Institut für Informatik
der Technischen Universität München

Text Analysis for Requirements Engineering

Leonid Kof

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender: Univ.-Prof. Nassir Navab, Ph. D.

Prüfer der Dissertation:

1. Univ.-Prof. Dr. Dr. h.c. Manfred Broy
2. Univ.-Prof. Michael Beetz, Ph. D.

Die Dissertation wurde am 5.07.2005 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 21.11.2005 angenommen.

Kurzfassung

Requirements Engineering ist die Achillesferse des gesamten Prozesses der Softwareentwicklung. Es erfordert Interaktion vieler Beteiligten und beinhaltet nicht nur technische, sondern auch soziologische und psychologische Aktivitäten. Auch wenn alle Beteiligte zu einem Konsens kommen, ist das resultierende Anforderungsdokument meist informell. In den frühen Projektphasen ist die Funktionalität der zu erstellenden Software noch nicht genau genug verstanden. Das macht den Prozess der Formalisierung der Anforderungen zu einem Lernprozess.

Wie die Studie von Mich et al. [MFN04] zeigt, wird die überwiegende Menge der Anforderungen in natürlicher Sprache geschrieben. In der Praxis sind solche Dokumente meistens vage und enthalten viele Inkonsistenzen. Missverständnisse und Fehler aus der Requirements Engineering-Phase wirken sich in späteren Projektphasen aus und können potentiell zum Misserfolg des gesamten Projekts führen.

Um Missverständnisse in den Griff zu bekommen und den Schritt von informellen Anforderungen zu einem formalen Modell zu unterstützen, wird in dieser Dissertation ein neuer Ansatz zur Extraktion der domänenspezifischen Ontologie aus Anforderungsdokumenten vorgeschlagen. Eine Ontologie besteht aus einer Menge von Termen und Relationen zwischen diesen Termen. Sie gibt, verglichen mit einem Glossar, präzisere Begriffsdefinitionen und lässt weniger Interpretationsspielraum. Nach der Extraktion muss die Ontologie auch validiert werden. Die validierte Ontologie wird zu *der* gemeinsamen Sprache für alle Projektbeteiligte.

Diese Dissertation macht zwei wichtige Beiträge zur Extraktion der Ontologie aus Anforderungsdokumenten:

- Sie implementiert eine halbautomatische Methode, die eine Ontologie aus einem Anforderungsdokument extrahiert und diese Anschließend validiert.
- Sie zeigt, wie der traditionelle Prozess der Anforderungsanalyse modifiziert werden soll, um die Ontologieextraktion in den Prozess zu integriereren.

Das vorgeschlagene Verfahren zur Ontologieextraktion wurde an drei Fallstudien evaluiert.

Abstract

Requirements Engineering is the Achilles' heel of the whole software development process. It involves many stakeholders and includes not only technical but also sociological and psychological activities. Even when all the stakeholders come to a consensus, the produced requirements are rather informal. In the early project phases the functionality of the prospective software is not yet understood in the precision necessary for formalization, which makes requirements formalization not only a refinement, but also a learning process.

As the survey by Mich et al. [MFN04] shows, the overwhelming majority of requirements are written in natural language. In practice these documents are often vague and contain a lot of ambiguities, which causes misunderstandings between project stakeholders. Misunderstandings and errors of the requirements engineering phase propagate to later development phases and can potentially lead to a project failure.

To alleviate misunderstanding and to support the step from informal requirements to a formal model this thesis proposes a novel approach to the extraction of a domain ontology from requirements documents in order to establish a common language for the project stakeholders. An ontology consists of a set of terms and relations between these terms. As compared to a glossary, a domain-specific ontology gives a more explicit definition of terms and relations between them. When the ontology is extracted, a domain expert validates it. The validated ontology becomes both *the* common language for all the project stakeholders and a valuable resource for later development steps.

The thesis makes two key contributions to ontology extraction as a part of requirements analysis:

- It implements a semiautomatic method, extracting an ontology from a requirements document and validating the extracted ontology.
- It shows how traditional requirements analysis process should be modified to include ontology extraction and validation.

The feasibility of the proposed approach was evaluated on three comprehensive case studies.

Acknowledgements

This thesis was made possible by help and cooperation of many people. Here I want to use the opportunity to thank them. Without their help and support this thesis had never been written.

First of all, I want to thank Professor Manfred Broy who offered me a position at his chair. Discussions with colleagues in this great research group gave me a lot of helpful suggestions for my research. Furthermore, I want to thank Professor Manfred Broy for his support of research that does not belong to the research mainstream at the chair. Without his support and invaluable feedback I could never have carried out the research presented in this thesis.

My thanks also go to Professor Michael Beetz, who, when asked, readily agreed to participate at the dissertation committee. His extremely short review cycles and constructive reviews enabled almost on-the-fly improvements in the final stage of thesis writing. Fruitful discussions with him contributed a lot to the final structuring of the thesis and to producing a clear line of argument.

Some phases of my work were really disappointing, as it seemed almost impossible to publish my ideas. It was Markus Pizka who encouraged me in such phases not to give up and gave valuable tips on how to write papers. I want to thank him here.

I am also obliged to the colleagues who agreed to review the almost final versions of my dissertation. Daniel Ratiu, Tilman Seifert, Jorge Fox, David Cruz, and Stefan Wagner contributed a lot to polishing the thesis and improving understandability.

Case studies on different requirements documents were an important part of my work. It was really difficult to find documents suitable for case studies and at the same time not completely secret. I am really grateful to the people who provided ideas and documents for case studies: Alexander Pretschner, Franz Huber, Jan Philipps, Markus Pister, Jewgenij Botaschanjan, Andreas Fleischmann, and Brian Berenbach. Without their case study ideas this thesis could not have been completed.

The case studies gave rise to the questions whether the documents prepared for automated text analysis are still human readable. Some of my helpful colleagues agreed to read and evaluate different versions of different documents. For this painstaking work I want to thank Stefan Berghofer, Martin Deubler, Norbert Diernhofer, Ulrike Hammerschall, Jan Jürjens, Michael Meisinger, Jan Philipps, Yuri Riabov, Maurice Schoenmakers, Tilman Seifert, Oscar Slotosch, Martin Strecker, Stefan Wagner, Martin Wildmoser, Guido Wimmel, and Alexander Ziegler.

The tools that I used during my research were all research tools, some of them not available off-the-shelf. It is the cooperation by the tool authors that made certain parts of my work possible. I am deeply grateful for this cooperation to Helmut Schmid, Sabine Schulte im Walde, David Faure, Claire Nédellec, Alain-Pierre Manine, Philipp Cimiano, Johanna Völker, Thomas Büchner, Tobias Hain, Alexander Klitni and Armand Wendt. I had solely e-mail contact with most of these people, which makes their readiness to cooperate even more worthily.

I also owe a special thank to Barbara Kalter, who helped a lot with the formalities of the dissertation submission and diminished my chaotic tendencies.

Last but not least, I want to thank my friends and my parents for their continuous support and encouragement during the work.

