
Attribute assignment				
... by predefined attributes	+	0	+	+
... by individual attributes	+	+	+	+
... automatically	-	+	0	+
... manually	+	+	+	+
Import of requirements from documents	-	-	0	-
Creation of glossaries, request templates, data dictionaries	0	0	0	-
Special consideration of non-functional requirements	0	-	-	+
Metamodel for requirements				
... as predefined classification	+	-	0	+
... by user classification	+	0	+	-
Structuring of requirements/ information	+	+	+	+
Creation of solutions approaches	0	0	0	+
To be continued on the following page.				

Table 15.1: Comparison of RE tools and OntoReq

Feature	**RequisitePro**	**DOORS**	**Polarion Requirements**	**OntoReq**
Providing templates				
... for documents	+	+	0	-
... Projects	+	+	0	-
Collaboration				
... through automatic user notification	+	+	+	-
... by parallel use of the tool	+	+	0	-
... by distributed use of the tool	+	0	+	-
... by commenting or review of data	0	+	+	+
Validation of completeness, consistency, quality	-	0	0	+
Clarity of the user interface	0	0	-	0
... Comprehensibility of the tool	+	+	0	+
... Learnability of the tool	+	0	0	+
... Support of labor and resource planning	0	0	0	-
... Definition of project automatisms	0	+	+	-
... Support for project monitoring	-	0	+	+
... Project-specific analysis options	-	-	-	+
Generation				
... Queries	+	+	+	0
... views	0	0	+	0
... Tables, charts, graphs, calculations, charts	0	0	0	+
... Documents, reports	-	-	+	+
To be continued on the following page.				

Table 15.2: Comparison of RE tools and OntoReq

Feature	RequisitePro	DOORS	Polarion Requirements	OntoReq
Reuse of requirements				
... of previous projects	-	+	0	0
... of a project-independent requirements archive	-	-	-	0
... prescribed methods support	-	-	-	0
... Support of a formal change process	0	+	0	0
... Documentation/ traceability of changes	0	+	+	+
... Automatic notification on potential effects of changes	0	0	0	0
... Traceability of requirements	0	+	+	+
... Configuration of requirements	-	-	-	+
... Support of a formal configuration management	0	0	-	+
... Allocation of priorities, rank and sequences	+	+	+	+
... Compilation of requirements configuration for later analysis	-	-	-	+
... Progress control	+	+	+	+
... Query language for database/ repository	RequisitePro Query Language	DXL	Apache Lucene	SPARQL
Operating Systems supported	Windows	Windows, Linux, UNIX	Windows, Linux	Windows, Linux
Necessary additional software	-	0	0	-
Data exchange ...				
... between different projects	-	+	+	-
... with other tools	0	0	0	0

To be continued on the following page.

Table 15.3: Comparison of RE tools and OntoReq

Feature	RequisitePro	DOORS	Polarion Requirements	OntoReq
File formats supported for import	Word, CSV	Word, Text, RTF, CSV, TSV, FrameMaker	Word, Excel	RDF, XML
File formats for export	Word, CSV, XML	Word, Excel, Text, Outlook, PowerPoint, RTF, HTML, FrameMaker	Word, Excel, Text, CSV, PDF, XML, XMLHTML	RDF, XML
Installation effort	-	-	+	+
Documentation (manual, online)	+	+	-	+
Potential cost	-	-	0	x

Table 15.4: Comparison of RE tools and OntoReq

15.4 Summary

In this chapter, we compare features of three well-known RE tools (IBM Rational RequisitePro, IBM DOORS and Polarion) with OntoReq. We describe their benefits, main features and weaknesses. The comparison is based on a classification scheme that acknowledges various aspects of software for RE. This tool evaluation shows, that metamodels are only little provided and supported. In contrast to OntoReq none of the evaluated tools provide validation for completeness, consistency or quality.

16 Conclusions and Future Work

This thesis is concerned with the investigation of problems and shortcomings in Requirements Engineering and the development of ontology-driven methods for improving the completeness, consistency and quality of requirements knowledge. Section 16.1 revisits the research goals planned in Chapter 1 and summarizes the main results of this thesis. In addition, Section 16.2 briefly describes the contributions of this thesis to Goal Oriented Requirements Engineering (GORE). Finally, Section 16.3 clarifies the limitation of our work and Section 16.4 describes future work.

16.1 Confirmation of Theses

The work presented in this thesis has been motivated by two main movements: goal-driven approaches in Requirements Engineering and ontologies in general. Within this thesis, the work is based on three hypotheses (see Section 1.3):

1. The specification of goals and their relation to requirements artefacts supports efforts in improving the completeness, consistency and quality of requirements knowledge.
2. The formaliation of requirements knowledge allows for automated validation of requirements knowledge.
3. Ontologies provide means to structure and reason about requirements knowledge, facilitate traceability and enable the automated validation of completeness, consistency and quality criteria captured within.

With respect to Hypothesis 1, the work in this thesis is based on Goal Oriented Requirements Engineering (GORE). "Goals are declarative statements of intent to be achieved by the system under consideration" [129]. Goal identification is accomplished preliminary to requirements identification, but requirements may be elaborated at the same time and, thus, more goals may be identified when discussing requirements. This way, the specification of goals drives the identification of requirements since a goal should be satisfied by at least one requirement [145]. In contrast to requirements that are changing over time, goals capture stable information. Hence, if goals are not related to any requirement, it can be assumed that the requirements specification is incomplete and that requirements are missing. Furthermore, the specification of goals and goal contribution relations between requirements and goals allow for the identification of a set of requirements that best satisfy a given set of goals. Thus, goals and goal contribution relations are capable of improving the quality of requirements knowledge. Therefore, we decided to base our research on GORE.

Regarding Hypothesises 2 and 3, ontologies are capable of representing and interrelating various knowledge. Since RE involves knowledge capturing and analysis, there is a clear

synergy between the ontological modelling of a domain and the modelling of requirements knowledge [34]. Requirements knowledge always needs a particular conceptualization. Thus, they have much in common with ontologies that are constructed by using a formal language [34] and provide a shared conceptualization.
These hypothesises have been approved in the thesis. We modelled a Requirements Ontology (TBox) that serves as requirements metamodel. The ontology elements (e.g., classes, properties, instances of classes, relationships between instances) can be used to specify formalized requirements artefacts and their relationships in the ontology ABox. This ontology has been successfully evaluated regarding the aspect whether it is capable of capturing the requirements knowledge contained in a real Software Requirements Specification (SRS).
Furthermore, Ontology reasoning was applied to automatically identify incomplete and inconsistent information as well as quality flaws and to provide the requirements engineer with suggestions on how to resolve these problems. This automatic validation has also been proven feasible and lead to the desired and correct results.

16.2 Thesis Contributions

The following subsections summarize the main contributions of our research in chronological order as they appear in the thesis. However, the two main results of the thesis are:

1. Conceptualisation of requirements knowledge, facilitated by ontologies.
2. Automated validation and measurement of requirements knowledge regarding the internal completeness, consistency and quality.

Classification of Problems in RE and Deduction of Requirements for RE

In this work, we evaluated problems and shortcomings in RE that have been published in research articles or documented by industry. We developed a classification of these problems and distinguished six problem categories:

1. **Technological**: problems based on changes or challenges in general technology and/or RE, e.g. tools, IT technology, etc.
2. **Organisation/Management/Communication**: shortcomings due to business (process) organisation, management of resources (e.g. time, money, human resources), communication between stakeholders, customers and requirements engineers
3. **Requirements**: problems originating from the nature of requirements themselves
4. **Requirements Engineering Process**: challenges regarding the process of requirements elicitation, analysis, etc., structure of the process, methods, traceability and guidance
5. **Requirements Documentation**: problems with the type of documentation, documentation structure, traceability, models, reuse, comprehensibility, etc.
6. **Validation/Verification**: challenges regarding checking specified requirements *during* the RE process

Based on these categories and the analysed problems related to them, we deduced a set of general requirements for Requirements Engineering that may be interpreted for various aspects (e.g., developing a new RE method, improving a RE tool). A subset of these requirements has been chosen to be considered for the approach developed in this thesis.

Ontology Metamodel

We developed a requirements metamodel as ontology TBox. This Requirements Ontology consists of 54 classes, 25 object properties and 3 data properties to interrelate the requirements knowledge and acknowledges goal-driven methods. Thus, it does not only allow for specifying goals and requirements, but also provides sufficient structures to interrelate them in various ways, e.g., for establishing goal contribution relations. The Requirements Ontology comprises several requirements artefacts (requirements, goals, use-cases, metrics, stakeholder, etc.) and a set of requirements relationships that enable their interrelation. These *metadata* can be understood as empty slots in the TBox that must be filled (instantiated) during the Requirements Engineering process. This instantiation forms the ABox of the ontology. The Requirements Ontology allows for a sufficient structure of requirements knowledge and is reusable. Furthermore, it facilitates the traceability of requirements knowledge (e.g., decisions for refinements or conflict reasons).
The Requirements Ontology and the contained metadata are the basis for the automated validation of the requirements knowledge specified within.

Automatic validation of completeness, consistency and quality flaws in requirements knowledge

Based on Davis' quality criteria for RE [32] and important requirements metadata defined by Firesmith et. al. [42], we defined a set of metadata that is measuring the completeness, consistency of the requirements knowledge and facilitates the quality improvement of requirements knowledge. These metadata have been used to define a number of completeness, consistency and quality rules. In order to support the requirements engineer, we developed a software demonstration (OntoReq) that allows for the automatic detection of incompleteness, inconsistency or quality problems. Therefore, the Requirements Ontology with the instantiated requirements knowledge is reasoned and the satisfiability of the validation rules is checked. The validation rules have been implemented by Java methods and SPARQL queries and use NBox reasoning (Local Closed World Reasoning with Negation as Failure Box, [114]).

Guidance Support for Resolving Incompleteness, Inconsistencies and Quality Flaws in Requirements Knowledge

In order to support the requirements engineer in resolving the identified completeness, consistency and quality problems, we developed decision strategies that allow for automatically presenting concrete solution suggestion particular to the requirements knowledge at the time of validation. These decisions are evaluated in OntoReq in the background, for example, the recommendation of alternative requirements or the exclusion of optional requirements. The decisions satisfy a set of validation rules themselves, since a presented suggestion must also guarantee to be compliant to the validation rules, e.g. not establishing new conflicts.

Measuring Completeness, Consistency and Quality of Requirements Knowledge

The above validation rules involve the requirements engineer in improving the SRS by pointing to the specific problems. Although these validation rules help to exactly identify where improvements are necessary and provide solution suggestions, the requirements engineer will not exactly know how important the modifications were for the overall quality of the SRS in contrast to any state of the requirements knowledge before. Therefore, we provide measurements for the quality of the SRS, based on Davis' criteria and formulas in [32] and additional specified criteria.
Thus, we provide formulas for the quality criteria "internal completeness", "correctness", "verifiability", "internal consistency", "traceability" and "uncriticality" that compute the percentage of the particular criteria. These measurements can be executed in OntoReq. The requirements engineer is also equipped with hints on how to improve the particular criteria if necessary. These measurements can be executed after each validation and state the quality of the actual requirements knowledge and represent the satisfaction of each quality criteria. Thus, enhancement or degeneration of the requirements knowledge becomes visible to the requirements engineer in the form of numerical values that allow for an objective comparison during different stages of development.

Conceptual Description of Ontology-Driven Process Guidance for OntoReq

Furthermore, we developed an ontology-driven method to guide the requirements engineer through the process of GORE. Therefore, we defined a set of optional and mandatory tasks, based on our validation rules. To allow reasoning about the current state of the requirements knowledge and to identify completed and incompleted process steps, we defined pre- and post-conditions for each task. Thus, a task is completed if all the post-conditions are satisfied. Since Requirements Engineering is an ongoing process, the use of these post-conditions allows for setting an already completed task back to the status "incomplete" if one or more of the conditions are not satisfied anymore.
We distinguish step guidance and flow guidance. Step guidance offers guidelines for each task that explains how to accomplish this particular task and flow guidance guides the requirements engineer in choosing the next task. These tasks and their pre- and postconditions must be kept in a Guidance Ontology that is linked to the Requirements Ontology. Therefore, we defined the main ontology components for the Guidance Ontology. Finally, reasoning allows for the computation of a set of open tasks the requirements engineer can either follow chronological or flexibly choose from. However, our guidance approach guarantees that the specified requirements knowledge satisfies all validation rules if each of the open tasks has been successfully accomplished.

Software Demonstrator OntoReq and GUI Prototype

In order to demonstrate our approach, we developed OntoReq as a software demonstration. Except for the process guidance, it allows for the execution all of the previously described research results. Thus, OntoReq automatically identifies incomplete and inconsistent requirements knowledge and quality flaws. It points directly to the source of a particular problem and displays concrete solution suggestions for each detected fault. Furthermore, it measures several quality criteria and provides hints on how to improve them if necessary. OntoReq has been developed as Eclipse Plugin [37] and uses the OWL API framework

[53], Jena Reasoner [44] and TrOWL Reasoner [3]. Furthermore, we use closed world reasoning with negation NBox reasoning to allow the identification of missing information in the Requirements Ontology. SPARQL [112] is used to query the requirements knowledge within.
A prototypic user interface demonstrates the modification of requirements knowledge (adding requirements, artefacts, changing requirements knowledge, etc.) without the need to directly manipulate the ontology model. The requirements engineer is presented with a RE specific user interface, which itself provides means to formalize the requirements knowledge and to improve the completeness and consistency of the requirements knowledge. Therefore, we propose appropriate fields for the relevant metadata, depending on the particular requirements artefact that is modified.

