# Requirement Specifications Using Natural Languages

Bures, T., Hnetynka, P., Kroha, P., Simko, V.

Charles University
Faculty of Mathematics and Physics
Dep. of Distributed and Dependable Systems

Technical Report D3S-TR-2012-05
December 2012

# Contents

# List of Tables

# List of Figures

## Abstract

We discuss the part of the requirement specification process which is located between the textual requirements definition and the semi-formal diagrams of the requirements specifications.

It concerns the acquisition and the refinement of requirements, the modeling in UML, and the consensus improvement between the analyst and the user.

We point out open problems in this area that include natural language processing (e.g. automatic construction of UML diagrams from a parsed text of requirements), ontologies, constraints, solution scope and optional requirements, requirement inconsistency, querying for completness improvement and refinement of requirements, ambiguity, traceability of requirements, and validation feedbacks.

# Chapter 1

# Introduction

## 1.1 Introduction to Requirements Engineering

Requirements engineering identifies the purpose and properties of a software system. It creats documents in a form that is suitable to analysis, communication, and subsequent implementation [103]. Traceability of requirements, i.e. links between requirements and documents of design and implementation, is an important feature for maintenance of the implemented system.

If a software system has to be built so it has to be described in some way before the analysis, design, and implementation process will be started. Typically, these descriptions (contained in requirement documents) are far from representing the real business logic [11]. Instead, we have a set of statements that is:

- incomplete (forgotten features),
- inconsistent (included contradictions),
- ambiguous (more possible interpretations).

During the last twenty years, standards for measuring and certifying effective software development process have been introduced and popularized. Many books and articles on software development process have been published. Eventhough, many questions remained open:

- How do we explain the high incidence of the software project failure today?
- Why are many, if not most, software projects still plagued by delays, budget overruns, and quality problems?
- How can we improve the quality of the systems we build? It is known that our daily activities become increasingly dependent on them.

The answers are in the people, tools, and processes applied. Requirements management, more exactly its improvement, is often proposed as a solution to the ongoing problems of software development.

A software requirement can be defined as a condition or capability to which the system must conform, i.e. as a software capability needed by the user to solve a problem or achieve an objective. It must be met or possessed by the proposed system or system component to satisfy a contract, specification, standard, or other formally imposed documentation [19].

The decision to describe requirements in documents deserves some thought. On the one hand, writing is a widely accepted form of communication and, for most people, a natural thing to do. On the other hand, the goal of the project is to produce a system, not documents. Common sense and experience teach that the decision is not whether but how to document requirements.

Document templates provide a consistent format for requirements management. For example, the system Rational RequisitePro offers these templates and the additional feature of linking requirements within a document to a database containing all project requirements. This unique feature allows requirements to be documented naturally, making them more accessible and manageable in a database.

There are many problems occurring in the field of requirements engineering. The following list gives some of them:

- Requirements are not always obvious and have many sources.
- Requirements are not always easy to express clearly in words.
- Requirements are not always complete.
- Requirements are not always unique.
- Requirements are not always consistent.
- Requirements do not contain all initial solution boundaries and constraints.
- Many different types of requirements at different levels of detail must be managed.
- The number of requirements can become unmanageable.
- Requirements are related to one another.
- Requirements are neither equally important nor equally easy to meet.
- Many interested and responsible parties are involved in a project, which means that requirements must be managed by cross-functional groups of people.
- Requirements change.
- Requirements can be time-sensitive.

Requirements have many sources. Customers, partners, end users, domain experts, management, project team members, business policies, and regulatory agencies are some sources of requirements. It is important to know how to determine who the sources should be, how to get access to those sources, and how to elicit information from them. The individuals who serve as primary sources for this information are referred to as “stakeholders” in the project.

Requirements may be elicited through activities such as interviewing, brainstorming, conceptual prototyping, using questionnaires, and performing competitive analysis. The result of requirements elicitation is a list of requests or needs that are described textually and graphically and that have been given priority relative to one another.

To define the system means to translate and organize the understanding of stakeholder needs into a meaningful description of the system to be built. Early in system definition, decisions are made on what constitutes a requirement, documentation format, language formality, degree of requirements, request priority and estimated effort, technical and management risks, and scope. Part of this activity may include early prototypes and design models directly related to the most important stakeholder requests. A requirement description may be a written document, electronic file, picture, or any other representation meant to communicate system requirements. The outcome of system definition is a description of the system that is both natural language and graphical. Some suggested formats for the description are provided in later sections.

The scope of a project is defined by the set of requirements allocated to it. Managing project scope to fit the available resources (time, people, and money) is key to managing successful projects. Managing scope is a continuous activity that requires iterative or incremental development, which breaks project scope into smaller, more manageable pieces.

Using requirement attributes, such as priority, effort, and risk, as the basis for negotiating the inclusion of a requirement is a particularly useful technique for managing scope. Focusing on the requirement attributes rather than the requirements themselves helps desensitize negotiations that are otherwise contentious.

With an agreed-upon high-level system definition and a fairly well understood initial scope, it is both possible and economical to invest resources in more refined system definitions. Refining the system definition includes two key considerations: developing more detailed descriptions of the high-level system definition and verifying that the system will comply with stakeholder needs and behave as described.

The descriptions are often the critical reference materials for project teams. Descriptions are best done with the audience in mind. A common mistake is to represent what is complex to build with a complex definition, particularly when the audience may be unable or unwilling to invest the critical thinking necessary to gain agreement. This leads to difficulties in explaining the purpose of the system to people both inside and outside the project team. Instead, you may discover the need to produce different kinds of descriptions for different audiences.

No matter how carefully you define your requirements, they will change. In fact, some requirement change is desirable; it means that your team is engaging your stakeholders. Accommodating changing requirements is a measure of your teams stakeholder sensitivity and operational flexibility, team attributes that contribute to successful projects. Change is not the enemyunmanaged change is.

A changed requirement means that more or less time has to be spent on implementing a particular feature, and a change to one requirement may affect other requirements. Managing requirement change includes activities such as establishing a baseline, keeping track of the history of each requirement, determining which dependencies are important to trace, establishing traceable relationships between related items, and maintaining version control.

A requirement type is simply a class of requirements. The larger and more intricate the system, the more types of requirements appear. By identifying types of requirements, teams can organize large numbers of requirements into meaningful and more manageable groups. Establishing different types of requirements in a project helps team members classify requests for changes and communicate more clearly.

Usually, one type of requirement can be broken down, or decomposed, into other types. Business rules and vision statements can be types of high-level requirements from which teams derive user needs, features, and product requirement types. Use cases and other forms of modeling drive design requirements that can be decomposed to software requirements and represented in analysis and design models. Test requirements are derived from the software requirements and decompose to specific test procedures. When there are hundreds, thousands, or even tens of thousands of instances of requirements in a given project, classifying requirements into types makes the project more manageable.

Unlike other processes, such as testing or application modeling, which can be managed within a single business group, requirements management should involve everyone who can contribute their expertise to the development process. It should include people who represent the customer and the business expectations. Development managers, product administrators, analysts, systems engineers, and even customers should participate. Requirements teams should also include those who create the system solution  engineers, architects, designers, programmers, quality assurance personnel, technical writers, and other technical contributors.

Often, the responsibility for authoring and maintaining a requirement type can be allocated by functional area, further contributing to better large project management. The cross-functional nature of requirements management is one of the more challenging aspects of the discipline.

As implied in the description of requirement types, no single expression of a requirement stands alone. Stakeholder requests are related to the product features proposed to meet them. Product features are related to individual requirements that specify the features in terms of functional and nonfunctional behavior. Test cases are related to the requirements they verify and validate. Requirements may be dependent on other requirements or mutually exclusive.

In order for teams to determine the impact of changes and feel confident that the system conforms to expectations, these traceability relationships must be understood, documented, and maintained. Traceability is one of the most difficult concepts to implement in requirements management, but it is essential to accommodating change. Establishing clear requirement types and incorporating cross-functional participation can make traceability easier to implement and maintain.

Both individual requirements and collections of requirements have histories that become meaningful over time. Change is inevitable and desirable to keep pace with a changing environment and evolving technology. Recording the versions of project requirements enables team leaders to capture the reasons for changing the project, such as a new system release. Understanding that a collection of requirements may be associated with a particular version of software allows you to manage change incrementally, reducing risk and improving the probability of meeting milestones. As individual requirements evolve, it is important to understand their history: what changed, why, when, and even by whose authorization.

## 1.2 Why to Use Natural Language in Requirement Engineering

To get a contract for a large software project, the software house has to work out a feasibility study and a requirements specification. It is part of the offer to the customer that additionally includes schedule and price. Its purpose is to describe all features (functional and non-functional) of the new, proposed system that are necessary to be implemented for the customer to sign the contract.

Requirements are the project team's to-do list. They define what is needed and focus the project team. They are the primary method used to communicate the goals of the project to everyone on the team.

Requirements define what the stakeholders need and what the system must include to satisfy the stakeholders' needs. Requirements are the basis for capturing and communicating needs, managing expectations, prioritizing and assigning work, verifying and validating the system (acceptance), and managing the scope of the project.

Requirements may take different forms, including scenarios, unstructured text, structured text, or a combination, and they may be stated at different levels of granularity. At the highest level of granularity, features define the services that the system must provide to solve the customer's problem. These are captured as structured or unstructured text in the project vision. At the next level of granularity, use cases can be used to define the functionality that the system must provide to deliver the required features. Use cases describe the sequence of actions performed by the system to yield an observable result of value.

As mentioned, a system must perform according to the behavior that can be specified as use cases. However, there are system requirements that do not represent a specific behavior, also known as system-wide requirements, including:

- Legal and regulatory requirements, as well as application standards
- Quality attributes of the system to be built, including usability, reliability, performance, and supportability requirements
- Interface requirements to be able to communicate with external systems
- Design constraints, such as those for operating systems and environments and for compatibility with other software

Formal specification is ideal for the software developer, but it is not reasonable to require the author of the requirements document, who is seldom familiar with formal methods or even with the concept of specification, to provide a formal description. State of the art are informal requirements documents written in natural language.

Detailed software requirements should be written in such a form as can be understood by both the customers and the development team.

Many solutions (e.g., KaOS [89]) require users to write requirements in formal notations. KaOS presents an approach for using Semantic nets and temporal logic for formal analysis of requirements using a goal based approach. While they can perform a broad range of reasoning, their system needs the requirements to be inputted in a formal language. This restriction is not feasible in practice because requirements documents are written by semi-technical analysts and have to be signed-off by business executives. Hence, the communication medium is still natural language.

We have found that using the language of the customer to describe these software requirements is most effective in gaining the customers understanding and agreement. These detailed software requirements are then used as input for the system design specifications as well as for test plans and procedures needed for implementation and validation. Software requirements should also drive the initial user documentation planning and design.

Using a natural language is necessary because:

- Requirement specification are written by a software house analyst in cooperation with customer's experts and potential users. They do very probably not understand any more formal specification as a specification in natural language.
- A customer would not sign a contract where requirements specification is written e.g. only in the Z notation.

Once a contract has been awarded and after a feasibility study has been approved, a requirements specification must be written in more details that describes the properties of the new system, i.e. functional properties, non-functional properties, and constraints, in a more detailed way.

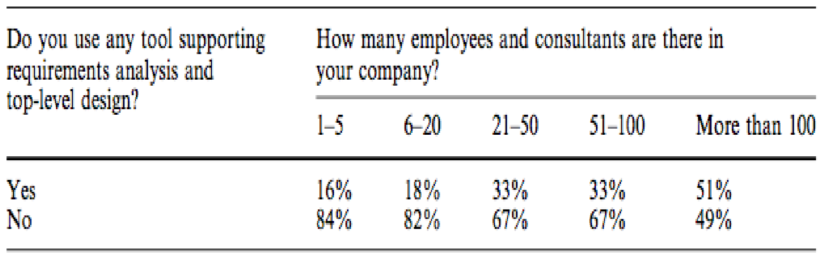
Very often, it will be written in some semi-formal graphical representation given by the CASE tool that is used in the software house.

The role of using natural language in requirement specifications is investigated in [100]. The statistics published in this paper shows that the market was open for more wide use of requirements engineering systems in the year 2003 - see Fig. 1.1. In Fig. 1.2 in [100], we can see that the percentual part of requirements described in natural language makes 79 %. In Fig. 1.3 in [100], the need of requirement engineering automation is documented.

We can conclude that the use of linguistic techniques and tools may perform a crucial role in providing support for requirements analysis.

It has been found that in a majority of cases it is necessary to use NLP systems capable of analysing documents in full natural language. If the language used in the documents is controlled (giving a subset of natural language), it is possible to use simpler and therefore less costly linguistic tools, which in some cases are already available. Instruments of this type can also be used to analyse documents in full natural language, even if in this case more analyst consultation is required to reduce the complexity of the language used in input documents or to intervene automatically in the models produced as output. Moreover, needed in many cases, besides an adequate representation of the shared/common knowledge, is specialised knowledge of the domain.

Effective requirements management includes the following project team activities:



| Do you use any tool supporting requirements analysis and top-level design? | How many employees and consultants are there in your company? | | | | |
|---|---|---|---|---|---|
| | 1–5 | 6–20 | 21–50 | 51–100 | More than 100 |
| Yes | 16% | 18% | 33% | 33% | 51% |
| No | 84% | 82% | 67% | 67% | 49% |

Figure 1.1: Use of tools for requirements analysis [100]

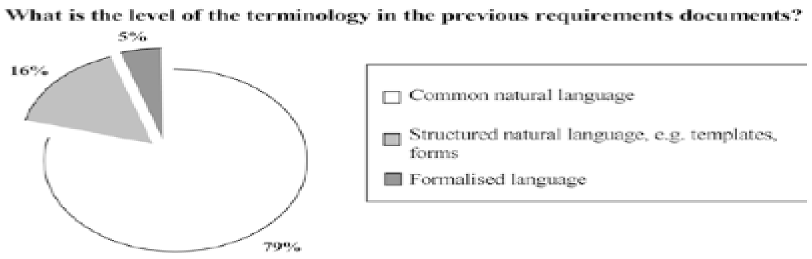
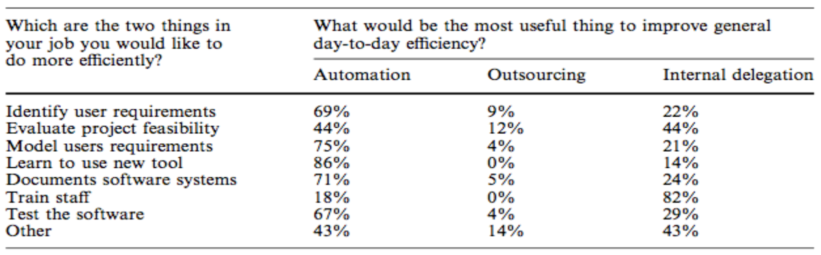


Figure 1.2: Using natural languages in requirement specifications [100]

- Agree on a common vocabulary for the project.
- Develop a vision of the system that describes the problem to be solved by the system, as well as its primary features.
- Elicit stakeholders needs in at least five important areas: functionality,usability, reliability, performance, and supportability.
- Determine what requirement types to use.
- Select attributes and values for each requirement type.
- Choose the formats in which requirements are described.
- Identify team members who will author, contribute to, or simply view one or more types of requirements.
- Decide what traceability is needed.
- Establish a procedure to propose, review, and resolve changes to requirements.
- Develop a mechanism to track requirement history.
- Create progress and status reports for team members and management.



| Which are the two things in your job you would like to do more efficiently? | What would be the most useful thing to improve general day-to-day efficiency? | | |
|---|---|---|---|
| | Automation | Outsourcing | Internal delegation |
| Identify user requirements | 69% | 9% | 22% |
| Evaluate project feasibility | 44% | 12% | 44% |
| Model users requirements | 75% | 4% | 21% |
| Learn to use new tool | 86% | 0% | 14% |
| Documents software systems | 71% | 5% | 24% |
| Train staff | 18% | 0% | 82% |
| Test the software | 67% | 4% | 29% |
| Other | 43% | 14% | 43% |

Figure 1.3: Efficiency of software development process [100]

These essential requirements management activities are independent of industry, development methodology, or requirements tools. They are also flexible, enabling effective requirements management in the most rigorous and the most rapid application development environments.