Contents

List of Figures

Chapter 1

Introduction

Construction of software systems is a non-trivial and error-prone task. In spite of the general understanding which steps are necessary in the development process (requirements engineering, architecture design, etc.), proper execution of these steps remains problematic. This problem becomes especially acute when constructing large software systems.

The understanding of the fact that development of large software systems requires a systematic approach, as opposed to ad-hoc programming, gave rise to the research field of software engineering. Software engineering is

> 1. The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.
> 2. The study of approaches as in (1).

(IEEE Standard 610-1990, see also [IEE05]). Software engineering traditionally subdivides the software development process in several phases, such as requirements engineering, design, implementation, and testing.

Although it makes no sense to say which phase is more important, it is rather obvious that requirements engineering is a crucial one: errors made in the requirements engineering phase propagate to all the later stages. For this reason correction of requirements engineering errors is also extremely expensive: according to Boehm [Les05], the cost of the error correction increases by the factor of 10 when the error is detected in a later project phase. Thus, a correction of a requirements engineering error in the design phase is 10 times more expensive than a direct correction in the requirements engineering phase, and the correction in the implementation phase is even 100 times more expensive. The later in the development process the error is detected, the higher the correction cost.

Zave [Zav97] defines requirements engineering as

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

Requirements engineering process poses manifold challenges, because it involves not only technical, but also psychological and sociological aspects, such as interaction of different stakeholders and requirements negotiation. As Jackson states, requirements engineering is "where informal meets formal" (cited after Berry [Ber03]). Supporting the step from informal to formal is one of the goals of the presented work.

The result of the early requirements engineering phases, namely requirements elicitation and negotiation, is a requirements document. As the survey by Mich et al. [MFN04] shows, the overwhelming majority of requirements are written in natural language. Practice shows that the natural language requirements documents mostly contain plenty of inconsistencies. In the requirements engineering phase it is vital to detect these inconsistencies and at least to establish an inconsistency-free common project language.

One of the possible definitions of a common project language would be a glossary of domain-specific terms. A glossary gives an informal natural language definition for each term. However, such definitions still leave room for interpretations. (An example of a possible misinterpretation will be given later, see Section 1.1.3.) A better, more explicit, term definition is an ontology. Contrary to the glossary, which is basically a plain term list, an ontology contains explicit relations between concepts. This thesis proposes ontology engineering as a promising way to define terms specific to the application domain. Wikipedia, the free encyclopedia, defines an ontology in the following way [Wik05b]:

> In computer science an ontology is an attempt to formulate an exhaustive and rigorous conceptual schema within a given domain, a typically hierarchical data structure containing all the relevant entities and their relationships and rules (theorems, regulations) within that domain.

The goal of the presented thesis is to build an application domain ontology on the basis of requirements documents.

The definition of an ontology is a first step towards a uniform project language. In order that the ontology can be really used as a common project language, it must be validated. Validation means in this context that an application domain expert approves the extracted terms and associations. The validation of the constructed ontology can take place in two ways: either via manual validation by a domain expert or via building an initial system model on the ontology basis and validating the model. The validated ontology then becomes *the* common language for all the project stakeholders. Furthermore, ontology validation indirectly contributes to the validation of the requirements document.

Ontology extraction, as proposed in this thesis, is based on the following scenario:

1. Requirements engineering starts mostly with rather vague ideas, and with different stakeholders having different ideas about the prospective project. Then, the goals of the different stakeholders are discussed and goal conflicts are detected. The conflicts must be negotiated and eliminated. The final result of this elicitation stage is a requirements document, agreed upon by all the stakeholders.

2. An ontology is extracted from this document. The process of ontology extraction consists of three steps:

   (a) term extraction (glossary extraction)

   (b) term classification, building of the term hierarchy

   (c) relation extraction

   The second and the third step are *interactive* and give the requirements analyst feedback on terminology inconsistencies. It is important to eliminate these inconsistencies *before* they find their way into the ontology (the requirements engineering process goes back to the step of requirements elicitation, negotiation, and writing). This interactive process of ontology extraction and document correction has the invaluable side-effect of validating the terminology that will be used in later project phases.

3. The extracted ontology is validated by all the stakeholders. The validated ontology becomes the communication basis (the common project language) for all the project participants.

The hypothesis underlying this approach is that the additional resources necessary to revise the requirements document and to extract an ontology will result in early discovery and elimination of inconsistencies in the requirements document. Furthermore, the extracted domain ontology, serving as the uniform project language, results in better understanding between project participants. The payoff is the absence of misunderstanding-based errors in later project phases. According to Boehm [Les05], this reduces the cost of error correction.

The remainder of this chapter gives an overview over the thesis: Section 1.1 sketches requirements engineering itself, Section 1.2 shows the role of ontology construction in requirements engineering, Section 1.3 discusses the differences of the presented thesis from other existing approaches, and Section 1.4 gives an overview over the remainder of the thesis.

1.1 Short Introduction to Requirements Engineering

A typical requirements engineering process includes initial requirements clarification and negotiation. These steps are presented in Section 1.1.1. The early requirements engineering steps, like requirements elicitation and negotiation, are rather psychological and sociological than technical activities. They deal with the identification of project stakeholders, integrating different stakeholders' views on the system, defining each party's project goals, finding and negotiating goal conflicts, etc. These activities are presented in Section 1.1.2.

When the requirements are negotiated and written down in a requirements document, they are analyzed for completeness and contradictions. To analyze the requirements, it is also necessary to understand the domain-specific terminology, possibly to build an application domain model, evaluate realizability of particular requirements and so on. The result of this analysis is either an approval of the requirements document or a return to the elicitation and negotiation phase. This analysis process is described in Section 1.1.3.

The requirements engineering process, as described above, is cyclic. This gives rise to the legitimate question when the requirements engineering is finished. To put it in a nutshell, it is finished when each requirement is validated and the requirements set is contradiction-free. This question is discussed in more detail in Section 1.1.4.

1.1.1 Requirements Engineering Process

The requirements engineering stage is an important part of any project, because requirements engineering ensures that the specification of the product to be built meets customer's wishes. (Are we building the *right* product?) The goal of the later development stages, to the contrary, is to ensure that a product is being built correctly *with respect to the specification*, produced in the RE phase. Therefore, requirements engineering errors can potentially lead to project failure or must be corrected in later phases, which is much more expensive than correction in the RE phase.