General Guidelines for improving completeness, consistency and quality in domain ontologies

Increasing the completeness, consistency and quality of knowledge is not only crucial to Requirements Engineering but to any kind of knowledge repository. Therefore, we generalized our approach and described the main steps to validate these criteria for any knowledge domain (e.g., medicine, architecture) whose data is captured in an ontology. We use domain-independent guidelines to sketch fundamental tasks such as building the ontology knowledge repository, evaluation and implementation of an application. Code snippets illustrate the main implementation aspects.

16.3 Limitations

In this thesis, we aimed to improve the *internal* completeness, consistency and quality of the requirements knowledge by applying ontology techniques. Thus, the appropriate analyses and validations are performed with respect to the ABox of the Requirements Ontology and its metadata. The completeness validation checks the completeness of requirements metadata, instead of making assumptions about whether requirements are missing in the SRS. Additionally, we enable the validation of the consistency of the requirements knowledge in a requirements *configuration*. Furthermore, it was not our aim to make any assumptions about the *external* completeness, consistency or quality of a SRS (e.g. whether a use-case has been described properly, readability of requirements, etc.).
OntoReq supports a part of the RE process where problems and shortcomings were stated popular and of significant negative effect. However, the Requirements Engineering process needs further inspection and probably support to address other known problems and to include various models, such as use-case or test-case models.
Although we provide sufficient data structures and information to support any effort in realising seamless traceability, we did not aim to provide seamless traceability of requirements for the whole Software Development process, but the Requirements Engineering process.

16.4 Future Work

The guidance support presented in this thesis comprises the automated validation of the requirements knowledge regarding completeness, consistency and quality as well as the suggestion of several solutions according to the present requirements knowledge. We furthermore describe a conceptual solution to guide the GORE process by facilitating ontology techniques. Besides the implementation of this concept, further enhancements may be interesting to guide the requirements engineer in GORE. Kavakli in [73] developed a systematic way of reasoning about the RE process in terms of goal modelling and supports the user by a process guidance framework. G-Map is a web-based software tool that supports the assembling and execution of goal-driven methods. A methodology roadmap is used as a metaphor to visualize alternative goal-driven ways of working [73]. However, G-Map only guides through the tasks of different GORE methods but is not connected to real requirements knowledge. Therefore, one possible enhancement of OntoReq would be to provide the process guidance in a similar way as described by Kavakli, with the difference that the graphical representation, navigation and display of tasks and guidelines is connected to the Requirements Ontology and, thus, allows for real guidance particular to the existing requirements knowledge.

The goal satisfaction relations we provide in the Requirements Ontology only distinguish between positive or negative goal contribution. However, it might be applicable to increase these relations by providing several goal contributions levels, e.g. weak, medium, strong. Other goal satisfaction approaches propose complex mathematical formulas and objective criteria to define the goal contribution (e.g., [89]). The Requirements Ontology may be extended by modelling appropriate ontology components and relations to allow for such complex goal contributions. Subsequently, OntoReq may be modified in a way that decisions regarding goal contribution become more precise and the selection of the most suitable requirements regarding a set of goals becomes possible.

All available guidelines and standards state the need that requirements must be unambiguous. This means that they must be described in a way that prevents misunderstanding or wrong interpretations. Such pitfalls are often specific to the use of (natural) language for description and can actually be easily corrected if identified. Some typical examples for ambiguous requirements descriptions are: (a) "The system shall respond in less than 2s *if possible*." and (b) "The user must be able to input *some* data.". Corrected, these requirements could for example be stated in the following ambiguous way: (a) The system must respond in less than 2s." , (b) "The user must be able to input address data.".
Our approach may be extended by providing validation mechanisms and measurement for this unambiguity and concrete solution suggestions that point to sources of ambiguity.

According to Parreiras et. al., UML and OWL comprise some constituents which are similar in many respects, like: classes, associations, properties, packages, types, generalization and instances [107]. However, both approaches have their advantages and disadvantages. While UML provides means to express dynamic behaviour, OWL does not. In contrast, OWL is capable of inferring the specified knowledge while UML does not allow for any kind of inference per se. Thus, a variety of approaches propose the transformation of ontologies into UML models and vice versa (e.g., [107]) in order to bridge both technologies. Especially for Requirements Engineering it might therefore be profitable to enable such model transformations. Thus, it would be interesting to allow a transformation of the Requirements Ontology and the data captured within into a UML metamodel for further specification of, for example, dynamic behaviour.

Finally, requirements knowledge often lacks a graphical representation that facilitates the comprehensibility of the knowledge. Since ontologies provide means to structure knowledge and allow for graphical visualizations (e.g., [81]), Requirements Engineering could profit from such a visualization and an improved navigation through particular complex knowledge structures with many interrelations.