Since software engineers are not specialists in the problem domain, their understanding of the problem is immensely difficult, especially if routine experiences cannot be used. It is a known fact [11] that projects completed by the largest software companies implement only about 42 % of the originally-proposed features and functions.

We argue that there is a gap between the requirements specification in a natural language and requirements specification in some semi-formal graphical representation. The analyst's and the user's understanding of the problem are usually more or less different when the project starts. The step from the requirements definition towards requirements specification will be done only in the brain of the analyst without being documented.

Usually, the user cannot deeply follow the requirements specification. The first possible time point when the user can validate the analyst's understanding of the problem, i.e. the first possible feedback, is when a prototype starts to be used and tested.

The phase of natural language requirement specifications pre-processing offers a feedback very soon, before the design and implementation started.

Using an appropriate tool during elicitation of requirements specification in natural language we can check and test requirement specifications during their genesis. This is the first feedback possible.

One approach of analyzing requirements is to treat them as a form of pseudo-code, or even a very high-level language, through which the requirements analyst is essentially beginning to program the solution envisioned by the stakeholders. In such a case, it would be possible to build tools that analyze requirements, just like compilers analyze software programs. This observation has been shared by a number of researchers in this space and a number of tools have been proposed (comprehensive survey of tools is available at [8]).

Errors, which arise from incorrect requirements, have become a significant development problem. Requirements errors are numerous: they typically make up 25 % to 70 % of total software errorsUS companies average one requirements error per function point [51]. They can be persistent: twothirds are detected after delivery. They can be expensive. The cost to fix them can be up to a third of the total production cost [5]. Moreover, many system failures are attributed to poor requirements analysis [52].

# Chapter 2

# Natural Language Processing

For using natural language processing in requirement specification analysis, it is recommanded to write sentences of requirements systematically in a consistent fashion starting with the agent/actor, followed by an action verb, followed by an observable result.

## 2.1 Tokens

The first step in the natural language analysis is a tokenization. This is a process that identifies words and numbers in sentences. It is necessary to specify what is the sentence delimiter.

## 2.2 Part-Of-Speech tagging

Part-Of-Speech-Tagging (POS Tagging) is the process of marking up a word in a text as corresponding to a particular part of speech, based on both its definition, as well as its context - i.e. relationship with adjacent and related words in a phrase, sentence, or paragraph. A simplified form of this is commonly taught to school-age children, in the identification of words as nouns, verbs, adjectives, adverbs, etc.

In computer science, this topic is investigated by computational linguistics. Part-of-speech tagging is harder than just having a list of words and their parts of speech, because some words can represent more than one part of speech at different times, and because some parts of speech are complex or unspoken. This is not rarein natural languages (as opposed to many artificial languages), a large percentage of word-forms are ambiguous.

The Penn Treebank Tagset for English [98] contains 36 tags. Currently, the most promissing tool we use is Stanford Parser [99].

## 2.3 Parsing

Parsing is the process that determines the parse tree (grammatical analysis) of a given sentence. The grammar for natural languages is ambiguous and typical sentences have multiple possible analyses. For a typical sentence there may be very many of potential parses. Most of them will seem completely nonsensical to a human but it is difficult to decide over their sense algorithmically.

## 2.4 Rule-based natural language processing

Prior implementations of language-processing tasks typically involved the direct hand coding of large sets of rules. It is possible to use methods of machine learning to automatically learn such rules through the analysis of large corpora of typical real-world examples. A corpus (plural, ”corpora”) is a set of documents that have been hand-annotated with the correct values to be learned.

The problem is that the rules are ambiguous. We will discuss it more in details in Section 12.

## 2.5 Statistical natural language processing

Statistical natural language processing uses statistical methods to resolve some of the difficulties discussed above, especially those which arise because longer sentences are highly ambiguous when processed with realistic grammars, yielding thousands or millions of possible analyses.

Methods for disambiguation often involve the use of corpora and Markov models. The technology for statistical NLP comes mainly from machine learning and data mining, both of which are fields of artificial intelligence that involve learning from data.

Textual documents of requirements (use cases, scenarios, user stories, transcriptions of conversations for requirements elicitation - often denoted as textual requirements descriptions) can reach several hundred of pages in large projects. Because of that it is useful to process the documents in some related groups obtained by semantic filtering. It is possible to extract sentences relevant for concept relationships, temporal organisation, control, causality [28].

# Chapter 3

# Modeling Static Structures from Requirements

In requirements, parts of UML model can be recognized and identified. It can be done partially automatically and partially manual. Class diagram (class hierarchy and associations), attributes, and methods (their existence) are parts of the UML model that describe the static structure.

Their identification can be done automatically with some necessary human interaction because of the complexity of real world semantics. The used method is based on the grammatical inspection [1]. This method offers a judgement saying that nouns in sentences of requirement specifications may have a link to classes in the corresponding UML model, verbs may represent methods or relationships, and adjectives may represent a value of an attribute. Because of the natural language complexity, this method works satisfactorily only when using a humen interaction.

Some help can be done by applying of part-of-speech tagging analysis, and searching for templates in the tree structure that represents the parsed sentence. Some experiments we made using TESSI are described in Section 16.8.4.

# Chapter 4

# Modeling Dynamic Structures from Requirements

Parts of UML model that contain sequence diagram (objects, messages), state diagram (states, transitions, events), collaboration diagram, and activity diagram describe the dynamic structures, i.e. the behavior.

The identification of these structures from requirements is more complex.

Formalization of textual behavior description can reveal deficiencies in requirements documents. Formalization can take two major forms:

- based on interaction sequences (translation of textual scenarios to interaction sequences (Message Sequence Charts, or MSCs) was presented in works [67], [68], [69]),
- based on automata (survey in [66]).

To close the gap and to provide translation techniques for both formalism types, an algorithm translating textual descriptions of automata to automata themselves is necessary [60].

# Chapter 5

# Constraints

Constraints, which are part of requirements and later parts of the UML model, describe restriction rules of requirements restricting both static structure (e.g. range of attribute values) and dynamic structure (limits of behavior).

Constraints can be inconsistent. There are many reasons for that, e.g. requirements are written by many analysts, an analyst cannot recognize that some assertions written by himself are contradictory either among each other or to domain assertions.

Constraints are described in OCL (Object Constraint Language) which is stronger than SWRL (Semantic Web Rule Language) used in Web engineering. As we will describe below, description logics have different expresiveness. The reason is that the computational complexity of the reasoning, i.e. the computational complexity of the decision whether the system is correct and consistent, may explode and we never obtain the result guarenteed if the expressivenes of the used description logic is too high.

The related work is discussed in more details in Section Inconsistency 11 and in Section Related Work to Checking Inconsistency 16.3.

# Chapter 6

# Using Ontologies

## 6.1 Ontologies

An ontology is a specification of a conceptualization [36]. It makes possible to describe a domain including all its concepts with their attributes and relationships. As a standard description formalism, the OWL language [144], [145] will be used that is based on RDF (Resource Description Framework) [146], [147], [148], [149].

Additionally to RDF, OWL makes possible to make reasoning by inference machine about elements of ontologies (classes, properties, individuals, relationships). OWL has three variants (sublanguages) that differ in levels of expressiveness. These are OWL Lite, OWL DL and OWL Full (ordered by increasing expressiveness). Each of these sublanguages is a syntactic extension of its simpler predecessor.

OWL Lite was originally intended to support those users primarily needing a classification hierarchy and simple constraints.

OWL DL was designed to provide the maximum expressiveness possible while retaining computational completeness, decidability (there is an effective procedure to determine whether an assertion is derivable or not), and the availability of practical reasoning algorithms. OWL DL is so named due to its correspondence with description logic, a decidable subset of predicate logic of the first order.

OWL Full is based on a different semantics from OWL Lite or OWL DL, and was designed to preserve some compatibility with RDF Schema. OWL Full allows an ontology to augment the meaning of the pre-defined (RDF or OWL) vocabulary. It is unlikely that any reasoning software will be able to support complete reasoning for OWL Full.

Description logics (DLs) are a family of logics that are decidable fragments of first-order logic with attractive and well-understood computational properties. OWL DL and OWL Lite semantics are based on DLs. They combine a syntax for describing and exchanging ontologies, and formal semantics that gives them meaning. Reasoners (i.e. systems which are guaranteed to derive every consequence of the knowledge in an ontology) exist for these DLs.

Querying in ontologies is based on querying in RDF-graphes implemented in a query language SPARQL (Simple Protocol and RDF Query Language) [150], [151].

## 6.2 Terminology of Ontologies

Languages in the OWL family are capable of creating classes, properties, defining instances and its operations.

A class is a collection of objects. It corresponds to a description logic (DL) concept. A class may contain individuals, instances of the class. A class may have any number of instances. An instance may belong to none, one or more classes. A class may be a subclass of another, inheriting characteristics from its parent superclass. This corresponds to logical subsumption and DL concept inclusion notated . All classes are subclasses of owl: Thing (DL top notated ), the root class. All classes are subclassed by owl: Nothing (DL bottom notated ), the empty class. No instances are members of owl: Nothing. Modelers use owl: Thing and owl: Nothing to assert facts about all or no instances.

An instance is an object. It corresponds to a description logic individual.

A property is a directed binary relation that specifies class characteristics. It corresponds to a description logic role. They are attributes of instances and sometimes act as data values or link to other instances. Properties may possess logical capabilities such as being transitive, symmetric, inverse and functional. Properties may also have domains and ranges.

Datatype properties are relations between instances of classes and RDF literals or XML schema datatypes.

Object properties are relations between instances of two classes.

Languages in the OWL family support various operations on classes such as union, intersection and complement. They also allow class enumeration, cardinality, and disjointness.

## 6.3 Inference in Ontologies

Inference in ontologies is based on concepts developed in Description Logic (DL) and frame-based systems and is compatible with RDFS (RDFS is a general-purpose language for representing simple RDF vocabularies on the Web [112]).

Systems providing inference in ontologies, e.g. OntoSem, use the following knowledge resources [80]:

- ontology, a language-independent tangled hierarchy (lattice) of concepts, each with a set of properties, representing the theory of the world,
- lexicons for specific natural languages, with most lexical entries anchored in an ontological concept, often with the constraints on their properties,
- lexicons for proper names for specific natural languages,
- language-independent text-meaning representation (TMR) language for representing the meaning of a text in ontological terms;

- fact repository (FR), the database of recorded TMRs.

The inference process consists of expanding and subsequent matching of TMR modules corresponding to input-text TMR (TMRI) and query TMR (TMRQ).

### 6.3.1 Why to Use Ontology for Checking Requirements Specifications

Ontologies seem to be the right tool because they are designed to capture natural language descriptions of domains of interest. An ontology consists of:

- Description part - a set of concepts (e.g. entities, attributes, processes), their definitions and their inter-relationships. This is referred to as a conceptualization. Here, ontology represents the domain knowledge (domain ontology) and requirements can be seen as a specialized subset of it (as problem ontology in our text).
- Reasoning part - a logical theory that constrains the intended models of logical language containing:
  - integrity rules of the domain model representing the domain knowledge,
  - derivation rules and constraint rules of the problem model.

  Reasoning in ontologies brings the inferential capabilities that are not present in taxonomies used for modeling formerly. It makes possible to search for contradictions that indicate inconsistencies.

Requirements are based on knowledge of domain experts and users' needs and wisches. One possible way to classify this knowledge and then fashion it into a tool is through ontology engineering.

Ontologies are specifications of a conceptualization in a certain domain. An ontology seeks to represent basic primitives for modeling a domain of knowledge or discourse. These primitives are typically concepts, attributes, and relations among concept instances. The represented primitives also include information about their meaning and constraints on their logically consistent application. A domain ontology for guiding requirements elicitation depicts the representation of knowledge that spans the interactions between environmental and software concepts. It can be seen as a model of the environment, assumptions, and collaborating agents, within which a specified system is expected to work. From a requirements elicitation viewpoint, domain ontologies are used to guide the analyst on domain concepts that are appropriate for stating system requirements.

Ontologies can be seen as explicit formal specifications of the terms in the domain and relationships among them. They care for a shared understanding of some domain of interest [141]. Such an understanding can serve as the basis for communication in requirements development. Ontologies are a guarantee of consistency [36] and enable reasoning. An ontology-based requirements specification tool may help to reduce misunderstanding, missed information, and help to overcome some of the barriers that make successful acquisition of requirements so difficult.

Simplified, ontologies are structured vocabularies having possibility of reasoning. It includes definitions of basic concepts in the domain and relations among them. It is important that the definitions are machine-interpretable and can be processed by algorithms.

Why would someone want to develop an ontology?

Some of the reasons are:

- To share common understanding of the structure of information among people or software agents
- To enable reuse of domain knowledge
- To make domain assumptions explicit
- To separate domain knowledge from the operational knowledge
- To analyze domain knowledge

Currently, ontology research has primarily focused more on the act of engineering ontologies or it has been explored more for use in domains other than requirements elicitation, specification, checking, and validation. Other interesting papers in this field are [78], [24], [25].

## 6.4 The Idea of Checking Requirements

In ontology-based requirements engineering, the correctness, completeness, consistency and unambiguity of ontology should be guaranteed so far that it can be used to guide the requirements elicitation and requirements evolution(!). As we will show below, the guarancy is difficult, especially the guarancy of completeness. The requirements evolution will often be forgotten but the never ending changing of environment causes that requirements change and than software systems have to be changed constantly. In this context, the traceability of requirements evolution and the traceability of requirements implementation is very important.

The goal is not only to develop requirements specification but to create (or to have available) a domain ontology in every project before the requirements specification process will be started. Software houses are usually specialized on producing software systems that solve a specific set of problems, e.g. information systems for financial institutions. Therefore the objective is to have an ontology domain available for a given field of applications and to check requirements of all projects being developed for this field.

For an ontology being succesfully used in requirements checking, it has to have the following properties: completness, correctness, consistency, and unambiguity.

The intuitive meaning is:

- correctness means that the knowledge in ontology do not violate the rules in domain that correctly represent the reality,

- consistency means that there are no contradictory definitions in ontology,
- completeness means that the knowledge in ontology describes all aspects of the domain,
- unambiguity means that the ontology have defined an unique or unambiguous terminology. There are not obscure definitions of concepts in ontology, i.e. each entity is denoted by only one, unique name, all names are clearly defined and have the same meaning for the analyst and all stakeholders.

Correctness and consistency are logical properties that can be checked by some reasoning mechanism under assumption that this mechanism works correctly.

This is what we can use for improving the quality of requirements. After we have checked the correctness and consistency of the corresponding domain ontology (domain knowlegde) we can check whether the modeled requirements (transformed into an ontology) are correct and consistent mutual and correct and consistent to the given domain ontology.

We cannot be sure that our ontology is complete but we can suppose that it is close to be complete if it has been used succesfully in many applications. The situation is similar to the problem of library package verification.

Our goal is to use an ontology in requirements engineering so we have to say what completness of requirements means.
Completeness for requiremens means that:

- all categories of requirements (functional and non-functional requirements) are addressed,
- all responsibilities allocated from higher-level specifications are recognized,
- all use cases, all scenarios, and all states are recognized,
- all assumptions and constraints are documented.

This is a good intuitive definition but it is not constructive. We cannot to use it to decide whether our requirements are complete.

The problem is in the "'all"', of course. What we can do is to hope that the domain ontology is "'more complete"' than the developed requirements so that we can check the requirements by comparing them with the ontology and find (perhaps) that there are some aspects described in the ontology but not described in the requirements.

Ontologies can be used:

- to specify classes and properties of object that should be found in the textual documents of requirement specifications [154]
- to check by reasoning in descriptive logics whether the problem ontology specified by requirement specifications is a subset of the domain-specific ontology that is common for all application in the domain [87]

In [87], we described the last version of TESSI that was constructed to support the following processes:

- building a domain ontology in OWL by using Protege (very briefly),
- checking a domain ontology for corretness and consistency by using Racer and Jess,
- building a UML model of requirements from textual description of requirements,
- conversion of requirements described as a UML model to a requirements ontology in OWL and its limits,
- checking requirements tranformed into the requirements ontology for mutually correctness and consistency checking and for checking with respect to the domain ontology,
- identifying correctness and consistency problems,
- finding the corresponding parts in the former textual description of requirements and correcting them,
- building a new UML model based on corrected textual description of requirements,
- after iterations when no ontology conflicts will be found a new textual description of requirements will be automatically generated that corresponds to the last UML model,
- before the UML model will be used for design and implementation the customer and the analyst will read the generated textual description of requirements and look for missing features or misunderstandings,
- problems found can start the next iteration from the very beginning,
- after no problems have been found the UML model in form of a XMI-file will be sent to Rational Modeler to further processing.