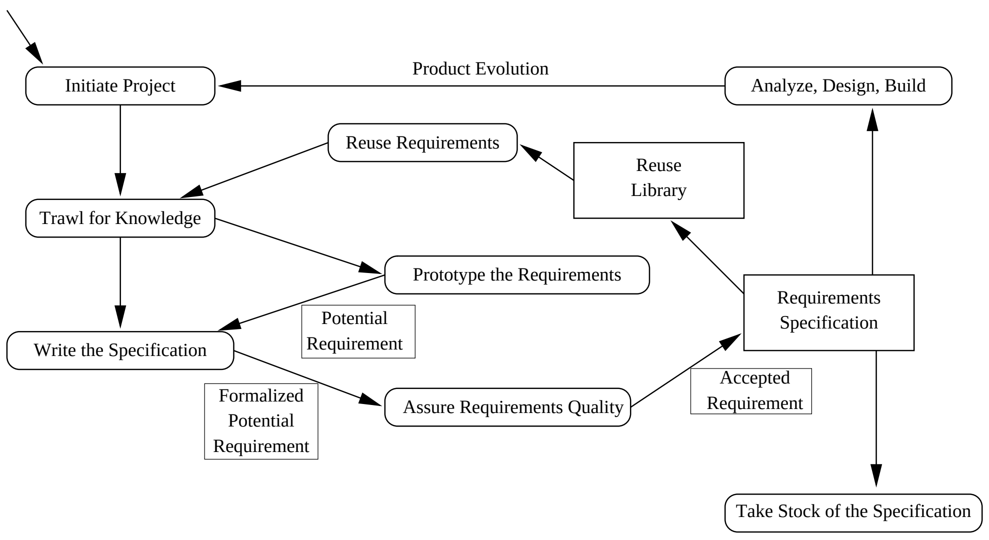


Figure 1.1: Requirements engineering process according to Robertson and Robertson [RR99]

Requirements engineering basically consists of two major activities: requirements elicitation and requirements analysis. Figure 1.1 shows the requirements engineering process suggested by Robertson and Robertson [RR99]. All the steps of this process can be subdivided into three categories:

requirements elicitation consists of the steps “trawl for knowledge”, “prototype the requirements”, and “reuse requirements”. The goal of the steps is to acquire domain knowledge and to gather requirements to the prospective system.

requirements analysis consists of the steps “assure requirements quality” and “take stock of the specification”. The goal of these steps is, on the one hand, to check that the requirements satisfy certain quality criteria, and, on the other hand, to make the requirements specification the information source for later project stages.

supportive steps are “initiate the project”, “write the specification”, and “analyze, design, build”. “Write the specification” belongs neither to requirements elicitation nor to requirements analysis, but it connects these two activities. “Initiate the project” and “analyze, design, build” are not a part of the requirements engineering process. They just connect requirements engineering to other project activities.

The following two sections present requirements elicitation and requirements analysis in more detail.

1.1.2 Requirements Elicitation

Requirements elicitation is the first step in the requirements engineering process. The primary goal of requirements elicitation is to identify the customers' objectives. Elicitation would not be an issue if there existed a single person knowing everything that the system under development is supposed to do. However, in practice there are often many stakeholders, each having a different view on the prospective system. For this reason requirements elicitation approaches have to focus on the social aspects of stakeholder interaction.

At the beginning of the elicitation process each stakeholder explicitly states his business goals. The idea behind the definition of the goals is that they are more stable than requirements. Each business goal may be satisfied by a different set of requirements, which, in turn, may be implemented via a different system. Van Lamsweerde [vL01] defines a business goal as an "objective the system under consideration should achieve". The goals are needed, for example, to achieve requirements completeness, to avoid irrelevant requirements and to explain requirements to stakeholders.

Goals and requirements of different stakeholders often interfere. As long as there are contradictions in the goals or requirements of a single stakeholder, it is up to this particular stakeholder to resolve the contradictions. However, if there exist conflicting goals/requirements of different stakeholders, a special procedure is necessary to come to an agreement. Grünbacher, Boehm and Briggs [GBB02] proposed a procedure for such an agreement. Their procedure (EasyWinWin) consists of the following steps:

Review and expand negotiation topics: Stakeholders jointly refine and customize the outline of negotiation topics based on a domain taxonomy of software requirements. The shared outline helps to stimulate thinking, to organize negotiation results, and serves as a completeness checklist during negotiations.

Brainstorm stakeholder interests: Stakeholders share their goals, perspectives, views, and expectations by gathering statements about their win conditions.[1]

Converge on win conditions: The team jointly craft a non-redundant list of clearly stated, unambiguous win conditions by considering all ideas contributed in the brainstorming session.

[1]A stakeholder's win condition is some goal he wants to achieve (not necessarily a business goal). Win-Win means that at the end of the negotiation process everybody perceives himself as a winner.

Capture a glossary of terms: Stakeholders define and share the meaning of important keywords of the project/domain in a glossary.

Prioritize win conditions: The team prioritizes the win conditions to define and narrow down the scope of work and to gain focus.

Reveal issues and constraints: "Issues" in the EasyWinWin terminology are conflicting win conditions for different stakeholders. Stakeholders should surface and understand issues by analyzing the prioritization poll.

Identify issues, options: Stakeholders register constraints and conflicting win conditions as Issues and propose Options to resolve these issues.

Negotiate agreements: The captured decision rationale provides the foundation to negotiate agreements.

The third step, *converge on win conditions*, is the most important one: only when everybody knows what is important for other stakeholders, it is possible to reach a consensus perceived as a "win" for everybody.

At the end of this negotiation process requirements are written down. The resulting document contains all the requirements of all the stakeholders. In the next requirements engineering step this document is analyzed in order to better understand the application domain and the prospective system.

1.1.3 Requirements Analysis and Domain Modelling

The goal of the requirements analysis is a precise description of the application domain, of the interface between the system to be developed and its environment, and of the interaction of the system with the environment. The following presentation focuses on application domain description as requirements engineering task, just because domain modelling is the core of the presented thesis.

Description of the application domain includes definition of the domain-specific terminology. In practice, terminology definition is mostly done in the form of glossary, but glossaries are not sufficiently precise. An example by Zave and Jackson [ZJ97] illustrates this potential glossary imprecision. Their example handles a hypothetical university information system and the definitions of a "student" and the binary relation "enrolled" for this system:

Able: Two important basic types are *student* and *course*. There is also a binary relation *enrolled*. If types and relations are formalized as predicates, then

$$\forall s\ \forall c\ (enrolled(s, c) \Rightarrow student(s) \land course(c)).$$

Baker: Do only students enroll in courses? I don't think that's true.
Able: But that's what I mean by *student*!

This example shows that it is not sufficient solely to introduce a glossary with application domain terms. Although Able and Baker *do* agree that the term "student" is an important domain concept, they disagree on the meaning of this concept. A more appropriate definition of domain concepts and relations between them would be an *ontology*.

In computer science an ontology consists of concepts specific to a particular domain and relations between these concepts. The presence of explicit relations between the concepts is the key difference of an ontology from a simple glossary. Relations between terms can be the subsumption ("is-a") relation as well as other relation types. The above example shows the importance of relations: explicit definition of a "student" as a special case of a "course participant" would prevent the misunderstanding between Able and Baker.

The usefulness of an ontology as a requirements engineering product was recognized also by other scientists. For example, Breitman and Sampaio do Prado Leite [BS03] see an application ontology as one of the products of the requirements engineering activity. Surely, the ontology is not the only result of requirements engineering, but a very important one.

In general, both the requirements and the system model can be subdivided into a static and a dynamic component, the ontology being the static component. Other results of requirements engineering are, for example, application scenarios specifying interactions between different agents and system components. Analysis of the dynamic (behavioral) part is beyond the scope of the presented thesis.