Appendix

A.1 Requirements Ontology RDF Data Model

```


<?xml version="1.0"?>


<!DOCTYPE rdf:RDF [
    <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
    <!ENTITY swrl "http://www.w3.org/2003/11/swrl#" >
    <!ENTITY swrlb "http://www.w3.org/2003/11/swrlb#" >
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY owl2xml "http://www.w3.org/2006/12/owl2-xml#" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
    <!ENTITY test_ontology "http://www.semanticweb.org/ontologies/2012/4/
        test_ontology.owl#" >
]>


<rdf:RDF xmlns="http://www.semanticweb.org/ontologies/2012/4/test_ontology.
    owl#"
     xml:base="http://www.semanticweb.org/ontologies/2012/4/test_ontology.
         owl"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:swrl="http://www.w3.org/2003/11/swrl#"
     xmlns:owl2xml="http://www.w3.org/2006/12/owl2-xml#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:swrlb="http://www.w3.org/2003/11/swrlb#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:test_ontology="http://www.semanticweb.org/ontologies/2012/4/
         test_ontology.owl#">
    <owl:Ontology rdf:about=""/>

    <!--
    ///////////////////////////////////////
    //
    // Object Properties
    //
    ///////////////////////////////////////
     -->

    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        NegativeContributedBy -->

    <owl:ObjectProperty rdf:about="#NegativeContributedBy">
        <rdfs:domain rdf:resource="#Goal"/>
        <rdfs:range rdf:resource="#Requirement"/>
    </owl:ObjectProperty>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        PositiveContributedBy -->

    <owl:ObjectProperty rdf:about="#PositiveContributedBy">
        <rdfs:domain rdf:resource="#Goal"/>
        <rdfs:range rdf:resource="#Requirement"/>
    </owl:ObjectProperty>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        describesRequirement -->

    <owl:ObjectProperty rdf:about="#describesRequirement">
        <rdfs:range rdf:resource="#Requirement"/>
        <rdfs:domain rdf:resource="#Story"/>
    </owl:ObjectProperty>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        excludes -->

    <owl:ObjectProperty rdf:about="#excludes">
        <owl:inverseOf rdf:resource="#isExcludedBy"/>
    </owl:ObjectProperty>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        hasGoal -->

    <owl:ObjectProperty rdf:about="#hasGoal">
        <rdfs:range rdf:resource="#Goal"/>
        <rdfs:domain rdf:resource="#Requirement"/>
        <owl:propertyChainAxiom rdf:parseType="Collection">
            <rdf:Description rdf:about="#isAlternativeTo"/>
            <rdf:Description rdf:about="#hasGoal"/>
        </owl:propertyChainAxiom>
        <owl:propertyChainAxiom rdf:parseType="Collection">
            <rdf:Description rdf:about="#isRefinementOf"/>
            <rdf:Description rdf:about="#hasGoal"/>
        </owl:propertyChainAxiom>
    </owl:ObjectProperty>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        hasObstacle -->

    <owl:ObjectProperty rdf:about="#hasObstacle">
        <rdfs:range rdf:resource="#Obstacle"/>
        <rdfs:domain rdf:resource="#Requirement"/>
    </owl:ObjectProperty>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        hasRefinementReason -->

    <owl:ObjectProperty rdf:about="#hasRefinementReason">
        <rdfs:domain rdf:resource="#Refinement"/>
        <rdfs:range rdf:resource="#RefinementReason"/>
    </owl:ObjectProperty>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        hasRefinementSource -->

    <owl:ObjectProperty rdf:about="#hasRefinementSource">
        <rdfs:domain rdf:resource="#Refinement"/>
        <rdfs:range rdf:resource="#Requirement"/>
    </owl:ObjectProperty>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        hasRefinementTarget -->

    <owl:ObjectProperty rdf:about="#hasRefinementTarget">
        <rdfs:domain rdf:resource="#Refinement"/>
        <rdfs:range rdf:resource="#Requirement"/>
    </owl:ObjectProperty>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        hasScenario -->

    <owl:ObjectProperty rdf:about="#hasScenario">
        <rdfs:domain rdf:resource="#Requirement"/>
    </owl:ObjectProperty>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        hasSoftMetric -->

    <owl:ObjectProperty rdf:about="#hasSoftMetric">
        <rdfs:domain rdf:resource="#Requirement"/>
        <rdfs:range rdf:resource="#SoftMetric"/>
    </owl:ObjectProperty>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        hasSource -->

    <owl:ObjectProperty rdf:about="#hasSource">
        <rdfs:range rdf:resource="#Source"/>
        <rdfs:domain rdf:resource="&owl;Thing"/>
    </owl:ObjectProperty>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        hasTrigger -->

    <owl:ObjectProperty rdf:about="#hasTrigger">
        <rdfs:domain rdf:resource="#Requirement"/>
        <rdfs:range rdf:resource="#Trigger"/>
    </owl:ObjectProperty>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        isAlternativeOf -->

    <owl:ObjectProperty rdf:about="#isAlternativeOf">
        <owl:inverseOf rdf:resource="#isAlternativeTo"/>
    </owl:ObjectProperty>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        isAlternativeTo -->

    <owl:ObjectProperty rdf:about="#isAlternativeTo">
        <rdf:type rdf:resource="&owl;SymmetricProperty"/>
    </owl:ObjectProperty>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        isAuthoredBy -->

    <owl:ObjectProperty rdf:about="#isAuthoredBy">
        <rdfs:range rdf:resource="#Stakeholder"/>
    </owl:ObjectProperty>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        isCoexistentWith -->

    <owl:ObjectProperty rdf:about="#isCoexistentWith">
        <rdf:type rdf:resource="&owl;SymmetricProperty"/>
    </owl:ObjectProperty>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        isConnectedWithTestCase -->

    <owl:ObjectProperty rdf:about="#isConnectedWithTestCase">
        <rdfs:domain rdf:resource="#Requirement"/>
        <rdfs:range rdf:resource="#TestCase"/>
        <owl:propertyChainAxiom rdf:parseType="Collection">
            <rdf:Description rdf:about="#isRefinementOf"/>
            <rdf:Description rdf:about="#isConnectedWithTestCase"/>
        </owl:propertyChainAxiom>
    </owl:ObjectProperty>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        isConnectedWithUseCase -->

    <owl:ObjectProperty rdf:about="#isConnectedWithUseCase">
        <rdfs:domain rdf:resource="#Requirement"/>
        <rdfs:range rdf:resource="#UseCase"/>
        <owl:propertyChainAxiom rdf:parseType="Collection">
            <rdf:Description rdf:about="#isRefinementOf"/>
            <rdf:Description rdf:about="#isConnectedWithUseCase"/>
        </owl:propertyChainAxiom>
    </owl:ObjectProperty>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        isExcludedBy -->

    <owl:ObjectProperty rdf:about="#isExcludedBy"/>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        isInConflictWith -->

    <owl:ObjectProperty rdf:about="#isInConflictWith">
        <rdf:type rdf:resource="&owl;SymmetricProperty"/>
    </owl:ObjectProperty>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        isNegativeContributionToGoal -->

    <owl:ObjectProperty rdf:about="#isNegativeContributionToGoal">
        <rdfs:range rdf:resource="#Goal"/>
        <owl:inverseOf rdf:resource="#NegativeContributedBy"/>
        <rdfs:domain rdf:resource="#Requirement"/>
    </owl:ObjectProperty>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        isPositiveContributionToGoal -->

    <owl:ObjectProperty rdf:about="#isPositiveContributionToGoal">
        <rdfs:range rdf:resource="#Goal"/>
        <owl:inverseOf rdf:resource="#PositiveContributedBy"/>
        <rdfs:domain rdf:resource="#Requirement"/>
    </owl:ObjectProperty>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        isRefinementOf -->

    <owl:ObjectProperty rdf:about="#isRefinementOf">
        <owl:inverseOf rdf:resource="#refinesTo"/>
    </owl:ObjectProperty>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        operationalisesTo -->

    <owl:ObjectProperty rdf:about="#operationalisesTo">
        <rdfs:range rdf:resource="#RequirementsArtefact"/>
        <rdfs:domain rdf:resource="#RequirementsArtefact"/>
    </owl:ObjectProperty>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        refinesTo -->

    <owl:ObjectProperty rdf:about="#refinesTo">
        <rdfs:domain rdf:resource="&owl;Thing"/>
        <rdfs:range rdf:resource="&owl;Thing"/>
    </owl:ObjectProperty>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        throwsErrorException -->

    <owl:ObjectProperty rdf:about="#throwsErrorException"/>


    <!--
    ///////////////////////////////////////////////////////////////////////////////////////
    //
    // Data properties
    //
    ///////////////////////////////////////////////////////////////////////////////////////
     -->


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        isMandatory -->

    <owl:DatatypeProperty rdf:about="#isMandatory">
        <rdfs:domain rdf:resource="#Requirement"/>
        <rdfs:range rdf:resource="&xsd;boolean"/>
    </owl:DatatypeProperty>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        isOptional -->

    <owl:DatatypeProperty rdf:about="#isOptional">
        <rdfs:domain rdf:resource="#Requirement"/>
    </owl:DatatypeProperty>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        isValid -->

    <owl:DatatypeProperty rdf:about="#isValid">
        <rdfs:domain rdf:resource="#Requirement"/>
        <rdfs:range rdf:resource="&xsd;boolean"/>
    </owl:DatatypeProperty>


     <!--
    ///////////////////////////////////////////////////////////////////////////////////////
    //
    // Classes
    //
    ///////////////////////////////////////////////////////////////////////////////////////
     -->


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        AccessibilityRequirement -->

    <owl:Class rdf:about="#AccessibilityRequirement">
        <rdfs:subClassOf rdf:resource="#NonFunctionalRequirement"/>
    </owl:Class>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        AlternativeRequirement -->

    <owl:Class rdf:about="#AlternativeRequirement">
        <owl:equivalentClass>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#isAlternativeTo"/>
                <owl:someValuesFrom rdf:resource="#Requirement"/>
            </owl:Restriction>
        </owl:equivalentClass>
        <rdfs:subClassOf rdf:resource="#Requirement"/>
    </owl:Class>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        Attribute -->

    <owl:Class rdf:about="#Attribute">
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#isAuthoredBy"/>
                <owl:someValuesFrom rdf:resource="#Stakeholder"/>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        BusinessGoal -->

    <owl:Class rdf:about="#BusinessGoal">
        <rdfs:subClassOf rdf:resource="#Goal"/>
    </owl:Class>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        CompleteRequirement -->

    <owl:Class rdf:about="#CompleteRequirement">
        <owl:equivalentClass>
            <owl:Class>
                <owl:intersectionOf rdf:parseType="Collection">
                    <rdf:Description rdf:about="#LevelOfPriority"/>
                    <rdf:Description rdf:about="#LevelOfRisk"/>
                    <owl:Restriction>
                        <owl:onProperty rdf:resource="#hasSource"/>
                        <owl:someValuesFrom rdf:resource="#Source"/>
                    </owl:Restriction>
                    <owl:Restriction>
                        <owl:onProperty rdf:resource="#isAuthoredBy"/>
                        <owl:someValuesFrom rdf:resource="#Stakeholder"/>
                    </owl:Restriction>
                    <owl:Restriction>
                        <owl:onProperty rdf:resource="#
                            isConnectedWithTestCase"/>
                        <owl:someValuesFrom rdf:resource="#TestCase"/>
                    </owl:Restriction>
                    <owl:Restriction>
                        <owl:onProperty rdf:resource="#
                            isConnectedWithUseCase"/>
                        <owl:someValuesFrom rdf:resource="#UseCase"/>
                    </owl:Restriction>
                </owl:intersectionOf>
            </owl:Class>
        </owl:equivalentClass>
        <rdfs:subClassOf rdf:resource="#Requirement"/>
    </owl:Class>



    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        ConflictingRequirement -->

    <owl:Class rdf:about="#ConflictingRequirement">
        <owl:equivalentClass>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#isInConflictWith"/>
                <owl:someValuesFrom rdf:resource="#Requirement"/>
            </owl:Restriction>
        </owl:equivalentClass>
        <rdfs:subClassOf rdf:resource="#Requirement"/>
    </owl:Class>



    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        EfficiencyRequirement -->

    <owl:Class rdf:about="#EfficiencyRequirement">
        <rdfs:subClassOf rdf:resource="#NonFunctionalRequirement"/>
    </owl:Class>



    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        ExcludedRequirement -->

    <owl:Class rdf:about="#ExcludedRequirement">
        <owl:equivalentClass>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#isExcludedBy"/>
                <owl:someValuesFrom rdf:resource="#Requirement"/>
            </owl:Restriction>
        </owl:equivalentClass>
        <rdfs:subClassOf rdf:resource="#Requirement"/>
    </owl:Class>



    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        ExcludingRequirement -->

    <owl:Class rdf:about="#ExcludingRequirement">
        <owl:equivalentClass>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#excludes"/>
                <owl:someValuesFrom rdf:resource="#Requirement"/>
            </owl:Restriction>
        </owl:equivalentClass>
        <rdfs:subClassOf rdf:resource="#Requirement"/>
    </owl:Class>



    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        FunctionalRequirement -->

    <owl:Class rdf:about="#FunctionalRequirement">
        <rdfs:subClassOf rdf:resource="#Requirement"/>
    </owl:Class>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        Goal -->

    <owl:Class rdf:about="#Goal">
        <rdfs:subClassOf rdf:resource="#RequirementsArtefact"/>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#refinesTo"/>
                <owl:someValuesFrom rdf:resource="#Goal"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#operationalisesTo"/>
                <owl:someValuesFrom rdf:resource="#RequirementsArtefact"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#operationalisesTo"/>
                <owl:someValuesFrom rdf:resource="#Requirement"/>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        HighCost -->

    <owl:Class rdf:about="#HighCost">
        <rdfs:subClassOf rdf:resource="#LevelOfCost"/>
    </owl:Class>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        HighPriority -->

    <owl:Class rdf:about="#HighPriority">
        <owl:equivalentClass rdf:resource="#ImportantRequirement"/>
        <rdfs:subClassOf rdf:resource="#LevelOfPriority"/>
    </owl:Class>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        HighRisk -->

    <owl:Class rdf:about="#HighRisk">
        <rdfs:subClassOf rdf:resource="#LevelOfRisk"/>
    </owl:Class>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        ImportantRequirement -->

    <owl:Class rdf:about="#ImportantRequirement">
        <owl:equivalentClass>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#isMandatory"/>
                <owl:hasValue rdf:datatype="&xsd;boolean">true</owl:
                   hasValue>
            </owl:Restriction>
        </owl:equivalentClass>
        <rdfs:subClassOf rdf:resource="#Requirement"/>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#operationalisesTo"/>
                <owl:someValuesFrom rdf:resource="#RequirementsArtefact"/>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>



    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        InconsistentRequirement -->

    <owl:Class rdf:about="#InconsistentRequirement">
        <owl:equivalentClass>
            <owl:Class>
                <owl:intersectionOf rdf:parseType="Collection">
                    <owl:Restriction>
                        <owl:onProperty rdf:resource="#isMandatory"/>
                        <owl:hasValue rdf:datatype="&xsd;boolean">true</owl
                           :hasValue>
                    </owl:Restriction>
                    <owl:Restriction>
                        <owl:onProperty rdf:resource="#isOptional"/>
                        <owl:hasValue rdf:datatype="&xsd;boolean">true</owl
                           :hasValue>
                    </owl:Restriction>
                </owl:intersectionOf>
            </owl:Class>
        </owl:equivalentClass>
        <rdfs:subClassOf rdf:resource="#Requirement"/>
    </owl:Class>



    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        LevelOfCost -->

    <owl:Class rdf:about="#LevelOfCost">
        <rdfs:subClassOf rdf:resource="#Attribute"/>
    </owl:Class>



    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        LevelOfPriority -->

    <owl:Class rdf:about="#LevelOfPriority">
        <rdfs:subClassOf rdf:resource="#Attribute"/>
    </owl:Class>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        LevelOfRisk -->

    <owl:Class rdf:about="#LevelOfRisk">
        <rdfs:subClassOf rdf:resource="#Attribute"/>
    </owl:Class>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        LowCost -->

    <owl:Class rdf:about="#LowCost">
        <rdfs:subClassOf rdf:resource="#LevelOfCost"/>
    </owl:Class>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        LowPriority -->

    <owl:Class rdf:about="#LowPriority">
        <rdfs:subClassOf rdf:resource="#LevelOfPriority"/>
    </owl:Class>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        LowRisk -->

    <owl:Class rdf:about="#LowRisk">
        <rdfs:subClassOf rdf:resource="#LevelOfRisk"/>
    </owl:Class>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        MediumCost -->

    <owl:Class rdf:about="#MediumCost">
        <rdfs:subClassOf rdf:resource="#LevelOfCost"/>
    </owl:Class>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        MediumPriority -->

    <owl:Class rdf:about="#MediumPriority">
        <rdfs:subClassOf rdf:resource="#LevelOfPriority"/>
    </owl:Class>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        MediumRisk -->

    <owl:Class rdf:about="#MediumRisk">
        <rdfs:subClassOf rdf:resource="#LevelOfRisk"/>
    </owl:Class>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        NegativeRequirement -->

    <owl:Class rdf:about="#NegativeRequirement">
        <owl:equivalentClass>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#isNegativeContributionToGoal
                    "/>
                <owl:someValuesFrom rdf:resource="#Goal"/>
            </owl:Restriction>
        </owl:equivalentClass>
        <rdfs:subClassOf rdf:resource="#Requirement"/>
    </owl:Class>



    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        NoAlternativeRequirement -->

    <owl:Class rdf:about="#NoAlternativeRequirement">
        <owl:equivalentClass>
            <owl:Class>
                <owl:unionOf rdf:parseType="Collection">
                    <rdf:Description rdf:about="#ConflictingRequirement"/>
                    <rdf:Description rdf:about="#ExcludedRequirement"/>
                    <rdf:Description rdf:about="#NegativeRequirement"/>
                </owl:unionOf>
            </owl:Class>
        </owl:equivalentClass>
        <rdfs:subClassOf rdf:resource="#Requirement"/>
    </owl:Class>



    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        NonFunctionalRequirement -->

    <owl:Class rdf:about="#NonFunctionalRequirement">
        <rdfs:subClassOf rdf:resource="#Requirement"/>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#hasSoftMetric"/>
                <owl:someValuesFrom rdf:resource="#SoftMetric"/>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>



    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        Obstacle -->

    <owl:Class rdf:about="#Obstacle">
        <rdfs:subClassOf rdf:resource="#RequirementsArtefact"/>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#operationalisesTo"/>
                <owl:someValuesFrom rdf:resource="#Requirement"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#refinesTo"/>
                <owl:someValuesFrom rdf:resource="#Obstacle"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#operationalisesTo"/>
                <owl:someValuesFrom rdf:resource="#RequirementsArtefact"/>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        PerformanceRequirement -->

    <owl:Class rdf:about="#PerformanceRequirement">
        <rdfs:subClassOf rdf:resource="#NonFunctionalRequirement"/>
    </owl:Class>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        PlatformRequirement -->

    <owl:Class rdf:about="#PlatformRequirement">
        <rdfs:subClassOf rdf:resource="#Requirement"/>
    </owl:Class>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        PositiveGoal -->

    <owl:Class rdf:about="#PositiveGoal">
        <rdfs:subClassOf rdf:resource="#Goal"/>
    </owl:Class>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        ProcessGoal -->

    <owl:Class rdf:about="#ProcessGoal">
        <rdfs:subClassOf rdf:resource="#Goal"/>
    </owl:Class>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        RefinedRequirement -->

    <owl:Class rdf:about="#RefinedRequirement">
        <owl:equivalentClass>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#isRefinementOf"/>
                <owl:someValuesFrom rdf:resource="#Requirement"/>
            </owl:Restriction>
        </owl:equivalentClass>
        <rdfs:subClassOf rdf:resource="#Requirement"/>
    </owl:Class>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        Refinement -->

    <owl:Class rdf:about="#Refinement">
        <rdfs:subClassOf rdf:resource="&owl;Thing"/>
    </owl:Class>



    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        RefinementReason -->

    <owl:Class rdf:about="#RefinementReason">
        <rdfs:subClassOf rdf:resource="&owl;Thing"/>
    </owl:Class>



    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        Requirement -->

    <owl:Class rdf:about="#Requirement">
        <rdfs:subClassOf rdf:resource="&owl;Thing"/>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#isConnectedWithTestCase"/>
                <owl:someValuesFrom rdf:resource="#TestCase"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#hasObstacle"/>
                <owl:someValuesFrom rdf:resource="#Obstacle"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#hasScenario"/>
                <owl:someValuesFrom rdf:resource="#Scenario"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#isAuthoredBy"/>
                <owl:someValuesFrom rdf:resource="#Stakeholder"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#isConnectedWithUseCase"/>
                <owl:someValuesFrom rdf:resource="#UseCase"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#isPositiveContributionToGoal
                    "/>
                <owl:someValuesFrom rdf:resource="#Goal"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#isValid"/>
                <owl:someValuesFrom rdf:resource="&xsd;boolean"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#hasSoftMetric"/>
                <owl:someValuesFrom rdf:resource="#SoftMetric"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#refinesTo"/>
                <owl:someValuesFrom rdf:resource="#Requirement"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#isMandatory"/>
                <owl:someValuesFrom rdf:resource="&xsd;boolean"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#hasSource"/>
                <owl:someValuesFrom rdf:resource="#Source"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#isNegativeContributionToGoal
                    "/>
                <owl:someValuesFrom rdf:resource="#Goal"/>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        RequirementsArtefact -->

    <owl:Class rdf:about="#RequirementsArtefact">
        <rdfs:subClassOf rdf:resource="&owl;Thing"/>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#isAuthoredBy"/>
                <owl:someValuesFrom rdf:resource="#Stakeholder"/>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        Scenario -->

    <owl:Class rdf:about="#Scenario">
        <rdfs:subClassOf rdf:resource="#Story"/>
    </owl:Class>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        SecurityRequirement -->

    <owl:Class rdf:about="#SecurityRequirement">
        <rdfs:subClassOf rdf:resource="#NonFunctionalRequirement"/>
    </owl:Class>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        SoftMetric -->

    <owl:Class rdf:about="#SoftMetric">
        <rdfs:subClassOf rdf:resource="#RequirementsArtefact"/>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#refinesTo"/>
                <owl:someValuesFrom rdf:resource="#SoftMetric"/>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        Source -->

    <owl:Class rdf:about="#Source">
        <rdfs:subClassOf rdf:resource="#RequirementsArtefact"/>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#refinesTo"/>
                <owl:someValuesFrom rdf:resource="#Source"/>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        Stakeholder -->

    <owl:Class rdf:about="#Stakeholder">
        <rdfs:subClassOf rdf:resource="&owl;Thing"/>
    </owl:Class>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        State -->

    <owl:Class rdf:about="#State">
        <rdfs:subClassOf rdf:resource="#Attribute"/>
    </owl:Class>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        Story -->

    <owl:Class rdf:about="#Story">
        <rdfs:subClassOf rdf:resource="#RequirementsArtefact"/>
        <rdfs:subClassOf>
            <owl:Restriction>
```

```
                <owl:onProperty rdf:resource="#hasSource"/>
                <owl:someValuesFrom rdf:resource="#Source"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#describesRequirement"/>
                <owl:someValuesFrom rdf:resource="#Requirement"/>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        SystemGoal -->

    <owl:Class rdf:about="#SystemGoal">
        <rdfs:subClassOf rdf:resource="#Goal"/>
    </owl:Class>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        TestCase -->

    <owl:Class rdf:about="#TestCase">
        <rdfs:subClassOf rdf:resource="#RequirementsArtefact"/>
    </owl:Class>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        TraceableRequirement -->

    <owl:Class rdf:about="#TraceableRequirement">
        <owl:equivalentClass>
            <owl:Class>
                <owl:intersectionOf rdf:parseType="Collection">
                    <owl:Restriction>
                        <owl:onProperty rdf:resource="#hasSource"/>
                        <owl:someValuesFrom rdf:resource="#Source"/>
                    </owl:Restriction>
                    <owl:Restriction>
                        <owl:onProperty rdf:resource="#isAuthoredBy"/>
                        <owl:someValuesFrom rdf:resource="#Stakeholder"/>
                    </owl:Restriction>
                    <owl:Restriction>
                        <owl:onProperty rdf:resource="#
                            isConnectedWithTestCase"/>
                        <owl:someValuesFrom rdf:resource="#TestCase"/>
                    </owl:Restriction>
                    <owl:Restriction>
                        <owl:onProperty rdf:resource="#
                            isConnectedWithUseCase"/>
                        <owl:someValuesFrom rdf:resource="#UseCase"/>
                    </owl:Restriction>
                </owl:intersectionOf>
            </owl:Class>
        </owl:equivalentClass>
        <rdfs:subClassOf rdf:resource="#Requirement"/>
    </owl:Class>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        Trigger -->

    <owl:Class rdf:about="#Trigger">
        <rdfs:subClassOf rdf:resource="#Attribute"/>
    </owl:Class>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        UsabilityRequirement -->

    <owl:Class rdf:about="#UsabilityRequirement">
        <rdfs:subClassOf rdf:resource="#NonFunctionalRequirement"/>
    </owl:Class>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        UseCase -->

    <owl:Class rdf:about="#UseCase">
        <rdfs:subClassOf rdf:resource="#Story"/>
    </owl:Class>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        ValidatedRequirement -->

    <owl:Class rdf:about="#ValidatedRequirement">
        <rdfs:subClassOf rdf:resource="#RequirementsArtefact"/>
    </owl:Class>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        VerifiableRequirement -->

    <owl:Class rdf:about="#VerifiableRequirement">
        <owl:equivalentClass>
            <owl:Class>
                <owl:unionOf rdf:parseType="Collection">
                    <owl:Restriction>
                        <owl:onProperty rdf:resource="#hasSoftMetric"/>
                        <owl:someValuesFrom rdf:resource="#SoftMetric"/>
                    </owl:Restriction>
                    <owl:Restriction>
                        <owl:onProperty rdf:resource="#
                            isConnectedWithTestCase"/>
                        <owl:someValuesFrom rdf:resource="#TestCase"/>
                    </owl:Restriction>
                </owl:unionOf>
            </owl:Class>
        </owl:equivalentClass>
        <rdfs:subClassOf rdf:resource="#Requirement"/>
    </owl:Class>


    <!-- http://www.semanticweb.org/ontologies/2012/4/test_ontology.owl#
        VerificationMethod -->

    <owl:Class rdf:about="#VerificationMethod">
        <rdfs:subClassOf rdf:resource="#Attribute"/>
    </owl:Class>


    <!-- http://www.w3.org/2002/07/owl#Thing -->

    <owl:Class rdf:about="&owl;Thing"/>


    <!--
    ///////////////////////////////////
    //
    // General axioms
    //
    //////////////////////////////////
     -->

    <rdf:Description>
        <rdf:type rdf:resource="&owl;AllDisjointClasses"/>
        <owl:members rdf:parseType="Collection">
            <rdf:Description rdf:about="#InconsistentRequirement"/>
            <rdf:Description rdf:about="#Refinement"/>
            <rdf:Description rdf:about="#RefinementReason"/>
        </owl:members>
    </rdf:Description>
</rdf:RDF>
```