# Chapter 7

# Querying to Improve Completeness

Using domain ontology, there seems to be a possibility to generate some requests automatically that should complete the gaps between knowledge stored in the domain ontology and the problem ontology that corresponds to the requirement specifications. Currently, this is an open problem.

The querying supported in the tool Rational RequisitePro makes only possible that the analyst can ask for existence and content of a requirement.

# Chapter 8

# Ontology and the Cyc Platform

In practice, some ontologies are available. The possibility should be investigated how to use them in linking ontologies and requirements engineering.

Cyc [13] is the world's largest and most complete general knowledge base and common sense reasoning engine is available.

Cyc can be used as the basis for a wide variety of intelligent applications such as :

information extraction and concept tagging, content/knowledge management, business intelligence, support of analysis tasks, semantic database integration, natural language understanding and generation, rapid ontology and taxonomy development, learning and knowledge acquisition, filtering, prioritizing, routing, summarization, and annotating of electronic communications.

The latest release of Cyc includes:

- 500,000 concepts, forming an ontology in the domain of human consensus reality,
- nearly 5,000,000 assertions (facts and rules), using 26,000 relations, that interrelate, constrain, and, in effect, (partially) define the concepts,
- a compiled version of the Cyc Inference Engine and the Cyc Knowledge Base Browser,
- natural language parsers and CycL-to-English generation functions,
- a natural language query tool, enabling users to specify powerful, flexible queries without the need to understand formal logic or complex knowledge representations,
- an Ontology Exporter that makes it simple to export specified portions of the knowledge base to OWL files,
- documentation and self-paced learning materials to help users achieve a basic- to intermediate-level understanding of the issues of knowledge representation and application development using Cyc,

- a specification of CycL, the language in which Cyc (and hence ResearchCyc) is written (there are CycL-to-Lisp, CycL-to-C, etc. translators),
- a specification of the Cyc API, by calling which a programmer can build an ResearchCyc application.

# Chapter 9

# Concepts

Related words may build specific clusters called concepts. The relationship is usually given by similar statistical properties of these words, e.g. their frequency of occurence in one sentence, in one document. It is known from information retrieval and from text mining how to find concepts using term-document matrix and its singular decomposition.

In requirements engineering, we can meet the problem when more people write requirements and use different vocabulary. Concerning paper is[71].

# Chapter 10

# Completeness

Requirements may be incomplete. This problem is caused by missing some requirements of a certain type. For example, it is often the case that performance requirements for a system are omitted either due to the lack of knowledge among the stakeholders or because the requirements analysts fail to elicit them.

This leaves the technical designers and developers to make design choices about the software system, which may or may not meet the stakeholders approval.

Another case is that some system features are not mentioned by the stakeholders because they think everybody knows them including the analyst. Often, it is not the case and the analyst works having incomplete requirements. Concerning paper is [75].

# Chapter 11

# Inconsistency and Contradictions

Requirements may be inconsistent which means that they are either conflicting with each other or with some policy or business rule (contained e.g. in domain rules, in domain ontology). Because of that terms should be used consistently and as defined in the glossary. Different phrases or words should not refer to the same thing [73], [108].

In [86], [87], we investigated how the methods developed for using in Semantic Web technology could be used in validating of requirements specifications. The goal of our investigation was to do some (at least partial) checking and validation of the UML model using a predefined domain-specific ontology in OWL, and to process some checking using the assertions in descriptive logic.

We argue that the feedback caused by the UML model checked by ontologies and OWL DL reasoning has an important impact on the quality of the outgoing requirements.

The paper [87] describes not only methods but also implementation of our tool TESSI (in Protégé, Pellet, Jess) and practical experiments in consistency checking of requirements.

## 11.1 Implementation

As we already mentioned above we needed to implement:

- converting UML model into problem ontology model,
- checking ontology class hierarchy,
- checking consistency of ontology rules.

The component of TESSI containing the ontology-based consistency checking of requirements specification has been implemented in [44].

### 11.1.1 Using ATL for Converting UML to OWL

Our goal was to convert the UML model obtained from the textual requirements into a corresponding problem ontology model that can be compared with the domain ontology model. The comparison results in consideration whether some new knowledge concerning the correctness, consistency, completness, and unambiguity could be made.

There are some tools available. The UMLtoOWL tool by Gasevic [**?**] converts UML model description in extended Ontology UML Profile (OUP) using the XML Metadata Interchange (XMI) format to The Web Ontology Language (OWL) ontologies. The tool is implemented using eXtensible Stylesheet Language Transformation (XSLT).

We have used the Eclipse Framework and the ATL Use Case UML2OWL by Hillairet [38]. He implemented a transformation according to the ODM specification. It consists of two separate ATL transformations. The first transformation UML2OWL takes an UML model as input and produces an ontology as OWL metamodel. The second transformation is an XML extractor that generates an XML document according to the OWL/XML specification by the W3C. We have extended Hillairets scripts to fit the UML models of TESSI and added support for SWRL contraints. These constraints are converted to SWRL/XML syntax to fit inside the OWL. This is done by an ANTRL parser and compiler that can convert SWRL rules in informal syntax entered in TESSI into the correct OWL/SWRL syntax. The use of SWRL rules provides us further posibilities for checking our model.

### 11.1.2 Using Pellet for Checking Ontology

Pellet is a tool that allows ontology debugging in the sense that it indicates the relation between unsatisfiable concepts or axioms that cause an inconsistency. We use it to check whether the requirements problem ontology subsumes the domain ontology. Because our problem ontology is generated from the UML model by the convertor ATL, there are no problems to be expected in the structure of the problem ontology because the UML model has been built under respecting rules for well-formed UML model.

The OWL files generated in the previous step can be loaded into Protégé. From there they can be transferred to a reasoner using the DIG description logic reasoner interface. The DIG interface is an emerging standard for providing access to description-logic reasoning via an HTTP-based interface to a separate reasoning process. Current releases of Protégé already include the Pellet reasoner, since it is robust and scalable, and is availabl under an open-source license.

### 11.1.3 Using Jess for Reasoning in Ontology

To find inconsistencies in ontology rules we need an inference machine. We used the Jess rule engine [23]. Jess was inspired by the CLIPS expert shell system and adds additional access to all the powerful Java APIs for networking, graphics, database access, and so on. Jess can be used free of charge for educational purposes. Because Protégé and Jess are implemented in Java, we can run them together in a single Java virtual machine. This approach lets us use Jess as an interactive tool for manipulating Protégé ontologies and knowledge bases.

Protégé offers two ways to communicate with Jess. The first one is the plugin JessTab, which provides access to the Jess console and supports manual mapping of OWL facts into Jess and back. We used the second plugin SWRLTab. It is a development envirionment for SWRL rules in Protégé and supports automatical conversion of rules, classes, properties and individuals to Jess. From there you can control the Jess console and look up the outputs. This is done by a plugin for SWRLTab called SWRLJessTab, which contains a SWRL to Jess bridge and a Jess to Java bridge. This allows the user to add additonal functions to their SWRL rules by defining the corresponding functions as Java code and use them inside Protégé. SWRLTab lets you also insert the inferred axioms back into your ontology. This way it is possible to use complex rules to infer new knowledge.

### 11.1.4 Interaction of the used tools

All these described tools are put together during the requirement analysis. Figure 11.1 shows how this is done. Starting with the textual description and an ontology describing the domain the analyst can use TESSI to create a model of the planned system. This model can also contain constraints which will be compiled into SWRL rules by an ANTLR parser. The rest of the model will be transformed into an UML model [152], which will later be converted into an ontology. The ontology is merged with the SWRL rules and can then be opened in Protégé. From there the analyst can check the model consistency with Pellet and validate the rules with SWRLTab and Jess. The knowledge gained will then be used to make corrections to the TESSI model.

## 11.2 Experiments

For experiments, we used a requirements specification of a library [87]. The text describes fictional requirements for a library management system. It contains aspects of media and user management, describing several special use cases and state machines for selected classes. The text was developed to show the posibilities TESSI provides for requirement analysis.

### 11.2.1 Checking with rules

As an example of checking rules, we have the following case. There is a relation ”borrow” between Librarian and User. But if we model Librarian as a subset of class User, because a librarian may also borrow books, the Librarian (as an instance) could borrow a book himself. This is not what we want. Usually, we do not allow that a clerk in a bank can give a loan to himself, we do not want that a manager decides about his salary etc. The solution is that we do not allow some relations to be reflexive in the domain ontology, e.g. the relation ”borrow”. Any problem ontology that does not contain the condition that a librarian must not borrow a book to himself will be found to be inconsistent to the domain ontology.

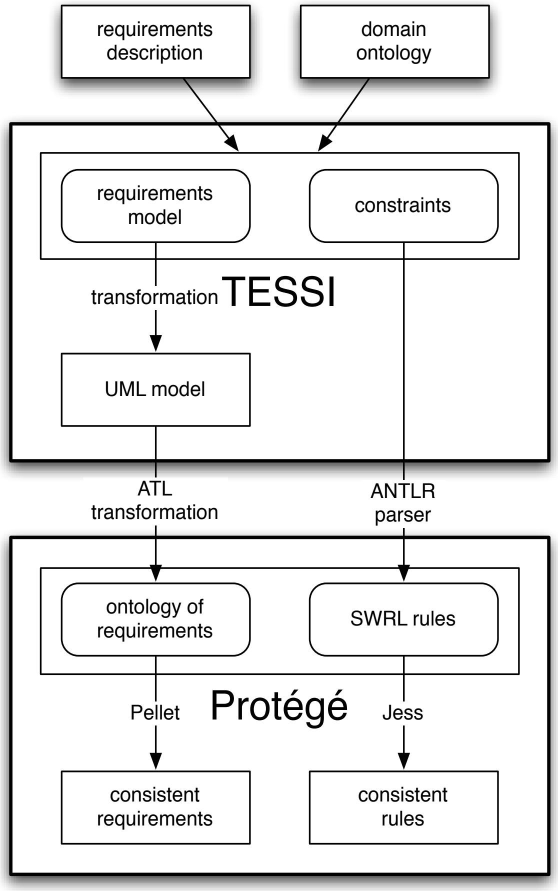


Figure 11.1: Interaction of the used tools

This example can be checked in TESSI by modelling the two classes User and Librarian. We decide that a Librarian is a specialization of an User with addional possibilities to manage the library. Then we define an association between these two and name the direction from Librarian to User borrowsTo. After that we can use SWRL-Rules to describe the desired behavior. The first rule we need will set the relation between every possible Librarian and User pair:

$$\text{Librarian}(?x) \wedge \text{User}(?y) \rightarrow \text{borrowsTo}(?x, ?y)$$

The second rule will be used to check if any of the librarians is able to borrow a book to himself:

$$\text{borrowsTo}(?x, ?y) \wedge \text{sameAs}(?x, ?y) \rightarrow \text{error}(?x, \text{"self"})$$

Now we can create an UML model based on our example and then generate an ontologie with this content. The ontology will then be loaded into Protégé.

The Protégé plugin SWRLTab offers several ways to work with SWRL rules. It also allows us to communicate with the Jess rule engine. Using this plugin we can transform the knowledge of the ontology in facts for Jess. Running Jess will then cause new facts to be inferred.

In our case it will set up the borrowsTo relationship for all Users and Librarians and then test for Librarians that borrow to theirselves. The Inferred Axioms window in Protégé will then list all possible errors and we can use this information to make corrençtions to the model in TESSI. In this case we can remove the subclass from User and after a further test Jess will get no errors.

### 11.2.2 Checking with restrictions

A second example will show the posibility to check restrictions. In our library a user can borrow books or reserve them if they are not available. In order to limit users to a fixed amount of reservations the reserve relation should be restricted.

In TESSI these conditions can be modeled with associations. Therefor we use the artifact dialog for associations to create a new Instance at the corresponding position in the requirements text. We set User and MediumInstance as association ends. The direction from User to MediumInstance will be labeled with reservedMedia an gets the cardinality 0 to $n$, in this example $n$ is set to 3. Both classes User and MediumInstance must have set some equivalents in the domain ontology to access the corresponding individuals later. To provide some test data we need to add a constraint that fills the reservedMedia relation:

$$\text{User}(?x) \wedge \text{MediaInstance}(?y) \rightarrow \text{reservedMedia}(?x, ?y)$$

After converting the model to UML and to an ontology we use the SWRLTab to infer the new axioms and then use the Jess to OWL button to include the new knowledge into our ontology. Afterwards we can check the results on the individuals tab in Protégé. It will show red borders around properties which do not meet the defined restrictions.

Based on these observations either the restrictions must be corrected or the test data is wrong and the constraint for filling it must be adopted.

One of the problems that may occur is that the restriction rules of requirements (called constraints) are described in OCL (Object Constraint Language) which is stronger than SWRL. As we will describe below description logics have different expresiveness. The reason is that the computational complexity of the reasoning, i.e. of the decision whether the system is correct and consistent, may explode and we never obtain the result guarenteed if the expressivenes of the used description logic is too high.

An other problem is the necessarily use of individuals to process SWRL rules. This requires to add several individuals of every class the the domain ontology whithout knowing what rules will later be modeled in TESSI. It also requires to have some meaningful properties set to these objects. Otherwise it will not be possible to validate the model with SWRL rules.

SWRL also offers only limited possibilities to express rules. The formulas are based on first oder logic but can only contain conjunctions of atomic formulas. There is no support for quantiviers or more complex terms. SWRL also can't express negations, which requires the user to create formulas on a special way and limits the expressiveness of SWRL rules.

Ontology research has primarily focused on the act of engineering ontologies or it has been explored for use in domains other than requirements elicitation, specification, checking, and validation. Using ontologies supports consistency which is critical to the requirements engineering process. Consistent understanding of the domain of discourse reduces ambiguity and lessens the impact of contextual differences between participants.

# Chapter 12

# Ambiguity

Text information usually enables more than one interpretation. To find the one correct interpretation that should be programmed we use the context and interaction with the customer or domain expert. This is very problematic for automatic systems, of course [70], [32], [7], [54]. To describe ambiguity we distinguish syntactic similarity and sematic similarity of words.

## 12.1 The Syntactic Similarity

Usually, when people talk about the similarity of words, they mean semantic similarity (e.g. synonomy). However, it is also useful to think about the syntactic similarity of words, i.e. how similar are two words with respect to their syntactic function or role? You can think of traditional part-of-speech tags as a coarse theory of syntactic similarity, e.g. all personal pronouns have similar syntactic roles. Still, it would be nice to have a quantitative measure of the exact degree of syntactic similarity between two words.

The method explored in [35] (and a similar method described in [125]) proposes to compute syntactic similarity as the cosine distance between the syntactic behavior of words represented as normalized feature vectors of the frequency of unique parse tree paths in large corpora of syntactically parsed text.

The syntactic similarity will be used in solving the plagiarism problem. The problem of detecting web documents that have some similarity degree with a given input document is very important. Search engines avoid indexing similar documents in their document bases, people wish to find documents that originated an input text, or even detect plagiarism between several documents obtained from the Web, among others [107].

The syntactic ambiguity is given by ambiguity of the language structure. In [41], the following example is given: The cop saw the robber with the binoculars. This sentence could mean either the cop was using the binoculars, or the robber had binoculars. In these cases, the discourse level of information is needed.

The disadvantage of syntactic similarity is that two sentences having the same words in different order can have a high syntactic similarity but a completely different meaning. Because of that semantic similarity will be used even though the syntactic similarity can be easier computed and in many cases can bring good results.

## 12.2 The Semantic Similarity

Determining semantic similarity of two sets of words that describe two entities is an important problem in web mining (search and recommendation systems), targeted advertisement and domains that need semantic content matching [138]. Usually, the content of documents is represented using the model ”bag of words”. This causes that relationships that are not explicit in the representations are usually ignored. Furthermore, these mechanisms cannot handle entity descriptions that are at different levels of granularity or abstractions as the implicit relationship between the concepts is ignored.