1.1.4 When is Requirements Engineering Complete?

This thesis focuses on the extraction of an application domain ontology form requirements documents. Does ontology extraction solve all the requirements engineering problems? Definitely not. The question how to solve all the requirements engineering problems is strongly connected to the question when the requirements engineering process can be said to be finished. Intuitively, requirements engineering is complete when both the static and the dynamic parts are complete:

- The static part describes the components, actors, objects and relations between them.

- The dynamic part describes the behavior of single components and their interaction.

Zave and Jackson [ZJ97] give more formal criteria for completeness of the requirements engineering process:

1. There is a set R of requirements. Each member of R has been validated (checked informally) as acceptable to the customer, and R as a whole has been validated as expressing all the customer's desires with respect to the software development project.
2. There is a set K of statements of domain knowledge. Each member of K has been validated (checked informally) as true of the environment.
3. There is a set S of specifications. The members of S do not constrain the environment; they are not stated in terms of any unshared actions or state components; and they do not refer to the future.
4. A proof shows that
   $$S, K \vdash R.$$
   This proof ensures that an implementation of S will satisfy the requirements.
5. There is a proof that S and K are consistent. This ensures that the specification is internally consistent and consistent with the environment. Note that the two proofs together imply that S, K, and R are consistent with each other.

Surely ontology extraction and validation does not answer all the above questions. A domain ontology obviously does not say anything about the requirements to the system under construction, so it does not address the first item of the above list. The ontology contains concepts of the application domain and the prospective system, without their behavior. Thus, a domain ontology provides the static part of items 2 and 3. Furthermore, a *validated* ontology does not contain contradictions between environment concepts and system concepts and therefore satisfies the static part of item 5.

1.1.5 Requirements Engineering, Summary

Requirements engineering is undoubtedly an extremely important project phase, influencing all the later phases. One of the goals of requirements engineering is to build an explicit model of the application domain. This domain model serves as a common knowledge base for all the stakeholders. A glossary is not sufficient for this purpose, so a more comprehensive model, an ontology, is a better choice. Construction of a domain ontology on the basis of the available requirements documents is the goal of the presented work. The remaining part of this chapter shows how an ontology can be extracted and how the extraction procedure helps to improve requirements documents.

1.2 Proposed Ontology Extraction Approach

This thesis proposes a semiautomatic method of ontology acquisition as a part of the requirements analysis. The proposed method is based on analysis of requirements documents, written in a natural language. The result of this analysis is an application domain ontology. In computer science an ontology is an explicit representation of concepts specific to a particular domain and relations between these concepts. The proposed approach extracts an ontology consisting of the following elements:

- a set of terms (concepts),
- "is-a"–relation between the terms,
- a set of (more general) relations.

These three ontology constituents imply three major ontology extraction step, namely term extraction, construction of the term hierarchy and extraction of more general relations. To give the reader a flavor of the obtained results and of the approach itself, Section 1.2.1 presents an example of an extracted ontology and Section 1.2.2 gives an overview over the extraction approach.

1.2.1 Ontology Extraction Example

The idea of ontology as a domain model, consisting of concepts and relations, is illustrated in Figure 1.2. It shows a part of the ontology extracted from the steam boiler specification [ABL96b] (Figure 1.2 shows for the clarity purpose just an excerpt from the steam boiler ontology, see Chapter 4 for details). Rectangular boxes represent concepts and hexagonal boxes represent relations between concepts. The very common "is-a"–relation is shown by means of variable-width lines, the thin end pointing to the more general concept and the thick end pointing to the less general one. The ontology in Figure 1.2 shows relations "waiting state" is an "operation mode", "potentially faulty hardware" (abbreviated as "faulty") is a "physical unit", and "pump controller" is both "message source" and "faulty". Figure 1.2 shows also other kinds of relations representable in an ontology: it shows the relation between a hardware unit and its possible state ("pump_failure" → "isA_FailureOf" → "pump") and possible failure consequences ("transmission_failure" → "causes" → "emergency_stop_mode"). The

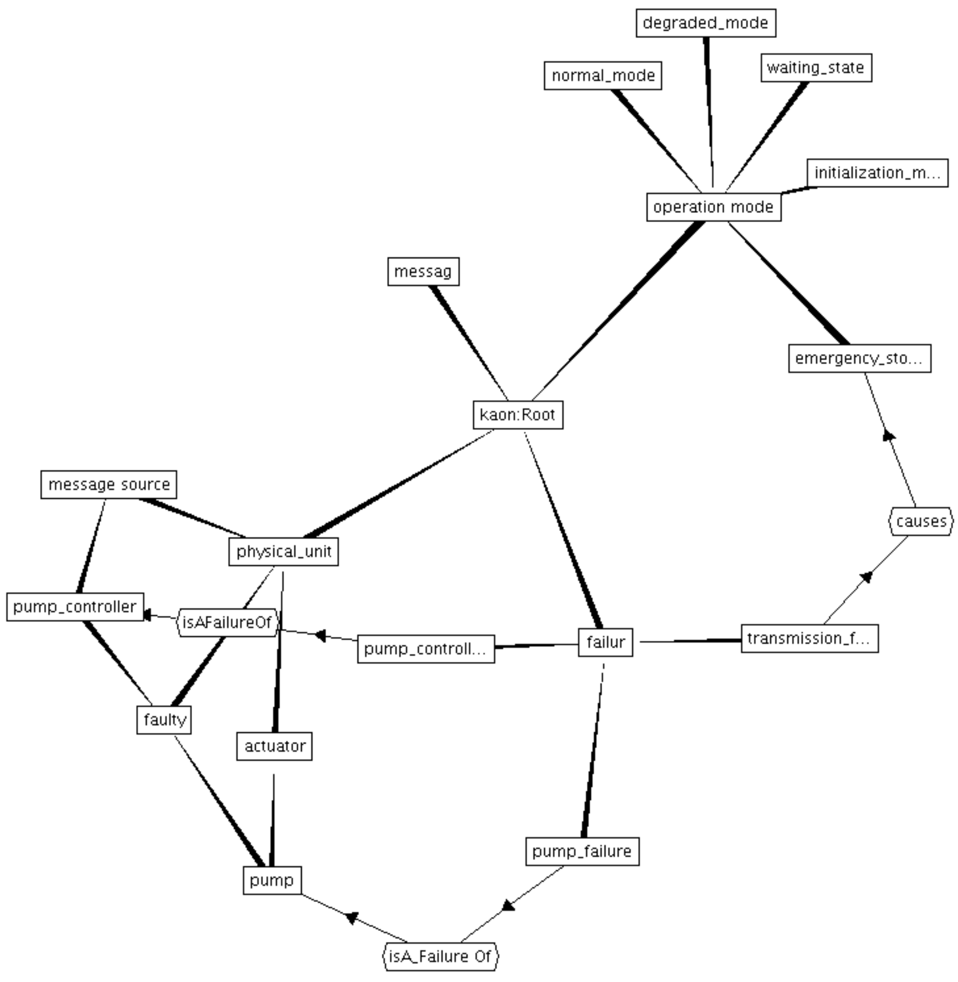


Figure 1.2: Example ontology, an excerpt

construction of such relations will be sketched in Section 1.2.2 and explained in detail in Chapters 2 and 4. All these relations would be missing in a plain glossary. This example shows that compared to a glossary an ontology gives a better (more explicit) domain model.