A.2 Completeness Rules

1. **AT LEAST ONE goal must be specified.**

   IF NO goal is specified
   THEN print error: ”You did not specify any goal.”
   ”Please specify at least one goal.”

   a) **Every goal must have AT LEAST ONE author.**

      IF goal has NO author
      THEN print error: ”You did not specify an author for the following goals: [G_n].”
      ”Please specify an author for the following goals: [G_n].”

   b) **Every goal should be related to a Use-Case.**

      IF goal is NOT linked to an Use-Case
      THEN print warning: ”You did not relate the following goals a Use-Case: [G_n].”
      ”You should link the following goals to a Use-Case: [G_n].”

2. **AT LEAST ONE Use-Case must be specified.**

   IF NO Use-Case is specified
   THEN print error: ”You did not specify any Use-Case.”
   ”Please specify at least one.”

   a) **Every Use-Case must have AT LEAST ONE author.**

      IF Use-Case has NO author
      THEN print error: ”You did not specify an author for the following Use-Cases [UC_n]”.
      ”Please specify at least one author for the Use-Cases: [UC_n].”

b) **Every Use-Case must have AT LEAST ONE scenario.**

IF Use-Case has NO scenario
THEN print error: ”You did not specify any scenario for the following Use-Cases $[UC_n]$”.
”Please specify at least one scenario for the Use-Cases: $[UC_n]$.”

c) **Every Use-Case should be connected to AT LEAST ONE goal.**

IF Use-Case is NOT connected to any goal
THEN print warning: ”You did not connect the Use-Case to any goal.”
”Please specify at least one goal that is described by the following Use-Cases: $[UC_n]$.”

d) **Every Use-Case must describe AT LEAST ONE requirement.**

IF Use-Case is NOT connected to any goal
THEN print warning: ”You did not connect the Use-Case to any requirement.”
”Please specify at least one requirement that is described by the following Use-Cases: $[UC_n]$.”

3. **AT LEAST ONE Scenario must be specified.**

IF NO Scenario is specified
THEN print error: ”You did not specify any Scenario.”
”Please specify at least one.”

a) **Every Scenario must have an Author.**

IF Scenario has NO Author
THEN print error: ”You did not specify an author for the following Scenarios $[S_n]$”.
”Please specify at least one author for the Scenarios: $[S_n]$.”

b) **Every Scenario should be connected to AT LEAST ONE goal.**

IF Scenario is NOT connected to any Goal
THEN print warning: ”You did not connect the Scenario to any goal.”
”Please specify at least one goal for the following Scenarios: $[S_n]$.”

c) **Every Scenario must describe AT LEAST ONE requirement.**

IF Scenario does NOT describe AT LEAST ONE Requirement
THEN print error: ”You did not relate the Scenario to any requirement.”
”Please specify at least one requirement for the following Scenarios: $[S_n]$.”

4. **AT LEAST ONE Functional Requirement (FR) must be specified.**

   IF NO FR is specified
   THEN print error: ”You did not specify any FR.
   ”Please specify at least one Functional Requirement.”

5. **AT LEAST ONE Non-Functional Requirement (NFR) must be specified.**

   IF NO NFR is specified
   THEN print error: ”You did not specify any NFR.
   ”Please specify at least one Non-Functional Requirement.”

6. **AT LEAST ONE Platform Requirement (PR) should be specified.**

   IF NO PR is specified
   THEN print warning: ”You did not specify any PR.
   ”Please specify at least one Platform Requirement.”

7. **Every requirement must define whether it is mandatory or optional.**

   IF requirement is not mandatory AND not optional
   THEN print error: ”You did not specify whether the following requirements are mandatory or optional”
   ”Please specify whether the following requirements are mandatory or optional: [R_n].”

8. **Every requirement should be connected to AT LEAST ONE goal.**

   IF requirement is NOT connected to any goal
   THEN print warning: ”You did not connect the FR to any goal.”
   ”Please specify at least one goal that describes the following FRs: [FR_n].”

9. **Every requirement should have a priority**.

   IF requirement has NO priority
   THEN print warning: ”You did not define a priority for the requirements [R_n].
   ”Please choose a priority for the following requirements: [R_n].”

10. **Every requirement should have an author.**

    IF requirement has NO author
    THEN print warning: ”You did not specify the author for the requirements [R_n].
    ”Please specify the author for the following requirements: [R_n].”

11. **Every requirement should define its risk.**

    IF requirement has NO risk defined
    THEN print warning: "You did not specify a risk level for the requirements $[R_n]$.
    "Please specify the risk for the following requirements: $[R_n]$."

12. **Every requirement must specify AT LEAST ONE property *has_requirement_relationship.***

    IF requirement has NOT specified a property "has_requirement_relationship"
    THEN print out error: "You did not specify any requirement relationship for the requirements $[R_n]$.
    "Please check the relationships for the following requirements: $[R_n]$."

13. **Every requirement should be linked to a metric or soft-metric.**

    IF requirement is NOT linked to a metric
    THEN print out warning: "You did not specify a metric or soft-metric for the requirements $[R_n]$.
    "Please specify a metric or soft-metric for the following requirements: $[R_n]$."

14. **Every requirement should be connected to AT LEAST ONE Use-Case**.

    IF requirement is NOT connected to AT LEAST ONE USE-Case
    THEN print out warning: "You did not specify a USE-Case for the requirements $[R_n]$.
    "Please specify a Use-Case for the following requirement: $[R_n]$."

15. **Every requirement should have AT LEAST ONE Test-Case.**

    IF requirement is NOT linked to AT LEAST ONE Test-Case
    THEN print out warning: "You did not specify a Test-Case for the requirement $[R_n]$.
    "Please specify a Test-Case for the following requirements: $[R_n]$."

16. **Every refinement of a requirement must define a refinement reason.**

    IF requirement is refined and NOT linked to a refinement reason
    THEN print out warning: "You did not specify a refinement reason for the following refined requirement $[R_n]$.
    "Please specify a refinement reason for the following requirements: $[R_n]$."

17. **At least one stakeholder must be specified.**

    IF NO stakeholder is specified

THEN print error: ”You did not specify any Stakeholder.”
”Please specify at least one Stakeholder.”

18. **At least one Source must be specified.**

    IF NO source is specified
    THEN print error: ”You did not specify any source.”
    ”Please specify at least one source.”

A.3 Consistency Rules

1. **All mandatory requirements must be included in the requirements configuration.**

   IF NOT all mandatory requirements are included
   THEN print error: ”The following requirements are mandatory and should therefore be included in the requirements configuration as well: $[R_n]$.”
   ”Please include these requirements or revise their mandate.”

2. **All coexistent requirements must be included in the requirements configuration.**

   IF NOT all coexistent requirements are included
   THEN print error: ”There are unsatisfied requirements relationships. The following requirements are coexistent with the ones included in the requirements configuration and should therefore be included as well: $[R_n]$.”
   ”Please include these requirements in your requirements configuration or revise the relationships of $[[R_x, R_y]$,...] .”

3. **Excluding requirements must not be included in the requirements configuration.**

   IF excluding requirements are included in the requirements configuration
   THEN print error: ”The following requirements exclude others of the requirements configuration$[R_n]$.”
   ”Please choose one of the following options:

   - Exclude the following requirements: $[R_n]$,
   - find alternatives for $[R_n]$ or
   - revise the requirement relationships of $[[R_x, R_y]$,...].”

4. **Conflicting requirements should not be included in the requirements configuration.**

IF any requirement is in conflict with any other requirement in the requirements configuration
THEN print error: ”The following requirements are in conflict: [[R_x, R_y],...].
”Please solve these conflicts and revise the requirement relationships of [[R_x, R_y],...].”

5. **Requirements with negative contributions to other requirements should be avoided.**

   IF any requirement has a negative contribution to any other requirement
   THEN print warning: ”The following requirements are a negative contribution to other requirements: [[$R_n to R_m$], ...].”
   ”Please reconsider your requirements configuration.”

6. **Refinements of requirements should be considered.**

   IF a refinement for a requirement exists
   THEN print warning: ”A refinement exists for the following requirements: [R_n].”
   ”Please consider to include these refined requirements in the requirements configuration: [R_m].”

A.4 Quality Rules

1. **AT LEAST EACH most refined requirement must be described by a use-case.**
   Error: "‘The following requirements have no use-case assigned: [R_n]
   "‘Please choose one of the following options:
   - Assign or extend an existing use-case for these requirements,
   - Specify a use-case for the requirements and assign them to the appropriate requirements.
2. **EVERY most refined requirement must have a test-case or metric assigned.**
   Error: "‘The following requirements are not assigned to any test-case or metric: [...]
   “Please choose one of the following options:
   - Assign or extend an existing test-case for these requirements,
   - Assign an existing metric to the requirements,
   - Specify a test-case and assign it to the appropriate requirement,
   - Specify a metric and assign it to the appropriate requirement.”
3. **There should be no requirement that is a negative contribution to a goal to be achieved.**
   WARNING: "‘The following requirements are a negative contribution to a goal: [r_1 on the goal g_1, ...]
   “Please choose one of the following options:
   - Exclude the optional requirements [...] from the requirements configuration,
   - Choose one of the alternative requirements instead of [r_1]: [r_x],

- Choose one of the alternative requirements instead of [r_2]: [r_y],
- Revise the goal satisfaction relationship."

4. **There should be no optional requirement with a high risk or high cost.**
   WARNING: "'The following optional requirements have a high risk and/or high cost [R_n]
   "'Please choose one of the following options:
   - Exclude the following optional requirements from the requirements configuration [R_n]
   - Choose one of the alternative requirements instead of [r_1]: [r_x], ...
   - Revise the cost and/or risk of these requirements,
   - Specify the requirements as mandatory (if reasonable).