To define semantics of a word the same words of two sentences have to be compared included their context.

In [95], an algorithm is given that can solve this problem for English and for database WordNet4. After the semantics of words in both sentences will be specified, the sematic similarity can be calculated based on distance metrics [131], [4].

# Chapter 13

# Part-of-Speech Analysis

A Part-Of-Speech Tagger (POS Tagger) is a piece of software that reads text in some language and assigns parts of speech to each word (and other token), such as noun, verb, adjective, etc., although generally computational applications use more fine-grained POS tags like 'noun-plural'. Concerning papers are [72], [74], [75].

There is a successful Stanford Log-linear Part-Of-Speech Tagger [139], [140]. Several downloads are available. The basic download contains two trained tagger models for English. The full download contains three trained English tagger models, an Arabic tagger model, a Chinese tagger model, and a German tagger model. Both versions include the same source and other required files. The tagger can be retrained on any language, given POS-annotated training text for the language. The English taggers use the Penn Treebank tag set.

Another successful tagger is the Tree Tagger form University of Stuttgart [127], [128] that is fre available, too.

# Chapter 14

# Validation of Requirements

The validation process will be explained usually im context of a verification process. The verification process means that software product properties are checked against its specification. After a successfully finished verification, we can say that the product is conform to its specification. The problem is that the specification may be incomplete. So, the validation process specifies how the customer is satisfied [64].

## 14.1 Validation of Requirements by Text Generation in TESSI

In this section, we describe our approach to textual feedback in requirement specification that we developed and published in more details in [88].

UML model is used for the synthesis of a text that describes the analyst's understanding of the problem, i.e. a new, model-derived requirements description will automatically be generated. Now, the user has a good chance to read it, understand it and validate it. His/her clarifying comments will be used by the analyst for a new version of the requirements description. The process is repeated until there is a consensus between the analyst and the user. This does not mean that the requirements description is perfect, but some mistakes and misunderstandings are removed.

We argue that the textual requirements description and its preprocessing by our tool will positively impact the quality and the costs of the developed software systems because it inserts additional feedbacks into the development process.

This document represents the analyst's understanding of the problem. It is very likely that the analyst and the user understand some words (some concepts) differently, it is very likely that the user holds some facts for self-evident and thinks they are not worth being mentioned. It is also very likely that some requirements have been forgotten. The document is a starting point to the next analysis. Using our tool TESSI the analyst identifies classes, methods, and attributes in the way how he/she understands the textual requirements and stores them into a UML-model. Our new approach is that from this UML-model a text can be generated that reflects how the analyst modeled the problem. The generated text is given to the user.

The user does not understand the UML-model but he/she can read the text generated and can decide whether it corresponds to his/her wishes. He/she discussed it with the analyst and the next iteration of the process of requirements refinement starts. Additionally, our tool can generate some simple questions, e.g. concerning constrains of attributes. These questions can influence the next iteration text, too.

After some iterations, when the user and the analyst can not find any disproportions, the last UML-model will be exported to the next processing. We use an interface to Rational Software Modeler (IBM). This tool produces diagrams of any kind, fragments of code in different programming languages, etc. The fragments of code have to be further completed and developed to a prototype. The prototype will be validated by the user and his/her comments will be inserted into the textual description of requirements.

As we can see our approach means that we use one additional feedback during the modeling before an executable prototype is available. It is very well known that the mistakes from requirements are very expensive because:

- it is expensive to find them because the costs grow up in exponential proportion to the distance between the time point when the mistake occured and the time point when the mistake was corrected,
- it is very likely that parts of the design and programming efford have been invested in vain and these parts have not only to be corrected but they have to be developed again.

The implemented component for text generation is a part of our CASE tool. As mentioned above, in the first phase of requirements acquisition, a text containing knowledge about the features of the system to be developed, is written in cooperation between the analysts, domain experts, and users. The analyst processes this text and, using the MODEL component, decides which parts of the text can be associated to which parts of the UML model. Then the GENERATOR component generates a text corresponding to the UML model and the user validates it. This process can iterate (see Fig. 16.3) until the differences disappear.

## 14.2 Generate natural language text from UML model

For the purpose of paraphrasing the specified UML model for users and domain experts, an NL text is generated from the UML model.

Differently from works that use templates completed with informationfrom assigned isolated model elements our linguistic approach can collect and combine pieces of information from the whole model for using them together in sentences of the generated text.

For the component GENERATOR we used the standard pipeline architecture [114] for NL generation, extended by an additional module used for NL analysis tasks [18]. Three modules are arranged in a pipeline where each module is responsible for one of the three typical NL generation subtasks, which include, in this order, document planning, micro planning and surface realization (Fig. 14.1). The output of one module serves as input for the next one.

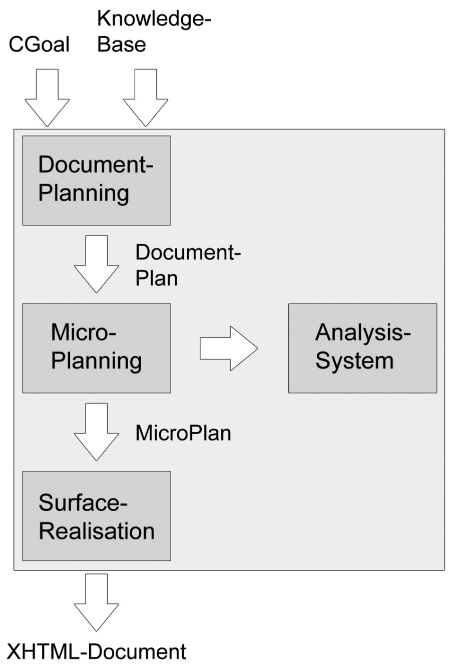


Figure 14.1: Architecture of the text generator component.

The input to the document planner is a communicative goal which is to be fulfilled by a text generation process. The communicative goal is basis for the selection of information (content determination) from a knowledge base.

In our case, the goal is to validate a requirements model and the knowledge base is the model itself.

Output of the document planner is:

- a document plan,
- a tree structure with message nodes,
- structural nodes.

Message nodes store pieces of information (NL text fragments) to be expressed in a sentence, structural nodes indicate the composition of the text and the order in which the sentences must occur.

The micro planner accepts a document plan as its input and transforms it into a micro plan by processing message nodes. Sentence specifications are produced, which can be either strings or abstract representations describing the underlying syntactic structure of a single sentence. In the latter case, this is done by a complex process (as described below) involving the tasks of NL parsing, linguistic representation and aggregation of text fragments as well as choosing additional lexems and referring expressions (articles).

A micro plan is transformed into the actual text (surface text) of a certain target language by a surface realizer. During structural realization the output format of the text is developed. The process of linguistic realization performs the verbalization of abstract syntactic structures by determining word order, adding function words and adapting the morphological features of lexems (e.g. endings).

### 14.2.1 The approach

First, we wrote the presupposed text that should be generated in our case study using semantic relations between its parts, which can be derived from the UML model. There are semantic relations in UML models between the following elements:

- use case and sequence diagram
- class and state machine
- use case and transition in a state machine

Examples are given in Section 14.3.

After this, we analyzed possibilities to derive the target text from an existing UML model. We found that there are:

- fixed text fragments that specify the structure of the generated text
- directly derivable text fragments that can be copied, e.g. names of classes
- indirectly derivable text fragments that depend on syntax and morphology rules
- not derivable text fragments that cannot be derived from the model

We noticed that a minor part of the text could be produced by a simple template-based approach. This is the case for sentences combining fixed text fragments and directly derivable text fragments.

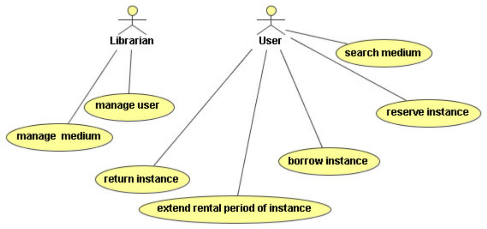
To simplify the generation process where possible we made our text generator capable to perform template-based generation as well. However, a generation based on templates was not sufficient for the major part of the text. In cases where sentences contain indirectly derivable text fragments, a method depending on linguistic knowledge was needed.

## 14.3 Case study

To illustrate our text generation method, we now apply it to a contrived specification of a library automation system.

As an example, we combine information from use case diagrams and state machine diagrams in the following way:

- Use case diagram ... “borrow instance”
- State machine diagram ... “available”, “available and reserved”



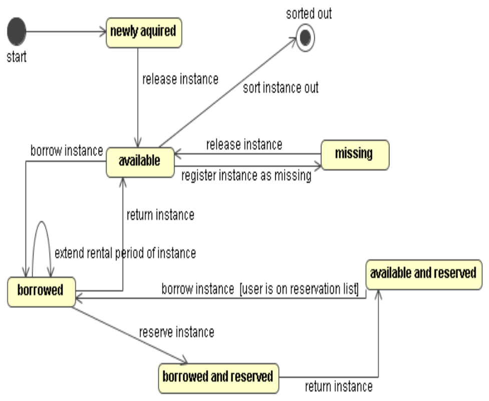


Figure 14.3: State machine diagram.

- Generated text: “Only instances can be borrowed which are available, or available and reserved and the user is the first on the reservation list”.

  “If the instance is signed as available the user can borrow the instance. The instance will afterwards be signed as borrowed. Alternatively, if the instance is signed as available and reserved and the user is on the reservation list the user can borrow the instance. The instance will afterwards be signed as borrowed.”

## 14.4 Implementation

The current system is a Java/Eclipse application. The NL generation component has been developed as a module and integrated into the system as a plug-in component.

The generator produces output texts in the English language. The results of the different generation steps (document plan, micro plan, output text) are represented using XML technology.

The task of document planning is managed by schemata, each of them is responsible for the generation of a certain part of the document plan. To fulfill the communicative goal Describe_Dynamic_Model several schemata exist which specify the interwoven steps of content determination and document structuring.

According to the two main subtasks the micro planner performs, the module includes a ProtoSentenceSpecBuilder and a SentenceSpecBuilder. The ProtoSentenceSpecBuilder processes an input message and produces a proto sentence specification.

After that, the SentenceSpecBuilder transforms the proto sentence specification into a sentence specification. The SentenceSpecBuilder provides an interface that is realized by several components according to the different types a proto sentence specification may have. Thus, the individual implementations encapsulate the complete knowledge needed for the creation of the DSyntS of a specific sentence type from data stored in the proto sentence specification.

For NL analysis the Stanford parser [58], [59] is used. This parser provides two different parsing strategies (lexicalized factored-model, unlexicalized PCFG) that both can be chosen for the task of preprocessing (micro planning). Access to the parser and corresponding components used for the processing of dependency structures and DSyntS is granted by the interface the AnalysisSystem provides.

Our generator component produces output texts formatted in XHTML. The mark up is developed in the stage of structural realization performed by the XHTMLRealiser. To accomplish linguistic realization and produce the surface form of the output texts RealPro [113] is used.

## 14.5 Example of the text generated for validation

As an example, we show a fragment of a generated text that is a part of a generated Library system description, i.e. the text is generated from UML-model of a Library system. In the following there is the description of the function BorrowInstance:

BorrowInstance

This function can be done by a user.

Preconditions and effects:

If the instance is signed as available the user can borrow the instance. The instance will afterwards be signed as borrowed. Alternatively, if the instance is signed as available and reserved and the user is on the reservation list the user can borrow the instance. The instance will afterwards be signed as borrowed.

Procedure:

1. The user identifies himself.
2. The user specifies the instance by the shelfmark.
3. A component (User Administration part of the Library system) registers the instance in the borrowed-list of the user account.
4. A component (Media Administration part of the Library system) registers the user in the borrowed-list of the instance.

5. A component (Media Administration part of the Library system) changes the status of the instance.
6. A component (Media Administration part of the Library system) returns the receipt.

## 14.6 Achieved Results and Conclusion

A component has been designed and implemented in [88] which serves as an important basis for sophisticated NL text generation with the purpose of validating requirements analysis models. The text generator performs text generation in three consecutive steps: document planning, micro planning and surface realization. It presents an approach to text generation based on textual input data using NL-analysis- and NL-generation-techniques. Compared to texts produced by the pre-existing template-based text generator, texts generated by the new non-trivial text generator are definitely more structured, more clearly arranged and more readable.

Further, the vocabulary used should be more understandable for people outside the software industry. As far as it has been considered possible, generated texts do not contain terms specific to software engineering. Due to the usage of RealPro for surface realization, the grammar of generated sentences is also more correct than before.

Currently, the text generator is capable of producing NL texts from use cases, sequence diagrams and state machines. As the architecture has been designed with the aim of easy extensibility, it should not be too difficult to integrate text generation functionality for other UML model elements as well. Furthermore, it is possible to adapt the text generator to other target languages.

A number of open issues may be addressed in the future: prevention of generation errors caused by the NL parser, improvement of the micro planner, integration of text schemata for other model elements (such as static structures like classes).

Further it is desirable to evaluate our proposed validation approach by application to real-world projects. This is not easy because it is necessary to persuade the management in a software house to run a project in two teams (one team should use our tool) and then to compare the results.

# Chapter 15

# Traceability of requirements

Traceability of requirements means that for any change in a requirement all its impacts can be found. There are impacts in other requirements, there are impacts in design, implementation, and test cases [133].

In software evolution and adaptive maintenance, traceability of requirements is very important. When we move from analysis to design, we assign requirements to the design elements that will satisfy them. When we test and integrate code, our concern is with traceability to determine which requirements led to each piece of code. The tool TESSI builds and maintains some important relationships between requirements and parts of the resulted system, e.g. bi-directional links between the identified entities in sentences of textual requirements and the corresponding entities of the model. These links will be followed during an adaptive maintenance. This helps to hold requirements and programs consistent and supports the concept of software evolution in which every change in the software system should start with the change of the requirements specification and follow the life-cycle of development.

Requirements traceability paths:

- Trace top level requirements into detailed requirements
- Trace requirements into design
- Trace requirements into test procedures
- Trace requirements into user documentation plan

Traceability links requirements to related requirements of same or different types. RequisitePros traceability feature makes it easy to track changes to a requirement throughout the development cycle. Without traceability, each change would require a review of your documents to determine which, if any, elements need updating.

If either end-point of the connection is changed, the relationship becomes suspect. If you modify the text or selected attributes of a requirement that is traced to or traced from another requirement, RequisitePro marks the relationship between the two requirements suspect.

Traceability relationships cannot have circular references. For instance, a traceability relationship cannot exist between a requirement and itself, nor can a relationship indirectly lead back to a previously traced from node. RequisitePro runs a check for circular references each time you establish a traceability relationship.

The trace to/trace from state represents a bidirectional dependency relationship between two requirements. The trace to/trace from state is displayed in a Traceability Matrix or Traceability Tree when you create a relationship between two requirements.

Traceability relationships may be either direct or indirect. In a direct traceability relationship, a requirement is physically traced to or from another requirement. For example, if Requirement A is traced to Requirement B, and Requirement B is traced to Requirement C, then the relationships between Requirements A and B and between Requirements B and C are direct relationships.The relationship between Requirements A and C is indirect. Indirect relationships are maintained by RequisitePro; you cannot modify them directly. Direct and indirect traceability relationships are depicted with arrows in traceability views. Direct relationships are presented as solid arrows, and indirect relationships are dotted and lighter in color.

A hierarchical relationship or a traceability relationship between requirements becomes suspect if RequisitePro detects that a requirement has been modified. If a requirement is modified, all its immediate children and all direct relationships traced to and from it are suspect. When you make a change to a requirement, the suspect state is displayed in a Traceability Matrix or a Traceability Tree. Changes include modifications to the requirement name, requirement text, requirement type, or attributes.

# Chapter 16

# Related Work to Requirements Engineering

Many books (e.g. [118], [90], [110]) and many articles on software development process have been published. In [15], they found 5,198 publications about requirement engineering spanning the years from 1963 through 2008.

## 16.1 Related Work to the Concept of Requirements Engineering

To obtain semantic information directly from a text, manual or automatic methods can be used. Many papers and books [10] support manual methods. For example, the method of grammatical inspection of a text can be mentioned. It analyzes the narrative text of use cases, identifies and classifies the participating objects [43]. In this method, nouns will be held for candidates for objects or classes, adjectives will be held for candidates for attributes, and verbs will be held for candidates for methods or relationships. The Object Behaviour Analysis [119] belongs to this group of methods, too.