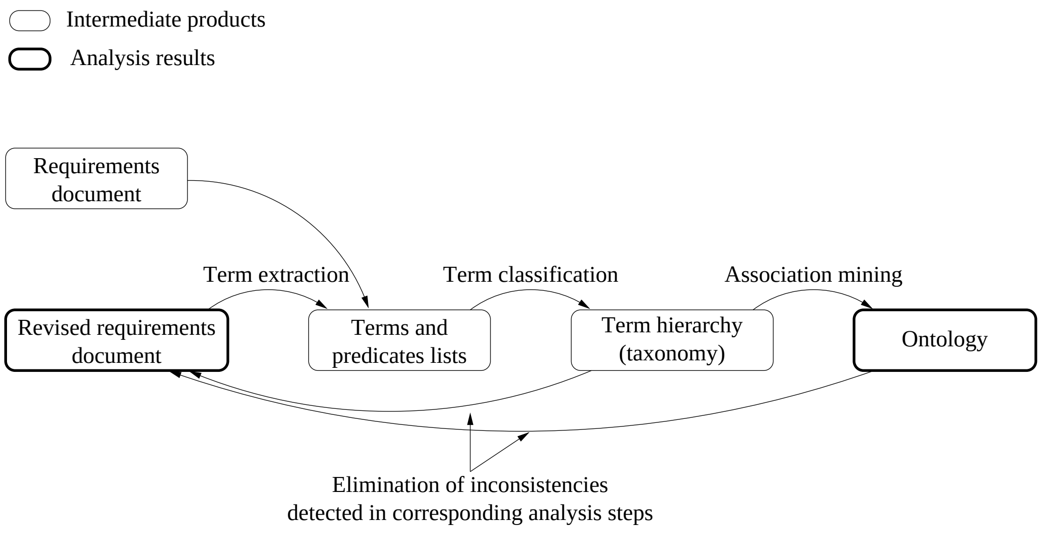
The extracted steam boiler ontology is natural, in the sense that a human requirements analyst would classify the concepts occurring in the specification document most probably in the same way. He would also identify messages, operation modes, hardware components, etc. as concept classes. For a small specification, as in the case of the steam boiler, it is easy to build the ontology manually. However, for larger specification documents, it is desirable to give the requirements analyst some automated support. Provision of such support is the goal of the presented thesis.

1.2.2 Ontology Extraction in a Nutshell

This thesis addresses ontology extraction from requirements documents as a means to establish a common language for all the project participants. As stated in the introduction to this chapter, ontology extraction consists of the following steps:

1. term extraction (glossary extraction) (completely automatic)
2. term classification, building of the term hierarchy (interactive)
3. relation extraction (interactive)

These steps of document processing are illustrated in Figure 1.3. The analysis starts with the requirements document. Then, terms are extracted and classified. When interactive term classification brings to light terminological inconsistencies, the requirements document is revised and re-analyzed. The same feedback loop is possible for inconsistencies detected during interactive relation extraction. The result of this analysis process is a domain-specific ontology *and* a revised requirements document.



Intermediate products
Analysis results
Requirements document
Term extraction
Term classification
Association mining
Revised requirements document
Terms and predicates lists
Term hierarchy (taxonomy)
Ontology
Elimination of inconsistencies detected in corresponding analysis steps

Figure 1.3: Main steps of the document analysis process

To extract the terms, each sentence of the requirements document is parsed. Good parsers, as for example the one developed by Collins [Col99], reach the coverage of about 90% on Wall Street Journal texts (i.e., about 90% of the Wall Street

Journal section of the Penn Treebank [MTM+99] are correctly parsed). Sentences in requirements documents are usually shorter and simpler, so the parser provides even better coverage on requirements texts.

After parsing each parse tree is analyzed to extract sentence subjects and objects. For example, from the sentence "Component X sends message Y", "component X" is extracted as the subject and "message Y" as the direct object. Then, the concepts are clustered according to verbs they are used with: if there are several sentences of the form "⟨Some component⟩ sends ⟨some message⟩", all the components are joined in a cluster, as well as all the messages. For example, the steam boiler specification, introduced above, contains the following sentences:

> As soon as the program recognizes a failure of the water level measuring unit it **goes into** rescue mode.
>
> As soon as the program sees that the water level measuring unit has a failure, the program **goes into** mode rescue.
>
> The program **goes into** emergency stop mode if it realizes that one of the following cases holds: the unit which measures the outcome of steam has a failure, or the units which control the pumps have a failure, or the water level risks to reach one of the two limit values.

The objects of "goes into" in these three sentences produce the concept cluster

$$\{rescue\ mode,\ mode\ rescue,\ emergency\ stop\ mode\},$$

which represents possible operation modes of the steam boiler.

By examining clusters the requirements analyst can detect terminology inconsistencies: when unrelated or just "suspiciously looking" terms are put into the same cluster, it indicates a potential terminology inconsistency. The detected inconsistencies should be corrected in the document. This is a prerequisite for a sensible continuation of text analysis. For example, the above example cluster contains both "rescue mode" and "mode rescue", which is a strong indication that at least two different names are used for the concept "rescue mode".

In the next step of ontology construction the clusters are used to build a concept hierarchy (a taxonomy). Clusters are investigated pairwise to find cluster intersections (concepts contained in both clusters). The intersections indicate that concepts contained in both clusters are somehow related. The requirements analyst may join the clusters to get a larger cluster, representing a more general concept. For example, the steam boiler specification produced the following clusters:

subjects of "detect":

$$\{program,\ water\ level\ measuring\ unit,\ steam\ level\ measuring\ unit\}$$

subjects of "work":

$$\{program,\ steam\!-\!boiler,\ water\ level\ measuring\ unit,\ pump\}$$

First of all, these clusters contain both the term "program" and hardware components, which is an indicator of terminology inconsistency. After the correction of sentences that caused inconsistent clusters, the clusters

$$\{water\ level\ measuring\ unit,\ steam\ level\ measuring\ unit\}$$

and

$$\{steam\!-\!boiler,\ water\ level\ measuring\ unit,\ pump\}$$

may be joined to a larger cluster representing hardware.

If the intersecting clusters are unrelated, it is again a sign of an inconsistency in one of the clusters (or in both of them). This can be the case when the same name is used to represent two different concepts. For example, if the name "measurement failure" is used both for a possible state of the measurement unit and for the message indicating this state, then the "states"–cluster and the "messages"–cluster would overlap. As in the case of inconsistencies detected in basic clusters, the name conflict should be eliminated before text analysis continues. After the elimination of all the naming inconsistencies a taxonomy ("is-a"–relation) is built as a cluster hierarchy. A cluster hierarchy is equivalent to a concept hierarchy because the analyst gives each cluster a collective name representing the concepts in this cluster.