5. **ALL requirements must state their priority.**
   Error: "'The following requirements do not state their level of priority [R_n]
   "'Please add the level of priority to these requirements.

6. **ALL requirements must state their mandate (optional or mandatory).**
   Error: "'The following requirements do not specify their mandate [R_n]
   "'Please add a mandate to these requirements.

7. **Requirements with a positive contribution to a goal should be included in the requirements configuration.**
   Warning: "'The following requirements are a positive contribution to a challenge or goal: [r_1 to g_1, ...]
   "'Please consider to include them in the requirements configuration.

8. **Requirements must be complete.**
   Error: "'The following requirements miss relevant information [R_n]
   "'Please execute the completeness validation and add missing information.

9. **The requirements configuration must be consistent.**
   Error: "'The requirements configuration is inconsistent.
   "'Please execute the consistency validation and resolve inconsistency.

A.5 OntoReq Exemplar Complete Requirements Knowledge

Italic relations are inferred by the reasoner and not explicitly stated.

ID	**Relation (Object/ Data Property)**	**Priority**	**Cost**	**Risk**
Goal1_IncreaseSatisfaction	hasSource Source1_Survey, isAuthoredBy Author_Katja, *isPositiveContributedBy UC2_IdentifyProduct*, *isPositiveContributedBy UC1_LocalizeProduct*			
Goal2_CustomerGuidance	isAuthoredBy Author_Katja, *isPositiveContributedBy FR8_AudioOutput*, *isPositiveContributedBy*, *UC2_IdentifyProduct*, *isPositiveContributedBy UC1_LocalizeProduct*			
Goal3_ImproveOrientation	refinesTo Goal3.1_SupportProdIdentification, refinesTo Goal3.2_SupportProdLocalization, isAuthoredBy Author_Katja, hasRefinementReason Ref2, hasObstacle O1, *isPositiveContributedBy UC1_LocalizeProduct*			
Goal3.1_SupportProdIdentification	isAuthoredBy Author_Katja, hasSource Source1_Survey, hasObstacle O1, *hasRefinementReason Ref2*, *isPositiveContributedBy UC2_IdentifyProduct*, *isPositiveContributedBy FR5.2_SearchByCategory*, *isRefinementOf Goal3_ImproveOrientation*			

ID	**Relation (Object/ Data Property)**	**Priority**	**Cost**	**Risk**
Goal3.2_SupportProdLocalization	hasSource Source1_Survey, isAuthoredBy Author_Katja, hasObstacle O1, *hasRefinementReason Ref2*, *isPositiveContributedBy UC1_LocalizeProduct*, *isRefinementOf Goal3_ImproveOrientation*			
FR1_CustomerInfo	hasRefinementReason Ref1, hasSource Source2_Thesis, refinesTo FR1.2_CustomerInfoAsAudio, refinesTo FR1.3_CustomerInfoAsPic, refinesTo FR1.1_CustomerInfoAsText, isAuthoredBy Author_Katja, isOptional “true”, *isConnectedWithUseCase UC3_Advertisement*	M		
FR1.1_CustomerInfoAsText	isAlternativeTo FR1.2_CustomerInfoAsAudio, isAlternativeTo FR1.3_CustomerInfoAsPic, isAuthoredBy Author_Katja, hasSource Source2_Thesis, isMandatory “true”, *isAlternativeOf FR1.3_CustomerInfoAsPic*, *isAlternativeOf FR1.2_CustomerInfoAsAudio*, *hasRefinementReason Ref1*, *isConnectedWithUseCase UC3_Advertisement*, *isRefinementOf FR1_CustomerInfo*	H	L	L
FR1.2_CustomerInfoAsAudio	isAuthoredBy Author_Katja, excludes FR1.3_CustomerInfoAsPic, isAlternativeTo FR1.3_CustomerInfoAsPic, hasSource Source2_Thesis, isOptional “true”, *isAlternativeOf FR1.1_CustomerInfoAsText*, *isAlternativeOf FR1.3_CustomerInfoAsPic*, *hasRefinementReason Ref1*, *isConnectedWithUseCase UC3_Advertisement*, *isRefinementOf FR1_CustomerInfo*, *isAlternativeTo FR1.1_CustomerInfoAsText*	M	H	L

ID	**Relation (Object/ Data Property)**	**Priority**	**Cost**	**Risk**
FR1.3_CustomerInfoAsPic	hasSource Source2_Thesis, isAuthoredBy Author_Katja, isMandatory "true", *isAlternativeOf FR1.1_CustomerInfoAsText*, *isAlternativeOf FR1.2_CustomerInfoAsAudio*, *hasRefinementReason Ref1*, *isExcludedBy FR1.2_CustomerInfoAsAudio*, *isConnectedWithUseCase UC3_Advertisement*, *isRefinementOf FR1_CustomerInfo*, *isAlternativeTo FR1.1_CustomerInfoAsText*, *isAlternativeTo FR1.2_CustomerInfoAsAudio*		M	
FR2_PhysicalAvailabilityOfProducts	isCoexistentWith NFR1_Modifiability, isAuthoredBy Author_Katja, isMandatory "true", *isConnectedWithTestCase TC2_Learnability_Localization*, *isConnectedWithUseCase UC1_LocalizeProduct*	H	M	
FR4_ProductPlaceDescription	refinesTo FR4.2_InteractiveRoute, isAuthoredBy Author_Katja, hasRefinementReason Ref3, refinesTo FR4.1_ProductPicture, isMandatory "true"			
FR4.1_ProductPicture	isAuthoredBy Author_Katja, isConnectedWithTestCase TC2_Learnability_Localization, isConnectedWithUseCase UC2_IdentifyProduct, isAlternativeTo FR4.2_InteractiveRoute, isConnectedWithUseCase UC1_LocalizeProduct, isMandatory "true", *isAlternativeOf FR4.2_InteractiveRoute*, *hasRefinementReason Ref3*, *isRefinementOf FR4_ProductPlaceDescription*	H	L	
FR4.2_InteractiveRoute	isAuthoredBy Author_Katja, isOptional "true"	L	H	H

ID	**Relation (Object/ Data Property)**	**Priority**	**Cost**	**Risk**
FR5_ProductSearch	refinesTo FR5.1_SearchByTerm, refinesTo FR5.2_SearchByCategory, isAuthoredBy Author_Katja, isMandatory “true”, *isInConflictWith FR8_AudioOutput*, *isInConflictWith NFR1_Modifiability*, *isAlternativeOf FR4.1_ProductPicture*, *hasRefinementReason Ref3*, *isConnectedWithTestCase TC2_Learnability_Localization*, *isRefinementOf FR4_ProductPlaceDescription*, *isAlternativeTo FR4.1_ProductPicture*		L	
FR5.1_SearchByTerm	isAuthoredBy Author_Katja, isMandatory “true”, *isConnectedWithTestCase TC2_Learnability_Localization*, *isConnectedWithUseCase UC1_LocalizeProduct*, *isRefinementOf FR5_ProductSearch*	H		
FR5.2_SearchByCategory	isPositiveContributionToGoal Goal3.1_SupportProdIdentif, isAuthoredBy Author_Katja, hasSource Source2_Thesis, isOptional “true”, *isConnectedWithTestCase TC2_Learnability_Localization* *isConnectedWithUseCase UC1_LocalizeProduct* *isRefinementOf FR5_ProductSearch*, *isMandatory “true”*			L
FR6_SelectionOfDesire	isAuthoredBy Author_Katja, isMandatory “true” *isConnectedWithUseCase UC2_IdentifyProduct* *isConnectedWithUseCase UC1_LocalizeProduct*		L	
FR7_CustomerDialogue	isCoexistentWith NFR1_Modifiability, isCoexistentWith NFR2_Comprehensibility, isAuthoredBy Author_Katja, isMandatory “true”, *isInConflictWith FR8_AudioOutput*, *isConnectedWithTestCase TC1_Learnability_Identification*, *isConnectedWithUseCase UC2_IdentifyProduct*	H		

ID	**Relation (Object/ Data Property)**	**Priority**	**Cost**	**Risk**
FR8__AudioOutput	isInConflictWith FR7__CustomerDialogue, isPositiveContributionToGoal Goal2__CustomerGuidance, isInConflictWith FR4.2__InteractiveRoute, hasConflictReason Conf1, *isMandatory "true"*	H		
NFR1__Modifiability	isInConflictWith FR4.2__InteractiveRoute, isAuthoredBy Author__Katja, isMandatory "true", *isCoexistentWith FR7__CustomerDialogue*, *isCoexistentWith FR2__PhysicalAvailabilityOfProducts*	H	M	
NFR2__Comprehensibility	isAuthoredBy Author__Katja, isMandatory "true" , *isCoexistentWith FR7__CustomerDialogue*	M		
NFR3__Learnability	isAuthoredBy Author__Katja, hasSoftMetric SM1, isMandatory "true" , *isConnectedWithTestCase TC2__Learnability__Localization*, *isConnectedWithTestCase TC1__Learnability__Identification*	H		
Author__Katja				
Source1__Survey	isAuthoredBy Author__Katja			
Source2__Thesis	isAuthoredBy Author__Katja			
SM1	isAuthoredBy Author__Katja, *isSoftMetricOf NFR3__Learnability*			
UC1__LocalizeProduct	describesRequirement FR2__PhysicalAvailabilityOfProducts, describesRequirement FR5.2__SearchByCategory, isPositiveContributionToGoal Goal1__IncreaseSatisfaction, isAuthoredBy Author__Katja, describesRequirement FR6__SelectionOfDesire, isPositiveContributionToGoal Goal3__ImproveOrientation, describesRequirement FR5.1__SearchByTerm, describesRequirement FR5__ProductSearch, isPositiveContributionToGoal Goal2__CustomerGuidance, isPositiveContributionToGoal Goal3.2__SupportProdLocalization, *describesRequirement FR4.1__ProductPicture*			

ID	**Relation (Object/ Data Property)**	**Priority**	**Cost**	**Risk**
UC2_IdentifyProduct	isPositiveContributionToGoal Goal3.1_SupportProdIdentif, isAuthoredBy Author_Katja, describesRequirement FR7_CustomerDialogue, describesRequirement FR6_SelectionOfDesire, isPositiveContributionToGoal Goal1_IncreaseSatisfaction, isPositiveContributionToGoal Goal2_CustomerGuidance, *describesRequirement FR4.1_ProductPicture*			
UC3_Advertisement	describesRequirement FR1.1_CustomerInfoAsText, describesRequirement FR1.3_CustomerInfoAsPic, isAuthoredBy Author_Katja, describesRequirement FR1_CustomerInfo, describesRequirement FR1.2_CustomerInfoAsAudio			
TC1_Learnability_Identification	testsRequirement FR7_CustomerDialogue, isAuthoredBy Author_Katja, testsRequirement NFR3_Learnability			
TC2_Learnability_Localization	testsRequirement FR5.1_SearchByTerm, testsRequirement FR5.2_SearchByCategory, testsRequirement FR5_ProductSearch, testsRequirement FR4.2_InteractiveRoute, testsRequirement NFR3_Learnability, testsRequirement FR2_PhysicalAvailabilityOfProducts, isAuthoredBy Author_Katja, testsRequirement FR4.1_ProductPicture			
O1	isAuthoredBy Author_Katja			
Ref1	isAuthoredBy Author_Katja, *isRefinementReasonOf FR1.1_CustomerInfoAsText*, *isRefinementReasonOf FR1.3_CustomerInfoAsPic*, *isRefinementReasonOf FR1_CustomerInfo*, *isRefinementReasonOf FR1.2_CustomerInfoAsAudio*			
Ref2	isAuthoredBy Author_Katja, *isRefinementReasonOf Goal3.1_SupportProdIdentification*, *isRefinementReasonOf Goal3.2_SupportProdLocalization*, *isRefinementReasonOf Goal3_ImproveOrientation*			

ID	**Relation (Object/ Data Property)**	**Priority**	**Cost**	**Risk**
Ref3	isAuthoredBy Author_Katja, *isRefinementReasonOf FR4.2_InteractiveRoute*, *isRefinementReasonOf FR4_ProductPlaceDescription*, *isRefinementReasonOf FR4.1_ProductPicture*			
Conf1				

A.6 Results of Consistency Validation (1)