There are some tools helping in requirements management (DOORS, Requisite Pro 16.6) that transform unstructured texts of requirements into structured texts. They neither build any models nor generate refining questions like our tool TESSI.

Automatically generated natural-language description of software models that is the main contribution of the tool TESSI is not a new idea. The first system of this kind was described in [134]. More recent projects include the ARIES [50] and the GEMA data-flow diagram describer [129].

Even if it is widely believed that the graphical representation is the best base for the communication between the analyst and the user, there are some serious doubts about it. It has been found in [56] that even a co-operation between experienced analysts and sophisticated users who are not familiar with the particular graphical language (which is very often the case) results in semantic error rates of about 25 % for entities and

70 % for relations. Similar results brings [109]. Some systems have been built which transform diagrams, e.g. ER-diagrams, into the form of a fluent English text. For example, a system MODEX has been described in [91], [92] which can be used for this purpose. Differently to our system, input to MODEX (MODel EXplainer) is based on the ODL standard from the ODMG group.

The first version of our system TESSI has been introduced in [83]. Additionally to MODEX, it supports the manual analysis of the requirements description by the analyst and the semi-automatic building of an OO-model in UML. Differently from MODEX, we have used simpler templates for generating the output text, because our focus was not in linguistics. The current version of TESSI generates refining questions and uses an XML-interface to Rational Rose.

## 16.2 Related Work to Ontologies in Requirements Specification

There are a number of research approaches to elicit and analyze domain requirements based on existing domain ontologies. For example, [93] used a domain ontology and requirements meta-model to elicit and define textual requirements. The system GOORE proposed in [130] represents an approach to goal-oriented and ontology-driven requirements elicitation. GOORE represents the knowledge of a specific domain as an ontology and uses this ontology for goal-oriented requirements analysis [66]. A shortcoming of these approaches is the need for a pre-existing ontology, as to our knowledge there is no suitable method for building this ontology for requirements elicitation in the first place in an at least semi-automated way.

## 16.3 Related Work to Checking Inconsistency

Using ontologies to shape the requirements engineering process is clearly not a new idea. In the area of knowledge engineering, ontology was first defined by [101].

An ontology-based approach to knowledge acquisition from text through the use of natural language recognition is discussed in [8], [48], in [53] and the last approach in [9]. In [141] they have constructed the Enterprise Ontology to aid developers in taking an enterprise-wide view of an organisation. The approach in [49] is intended to automate both interactions with users and the development of application models.

The ontologies used by [122] in their Oz system are domain models which prescribe detailed hierarchies of domain objects and relationships between them. Formal models for ontology in requirements are described in [46]. Domain rules checking is described in [96]. In [156] the inconsistency measurement is discussed. The ontology used by the QARCC system [6] is a decomposition taxonomy of software system quality attributes. QUARCC uses a specialized model for identifying conflicts between quality (non-functional) requirements. QuARS [34] presents an approach for phrasal analysis and classification of natural language requirements documents.

In [96] a formal model of requirements elecitation is discused that contains domain ontology checking.

Concerning inconsistencies the overview is given in [22], in [102], and lately in [132] but there is not an approach applying ontology in the sense of our way.

However, our work is not specifically addressing the issue of improving natural language communication between stakeholders in an interview in order to achieve more polished requirements as most of the related papers are. We investigate the possibility of combining UML model and OWL ontology for checking and validating requirements specifications, as we have already mentioned above.

## 16.4 Related Work to Linguistic Methods

In [57], three experiments are presented concerning domain class modeling. The tool used for the experiment named NL-OOPS extracts classes and associations from a knowledge base realized by a deep semantic analysis of a sample text.

In [142], a tool is introduced, called the Requirements Analysis Tool (RAT) that automatically performs a wide range of syntactic and semantic analyses on requirements documents based on industry best practices, while allowing the user to write these documents in natural language. RAT encourages users to write in a standardized syntax, a best practice, which results in requirements documents that are easier to read and understand. RAT performs a syntactic analysis by using a set of glossaries to identify syntactic constituents and flag problematic phrases. An experimental version of RAT also performs semantic analysis using domain ontologies and structured content extracted from requirements documents during syntactic analysis. The semantic analysis, which uses semantic Web technologies, detects conflicts, gaps, and interdependencies between different sections (corresponding to different subsystems and modules within an overall software system) in a requirements document.

## 16.5 Related Work to Requirement Text Generation

Automatically generated texts can be used for many purposes, e.g. error messages, help systems, weather forecast, technical documentation, etc. An overview is given in [106].

In most systems, text generation is based on templates corresponding to model elements (discussed in [16]). There are rules on how to select and instantiate templates according to the type and contents of an element of the model. String processing is used as a main method. We used it in the first version of our system and found that texts generated in this way were very large and boring.

Another disadvantage was that the texts we generated were often not right in the sense of grammar [83]. Building all possible grammatical forms for all possible instantiations had been too complex [40], [20]. Further, the terminology used in the generated texts also included specific terms from the software engineering domain, which reduces text understandability for users and domain experts. The maintenance and evolution of such templates was not easy.

Except for our previous work [84], [117], there are at least two similar approaches with the aim of generating natural language (NL) text from conceptual models in order to enable users to validate the models, which are briefly characterized in turn.

The system proposed by Dalianis [14] accepts a conceptual model as its input. A query interface enables a user to ask questions about the model, which are answered by a text generator. User rules are used for building a dynamic user model in order to select the needed information. Another system for conceptual modeling description is discussed in [39].

A discourse grammar [42] is used for creating a discourse structure from the chosen information. Further, a surface grammar is used for surface realization, i.e. for the realization of syntactic structures and lexical items.

ModelExplainer [91] is a web-based system that uses object-oriented data modeling (OODM) diagrams as starting point, from which it generates texts and tables in hypertext which may contain additional parts not contained in the model. The result text is corrected and adjusted by the RealPro system [92], which produces sentences in English. However, since NL generation is based on OODM diagrams alone, it is confined to static models. The problems concerning text structure are described in [97].

In the approach given in [143], a specific Requirement Specification Language is defined, and some parsers that can work with it. Textual requirements should be processed automatically. In our approach they are process semi-automatically. The analyst is still an important person and he/she has to understand the semantics of the problem to be solved and implemented.

Because of the disadvantages described above we used a linguistic approach [114] in our last version.

Currently, there are no systems available (we have not even found any experimental systems of that kind) that would follow the idea of using automatically generated textual description of requirements for feedback in modeling. The main application field is in information systems where requirements have to be acquired during an interview then collected, integrated often from many parts together, and processed.

## 16.6 The Tool Rational RequisitePro

Rational RequisitePro [115] is a requirements management tool that integrates a multiuser requirements database utility into the Windows-Word environment. The program allows to work simultaneously with a requirements database and requirements documents. As an editor, the Microsof Word is used.

Requirements management denotes:

- a systematic approach to eliciting, organizing, and documenting the requirements of the system,
- a process that establishes and maintains agreement between the customer and the project team on the changing requirements of the system.

It has been built to organize activities that are common to all users who view and query requirements. It supports creating and managing requirements throughout the entire development life cycle, and it addresses managing projects.

In difference to TESSI (Section 16.8), it does not offer a design using the corresponding UML-model. RequisitePro supports the transition from a set of not structured text documents to a hierarchically organized form of specification texts. Hierarchical requirement relationships are one-to-one or one-to-many, parent-child relationships between requirements of the same type. Use hierarchical relationships to subdivide a general requirement into more explicit requirements.

### 16.6.1 The model used in RequisitePro

The model used in RequisitePro contains:

- Packages

  Within each project, requirements artifacts are organized in packages. A package is a container that can contain requirements documents, requirements, views, and other packages. You can place related artifacts in a single package, and this organization makes it easier for you to view them and to manipulate the data. You can configure your packages as necessary to facilitate your work. An artifact cannot appear in more than one package, but you can move it from one package to another. You can create a package within another package. All project packages are shared by all project users. Within a package, artifacts are listed in the following order: documents (alphabetically by name), views (by type and then alphabetically within the type), and requirements (by type and then by tag).

- Documents

  Documents in RequisitePro are more or less documents in Word but a few changes were introduced to exercise security control and to prevent conflicts. RequisitePro manages requirements directly in the project documents. When a requirements document is built, RequisitePro dynamically links it to a database, which allows rapid updating of information between documents and views. Whenyou save revisions, they are available to team members and others involved with the project.With RequisitePros version tracking, you can easily review the change history for a requirement, a document, or the whole project.

- Requirement types - are used to classify similar requirements so they can be efficiently managed.

- Requirement attributes

  Each type of requirement has attributes, and each individual requirement has different attribute values. For example, requirements may be assigned priorities, identified by source and rationale, delegated to specific sub-teams within a functional area, given a degree-of-difficulty designation, or associated with a particular iteration of the system.

  Even without displaying the entire text for each requirement, we can learn a great deal about each requirement from its attribute values.

  In more detailed types of requirements, the priority and effort attributes may have more specific values (for example, estimated time, lines of code) with which to further refine scope. This multidimensional aspect of a requirement, compounded by different types of requirements (each with its own attributes) is essential to organizing large numbers of requirements and to managing the overall scope of the project.

Attribute information may include the following: the relative benefit of the requirement, the cost of implementing the requirement, the priority of the requirement, the difficulty or risk associated with the requirement, the relationship of the requirement to another requirement.

- Views

  Rational RequisitePro views use tables or outline trees to display requirements and their attributes or the traceability relationships between different requirement types. RequisitePro includes powerful query functions for filtering and sorting the requirements and their attributes in views. A view is an environment for analyzing and printing requirements. You can have multiple views open at one time, and you can scroll to view all requirements and attributes in the table or tree.

  RequisitePro views are windows to the database. Views present information about a project, a document, or requirements graphically in a table (matrix) or in an outline tree. Requirements, their attributes, and their relationships to each other are displayed and managed in views. RequisitePro includes query functions for filtering and sorting the requirements and their attributes in views.

  Three kinds of views can be created:

  The Attribute Matrix displays all requirements of a specified type. The requirements are listed in the rows, and their attributes appear in the columns.

  Traceability Matrix displays the relationships (traceability) between two types of requirements.

  Traceability Tree displays the chain of traceability to or from requirements of a specified type.

  All views display hierarchical relationships, and you can use the Traceability Matrix and Traceability Tree to display hierarchical relationships that are marked suspect.

### 16.6.2 Traceability in RequisitePro

Traceability links requirements to related requirements of same or different types. RequisitePros traceability feature makes it easy to track changes to a requirement throughout the development cycle. Without traceability, each change would require a review of your documents to determine which, if any, elements need updating.

If either end-point of the connection is changed, the relationship becomes suspect. If you modify the text or selected attributes of a requirement that is traced to or traced from another requirement, RequisitePro marks the relationship between the two requirements suspect.

After a view into the database of requirements is created, it can be refined by querying (filtering and sorting). There are not more advanced concepts (ontology technology) present.

## 16.7 The Tool RAT

In [142], the recommended analysis structure is specified in Fig. 16.1 as:

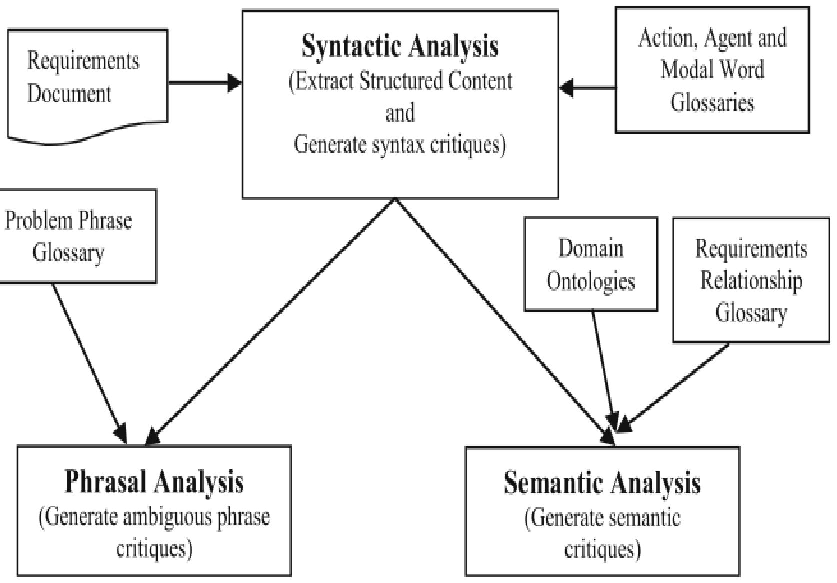

Requirements Document
Syntactic Analysis
(Extract Structured Content and Generate syntax critiques)
Action, Agent and Modal Word Glossaries
Problem Phrase Glossary
Domain Ontologies
Requirements Relationship Glossary
Phrasal Analysis
(Generate ambiguous phrase critiques)
Semantic Analysis
(Generate semantic critiques)

Figure 16.1: Natural language processing - overview

### 16.7.1 Controlled Syntax for Writing Requirements

The tool RAT [142] supports a set of controlled syntaxes for writing requirements. This set includes the following Standard Requirements Syntax. This is the most commonly used syntax for writing requirements and is of the form:

Standard Requirement syntax:

```
<agent> <modal word> <action> <rest>
```

Where,

```
<agent>, <action>, <modal word>
```

are phrases in their respective glossaries and

```
<rest>
```

is the remainder of the sentence and can consist of agents, actions or any other words and is defined as:

```
<rest>: [<anyword> | <agent> | <action>]*
```

Conditional Requirements Syntax: There are a number of conditional requirements supported by RAT. For brevity, we will only discuss the most common condition syntax:

```
<if> <condition> <then> <StandardRequirement>
```

For example, consider the following requirement: If the user enters the wrong password, then the system shall send an error message to the user. In the case, user enters the wrong password is the condition. The part after then is treated like a standard requirement.

Business Rules Syntax: RAT treats all requirements that start with all, only and exactly as business rules. An example is: Only the members of payroll department will be able to access the payroll database.

### 16.7.2 User-Defined Glossaries and Document Parsing

RAT uses three types of user glossaries to parse requirements documents:

- agent glossary - An agent entity is a broad term used to denote systems, subsystems, interfaces, actors and processes in a requirements document. The agent glossary contains all the valid agent entities for the requirements document. It captures the following information about each agent: name of the agent, immediate class and super-class of the agent and a description of the agent. The class and parent of the class field of the glossary are used to load the glossary into the semantic engine.

- action glossary- The action glossary lists all the valid actions for the requirements document. It has a similar structure to the agent glossary. Examples of actions are generate, send and allow.

- modal word glossary - The modal word glossary lists all the valid modal words in the requirements document. Examples of modal words are must and shall.

As a requirement is parsed according to the syntax rules given above, structured content is extracted for each requirement. This structured content is used for both the syntactic and semantic analyses. The extracted structured content contains:

- Type of requirement syntax (standard, conditional, business rule)

- All agents, actions and modal words for all the requirements

- Different constituents of conditional requirements.

### 16.7.3 Classification of problematic phrases

Certain phrases frequently result in requirements that are ambiguous, vague or misleading. The problematic use of such phrases has been well documented in the requirements literature. A classification of problematic phrases is presented in [34]. In [155], a list of such words is given and it is explained how to correct requirements that use them.

A list of problematic phrases is stored in a user-extensible glossary called the problem phrase glossary.

### 16.7.4 Semantic Analysis

Much of the domain knowledge can be captured using domain-specific ontologies to provide a deeper analysis of requirements document. The crux of the approach in [142] is to create a semantic graph for all requirements in the document based on the extracted content.

In RAT, users can use the requirements relationship glossary to enter domain specific knowledge. The requirements relationship glossary contains a set of requirement classification classes, its super-class, keywords to identify that class and the relationships between the classes.