The last phase of ontology extraction is the extraction of general relations. At this step the idea of data mining is applied to the text: if two terms often occur in the same sentence, it is assumed that there is a potential relation between them. For example, the terms "transmission failure" and "emergency stop mode" occur several times in the same sentence in the steam boiler specification. They co-occur in the sentence

> A transmission failure puts the program into the emergency stop mode,

which is repeated several times in the specification text. This yields the relation "transmission failure causes emergency stop mode" in Figure 1.2. The process of naming and adoptions of such relations into the ontology is interactive, because the data mining tool cannot find sensible names for potential relations, neither can it validate them. Only after the validation by the requirements analyst the relations are absorbed into the ontology.

The resulting ontology itself should be examined and validated by domain experts. The validated ontology then becomes the common language for all the project participants for all the later project steps.

To prove the feasibility of the proposed approach, three case studies were performed. The first case study analyzed the aforementioned steam boiler specification [ABL96b], the second one analyzed the much larger instrument cluster document [BHH$^+$04], the third one analyzed two industrial specifications.

The steam boiler specification introduces requirements to a control program, whose goal is to maintain the required water level in the steam boiler. Figure 1.2 shows a part of the ontology extracted from the steam boiler specification. The steam boiler ontology was produced after elimination of several inconsistencies. The most common error in the specification text was the usage of several different names for the same concept. After this name correction it was possible to extract a sensible ontology. However, such correction requires manual work, which gives rise to the question whether the approach scales.

The second case study was performed to explore the scalability of the approach. The scalability case study was important in order to see whether the amount of manual work necessary to build an ontology is acceptable for larger documents. An 80-page instrument cluster specification [BHH$^+$04] was chosen for this case study. The manual work is necessary both to detect and eliminate inconsistencies and to decide which cluster intersections and which relations are sensible. The second case study showed that the amount of the necessary manual work is reasonable even for larger documents: mere skimming the document took one working day, whereas ontology building took four additional working days for this document.

The first two case studies showed that the proposed approach is applicable to academic requirements documents. The third case study treated two real industrial documents and showed that the presented approach is applicable to industrial documents as well.

1.3 Contribution of the Thesis

The main features of the proposed ontology extraction approach, making important contributions to the research area of requirements engineering, are the following ones:

1. The proposed approach is able to analyze full-fledged English grammar and does not restrict the allowed expressions to predefined patterns.

2. It extracts an ontology, as opposed to a plain glossary. Compared to a glossary, an ontology gives more explicit term definitions.

3. It integrates inconsistency detection into the process of ontology extraction and prevents inconsistencies from being included into the ontology.

4. It does not rely on any previous knowledge of the application domain. For this reason it can be used to analyze requirements documents from any application area.

5. It shows how the traditional requirements analysis process should be modified in order to include ontology extraction and validation.

Each of the above features is important on its own. However, it is especially the combination of all the five features that makes the presented approach novel. Surely, interweaving of document analysis, inconsistency detection and ontology construction is of additional value for requirements engineering.

The above feature combination is not available in other text analysis approaches. Some researchers aim at extraction of domain-specific concepts from documents, as for example Abbott [Abb83], and Goldin and Berry [GB97]. Some others, like Chen [Che83] and Fuchs et al. [FSS99], analyze certain expression patterns, assuming that requirements documents can be restrained to a set of fixed patterns. There are also approaches aiming solely at inconsistency detection (e.g., [KBP01]). A more in-depth analysis of the related approaches is provided in Chapter 5. In particular, Section 5.5 gives a detailed comparison of the presented thesis with other existing approaches.

The techniques used to extract an ontology (parsing, clustering, etc., all but term extraction) were developed separately for purposes other than requirements documents analysis. However, chaining these separate techniques into a sequential process and enriching them with term extraction is of strong benefit for ontology extraction from requirements documents.

1.4 Outline

The thesis is organized as follows: Chapter 2 is the technical core of the thesis: it introduces the algorithms necessary to extract the ontology and the tools implementing these algorithms. It also shows how these tools can be chained to produce a sensible combination. Chapter 3 embeds the ontology extraction approach into the requirements engineering process. Chapter 4 shows feasibility of the approach on three case studies. The first case study is relatively small, because its goal was to test applicability of the method. The second case study is much larger and shows that the approach works for larger documents as well. The third case study demonstrates applicability of the proposed ontology extraction approach to real industrial requirements documents. Chapter 5 gives an overview of related approaches and shows where the presented work goes beyond the existing techniques. Chapter 6 summarizes the whole work and sketches some possibilities of further extensions.

Chapter 2

Ontology Extraction

Extraction of an application domain ontology from requirements documents is the core of the presented thesis. As there exist many definitions of an ontology, it is necessary to set up the context first. Originally, ontology is a branch of philosophy, addressing questions of being and existence, as well as fundamental categories of being. It tries to answer questions like

- What are fundamental categories of being?
- What is a physical object?
- What identifies a physical object?
- ...

Ontology as a philosophy branch deals with the most fundamental concepts like categorization, differences between the categories, and fundamental properties of objects in these categories. A short overview over ontology as a branch of philosophy can be found in [Wik05a].

In artificial intelligence an ontology is a set of concepts and relations between these concepts. Such a concept network serves as a common world model for communicating intelligent agents. Agents sharing a common world model can be sure, at least, that they talk about the same concept when they use the same term. This shared world model must be provided by a human knowledge engineer as a part of each agent's knowledge base. According to Gruber [Gru93], following aspects should be taken into account when constructing ontologies for intelligent agents:

Clarity: An ontology should effectively communicate the intended meaning of the defined terms.

Coherence: The defining axioms should be logically consistent.

Extendibility: Extension of the ontology by new terms should not require redefinition of the already defined terms.

Minimal encoding bias: Encoding bias is the dependency of the conceptualization on concrete ontology encoding or representation language. Such dependency should be avoided.

Minimal ontological commitment: The ontology should put as few constraints as possible on the modelled world, to allow the agents using this ontology for communication to define their own instantiations.

The above criteria give some guidelines on the evaluation of an already constructed ontology, but they do not provide any help in ontology construction itself.