Listing A.1: Consistency Validation (1)

```
13 inconsistency errors detected.

- The following requirements are mandatory and should be included in the
    requirements configuration:
[FR1.3_CustomerInfoAsPic, FR4.1_ProductPicture, FR5.1_SearchByTerm, FR5.2
    _SearchByCategory, FR8_AudioOutput, NFR3_Learnability]

----------------------------------------
- Error: FR8_AudioOutput and FR7_CustomerDialogue are specified as
    conflicting.
  You have the following options:
  - Revise the requirements FR8_AudioOutput or FR7_CustomerDialogue to
      solve the conflict
  - Revise the requirements relationship (conflict) between FR8_AudioOutput
      and FR7_CustomerDialogue
  - Choose one of the following alternative requirements instead of FR4.2
      _InteractiveRoute: [FR4.1_ProductPicture]

----------------------------------------
- Error: FR5_ProductSearch was refined to FR5.2_SearchByCategory which is
    not included in your requirements configuration.
  You have the following options:
  - Revise the requirements FR5.2_SearchByCategory or FR5_ProductSearch to
      solve the refinement problem
  - Revise the requirement relationship (refinement) between FR5.2
      _SearchByCategory and FR5_ProductSearch
  - Include the requirement FR5.2_SearchByCategory in your requirements
      configuration.

- Error: FR5_ProductSearch was refined to FR5.1_SearchByTerm which is not
    included in your requirements configuration.
  You have the following options:
  - Revise the requirements FR5.1_SearchByTerm or FR5_ProductSearch to
      solve the refinement problem
  - Revise the requirement relationship (refinement) between FR5.1
      _SearchByTerm and FR5_ProductSearch
  - Include the requirement FR5.1_SearchByTerm in your requirements
      configuration.

- Error: FR1_CustomerInfo was refined to FR1.3_CustomerInfoAsPic which is
    not included in your requirements configuration.
  You have the following options:
  - Revise the requirements FR1.3_CustomerInfoAsPic or FR1_CustomerInfo to
      solve the refinement problem
  - Revise the requirement relationship (refinement) between FR1.3
      _CustomerInfoAsPic and FR1_CustomerInfo
  - Include the requirement FR1.3_CustomerInfoAsPic in your requirements
      configuration.

- Error: FR1_CustomerInfo was refined to FR1.2_CustomerInfoAsAudio which is
    not included in your requirements configuration.
  You have the following options:
  - Revise the requirements FR1.2_CustomerInfoAsAudio or FR1_CustomerInfo
      to solve the refinement problem
  - Revise the requirement relationship (refinement) between FR1.2
      _CustomerInfoAsAudio and FR1_CustomerInfo
  - Include the requirement FR1.2_CustomerInfoAsAudio in your requirements
      configuration.

- Error: FR4_ProductPlaceDescription was refined to FR4.2_InteractiveRoute
    which is not included in your requirements configuration.
  You have the following options:
  - Revise the requirements FR4.2_InteractiveRoute or
      FR4_ProductPlaceDescription to solve the refinement problem
  - Revise the requirement relationship (refinement) between FR4.2
      _InteractiveRoute and FR4_ProductPlaceDescription
  - Include the requirement FR4.2_InteractiveRoute in your requirements
      configuration.

- Error: FR4_ProductPlaceDescription was refined to FR4.1_ProductPicture
    which is not included in your requirements configuration.
  You have the following options:
  - Revise the requirements FR4.1_ProductPicture or
      FR4_ProductPlaceDescription to solve the refinement problem
  - Revise the requirement relationship (refinement) between FR4.1
      _ProductPicture and FR4_ProductPlaceDescription
  - Include the requirement FR4.1_ProductPicture in your requirements
      configuration
```

Abrevations

FODA Feature Oriented Domain Analysis

GORE Goal Oriented Requirements Engineering

GUI Graphical User Interface

OWL Web Ontology Language

RE Requirements Engineering

RSG Requirements Specification Guidance

SPL Software Product Line

SRS Software Requirements Specification

UI User Interface

UML Unified Modeling Language

Bibliography

[1] Pellet Reasoner. http://clarkparsia.com/pellet/. (accessed 20.05.2011).

[2] Software Product Line. http://www.sei.cmu.edu/productlines/. (accessed 10.08.2013).

[3] TrOWL. http://trowl.eu/. (accessed 02.06.2012).

[4] Yoji Akao. *Quality Function Deployment QFD: Integrating Customer Requirements into Product Design*. Productivity Press, 1990.

[5] Ian F. Alexander and Richard Stevens. *Writing Better Requirements*. Pearson Education, 2002.

[6] A. I. Antó, E. Liang, and R. A Rodenstein. A web-based requirements analysis tool. In *Proceedings of the 5th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE'96*, WET-ICE '96, pages 238–244, Washington, DC, USA, 1996. IEEE Computer Society.

[7] Annie I. Antón. Goal-Based Requirements Analysis. In *ICRE '96: Proceedings of the 2nd International Conference on Requirements Engineering (ICRE '96)*, page 136, Washington, DC, USA, 1996. IEEE Computer Society.

[8] Annie I. Antón and Colin Potts. The Use of Goals to Surface Requirements for Evolving Systems. In *Proceedings of the 20th international conference on Software engineering*, ICSE '98, pages 157–166, Washington, DC, USA, 1998. IEEE Computer Society.

[9] Sohail Asghar and Mahrukh Umar. Requirement Engineering Challenges in Development of Software Applications and Selection of Customer-off-the-Shelf (COTS) Components. In *International Journal of Software Engineering (IJSE)*, volume 1, pages 32–50, 2010.

[10] Franz Baader, Ian Horrocks, and Ulrike Sattler. Description Logics for the Semantic Web. *IEEE Data Engineering Bulletin*, 16(25):4–9, 2001.

[11] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. The Goal Question Metric Approach. In *Encyclopedia of Software Engineering*. Wiley, 1994.

[12] Sean Bechhofer. OWL Reasoning Examples. http://owl.man.ac.uk/2003/why/latest/, 2003. (accessed 15.07.2011).

[13] Tom Beger. Evaluation von Werkzeugen fiür das Requirements Engineering bezueglich der Spezifizierbarkeit von nicht-funktionalen Anforderungen. Groï£¡er beleg, Technische Universitïät Dresden, September 2009.

[14] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 2001.

[15] Andreas Birk and Gerald Heller. Challenges for Requirements Engineering and Management in Software Product Line Development. In *Requirements Engineering: Foundation for Software Quality*, volume 4542/2007, pages 300–305. Springer Berlin / Heidelberg, 2007.

[16] Ryan Blace. OWL 2 in Action: Property Chains. http://semwebprogramming.org/?p=175, June 2009. (accessed 02.08.2013).

[17] B. W. Boehm, J. R. Brown, and M. Lipow. Quantitative Evaluation of Software Quality. In *Proceedings of the 2nd International Conference on Software Engineering*, ICSE '76, pages 592–605, Los Alamitos, CA, USA, 1976. IEEE Computer Society Press.

[18] Karin Breitman and Julio Cesar Sampaio do Prado Leite. Ontology as a Requirements Engineering Product. In *Requirements Engineering Conference, 2003*, pages 309–319. IEEE Computer Society, 2003.

[19] Christopher Brewster, Simon Jupp, Joanne Luciano, David Shotton, Robert Stevens, and Ziqi Zhang. Issues in Learning an Ontology From Text. *BMC Bioinformatics*, 10(Suppl 5):S1+, 2009.

[20] Ronald S. Carson. Requirements Completeness: A Deterministic Approach, 1995.

[21] Verónica Castañeda, Luciana Ballejos, Ma. Laura Caliusco, and Ma. Rosa Galli. The Use of Ontologies in Requirements Engineering. *Global Journal of Researches in Engineering*, 10:2–8, 2010.

[22] Jaelson Castro, Manuel Kolp, and John Mylopoulos. Towards Requirements-Driven Information Systems Engineering: The Tropos Project. *Information Systems*, 27(6), 2002.

[23] Lawrence Chung, Brian A. Nixon, Eric Yu, and John Mylopoulos. *Non-Functional Requirements in Software Engineering*, volume 5 of *International Series in Software Engineering*. Kluwer Academic Publishers, 1999.

[24] Racer Systems GmbH & Co. RacerPro User's Guide. http://www.racer-systems.com/products/racerpro/users-guide-2-0-0-preview.pdf, October 2012. (accessed 04.11.2013).

[25] Bill Curtis, Marc I. Kellner, and Jim Over. Process Modeling. *Commun. ACM*, 35(9):75–90, 1992.

[26] Luiz Marcio Cysneiros, Vera Werneck, and Eric Yu. Evaluating Methodologies: A Requirements Engineering Approach Through the Use of an Exemplar. In *Proc. of 7th Workshop on Requirements Engineering*, pages 40–55, 2004.

[27] K. Czarnecki and U.W. Eisenecker. *Generative Programming: Methods, Tools and Applications*. ACM Press/Addison-Wesley Publishing Co. New York, NY, USA, 2000.

[28] Mohamed Yehia Dahab, Hesham A. Hassan, and Ahmed Rafea. TextOntoEx: Automatic Ontology Construction From Natural English Text. *Expert Syst. Appl.*, 34(2):1474–1480, 2008.

[29] Anne Dardenne, Axel van Lamsweerde, and Stephen Fickas. Goal-directed Requirements Acquisition. *Sci. Comput. Program.*, 20:3–50, 1993.

[30] R. Darimont, E. Delor, P. Massonet, and A. van Lamsweerde. GRAIL/KAOS: an Environment for Goal-driven Requirements Engineering. In *Proceedings of the 19th international conference on Software engineering*, ICSE '97, pages 612–613, New York, NY, USA, 1997. ACM.

[31] Robert Darimont and Axel van Lamsweerde. Formal Refinement Patterns For Goal-driven Requirements Elaboration. *SIGSOFT Softw. Eng. Notes*, 21:179–190, 1996.

[32] Alan M. Davis. *Software Requirements: Analysis and Specification.* Prentice Hall Press, Upper Saddle River, NJ, USA, 2nd edition edition, 1993.

[33] Frank van Harmelen Deborah L. McGuinness. OWL Web Ontology Language Overview. http://www.w3.org/TR/owl-features/, 2004. W3C Recommendation.

[34] Glen Dobson and Peter Sawyer. Revisiting Ontology-Based Requirements Engineering in the Age of the Semantic Web. In *Dependable Requirements Engineering of Computerised Systems at NPPs*, 2006.

[35] A. Eberlein. *Requirements Acquisition and Specification for Telecommunication Services.* PhD thesis, UK: University of Wales, Swansea, 1998.

[36] Christof Ebert. *Systematisches Requirements Engineering und Management: Anforderungen Ermitteln, Spezifizieren, Analysieren und Verwalten.* dpunkt-Verl., Heidelberg, 2., aktualisierte und erw. aufl. edition, 2008.

[37] Eclipse. Eclipse modeling - emf. http://www.eclipse.org/modeling/emf/. (accessed 09.03.2011).

[38] F. Fabbrini, M. Fusani, S. Gnesi, and G. Lami. Quality Evaluation of Software Requirements Specifications. http://fmt.isti.cnr.it/WEBPAPER/paper8A2.pdf. (accessed 04.11.2013).

[39] Stefan Farfeleder, Thomas Moser, Andreas Krall, Tor Stålhane, Inah Omoronyia, and Herbert Zojer. Ontology-driven Guidance for Requirements Elicitation. In *Proceedings of the 8th Extended Semantic Web Conference on The Semantic Web: Research and Applications - Volume Part II*, ESWC'11, pages 212–226, Berlin, Heidelberg, 2011. Springer-Verlag.

[40] C.J. Fidge and A.M. Lister. The Challenges of Non-Functional Computing Requirements. http://sky.fit.qut.edu.au/ fidgec/Publications/fidge93c.pdf. (accessed 10.07.2012).

[41] Donald Firesmith. Specifying Good Requirements. *Journal of Object Technology*, 2:77–87, 2003.

[42] Donald Firesmith. Are Your Requirements Complete? *Journal of Object Technology*, 4(1):27–44, 2005.

[43] Donald Firesmith. Quality Requirements Checklist. *Journal of Object Technology*, 4(9):31–38, 2005.

[44] The Apache Software Foundation. Reasoners and Rule Engines: Jena Inference Support. http://jena.apache.org/documentation/inference/. (accessed 03.04.2013).

[45] Piotr Gawrysiak, Grzegorz Protaziuk, Henryk Rybinski, and Alexandre Delteil. Text Onto Miner - A Semi Automated Ontology Building System. In *ISMIS*, pages 563–573, 2008.

[46] Paolo Giorgini, John Mylopoulos, Eleonora Nicchiarelli, and Roberto Sebastiani. Reasoning with goal models. In *Proceedings of the 21st International Conference on Conceptual Modeling*, ER '02, pages 167–181, London, UK, UK, 2002. Springer-Verlag.

[47] Martin Glinz. A Glossary of Requirements. Engineering Terminology. https://files.ifi.uzh.ch/rerg/amadeus/publications/various/RE-Glossary_version_1.1b.pdf, 2011. (accessed 12.08.2011).

[48] Leah Goldin and Anthony Finkelstein. Abstraction-Based Requirements Management. In *Proceedings of the 2006 International Workshop on Role of Abstraction in Software Engineering*, ROA '06, pages 3–10. ACM, 2006.

[49] Orlena C. Z. Gotel and Anthony C. W. Finkelstein. An Analysis of the Requirements Traceability Problem. In *Proceedings of the First International Conference on Requirements Engineering*, pages 94–101, 1994.

[50] Sol Jaffe Greenspan. *Requirements Modeling: A Knowledge Representation Approach to Software Requirements Definition.* PhD thesis, Department of Computer Science, University of Toronto, 1984.

[51] M. Grüninger and M. Fox. Methodology for the Design and Evaluation of Ontologies. In *IJCAI'95, Workshop on Basic Ontological Issues in Knowledge Sharing, April 13, 1995*, 1995.

[52] Georges Grosz, Colette Rolland, S. Schwer, Carine Souveyet, Véronique Plihon, Samira Si-Said, Camille Ben Achour, and Christophe Gnaho. Modelling and Engineering the Requirements Engineering Process: An Overview of the NATURE Approach. *Requir. Eng.*, 2(3):115–131, 1997.

[53] W3C OWL Working Group. OWL 2 Web Ontology Language Document Overview. http://www.w3.org/TR/2012/REC-owl2-overview-20121211/, November 2012. (accessed 04.11.2013).

[54] Thomas R. Gruber. A Translation Approach to Portable Ontology Specifications. *KNOWLEDGE ACQUISITION*, 5:199–220, 1993.

[55] Tom Gruber. *Ontology.* Springer US, 2009.

[56] H. Verheul H. d. Vries and H. Willemse. Stakeholder Identification in IT Standardization Processes. In *Standard Making: A Critical Research Frontier for Information Systems MISQ Special Issue Workshop*, 2003.

[57] Mariele Hagen, Berit Jungmann, and Kim Lauenroth. Ein Prozessmodell für ein Agiles und Wiki-basiertes Requirements Engineering mit Unterstützung durch Semantic-Web-Technologien, February 2010.