The fundamental belief in [142] is that once a requirements document is transformed into a semantic graph (represented as an OWL ontology), users can query for different kinds of relationships that are important to them. Here are the steps that RAT uses to create the semantic graph (graphically depicted in Figure 16.2) from a requirements document:

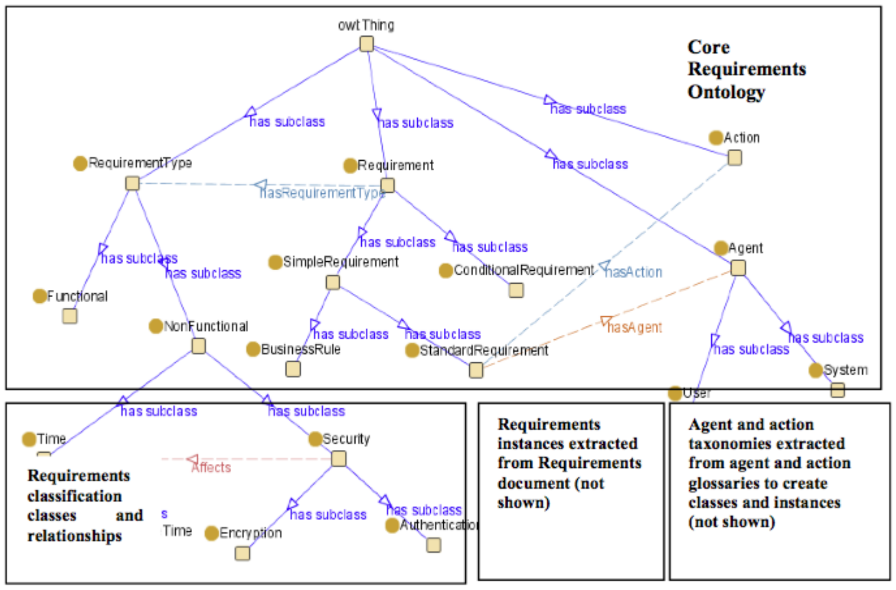

owl:Thing
Core Requirements Ontology
has subclass
RequirementType
Requirement
Action
hasRequirementType
Agent
Functional
NonFunctional
SimpleRequirement
ConditionalRequirement
hasAction
BusinessRule
StandardRequirement
hasAgent
User
System
Time
Security
Affects
Encryption
Authentication
Requirements classification classes and relationships
Requirements instances extracted from Requirements document (not shown)
Agent and action taxonomies extracted from agent and action glossaries to create classes and instances (not shown)

Figure 16.2: Core requirements ontology

- The core requirements ontology is in the Semantic Engine. It is basic a requirements ontology with different types of requirements formats (standard, business rule and conditional) and the information that each of them contain.
- Using the agent and action glossaries to create the agent and action classes and instances of agents and actions.
- Using the requirement relationship glossary to create requirements classification classes and their relationships.
- Using the extracted structured content (enhanced by requirement classification information) to create instances of requirements.

### 16.7.5 Implementation Details and Early User Evaluation of RAT

RAT has currently been created as a plug-in for Microsoft Word 2003/7 using Visual Basic for Applications. The Glossaries are currently also Word documents. The Semantic Engine leverages the reasoning capabilities of Jena [45] and is implemented in Java. Protege was used to create the OWL [145] ontologies. The Jena semantic engine is used for the reasoning and SPARQL [151] is used for the query language.

Currently, an early version of this tool without the full semantic analysis engine will be proved, at four client teams. In these pilots, 15 requirements analysts have used RAT to process more than 10,000 requirements.

## 16.8 The Tool TESSI

In tool TESSI [83], [84], [154], [87], [88], we offer a textual refinement of the requirements specification that can be called requirements description. Working with it, the analyst is forced by the supporting tool TESSI to complete and explain requirements and to specify the roles of words in the text in the sense of the object-oriented analysis.

During this process, a UML model will automatically be built by TESSI. The process is driven partially automatically and partially by the analyst's decisions. In the old release of TESSI, the process of UML-elements identification was based on human decision. In the last release of TESSI, the automation of this process was constructed using methods of natural languages processing; human interaction is often necessary to correct and complete the model.

### 16.8.1 Architecture and Dataflow

We argue that there is a gap between the requirements definition in a natural language and the requirements specification in some semi-formal graphical representation. The analyst's and the user's understanding of the problem are usually more or less different when the project starts. Usually, the first possible time point when the user can validate the analyst's understanding of the problem is when a prototype starts to be used and tested.

This model will be used:

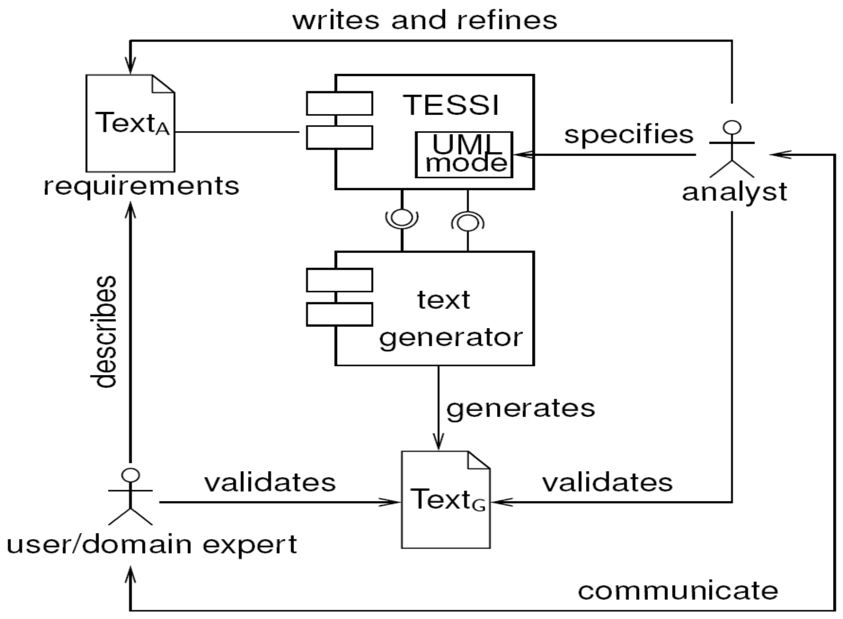


Figure 16.3: Architecture and dataflow of the existing tool TESSI

- for checking ambiguity of the described requirements,
- for checking for inconsistency and completness using the domain knowledge described in domain ontology [87]
- for the synthesis of a text that describes the analyst's understanding of the problem, i.e. a new, model-derived requirements (textual) description will automatically be generated [88]. Now, the user has a good chance to read it, understand it and validate it. His/her clarifying comments will be used by the analyst for a new version of the requirements description.

The process repeats until there is a consensus between the analyst and the user. This does not mean that the requirements description is perfect, but some mistakes and misunderstandings are removed.

We argue that the textual requirements description and its preprocessing by tool TESSI will positively impact the quality and the costs of the developed software systems because it inserts additional feedbacks into the development process.

The tool TESSI can be used in the following steps:

- Automatic identification of parts of text that correspond to model elements
- Grouping of the identified parts of text based on text similarity
- Structured visualization of the created groups

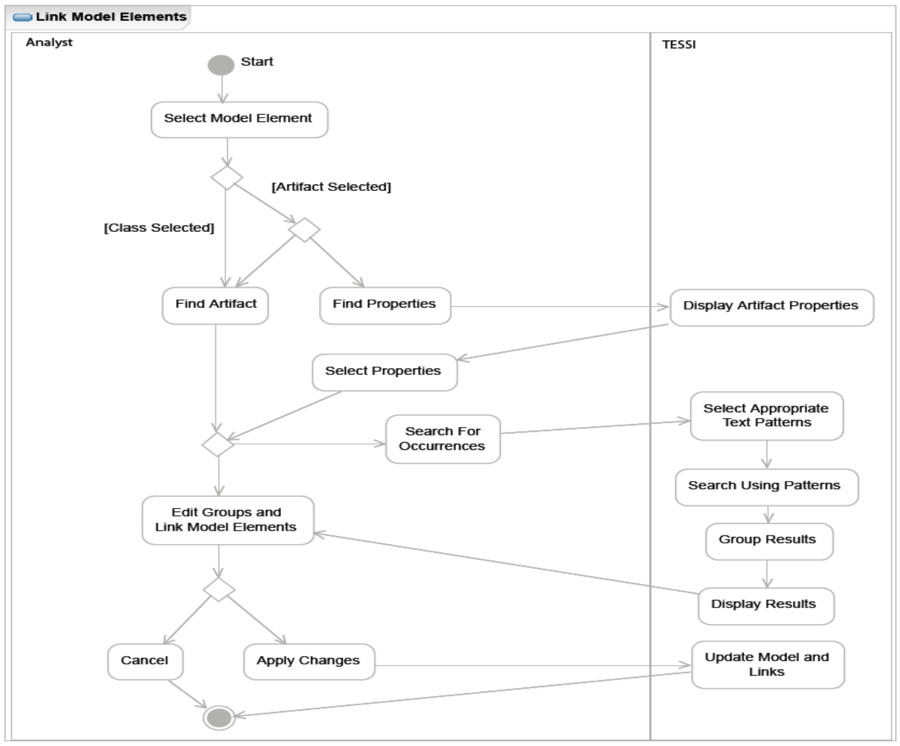


Figure 16.4: TESSI - Creating and relation between model elements

- Editing of the created groups
- Manuel identification of the relevant parts of text

  Searching by help of phrases or regular expressions

  Appending the text parts found to the corresponding group
- Linking a model element and a group

  Choice of the existing model element

  Creating of a new model element

The UML-elements identified are connected to the relevant parts of text (in the sequel we speak about words only, eventhough it can be a phrase). All occurrencies of these words are highlighted in color, so the analyst can see the impact of its modeling decision.

The problem is that occurrence of such a word is stated out of the context (without natural language parsing process). Here, the ambiguity of natural languages makes difficulties. Eventhough, the analyst should try to avoid the ambiguity when formulating requirements, mistakes are supposed to occur. This is one of reasons why methods of natural language processing are unavoidable in requirements engineering.

The automatic search for artefacts in textuell requirements needs artefacts identification templates (rules). Such templates (rules) can be described as an algorithm, as a part of text, e.g. as a regular expression. To define these templates, a template type definition using the language TPL (Tree Pattern Language [153]) is a suitable construction.

The language TPL uses regular expressions for information extraction from trees. It contains operators for tree traversing and methods for testing of node properties.

### 16.8.2 Natural language analysis in TESSI - using UIMA

In TESSI, the system for NLP processing UIMA (Unstructured Information Management Architecture) was used [26], [27], [153]. It is an OpenSource-Project and it offers an Apache UIMA-framework, a Java implementation, and a set of Eclipse-plugins.

UIMA is an architecture of analysis engines that can provide an analysis of a document in natural language. The basic data structure where all analysed objects are stored is called Common Analysis Structure (CAS). It is an object-oriented hierarchical structure. Parts of text are denoted by annotation as positions where the part of text beginns and ends.

Primitive Analysis Engines can be combined (aggregated) into Agregated Analysis Engines. In Figure 16.5, there is an aggregated analysis engine Full Parser.

The results of text document analysis using UIMA aggregated engines are inputs for functions that offers candidates of artefacts (model elements) produced from textual requirements automatically.

The artefacts can be stored and managed as parts of ontologies. So, the first model of textual requirements is an ontology.

### 16.8.3 Ontologies in TESSI for Building UML Model

Often, ontologie are used to represent domain concepts. In TESSI, ontologies contain instances of such concepts and our knowledge about these instances. These concepts determine types of requirements on the system to be implemented. We use concepts UseCase, Actor, Class, Object, Attribute known from object-oriented modeling. The import mechanism supports to recombine various ontologies.

Ontologies serve for collecting information during the requirments elicitation. They do not substitute the UML model. As we show later on, reasoning in ontologies is used in TESSI to provide the very first feedback of the requirements quality.

In TESSI, ontologies are implemented using the Jena-framework [45] for semantic web applications. Jena can manage the RDF-graphs completely in the main memory or in a relational database. This property enables processing of models regardless of their size.

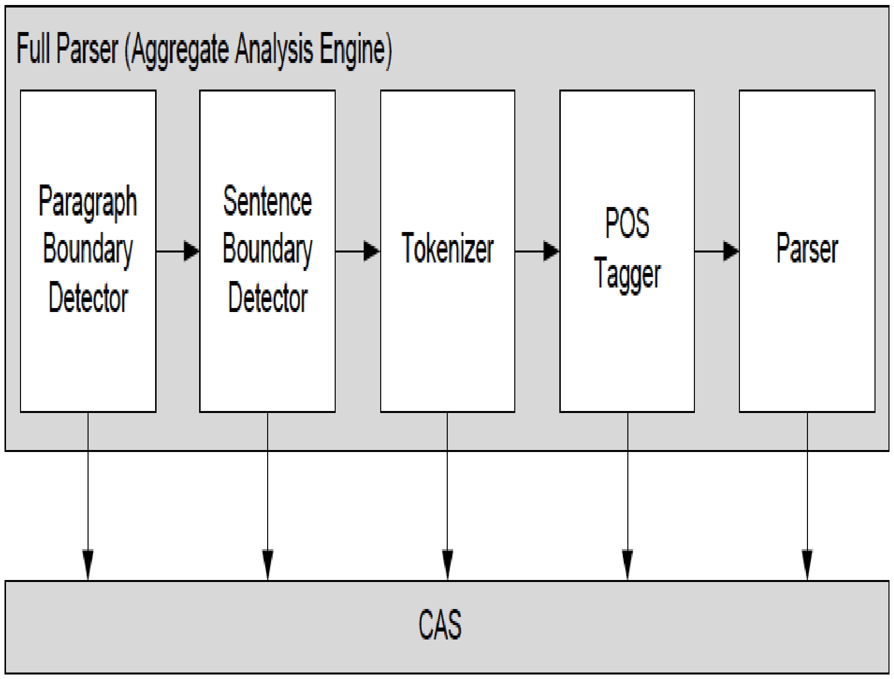

Full Parser (Aggregate Analysis Engine)
Paragraph Boundary Detector
Sentence Boundary Detector
Tokenizer
POS Tagger
Parser
CAS

Figure 16.5: UIMA - Aggregated Analysis Engine

### 16.8.4 Grammatical Templates for Identification of Parts of UML Model

Actions Find Artifact and Find Properties start searching for instances of concepts and their properties (realtionships) in text documents of requirement specifications. The user of TESSI has the possibility to complete or change the automatic search results [154] by manual corrections.

For searching artifacts that can represent the concept Class, the following TPL-expression [153] was used:

```
(Phrase & { tag == "NP" } !<< {tag =~ /^(NP|VP|CD|PR|CC|)/}) |
( {tag =~ /^NN/} >>
(Phrase &
{ tag == "NP" } << {tag =~ /^(NP|VP|CD|PR|CC|)/})
)
```

This expression (abbreviations from PennTreebank [98], [126]) has the following meaning:

Find such noun phrases (NP) that do not contain other noun phrases (NP), verbal phrases (VP), cadinal numerals (CD - e.g. one, two, 3), pronouns (PR*) or coordinating conjunctions (CC - e.g. and, or), and nouns (NN*) that do not belong to a noun phrase fullfilling the conditin above.

For searching artifacts that can represent the the artifact Book of concept Class according to the Property hasAttribute, the following TPL-expression [153] was used:

```
class:* >> (c:Clause << #class << vp:({tag == "VP"} ,, #class))
join[vp]
np:({tag == "NP"} !<< {tag =~ /^(NP|VP|CD|PR|CC|)/})
?. (
{tag == "PP"} <<
(np2:{tag =~ /^(NN|NP)/} !<<
{tag =~ /^(NP|VP|CD|PR|CC|)/})
)
-> np, np2
```

This TPL-expression searches in a sentence (Clause) given by the text context (class:*) that references the concept class. The sentence should contain a verbal phrase (VP) that follows the class reference in the same sentence. In the verbal phrase found, all noun phrase (NP) will be tested that fullfil a specific condition. This condition is based on grammatical inspection an co-ocurrence and is given in manual.

The OWL ontologies are based on description logic. The advantage is that there is a reasoning guaranted over used concepts and properties. It includes transitivity, inversion, and symmetry.

A similar approach is given in [143].

### 16.8.5 Ontologies in TESSI for Checking Consistency

Using experiences given in [96], we described the domain ontology in Protégé and apply ontology reasoning (e.g. the inference engine in Pellet - for checking classes) first for domain ontology checking, then for requirements problem ontology checking, and last for checking whether the requirements problem ontology subsumes the domain ontology.