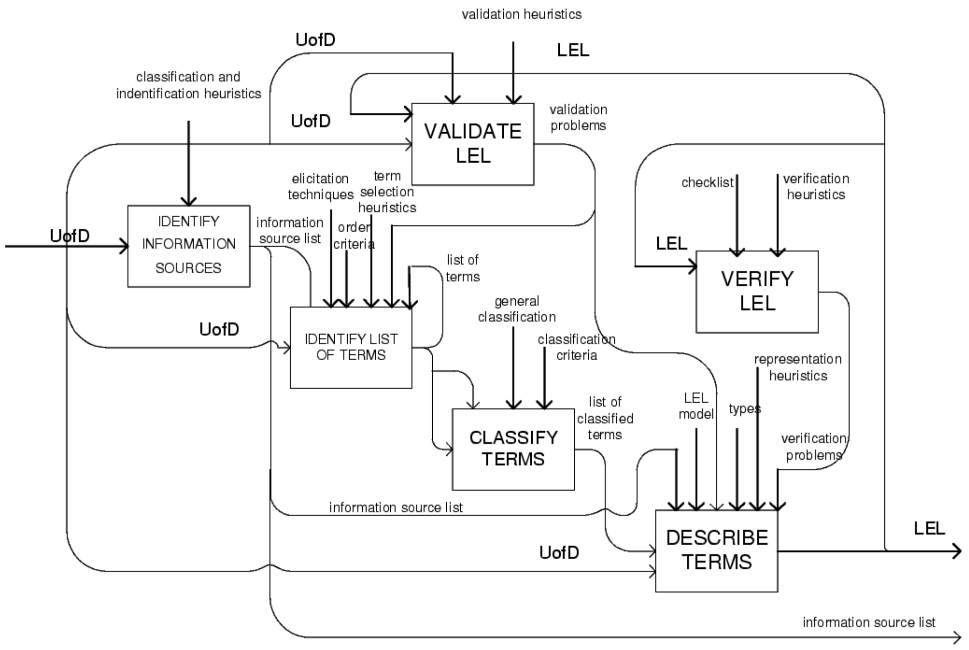


Figure 2.1: Steps of ontology construction [BS03]

Breitman and Sampaio do Prado Leite [BS03] go further and argue that an ontology can serve not only as a means of agent communication, but also as a common concept definition for different human stakeholders in a software development project. The situation in a software project is similar to communication of intelligent agents, in the sense that a common communication basis, accepted by all the stakeholders, is vital for any project. According to Breitman and Sampaio do Prado Leite, an application domain ontology should be one of the results of the requirements engineering phase. They list several ontology construction methodologies, all of which share the same basic steps as shown in Figure 2.1:

Identify information sources: In this step the information sources relevant to ontology building ("Universe of Discourse") are chosen. Additionally, a heuristic to identify and classify terms is provided for the future term extraction step.

Identify list of terms: The terms are identified using the documents and heuristics chosen above.

Classify terms: The identified terms are classified into four categories: subjects, objects, verbs, and states.

Describe terms: The identified terms are described. The descriptions are provided in natural language. When describing terms, it is necessary to use the terms from the identified term list, to achieve closeness of the description. This yields a Language Extended Lexicon (LEL).

Validate and verify LEL: Validation and verification of the produced Language Extended Lexicon are performed by the means of special reading techniques.

Besides these common steps, the various methodologies listed by Breitman and Sampaio do Prado Leite remain rather abstract in the sense that they do not specify *how* to identify information sources, *how* to classify terms, and so on.

The presented thesis makes the idea proposed by Breitman and Sampaio do Prado Leite concrete. It introduces an ontology extraction method, performing the above ontology construction steps. If ontology building is considered as a part of the requirements engineering process, the identification of the information sources is rather obvious: the primary source of information is the requirements document. As the survey by Mich et al. [MFN04] shows, the overwhelming majority of requirements are written in natural language. The concrete implementation of the remaining steps (term list identification, term classification, term description) strongly depends on the form of the ontology that should be constructed. The presented thesis adopts the following ontology view:

- An ontology consists of a taxonomy (concept hierarchy), augmented with some more general (not "is-a") relations.
- The taxonomy, in turn, consists of a term list and the "is-a"–relation (specialization, subtyping).

This implies three steps of ontology extraction:

1. term extraction,

2. term classification, building the “is-a”–relation,

3. extraction of more general relations.

The analysis steps are not atomic: term extraction needs parsed sentences, and parsing needs that each word is marked with a part-of-speech tag. The processes of tagging, parsing, and term extraction are explained in detail in Section 2.1. The second step, term classification, is not atomic either: it consists of term clustering (finding groups of related terms) and building a hierarchy of related clusters. Two clusters are seen as related if they contain a common term. Term clustering and taxonomy building is described in detail in Section 2.2. The last step, relation extraction, consists of generation of potential relations and their validation. It is introduced in Section 2.3.

It turned out in the case studies (see also Chapter 4), that the linear process, as described above, is feasible only if the analyzed document contains absolutely no terminology inconsistencies. In the case of an inconsistent document terminology inconsistencies become visible during the ontology extraction. For this reason the presented thesis proposes an iterative process. Two steps of the proposed ontology extraction process are interactive, namely taxonomy building and mining of general relations. In these interactive steps the analyst has to validate both the taxonomic and non-taxonomic relations proposed by the tool. This manual validation detects the inconsistencies inherent to the analyzed requirements document. The analyst should correct the inconsistencies before continuing the document analysis.

Figure 1.3 (page 26) clarifies this idea:

1. The analysis starts with the requirements document.

2. In the first analysis step the terms occurring in the text are extracted.

3. The next step classifies the terms and produces a term hierarchy (taxonomy). It is possible that some inconsistencies are detected during this classification. In this case it is necessary to go back to the text and to correct the inconsistencies.

4. The last step augments the taxonomy with some general relations. As in the case of the previous step inconsistencies detected as a result of the analysis should be eliminated in the document before the analysis continues.

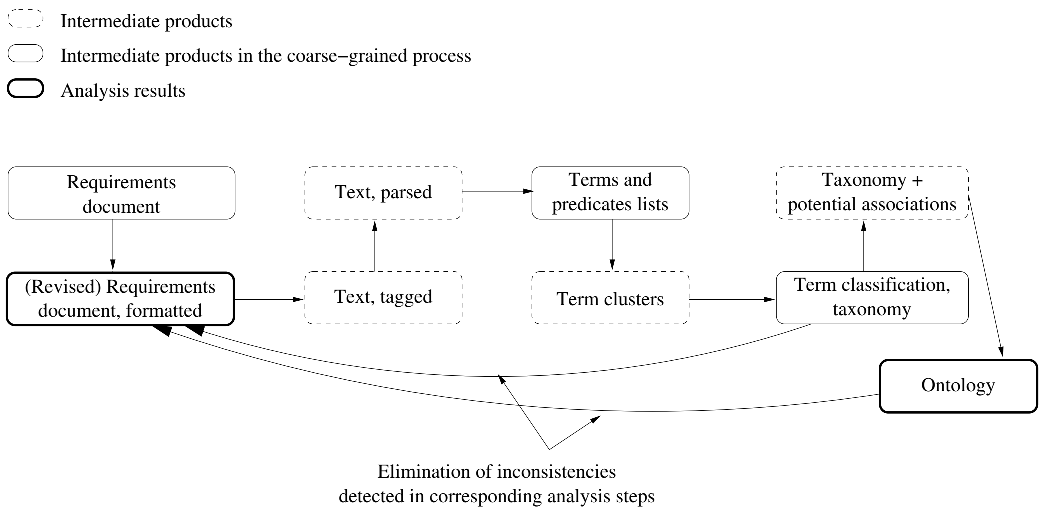

Intermediate products
Intermediate products in the coarse−grained process
Analysis results
Requirements document
(Revised) Requirements document, formatted
Text, tagged
Text, parsed
Terms and predicates lists
Term clusters
Term classification, taxonomy
Taxonomy + potential associations
Ontology
Elimination of inconsistencies detected in corresponding analysis steps