[58] Tracy Hall, Sarah Beecham, and Austen Rainer. Requirements Problems in Twelve Software Companies: An Empirical Analysis. *IEE Proceedings - Software*, 149(5):153–160, 2002.

[59] Terry Halpin. Ontological Modeling: Part 10. *Business Rules Journal*. LogicBlox and INTI International University.

[60] Jan Hendrik Hausmann, Reiko Heckel, and Gabi Taentzer. Detection of Conflicting Functional Requirements in a Use Case-Driven Approach: A Static Analysis Technique Based on Graph Transformation. In *Proceedings of the 24th International Conference on Software Engineering*, ICSE '02, pages 105–115, New York, NY, USA, 2002. ACM.

[61] Matthew Horridge and Sean Bechhofer. The OWL API: A Java API for OWL Ontologies. In *Semant. Web*, volume 2, pages 11–21, Amsterdam, The Netherlands, The Netherlands, 2011. IOS Press.

[62] IEEE. IEEE 829-1998 – IEEE Standard for Software Test Documentation. Standard, September 1998.

[63] IEEE. IEEE Recommended Practice for Software Requirements Specifications. Technical Report 830-1998, IEEE, 1998.

[64] Hoh In, Barry Boehm, Thomas Rodgers, and Michael Deutsch. Applying WinWin to Quality Requirements: A Case Study. In *Proceedings of the 23rd International Conference on Software Engineering*, ICSE '01, pages 555–564, Washington, DC, USA, 2001. IEEE Computer Society.

[65] Carnegie Mellon Software Engineering Institute. Software Product Lines. http://www.sei.cmu.edu/productlines/. (accessed 02.03.2013).

[66] Institute of Electrical and Electronics Engineers. *IEEE Standard Glossary of Software Engineering Terminology : IEEE Std 610.12-1990*. Institute of Electrical and Electronics Engineers, New York, NY, USA, 1990.

[67] Mark S. Fox Jinxin Lin and Taner Bilgic. A Requirement Ontology for Engineering Design. *Concurrent Engineering: Research and Applications*, 4:279–291, 1996.

[68] Ivan J. Jureta, John Mylopoulos, and Stéphane Faulkner. A Core Ontology for Requirements. *Appl. Ontol.*, 4(3-4):169–244, 2009.

[69] Haruhiko Kaiya and Motoshi Saeki. Ontology Based Requirements Analysis: Lightweight Semantic Processing Approach. In *Proc. Fifth International Conference on Quality Software (QSIC 2005)*, 2005.

[70] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-oriented Domain Analysis (FODA) Feasibility Study. Technical report, Carnegie-Mellon University Software Engineering Institute, 1990.

[71] M. Kassab, O. Ormandjieva, and M. Daneva. An Ontology Based Approach to Non-functional Requirements Conceptualization. *Software Engineering Advances, International Conference on*, 0:299–308, 2009.

[72] E. Kavakli and Loucopoulos P. Goal Driven Requirements Engineering: Evaluation of Current Methods. In *the 8th CAiSE/IFIP8.1 International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design (EMMSAD '03)*, 2003.

[73] Evangelia Kavakli. *Goal-driven Requirements Engineering: Modelling and Guidance.* PhD thesis, University of Manchester, 1999.

[74] Kahn Keller and Panara. Specifying SoftwareQuality Requirements with Metricsn. In R.H. Thayer and M. Dorfman, editors, *System and Software Requirements Engineering*, pages 145–163. IEEE Computer Society Press, 1990.

[75] Marc I. Kellner. *Process Guides: Effective Guidance For Process Participants*, volume 12 of *IESE-Report // Fraunhofer Einrichtung experimentelles Software Engineering.* Fraunhofer-IESE, 1998.

[76] Chang Hwan Peter Kim. *On the Relationship Between Feature Models and Ontologies.* University of Waterloo (Canada), 2006.

[77] B. Kitchenham, L. Pickard, and S. L. Pfleeger. Case Studies for Method and Tool Evaluation. *IEEE Software*, 12(4):52–62, 1995.

[78] Leonid Kof. Natural Language Processing For Requirements Engineering Applicability. In *Proceedings of the Workshops*, page 2004, 2004.

[79] M. Kossmann, R. Wong, M. Odeh, and A. Gillies. Ontology-driven Requirements Engineering: Building the OntoREM Meta Model. In *Information and Communication Technologies: From Theory to Applications, 2008. ICTTA 2008. 3rd International Conference on*, pages 1 – 6, 2008.

[80] A. Kott and J. Peasant. Representation and Management of Requirements: The RAPID-WS Project. *Concurrent Engineering*, 3(2):93–106, 1995.

[81] Simone Kriglstein. OWL Ontology Visualization: Graphical Representations of Properties on the Instance Level. In Ebad et. al. Banissi, editor, *IV*, pages 92–97. IEEE Computer Society, 2010.

[82] Ivan Kurtev, Jean Bézivin, and Mehmet Aksit. Technological Spaces: An Initial Appraisal. In *International Symposium on Distributed Objects and Applications, DOA 2002*, 2002.

[83] Giuseppe Lami and Gianluca Trentanni. An Automatic Tool for Improving the Quality of Software Requirements. http://www.ercim.eu/publication/Ercim_News/enw58/EN58.pdf, 2004. (accessed 04.11.2013).

[84] A. Lamsweerde, R. Darimont, and P. Massonet. Goal-Directed Elaboration of Requirements For a Meeting Scheduler: Problems and Lessons Learnt. In *Proceedings of the Second IEEE International Symposium on Requirements Engineering*, RE '95, pages 194–, Washington, DC, USA, 1995. IEEE Computer Society.

[85] Axel Van Lamsweerde and Laurent Willemet. Inferring Declarative Requirements Specifications From Operational Scenarios. *IEEE Transactions on Software Engineerin*, 24:1089–111, 1998.

[86] Alexei Lapouchnian. Goal-oriented Requirements Engineering: An Overview of the Current Research, 2005.

[87] Yuquin Lee and Wenyun Zhao. An Ontology-Based Approach for Domain Requirements Elicitation and Analysis. *Computer and Computational Sciences, International Multi-Symposiums on*, 2:364–371, 2006.

[88] Jens Lemcke, Andreas Friesen, and Tirdad Rahmani. Validating Component-based Implementations of Business Processes. In *Electronic Business Interoperability: Concepts, Opportunities and Challenges*, chapter 7, pages 124–151. IGI Global, 2011.

[89] Emmanuel Letier and Axel van Lamsweerde. Reasoning About Partial Goal Satisfaction For Requirements and Design Engineering. *SIGSOFT Softw. Eng. Notes*, 29:53–62, 2004.

[90] L. Liu and E. Yu. From Requirements to Architectural Design - Using Goals and Scenarios, 2001.

[91] Jacques Lonchamp. A Collaborative Process-centered Environment Kernel. In *CAiSE '94: Proceedings of the 6th international conference on Advanced information systems engineering*, pages 28–41, Secaucus, NJ, USA, 1994. Springer-Verlag New York, Inc.

[92] Pericles Loucopoulos and Vassilios Karakostas. *System Requirements Engineering.* McGraw-Hill, Inc., New York, NY, USA, 1995.

[93] Mich Luisa, Franch Mariangela, and Inverardi Pierluigi. Market Research for Requirements Analysis Using Linguistic Tools. *Requirement Engineering*, 9(1):40–56, 2004.

[94] Bharat B. Madan, Katerina Goseva-Popstojanova, Kalyanaraman Vaidyanathan, and Kishor S. Trivedi. Modeling and Quantification of Security Attributes of Software Systems. In *Proceedings of the 2002 International Conference on Dependable Systems and Networks*, DSN '02, pages 505–514, Washington, DC, USA, 2002. IEEE Computer Society.

[95] Alexander Maedche and Steffen Staab. Mining Ontologies from Text. In *EKAW '00: Proceedings of the 12th European Workshop on Knowledge Acquisition, Modeling and Management*, pages 189–202. Springer-Verlag, 2000.

[96] Zohar Manna, Nikolaj S. Bjørner, Anca Browne, Michael Colón, Bernd Finkbeiner, Mark Pichora, Henny B. Sipma, and Tomás E. Uribe. STeP: Deductive-algorithmic Verification of Reactive and Real-time Systems. In *In 8th CAV*, pages 415–418. Springer-Verlag, 1996.

[97] Mayank, Kositsyna, and Austin. Requirements Engineering and the Semantic Web: Part II. Representation, Management and Validation of Requirements and System-Level Architectures. Technical report, University of Maryland, 2004. ISR Technical Report 2004-14.

[98] Deborah L. McGuinness and Frank van Harmelen. OWL Web Ontology Language. http://www.w3.org/TR/owl-features/, November 2009. (accessed 10.11.2011).

[99] Krzysztof Miksa, Marek Kasztelnik, Pawel Sabina, and Tobias Walter. Towards Semantic Modeling of Network Physical Devices. In *MoDELS Workshops*, volume 6002 of *Lecture Notes in Computer Science*, pages 329–343, 2009.

[100] Mikyeong Moon, Keunhyuk Yeom, and Heung Seok Chae. An Approach to Developing Domain Requirements as a Core Asset Based on Commonality and Variability Analysis in a Product Line. *EEE Trans. Softw. Eng.*, 31:551–569, 2005.

[101] J. Mylopoulos, L. Chung, and B. Nixon. Representing and Using Nonfunctional Requirements: A Process-Oriented Approach. *IEEE Trans. Softw. Eng*, 18(6):483–497, 1992.

[102] John Mylopoulos, Alex Borgida, Matthias Jarke, and Manolis Koubarakis. Telos: Representing Knowledge About Information Systems. *ACM Trans. Inf. Syst.*, 8:325–362, 1990.

[103] John Mylopoulos, Lawrence Chung, and Eric Yu. From Object-oriented to Goal-oriented Requirements Analysis. *Communications of the ACM*, 42(1):31 – 37, 1999.

[104] Joost Noppen, Pim Van Den Broek, and Mehmet Aksit. Imperfect Requirements in Software Development. In *Proceedings of the 13th international working conference on Requirements engineering: foundation for software quality*, REFSQ'07, pages 247–261, Berlin, Heidelberg, 2007. Springer-Verlag.

[105] Natalya F. Noy and Deborah L. Mcguinness. Ontology Development 101: A Guide to Creating Your First Ontology. Technical Report KSL-01-05, Stanford Knowledge Systems Laboratory, 2001.

[106] Bashar Nuseibeh and Steve Easterbrook. Requirements Engineering: A Roadmap. In *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*, pages 35–46, New York, NY, USA, 2000. ACM.

[107] Fernando Silva Parreiras, Steffen Staab, and Andreas Winter. TwoUse: Integrating UML Models and OWL Ontologies. Technical Report 16/2007, Universität Koblenz-Landau, Fachbereich Informatik, 2007.

[108] Klaus Pohl. PRO-ART: Enabling Requirements Pre-Traceability. In *ICRE'96*, pages 76–85, 1996.

[109] Klaus Pohl. *Requirements Engineering. Grundlagen, Prinzipien, Techniken*, volume 1. Dpunkt Verlag, 2007.

[110] Names Project. Requirement Specification. http://names.mimas.ac.uk/documents/Names_Software_Requirements_11Jul2008.pdf, 2008.

[111] Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF. http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/, January 2008. (accessed 01.02.2011).

[112] Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF. http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/, March 2013. (accessed 04.11.2013).

[113] Balasubramaniam Ramesh and Matthias Jarke. Toward Reference Models of Requirements Traceability. *IEEE Trans. Software Eng.*, pages 58–9, 2001.

[114] Yuan Ren, Jeff Z. Pan, and Yuting Zhao. Closed World Reasoning for OWL2 with NBox. *Journal of Tsinghua Science and Technology*, 15(6), 2010.

[115] Thomas Riechert, Kim Lauenroth, and Jens Lehmann. Semantisch unterstütztes Requirements Engineering. In *Proceedings of the SABRE-07 SoftWiki Workshop*, 2007.

[116] C. Rolland and Colette Rolland. A Comprehensive View of Process Engineering. In *In Proc. 10 th Intl. Conf. Advanced Information Systems Engineering, (CAiSE 98)*, pages 1–24. Springer, 1998.

[117] Pere P. Sancho, Carlos Juiz, Ramon Puigjaner, Lawrence Chung, and Nary Subramanian. An Approach to Ontology-aided Performance Engineering Through NFR Framework. In *WOSP '07: Proceedings of the 6th international workshop on Software and performance*, pages 125–128, New York, NY, USA, 2007. ACM Press.

[118] Bruno Schienmann. *Kontinuierliches Anforderungsmanagement: Prozesse - Techniken - Werkzeuge*. Addison-WesleAddison-Wesley, München, 2002.

[119] F.T. Sheldon and H. Y. Kim. Validation of Guidance Control Software Requirements Specification for Reliability and Fault-Tolerance. In *IEEE annual proceedings on Reliability and Maintainability Symposium*, Washington, DC, USA, 2002.

[120] Samira Si-Said and Colette Rolland. Guidance for Requirements Engineering Processes. In *DEXA '97: Proceedings of the 8th International Conference on Database and Expert Systems Applications*, pages 643–652, London, UK, 1997. Springer-Verlag.

[121] Ian Sommerville and Pete Sawyer. *Requirements Engineering: A Good Practices Guide*. John Wiley & Sons, 1997.

[122] Carine Souveyet, Camille Ben Achour, Camille Ben Achour, Colette Rolland, and Colette Rolland. A Proposal for Improving the Quality of the Organisation of Scenarios Collections. In Eric Dubois, Andreas L. Opdahl, and Klaus Pohl, editors, *REFSQ*, number 98, pages 33–45. Presses Universitaires de Namur, 1998.

[123] John F. Sowa. Ontology. http://www.jfsowa.com/ontology/, 2010. (accessed 28.05.2010).