The steps of the requirements processing that uses ontologies are the following:

- building a domain ontology in OWL using Protégé, i.e. domain ontology description based on OWL and SWRL based is constructed by domain experts at first and then transformed into the concepts set of Pellet and rules set of Jess.
- checking the domain ontology for consistency using Pellet (class hierarchy) and Jess (rules),
- the analyst writes a text description of requirements based on interviews with users,
- the analyst builds the UML model from a textual description of requirements supported by our tool TESSI,
- conversion of requirements described as a UML model to a problem ontology in OWL using convertor ATL (we used ATL because RacerPro is not public),
- checking the problem ontology for mutually consistency,
- checking the problem ontology by confronting it with the domain ontology,
- identifying inconsistency problems,
- finding the corresponding parts in the former textual description of requirements and correcting them,
- building a new UML model based on corrected textual description of requirements,
- after iterations when no ontology conflicts will be found a new textual description of requirements will be automatically generated that corresponds to the last iteration of the UML model,
- before the UML model will be used for design and implementation the user and the analyst will read the generated textual description of requirements and look for missing features or misunderstandings,
- problems found can start the next iteration from the very beginning,
- after no problems have been found the UML model in form of a XMI-file will be sent to Rational Modeler for further processing.

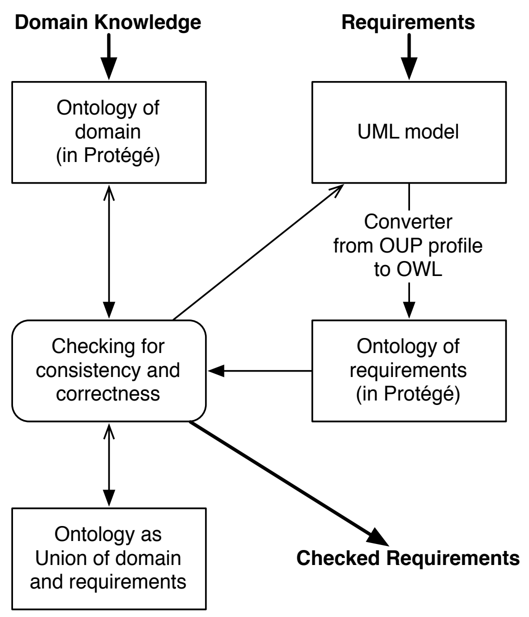

Domain Knowledge
Requirements
Ontology of domain (in Protégé)
UML model
Converter from OUP profile to OWL
Checking for consistency and correctness
Ontology of requirements (in Protégé)
Ontology as Union of domain and requirements
Checked Requirements

Figure 16.6: The component for consistency checking

### 16.8.6 Feedbacks in TESSI

The UML model will be used for the synthesis of a text that describes the analyst's understanding of the problem, i.e. a new, model-derived requirements description will automatically be generated. Now, the user has a good chance to read it, understand it and validate it. His/her righting comments will be used by the analyst for a new version of the requirements description. The process repeats until there is a consensus between the analyst and the user. This does not mean that the requirements description is perfect, but some mistakes and misunderstandings are removed.

We argue that the textual requirements description and its preprocessing by TESSI will positively impact the quality and the costs of the developed software systems because it inserts additional feedbacks in the development process.

More details are given in the Section 14.1.

# Chapter 17

# Open problems

- UML-model automatically derived from text of requirements by NLP

  In TESSI, we used templates (patterns) describing part-of-speech tagging corresponding to classes and to relationships. The used templates were specified manually following the grammatical inspection. We should find templates for other parts of the UML model automatically that should be derived from textual requirements. A similar approach is given in [143].

  Searching templates of POS-tags-trees can be done manual (as in TESSI) or automatically. The automated method assumes to use machine learning to get classified cases for supervised methods of classification. Experiments on large data will give statistical results (e.g. presicion, recall, F-measure).

- Inconsistency checking

  Is there any tool for checking consistency of OCL rules?

  Where is the limit of rule complexity for executable consistency checking?

- Completeness

  Using domain ontology, there is a possibility to generate some requests automatically that should complete the gaps between knowledge stored in the domain ontology and the problem ontology that corresponds to the requirement specifications.

  How to generate questions that can cause getting requirements more complete?

  How can be used domain ontology for improving completeness of requirements?

- Scope

  How to generate possible scopes for requirements? Th project scope is very important with respect to the limited budget of the customer. Requirements are not unambiguous in many ways. One of ambiguity is the solution "luxury" expectation of the customer and the modest implementation of the analyst.

Is it possible to reveal more variants of solution, may be more paths in ontologies, that solve the requirements but have various costs?

- Traceability

  How to generate links between text segments of requirements, UML-elements, and black-box test cases automatically?

The others open problem will be completed later.

## Acknowledgments

This work was partially supported by the Grant Agency of the Czech Republic project P103/11/1489.

# Bibliography

[1] Abbott, R.J.: Program design by informal english descriptions. Communications of the ACM, 26(11), pp. 882894, 1983.

[2] Berry, D.M.: Formal Methods: The Very Idea - Some Thoughts About Why They Work When They Work.

[3] Kalb, H. et al.: BlogForever: D4.1 User Requirements and Platform Specifications Report. Technische Universitt Berlin, 2011.

[4] Banerjee, S.: Adapting the Lesk Algorithm forWord Sense Disambiguation to WordNet. Master Thesis, University of Minnesota, 2002.

[5] Boehm, B.: Software Engineering Economics. Prentice-Hall, Englewood Cliffs, NJ.

[6] Boehm, B.: Identifying Quality Requirements Conflicts. IEEE Software, pp. 25-35, March 1996, 1996.

[7] de Bruijn, F., Dekkers, H.L.: Ambiguity in Natural Languages Software Requirements: A Case Study. In: Wieringa,R. and Persson, A. (Eds.): REFSQ 2010, LNCS 6182, pp. 233-247, Springer, 2010.

[8] Carreno, R.S., et al.: An ontology-based approach to knowledge acquisition from text. Cuadernos de Filologia Inglesa, 9(1), pp. 191-212 (in English), 2000.

[9] Christopherson, L.L.: Use of an ontology-based note-taking tool to improve communication between analysts and their clients. A Masters Paper for the M.S. in I.S.degree, University of North Carolina, November, 2005.

[10] Coad, P., Yourdon, E.: Object-Oriented Analysis. Prentice Hall, 1991.

[11] CHAOS Report, The Standish Group, 1995.

[12] Cranefield, S., Purvis, M.: UML as an Ontology Modelling Language. In: Proc. of the IJCAI99 Workshop on Intelligent Information Integration, Sweden, 1999.

[13] Cycorp Inc., ResearchCyc. http://research.cyc.com

[14] Dalianis, H.: A method for validating a conceptual model by natural language discourse generation. In: Loucopoulos, P. (Ed.): Advanced Inforamtion Systems Engineering, CAiSE92, Manchester, LNCS, Vol. 593, pp. 425-444, Springer, 1992.

[15] Davis, A., Hickey, A.: A Quantitative Assessment of Requirements Engineering Publications - 1963-2008. In: M. Glinz and P. Heymans (Eds.): REFSQ 2009, LNCS 5512, pp. 175189, Springer, 2009.

[16] Deemter, K. V., Krahmer, E., Theune, M.: Real versus template-based natural language generation: A false opposition? In: Computer Linguist., Vol. 31(1), pp.15-24, 2005.

[17] Deeptimahanti, D. K., Sanyal, R.: An Innovative Approach for Generating Static UML Models from Natural Language Requirements. Advances in Software Engineering, Springer, 2009

[18] DeSmedt, K., Horacek, H., and Zock, M.: Architectures for natural language generation: Problems and Perspectives. In: Trends in Natural Language Generation: An Articial Intelligence Perspective. Eds.:Adorni,G., Zock,M., Lecture Notes in Artificial Intelligence, 1036, pp. 17-46, Springer, 1996.

[19] Dorfman, M., and R.H. Thayer (eds.): Standards, Guidelines, and Examples on System and Software Requirements Engineering. IEEE Computer Society Press, Los Alamitos, CA, 1990.

[20] Dressler, W., de Beaugrande, R.: Introduction to text linguistics. Longman, London, 8th edition, 1996.

[21] Emmerich, W.,Kroha, P., Schäfer, W.: Object-Oriented Database Management Systems for Construction of CASE Environments. In: Marik, V. et al (Eds.): Proceedings of the 4th International Conference DEXA'93, Lecture Notes in Computer Sciences, No. 720, Springer, 1993.

[22] Easterbrook, S., Callahan, J., and Wiels, V.: V & V Through Inconsistency Tracking and Analysis. Proceedings of International Workshop on Software Specification and Design, Kyoto, 1998.

[23] Eriksson, H.: Using JessTab to integrate Protege and Jess, Intelligent Systems, Volume 18, Issue 2, pp. 43-50, IEEE Mar-Apr, 2003.

[24] Falbo, R.d.A., Guizzardi, G., Duarte, K.C.: An ontological approach to domain engi- neering. In: Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering, 2002.

[25] Faure, D., Nedellec, C.: Asium: Learning subcategorization frames and restrictions of selection. In: Y. Kodratoff (Ed.): 10th European Conference on Machine Learning (ECML 98) Workshop on Text Mining, Chemnitz Germany, April 1998.

[26] Ferrucci, D., Lally, A.: Accelerating Corporate Research in the Development, Application and Deployment of Human Language Technologies, Techn. Ber., IBM T.J. Watson Research Center, 2003.

[27] Ferrucci, D., et al.: Towards an Interoperability Standard for Text and Multi-Modal Analytics, Techn. Ber., IBM Research Division, November 2006.

[28] Flores, J.J.G.: Semantic Filtering of Textual Requirements Descriptions. In: Natural Language Processing and Information Systems, pp. 474483, 2004

[29] Gasevic, D., Djurevic, D., Devedzic, V.: Model Driven Architecture and Ontology Development, Springer, 2006.

[30] Gelhausen, T.: Modellextraktion aus naturlichen Sprachen. Dissertation am Institut fur Programmstrukturen und Datenorganisation, Lehrstuhl Prof. Dr. Walter F. Tichy, Fakultat fur Informatik, Karlsruher Institut Institut fur Technologie (KIT), (In German), 2010

[31] Gemeinhardt, L.: Connecting TESSI and Rational Rose by means of XML. Project Report, TU Chemnitz, 2000. (In German)

[32] Gervasi, V., Zowghi, D.: On the Role of Ambiguity in RE. In: Wieringa,R. and Persson, A. (Eds.): REFSQ 2010, LNCS 6182, pp. 248-254, Springer, 2010.

[33] Gildea, D., Jurafsky, D.: Automatic labeling of semantic roles. Computational Linguistics, Vol. 28, pp. 245-288, MIT Press, 2002.

[34] Gnesi, S., Lami, G., Trentanni, G.: An automatic tool for the analysis of natural language requirements. CSSE Journal 20(1), pp. 5362, 2005.

[35] Gordon, A., Swanson, R.: Generalizing semantic role annotations across syntactically similar verbs. Proceedings of the 2007 meeting of the Association for Computational Linguistics (ACL-07), Prague, pp. 23-30, 2007.

[36] Gruber, T.: A translation approach to portable ontologies. Knowledge Acquisition, Vol. 5(2), pp. 199220, 1993.

[37] Guizzardi, G., Falbo, R.A., Pereira Filho, J.G.: Using Objects and Patterns to Implement Domain Ontologies. In: Proc. of the 15th Brazilian Symposium on Software Engineering, Rio de Janeiro, Brazil, 2001.

[38] Hillairet, G.: ATL Use Case - ODM Implementation (Bridging UML and OWL), http://www.eclipse.org/m2m/atl/usecases/ODMImplementation.

[39] Hoppenbrouwers, J., van der Vos, A., Hoppenbrouwers, S.: Nl structures and conceptual modelling: the kiss case. In: Proceedings of the 2nd. International Workshop on the application of Natural Language to Information Systems (NLDB96). IOS Press, Amsterdam, The Netherlands, 1996.

[40] Horacek, H.: New Concepts in Natural Language Generation. Pinter Publishers, 1993.

[41] Houpt, J.: Ambiguous Prepositional Phrase Resolution by Humans. Master of Science Artificial Intelligence School of Informatics University of Edinburgh, 2006.

[42] Hovy, E.: Automated discourse generation using discourse structure relations. Artificial Intelligence, Vol. 63, Nr.1-2, 1993.

[43] Jacobson, I.: Object-Oriented Software Engineering: A Use Case Driven Approach. Addison-Wesley, 1992.

[44] Janetzko, R.: Applying ontology for checking of requirements specification, M.Sc. Thesis, Faculty of COmputer Science, TU Chemnitz, 2009. (In German)

[45] Jena Documentation, 2007. URL http://jena.sourceforge.net/documentation.html

[46] Jiang, D., Zhang, S., Wang, Y.: Towards a formalized ontologybased requirements model. Journal of Shanghai Jiaotong University (Science), 10(1), pp. 34-39, 2005.

[47] Jin, Z.: Ontology-based requirements elicitation automatically. Chinese J. Computers, 23(5), pp. 486-492, May 2000.

[48] Jin, Z.: Ontology-Based Requirements Elicitation. Journal of Computers, 23(5), pp. 486-492, 2003.

[49] Jin, Z., Bell, D.A., et al.: Automated requirements elicitation: Combining a model-driven approach with concept reuse. International Journal of Software Engineering and Knowledge Engineering, 13(1), pp. 53-82, 2003.

[50] Johnson, W.L., Feather, M.S., Harris, D.R.: Representation and presentation of requirements knowledge. IEEE Transactions on Software Engineering, pp. 853-869, 1992.

[51] Jones, C.: Software challenges. Comput. industry 28, 10, pp. 102103, 1995.

[52] Jones, C.: Patterns of Software Systems Failure and Success. International Thomson Computer Press, London, 1996.

[53] Kaiya, H., Saeki, M.: Using Domain Ontology as Domain Knowledge for Requirements Elicitation. In: Proceedings of 14th IEEE International Requirements Engineering Conference, Minnesota, pp. 186-195, 2006.

[54] Kamsties, E., Daniel M. Berry, D.M., Paech, B.: Detecting Ambiguities in Requirements Documents Using Inspections.

[55] Karpati, P., Sindre, G., Opdahl, A.L.: Visualizing Cyber Attacks with Misuse Case Maps. In:

[56] Kim, Y.-G.: Effects of Conceptual Modeling Formalism on User Validation and Analyst Modeling of Information Requirements. PhD. Thesis, University of Minnesota, 1990.

[57] Kiyavitskaya, N., Zeni, N., Mich, L., Mylopoulos, J.: Experimenting with Linguistic Tools for Conceptual Modelling: Quality of the Models and Critical Features. In: Meziane,F., Metais, E. (Eds.):Proceedings of NLDB 2004, LNCS 3136, pp. 135-146, 2004.

[58] Klein, D. and Manning, C.: Accurate unlexicalized parsing. In Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL 2003), pp.423-430, 2003.

[59] Klein, D. and Manning, C.: Fast exact inference with a factored model for natural language parsing. In: NIPS, Volume 15, MIT Press, 2003.

[60] Kof, L.: Translation of Textual Specifications to Automata by Means of Discourse Context Modeling. In: REFSQ '09 Proceedings of the 15th International Working Conference on Requirements Engineering: Foundation for Software Quality, Springer, 2009

[61] Kof, L.: Using Application Domain Ontology to Construct an Initial System Model. In: IASTED International Conference on Software Engineering, 2004.

[62] Kof, L.: Text Analysis for Requirements Engineering. Ph.D. Thesis, TU Mnchen, 2005.

[63] Kof, L.: Natural Language Processing: Mature Enough for Requirements Documents Analysis? In: Montoyo, A.; Munoz, R., Metais, E. (Eds.): NLDB, LNCS 3513, 91-102, Springer, 2005.

[64] Kof, L., Gacitua, R., Rouncefield, M., Sawyer, P.: Ontology and Model Aligment as a Means for Requirements Validation. In: Proceedings of ICSC'10, pp. =46-51, 2010.

[65] Kof, L.: From Requirements Documents to System Models: a Tool for Interactive Semi-Automatic Translation.

[66] Kof, L., Schatz, B.: Combining aspects of reactive systems. In: Broy, M., Zamulin, A.V. (eds.) PSI 2003. LNCS, Vol. 2890, pp. 344349, Springer, 2004.

[67] Kof, L.: Scenarios: Identifying missing objects and actions by means of computational linguistics. In: 15th IEEE International Requirements Engineering Conference, New Delhi, India, pp. 121130. IEEE Computer Society Conference Publishing Services, Los Alamitos, 2007.

[68] Kof, L.: Treatment of Passive Voice and Conjunctions in Use Case Documents. In: Kedad, Z.,Lammari, N., Metais, E., Meziane, F., Rezgui, Y. (eds.) NLDB 2007. LNCS, vol. 4592, pp. 181192, Springer, 2007.

[69] Kof, L.: From Textual Scenarios to Message Sequence Charts: Inclusion of Condition Generation and Actor Extraction. In: 16th IEEE International Requirements Engineering Conference, Barcelona, Spain, pp. 331332. IEEE Computer Society, 2008.

[70] Gleich, B., Creighton, O., Kof, L.: Ambiguity Detection: Towards a Tool Explaining Ambiguity Sources.

[71] Kof, L.: Requirements Analysis: Concept Extraction and Translation of Textual Specifications to Executable Models.

[72] Kof, L.: From Textual Scenarios to Massage Sequence Charts: Inclusion of Condition Generation and Actor Extraction

[73] Kof, L.: On the Identification of Goals in Stakeholders Dialogs

[74] Kof, L.: Treatment of Passive Voice and Conjunctions in Use Case Documents

[75] Kof, L.: Scenarios: Identifying Missing Objects and Actions by MEans of Computational Linguistics

[76] Kof, L.: Natural Language Processing: Mature Enough for Requirements Documents Analysis?

[77] Kof, L.: An Application of Natural Language Processing to Domain Modelling - Two Case Studies, 2004.

[78] Kof, L.: Application Domain Ontology to Construct an Initial System Model

[79] Korner S. J., Gelhausen T.: Improving Automatic Model Creation Using Ontologies. In Knowledge Systems Institute (ed.): Proceedings of the Twentieth International Conference on Software Engineering - Knowledge Engineering, pp. 691-696, 2008.

[80] Krachina, O., Raskin, V.: Ontology-Based Inference Methods. Tech report number CERIAS TR 2006-76.

[81] Kroha, P.: Objects and Databases. McGraw-Hill, 1993.

[82] Kroha, P.: Softwaretechnologie. Prentice Hall, 1997 (in German).

[83] Kroha, P., Strauß, M.: Requirements Specification Iteratively Combined with Reverse Engineering. In: Plasil, F., Jeffery, K.G. (Eds.): Proceedings of SOFSEM'97: Theory and Practice of Informatics, Lecture Notes in Computer Science, No. 1338, Springer, 1997.

[84] Kroha, P.: Preprocessing of Requirements Specification. In: Ibrahim, M., Kng, J., Revell, N. (Eds.): Proceedings of DEXA2000, London, Lecture Notes in Computer Science, No. 1873, Springer, 2000.

[85] Kroha, P., Rosenhainer, L.: Textuelle Anforderungen und Software Migration. In: Kaiser, U., Kroha, P., Winter, A. (Hrsg.): 3. Workshop Reengineering Prozesse (RePro 2006) Software Migration. Informatik Bericht Nr. 2/2006, Institut fr Informatik, Fachbereich 08, Johannes Gutenberg-Universitt Mainz, November 2006. Reprinted in: GI - Softwaretechnik-Trends, Band 27, Heft 1, Februar 2007.

[86] Kroha, P., Labra Gayo, J. E.: Using Semantic Web Technology in Requirements Specifications. Chemnitzer Informatik-Berichte CSR-08-02, ISSN 0947-5125, TU Chemnitz, November 2008.

[87] Kroha, P., Janetzko, R., Labra, J. E.: Ontologies in Checking for Inconsistency of Requirements Specification. In: Dokoohaki, N., Zavoral, F., Noll, J.(Eds.): Proceedings of the 3rd International Conference on Advances in Semantic Processing SEMAPRO 2009, IEEE Computer Society, Sliema, Malta, October 2009.

[88] Kroha, P., Rink, M.: Text Generation for Requirements Validation. In: Filipe, J., Cordeiro, J.(Eds.): Proceedings of the 11th International Conference Enterprise Information Systems - ICEIS 2009, Milano, Lecture Notes in Business Information Processing, Nr. 24, pp. 467-478, Springer, May 2009.

[89] Lamsweerde, A.V., Darimont, R., Letier, E.: Managing Conflicts in Goal-Driven Re- quirements Engineering. IEEE Trans. Software Eng. 24(11), 1998.

[90] Lamsweerde, A.V.: Requirements Engineering: From System Goals to UML Models to Software Specifications. Wiley, 2009.

[91] Lavoie, B., Rambow, O., Reiter,E.: The ModelExplainer. In: Demonstration Notes of International Natural Language Generation Workshop, INLG'96, Harmonceux Castle, Sussex, UK, 1996.

[92] Lavoie,B., Rambow, O., Reiter, E.: A Fast and Portable Realizer for next Generation Systems. In: Proceedings of the 5th Conference on Applied Natural Language Processing, ANLP'97, Washington, 1997.

[93] Lee, Y., Zhao, W.: An Ontology-Based Approach for Domain Requirements Elicitation and Analysis. In: Proceedings of the First International Multi-Symposiums on Computer and Computational Sciences, 2006

[94] Leidenfrost, S.: TESSI in Java. Project report, TU Chemnitz, 1999. (In German)

[95] Li, X., Szpakowicz, S., Stan Matwin, S.: A WordNet-based Algorithm for Word Sense Disambiguation, 1995.

[96] Li, Z., Wang, Z., Zhang, A., Xu, Y.: The Domain Ontology and Domain Rules Based Requirements Model Checking. International Journal of Software Engineering and Its Applications, Vol. 1, No. 1, July, 2007.

[97] Mann, W.: Text generation: The problem of text structure. In Natural Language Generation Systems, Eds.: D.D. McDonald, Bolc,L., pp. 47-68, Springer, 1988.

[98] Marcus, M.P., Santorini, B., Marcinkiewicz, M.A.: Building a Large Annotated Corpus of English: the Penn Treebank. Computational Linguistics, vol. 19, 1993.

[99] Marneffe, M., MacCartney, B., Manning, Ch.D.: Generating Typed Dependency Parses from Phrase Structure Parses. In Proceedings: The International Conference on Language Resources and Evaluation LREC 2006.

[100] Mich, L., Franch, M., Novi, I.P.: Market research for requirements analysis using linguistic tools. Requirements Engineering, Vol. 9, pp. 4056, 2004.

[101] Neches, R., Fikes, R.E., Finin, T.: Enabling technology for knowledge sharing. AI Magazine, 12 (3), pp. 36-56, 1991.

[102] Nuseibeh, B., Easterbrook, S., Russo, A.: Leveraging Inconsistency in Software Development. IEEE Computer, April 2000.

[103] Nuseibeh, B., Easterbrook, S.: Requirements Engineering: A Roadmap. Proceedings of International Conference on Software Engineering ICSE-2000, pp. 4-11, Limerick, Ireland, ACM Press, June 2000.

[104] Nuseibeh, B., Haley, C.B., and Foster C.: Securing the Skies: In Requirements We Trust. IEEE Computer, 42(9):64-72, September 2009.

[105] Omoronyia, I. et al.: A Domain Ontology Building Process for Guiding Requirements Elicitation. In: R. Wieringa, R., and Persson, A. (Eds.): Proceedings of REFSQ 2010, LNCS 6182, pp. 188202, Springer, 2010.

[106] Paiva, D.: A survey of applied natural language generation systems. http://citeseer.ist.psu.edu/paiva98 survey.html.

[107] Pereira, A.R., Ziviani, N.: Syntactic Similarity of Web Documents. In: Proceedings of the First Latin American Web Congress (LA-WEB 2003), IEEE, 2003.

[108] Perrouin, G., Brottier, ., aundry, B., Le Traon, Y.: Composing Models for Detecting Inconsistencies: A Requirements Engineering Perspective. In: M. Glinz and P. Heymans (Eds.): REFSQ 2009, LNCS 5512, pp. 89103, Springer, 2009

[109] Petre, M.: Why looking isn't always seeing: Readership skills and graphical programming. Communication of the ACM, Vol. 38, No. 6, pp. 33-42, 1995.

[110] Pohl, K.: Requirements Engineering - Fundamentals, Principles, and Techniques. Springer, 2010.

[111] Pomerantz, J.: A Linguistic Analysis of Question Taxonomies. Journal of the American Society for Information Science and Technology, 56(7), pp. 715-728, 2005.

[112] RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation 10, February 2004.

[113] Realpro: General english grammar, user manual. 2000. http://www.cogentex.com/papers/realpro-manual.pdf.

[114] Reiter, E., Dale, R.: Building natural language generation systems. In: Studies in Natural Language Processing, Journal of Natural Language Engineering, Vol. 3(1), pp.57-87, 1997.

[115] Rational RequisitePro - User's Guide. Rational Software Corporation, 2003.

[116] Rational Suite Analyst Studio - Getting Started. Rational Software Corporation, 2003.

[117] Rink, M.: Text generator implementation for description of UML 2 models. MSc. Thesis, TU Chemnitz, 2008, (In German).

[118] Robertson, S., Robertson, J.: Mastering the Requirements Process. Addison-Wesley, 1999.

[119] Rubin,K., Goldberg,A.: Object Behaviour Analysis. Communication of ACM, Vol. 35, No.9, pp. 48-62, September 1992.

[120] Robinson, M., Bannon, L.: Questioning representations. In: Proceedings of ECSCW, Amsterdam. http://www.ul.ie/ idc/library/papersreports/ LiamBannon/ 15/QuestFin.html.

[121] Robinson, W.: Interactive Decision Support for Requirements Negotiation. Concurrent Engineering: Research & Applications, 2, pp. 237-252, 1994.

[122] Robinson, W., Fickas, S.: Supporting Multiple Perspective Requirements Engineering. In: Proceedings of the 1st International Conference on Requirements Engineering (ICRE 94), IEEE Computer Society Press, pp.206-215, 1994.

/bibitemRobinson1997 Robinson,W.: I Didnt Know My Requirements were Consistent until I Talked to My Analyst. Proceedings of 19th International Conference on Software Engineering (ICSE-97), IEEE Computer Society Press, Boston, USA, May , pp. 17-24, 1997.

[123] Robinson, W., Pawlowski, S.: Managing Requirements Inconsistency with Development Goal Monitors. IEEE Transactions on Software Engineer, November/December, 1999.

[124] Robinson, W.N., Pawlowski, S.D., Volkov, V.: Requirements interaction management. ACM Comput. Surv. 35(2), pp. 132190, 2003.

[125] Sagae, K., Gordon, A.: Clustering Words by Syntactic Similarity Improves Dependency Parsing of Predicate-Argument Structures. International Conference on Parsing Technologies (IWPT-09), Paris, pp. 7-9, 2009.

[126] Santorini, B.: Part-of-Speech Tagging Guidelines for the Penn Treebank Project (3rd Revision, 2nd printing), Feb 1995.

[127] Schmid, H.: Probabilistic Part-of-Speech Tagging Using Decision Trees. Proceedings of International Conference on New Methods in Language Processing, Manchester, 1994.

[128] Schmid, H.: Improvements in Part-of-Speech Tagging with an Application to German. Proceedings of the ACL SIGDAT-Workshop. Dublin, 1995.

[129] Scott, D., de Souza, C.: Conciliatory planning for extended descriptive texts. Technical Report 2822, Philips Research Laboratory, Redhill.

[130] Shibaoka, M., Kaiya, H., Saeki, M.: GOORE: Goal-Oriented and Ontology Driven Requirements Elicitation Method. In: Hainaut, J.-L., Rundensteiner, E.A., Kirchberg, M., Bertolotto, M., Brochhausen, M., Chen, Y.-P.P., Cherfi, S.S.-S., Doerr, M., Han, H., Hartmann, S., Parsons, J., Poels, G., Rolland, C., Trujillo, J., Yu, E., Zimanyie, E. (eds.): ER Workshops, LNCS, Vol. 4802, pp. 225234, Springer, 2007.

[131] Simpson, T., Dao, T.: WordNet-based semantic similarity measurement. URL http://www.codeproject.com/cs/library/semanticsimilaritywordnet.asp

[132] Spanoudakis, G., Zisman, A.: Inconsistency Management in Software Engineering: Survey and Open Research Issues. In: Chang, S.K.(Ed.): Handbook of Software Engineering and Knowledge Engineering, World Scientific Publishing Co., pp. 329-380, 2001.

[133] Spence, I., Probasco, L.: Traceability Strategies for Managing Requirements with Use Cases. Rational Software White Paper, 2000

[134] Swartout,B.: GIST English Generator. In: Proceedings of the National Conference on Artificial Intelligence, AAAI'82.

[135] Thayer, R.H., Dorfman, M.: System and software requirements engineering. IEEE Computer Society Press, 1990.

[136] Thayer, R.H.: Software System Engineering: A Tutorial. Computer, April 2002.

[137] Thayer, R.H.: Software Engineering Management. In: Software Engineering: Volume 2: The Supporting Processes, (Eds.):Richard H. Thayer and Mark J. Christensen, IEEE Computer Society Press, Los Alamitos, CA, 2002.

[138] Thiagarajan, R., Manjunath, G., Stumptner, M.: Computing Semantic Similarity Using Ontologies. ISWC 08, The International Semantic Web Conference (ISWC), Karlsruhe, 2008.

[139] Toutanova, K., Manning,C.D.: Enriching the Knowledge Sources Used in a Maximum Entropy Part-of-Speech Tagger. In: Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP/VLC-2000), pp. 63-70, 2000.

[140] Toutanova, K., Klein, D., Manning, C., and Singer, Y.: Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network. In: Proceedings of HLT-NAACL 2003, pp. 252-259, 2003.

[141] Uschold, M., King, M., Moralee, S., Zorgios, Y.: The Enterprise Ontology. Knowledge Engineering Review, 13(1), pp. 31-89, 1998.

[142] Verma, K., Kass, A.: Requirements Analysis Tool: A Tool for Automatically Analyzing Software Requirements Documents. In: Sheth, A. et al. (Eds.): Proceedings of ISWC 2008, LNCS 5318, pp. 751763, Springer, 2008.

[143] Videira, C., Ferreira, D., da Silva, A.: A linguistic patterns approach for requirements specification. In: Proceedings of the 32nd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO-SEAA06), 2006.

[144] W3C: OWL Web Ontology Language Guide, Feb 2004.

[145] W3C: OWL Web Ontology Language Overview, Jan 2004.

[146] W3C: RDF Primer, Feb 2004.

[147] W3C: RDF Vocabulary Description Language 1.0: RDF Schema, Feb 2004.

[148] W3C: RDF/XML Syntax Specification, Feb 2004.

[149] W3C: Resource Description Framework (RDF): Concepts and Abstract Syntax, Feb 2004.

[150] W3C: SPARQL Protocol for RDF, October 2006.

[151] W3C: SPARQL Query Language for RDF, April 2006.

[152] Weidauer, J.: Implementierung eines Teilsystems fur die Transformation eines OWL-Modells nach UML 2 und Neuentwicklung des Hilfesystems im Rahmen des Projektes TESSI. Master Thesis, TU Chemnitz, 2008

[153] Wenzel, K.: Informationsextraktion aus semistrukturierten Dokumenten mit Hilfe von Mustern. Project-Report, Department of Computer Science, University of Technology, Chemnitz, 2006 (In German).

[154] Wenzel, K.: Implementierung eines neuen Prototyps fr das CASE-WErkzeug TESSI unter Integration eines Analysesystems fr natrliche Sprache und Ontologien. Master Thesis, Department of Computer Science, University of Technology, Chemnitz, 2007 (In German).

[155] Wiegers, K.: Software Requirements. Microsoft Press, 2003.

[156] Zhu, X., Zhi, Jin: Inconsistency Measurement of Software Requirements Specifications an Ontology-Based Approach. In: Proceedings of the 10th IEEE International Conference on engineering of Complex Computer Systems, 2005.