[124] A. Terry Bahill and Steven J. Henderson. Requirements Development, Verification, and Validation Exhibited in Famous Failures. *Syst. Eng.*, 8(1):1–14, 2005.

[125] T. Thüm, C. Kästner, S. Erdweg, and N. Siegmund. Abstract Features in Feature-Modeling. In *Software Product Line Conference (SPLC)*, pages 191–200, 2011.

[126] Erik Tittel. Ontology-based Guidance for Requirements Engineering, 2010.

[127] Dmitry Tsarkov. FaCT++. http://owl.man.ac.uk/factplusplus/. (accessed 2011).

[128] Mike Uschold, Michael Gruninger, Mike Uschold, and Michael Gruninger. Ontologies: Principles, Methods and Applications. *Knowledge Engineering Review*, 11:93–136, 1996.

[129] Axel van Lamsweerde. Requirements Engineering in the Year 00: A Research Perspective. In *International Conference on Software Engineering*, pages 5–19, 2000.

[130] Axel van Lamsweerde. Goal-Oriented Requirements Engineering: A Guided Tour. In *RE '01: Proceedings of the Fifth IEEE International Symposium on Requirements Engineering*, page 249, Washington, DC, USA, 2001.

[131] Axel van Lamsweerde. Reasoning About Alternative Requirements Options. In Alexander Borgida, Vinay K. Chaudhri, Paolo Giorgini, and Eric S. K. Yu, editors, *Conceptual Modeling: Foundations and Applications*, volume 5600 of *Lecture Notes in Computer Science*, pages 380–397. Springer, 2009.

[132] Axel van Lamsweerde, Robert Darimont, and Emmanuel Letier. Managing Conflicts in Goal-Driven Requirements Engineering. *IEEE Transactions on Software Engineering*, 24:908–926, 1998.

[133] Axel van Lamsweerde and Emmanuel Letier. From Object Orientation to Goal Orientation: A Paradigm Shift for Requirements Engineering. In *RISSEF*, pages 325–340, 2003.

[134] Philippe Vincke. *Multicriteria Decision-Aid.* Wiley, 1992.

[135] Wikipedia. Closed World Assumption. http://en.wikipedia.org/wiki/Closed-_world_assumption. (accessed 07.09.2011).

[136] Wikipedia. Foda. https://en.wikipedia.org/wiki/Feature-oriented_domain_analysis. (accessed 10.08.2013).

[137] Wikipedia. Ontology Merging. http://en.wikipedia.org/wiki/Ontology_merging. (accessed 13.09.2013).

[138] Wikipedia. Ontology TBox. http://en.wikipedia.org/wiki/Abox. (accessed 08.08.2013).

[139] Wikipedia. Open World Assumption. http://en.wikipedia.org/wiki/Open-_world_assumption. (accessed 13.08.2013).

[140] S. P. Wilson, T. P. Kelly, and J. A. McDermid. Safety Case Development: Current Practice, Future Prospects. In *OF SOFTWARE BASED SYSTEMS - TWELFTH ANNUAL CSR WORKSHOP*. Springer-Verlag, 1997.

[141] Yang Ying-ying, Li Zong-yon, and Wang Zhi-xue. Domain Knowledge Consistency Checking for Ontology-Based Requirement Engineering. In *CSSE '08: Proceedings of the 2008 International Conference on Computer Science and Software Engineering*, pages 302–305, Washington, DC, USA, 2008. IEEE Computer Society.

[142] Ralph Rowland Young. *The Requirements Engineering Handbook.* Artech House Inc., 2004.

[143] Eric S. Yu. An Organization Modelling Framework for Multi-Perspective Information System Design. Technical report, Tech. Rpt. DKBS-TR93 -2, Dept. of Comp. Sci., Univ. of Toronto, 1993.

[144] Eric S. K. Yu. Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering. In *Proceedings of the 3rd IEEE International Symposium on Requirements Engineering*, RE '97, pages 226–235, Washington, DC, USA, 1997. IEEE Computer Society.

[145] K. Yue. What Does It Mean to Say that a Specification is Complete? *Proc. IWSSD-4, Fourth International Workshop on Software Specification and Desig*, 1987.

[146] Pamela Zave. Classification of Research Efforts in Requirements Engineering. *ACM Comput. Surv.*, 29(4):315–321, 1997.

[147] Pamela Zave and Michael Jackson. Four Dark Corners of Requirements Engineering. *ACM Transactions on Software Engineering and Methodology*, 6(1):1–30, 1997.

[148] Zhang and Eberlein. Architectural Ddesign of an Intelligent Requirements Engineering Tool. In *Canadian Conference on Electrical and Computer Engineering, CCECE*, Montreal, 2003.

[149] Wei Zhang, Hong Mei, and Haiyan Zhao. Feature-driven Requirement Dependency Analysis and High-level Software Design. *Requir. Eng.*, 11(3):205–220, 2006.

[150] Xuefeng Zhu. Inconsistency Measurement of Software Requirements Specifications: An Ontology-Based Approach. In *ICECCS '05: Proceedings of the 10th IEEE International Conference on Engineering of Complex Computer Systems*, pages 402–410, Washington, DC, USA, 2005. IEEE Computer Society.

[151] Didar Zowghi and Vincenzo Gervasi. The Three Cs of Requirements: Consistency, Completeness, and Correctness. In *Proceedings of 8th International Workshop on Requirements Engineering: Foundation for Software Quality, (REFSQ'02)*, 2002.

Glossary

Application Domain: “A set of current and future applications which share a set of common capabilities and data” [70].

Coexistence (Implication) Relationship: A requirement implies (requires) one or more other requirements.

Completeness Rules: Completeness rules comprise three parts: rule definition, fault message and solution suggestion. The definition states all the requirement artefacts that need to be specified in the SRS and their associated metadata that must (or should) be specified. The fault message provides additional information of the concrete problem for each rule that fails and the solution suggestion proposes knowledge-specific opportunities for each incompleteness problem to be eliminated.

Conflict Relationship: A requirement r_1 conflicts with another requirement r_2 if the fulfilment of r_1 excludes the fulfilment of r_2 and vice versa. A conflicting relationship must also specify a conflict reason (p). This relationship is symmetric.

Consistency Rule: Consistency rules comprise three parts: rule definition, fault message and solution suggestion. The definition states a condition of the requirements configuration to ensure consistency. The fault message provides additional information of the concrete problem for each rule that fails and the solution suggestion proposes options for each inconsistency problem to be eliminated, based on the specified requirements knowledge. A requirements configuration is consistent if all consistency rules are satisfied.

Domain Analysis: “The process of identifying, collecting, organizing, and representing the relevant information in a domain based on the study of existing systems and their development histories, knowledge captured from domain experts, underlying theory, and emerging technology within the domain” [70].

Domain Model: “A definition of the functions, objects, data, and relationships in a domain” [70].

Exclusion Relationship: The exclusion of a requirement by another requirement. In contrast to the conflict relationship, this exclusion is non-symmetric and may define an exclusion reason (p).

External Completeness (of Requirements Specification): External Completeness is stated with regard to the whole SRS, which may include various documents and models [42].

Functional Requirement: A requirement concerning a result of be- haviour that shall be provided by a function of a system, a component or service [47].

Goal Satisfaction: The satisfaction of goals comprises the avoidance of negative goal contributions and the consideration of positive goal contributions.

Goal-oriented Requirements Engineering (GORE): Goal-oriented Requirements Engineering is the branch of systems engineering concerned with the development of requirements through a systematic, iterative and co-operative process. This process involves the elicitation, negotiation, specification and validation of real-world goals and requirements for, functions of, and constraints on software systems. It is also concerned with the relationship of these RE artefacts to precise specifications of software behaviour, their evolution over time and across software families. Requirements and all related artefacts are documented in a Requirements Specification that needs to be validated regarding customer wishes, correct understanding and accuracy.

Goal: Goals are declarative statements of intent to be achieved by the system under consideration.

GORE: See Goal-oriented Requirements Engineering.

Internal Completeness (of Individual Requirements): An individual requirement is complete if it contains all necessary information to avoid ambiguity and needs no amplification to enable proper implementation [42].

Internal Consistency of a Requirements Configuration: A requirements configuration is internally consistent if it is free of conflicting and excluding requirements. All mandatory requirements and coexistent requirements must be included. The requirements configuration must contain the most refined requirement of each particular requirement refinement and comply to the alternative relationships (or- and ex-alternative).

Mandatory Attribute: A requirement is mandatory if it is definitely required by the system and must be realised.

Metadata: Metadata comprise all attributes for requirements artefacts (e.g., priority, state, cost) and the relations between requirements artefacts (e.g., goal contribution, refinements).

Metric: A functional description of values that can be measured, e.g., response time.

Non-functional Requirement: A non-functional requirement (also quality requirement) is a requirement that pertains to a quality concern that is related to a functional requirement citeGlinz2011.

Obstacle: Obstacles are declarative statements of identified behaviour that have a negative effect on the satisfaction of goals or requirements.

ODRE: See Ontology-Driven Requirements Engineering.

Ontology-Driven Requirements Engineering (ODRE): Ontology-driven (or sometimes ontology-based) RE describes a RE process or at least a RE method comprehensively aided by ontologies. Therefore, ontologies are involved for some or all tasks of the RE process. ODRE clearly states the method how to integrate a proposed ontological technique into a continuous RE process.

Ontology: In the context of computer and information sciences, an ontology defines a set of representational primitives with which to model a domain of knowledge or discourse. The representational primitives are typically classes (or sets), attributes (or properties), and relationships (or relations among class members). [...] Ontologies are typically specified in languages that allow abstraction away from data structures and Ontology-Driven Requirements Engineering implementation strategies; in practice, the languages of ontologies are closer in expressive power to first-order logic than languages used to model databases. For this reason, ontologies are said to be at the "semantic" level, whereas database schema are models of data at the "logical" or "physical" level. Due to their independence from lower level data models, ontologies are used for integrating heterogeneous databases, enabling interoperability among disparate systems, and specifying interfaces to independent, knowledge-based services [55].

Optional Attribute: A requirement is optional if it is in scope but not necessarily required by the system. It is not mandatory.

Optional-Alternative Relationship: One requirement may be replaced by another.

Or-Alternative Relationship: A choice limited to at least one requirement of a set of requirements.

Platform Requirement: Platform Requirements (also System Requirements) refer to requirements regarding the platform of the software, e.g., RAM size, Operating System, Hardware Architecture, etc.

Process Requirement: Process requirements are constraints placed upon the development process of the system.

Quality Flaw: A quality flaw is an undesired characteristic in the requirements knowledge that decreases its quality.

Quality Rules: Quality rules comprise three parts: rule definition, fault message and solution suggestion. The definition states a condition of the requirements configuration to improve quality. The fault message provides additional information of the concrete problem for each rule that fails and the solution suggestion proposes options for each quality flaw detected, based on the specified requirements knowledge.

Refinement Relationship: A requirement r_1 refines a requirement r_2 if r_1 is derived from r_2 by adding more details to it. r_1 can be seen as an abstraction of the detailed requirement r_2. The triple relationship consists of the refinement source, the refinement target and the refinement reason (p).

Requirement: "(1) A condition or capability needed by a user to solve a problem or achieve an objective. (2) A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents. (3) A documented representation of a condition or capability as in (1) or (2)." [66].

Requirements artefacts: Requirements artefacts are parts of the requirements knowledge that hold specific types of information, e.g., requirements, test-cases, metrics, prioritiess.

Requirements Configuration (RC): A requirements configuration is a set of requirements to be implemented. A requirements configuration is valid if and only if it does not violate constraints imposed by the Requirements metamodel.

Requirements Engineering: Requirements Engineering is the branch of Systems Engineering concerned with the development of requirements through a systematic, iterative and co-operative process. This process includes the elicitation, negotiation, specification and validation of requirements. It is also concerned with the relationship of these RE artefacts to precise specifications of software behaviour, their evolution over time and across software families. Requirements and all related artefacts are documented in a Requirements Specification that needs to be validated regarding customer wishes, correct understanding and accuracy.

Requirements Specification: A Software Requirements Specification (SRS) is a comprehensive description of the intended purpose and environment for software under development. The SRS fully describes what the software will do and how it will be expected to perform.

Requirements Validation: "Ensuring that (1) the set of requirements is correct, complete, and consistent, (2) a model can be created that satisfies the requirements, and (3) a real-world solution can be built and tested to prove that it satisfies the requirements." [124].

Requirements Verification "Ensuring that the system complies with the system requirements and conforms to its design." [124].

Risk: A risk is an event that threatens the success of an endeavour, e.g., of developing or operating a system. A risk is typically assessed in terms of its probability and potential damage [47].

Scenario: A textual description of a small part of a Use-Case that leads to a desired (or undesired) result. A scenario describes a sequence of user actions in general terms.

set A collection of distinct objects.

Soft-Metric: A non-functional description of quality values that cannot be easily measured, but still provide important information for the verification of non- functional requirements, e.g., learnability, understandability.

Software Feature: "A distinguishing characteristic of a software item (e.g., performance, portability, or functionality)" [**?**].

Source: The origin of a requirements artefact (e.g., document, guidelines, law).

Stakeholder: A person or organization that has a (direct or indirect) influence on a systemť's requirements. Indirect influence also includes situations where a person or organization is impacted by the system [47].

Test-Case: A description of possible interactions between actors and the system to be tested for. Every test-case describes some functionality that the system must (or must not) provide for the actors involved in the test-case.

Use-Case: A description of the interactions possible between actors and a system that, when executed, provide added value. Use-cases specify a system from a userŠs (or other external actorŠs) perspective: every use case describes some functionality that the system must provide for the actors involved in the use case [47].

X-Alternative Relationship (XOR): A choice limited to exactly one requirement of a set of requirements.

Confirmation

I confirm that I independently prepared the thesis and that I used only the references and auxiliary means indicated in the thesis.

Dresden, March 13, 2015