Figure 2.2: Steps of the document analysis process, complete

As stated above, term extraction, term classification and relations extraction are complex procedures, consisting, in turn, of different smaller processing steps. Figure 2.2 shows the overall document analysis process in detail:[1]

1. In the first step the requirements document has to be formatted. For later parsing it is necessary that every non-empty line corresponds to exactly one sentence.
2. In the second step each word is marked with its POS (part-of-speech) tag. This is necessary for later parsing.
3. The POS-tagged text is parsed. Parser output is a parse tree for every sentence.
4. The parse trees are analyzed in order to extract predicates and terms used with these predicates. Predicates are used in the next step to classify terms.
5. Initial clusters are built on the basis of predicates the terms are used with. Terms that are used in the same role (subject vs. direct object vs. indirect object) with the same predicate (verb) are put into the same cluster.
6. A taxonomy is built as a cluster hierarchy. Thereby intersecting clusters are joined to larger ones.
7. Potential non-taxonomic relations are extracted from text.

[1]In order not to over-complicate the matters, Figure 2.2 does not differ between "requirements document" and "revised requirements document".

8. Potential non-taxonomic relations are validated and absorbed into the ontology.

The remainder of this chapter presents each major step of the extraction procedure, as shown in Figure 1.3, in more detail. Sections 2.1, 2.2 and 2.3 introduce term extraction, taxonomy building and association mining respectively.

2.1 Term Extraction

Term extraction is the first step of the analysis process. In order to enable the later analysis steps to classify the extracted terms, it is necessary to extract not only terms, but also their grammatical roles and predicates. Predicate and grammatical role extraction is necessary to cluster terms. The clustering and classification procedure will be explained in Section 2.2.

Figure 2.3 gives an overview over the term extraction process. It shows the steps of the global process (Figure 2.2) that are devoted to term and predicate extraction. As sketched in the introduction to this chapter, terms and predicates are extracted from parse trees, provided by the parser. The parser gives much structural information for the sentence, which eases the extraction of sentence predicates with their arguments (subjects and objects). An example parse tree is shown in Figure 2.4.[2] The tree gives enough information to extract the sentence subject (left subtree, "the steam-boiler"), the predicates (leftmost branches of the right subtree, "is" and "characterized"), and the object (right subtree, "the following elements"). Parsing is introduced in more detail in Section 2.1.2. Then, Section 2.1.3 shows in detail how the predicates and terms are extracted from parse trees.

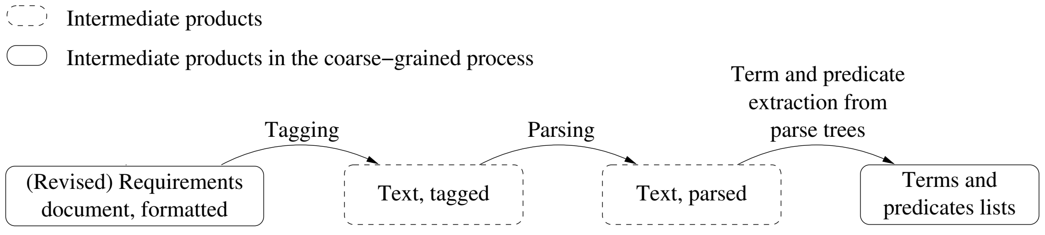

Intermediate products
Intermediate products in the coarse−grained process
Tagging
Parsing
Term and predicate extraction from parse trees
(Revised) Requirements document, formatted
Text, tagged
Text, parsed
Terms and predicates lists

Figure 2.3: Steps of the document analysis process, term and predicate extraction

The prerequisite for parsing is marking each word with its part-of-speech (POS) tag. This is necessary because in English the same word can adopt different roles in the sentence, depending on its part-of-speech. A good example is the famous sentence

[2]The meanings of the node tags will be introduced in Section 2.1.2.

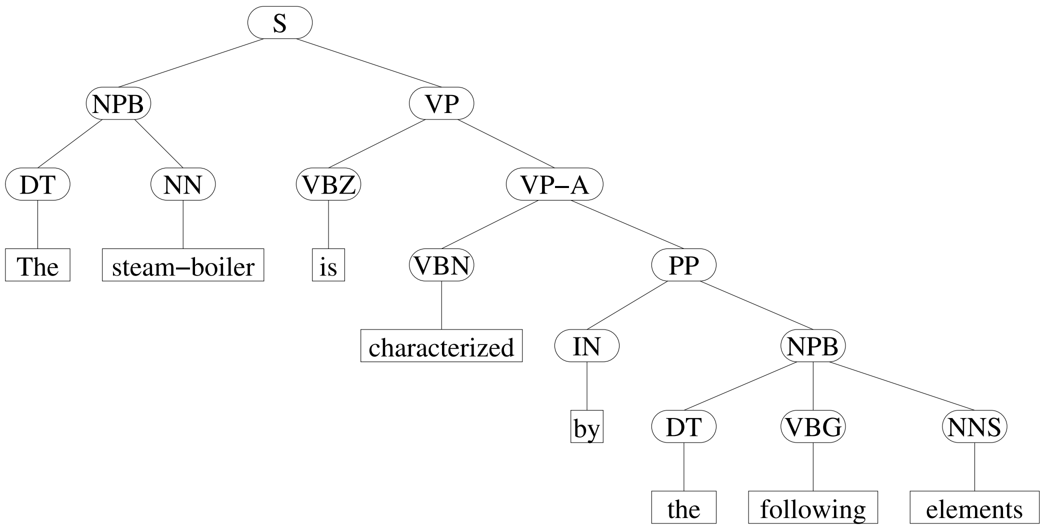


Figure 2.4: The parse tree for the sentence "The steam-boiler is characterized by the following elements"

Fruit flies like bananas,

where "flies" can be a verb (and, in this case, the predicate) or a noun (and, in this case, a part of the subject). From the pragmatic point of view POS tagging is necessary just because the parser used for this work requires such tagging. From the theoretical point of view POS tagging is necessary to improve parsing results. POS tagging is introduced in Section 2.1.1.

The term extraction technique introduced in this chapter works at the moment for English only. Language-dependent parts are the POS tagger and the parser, and English taggers and parsers provide the best available analysis quality at the moment. The presented ontology extraction technique can become applicable to other languages as soon as parsers of comparable quality become available.

2.1.1 Part-of-Speech Tagging

The goal of part-of-speech tagging is to assign a part-of-speech tag to every word. The problem is that in English the same word can be a different part-of-speech, depending on the context. For example, consider the following sentences:

1. This function is continuous.
2. The motor needs oil to function.

In the first sentence the word "function" is a substantive, whereas in the second one it is a verb.

For the presented work the POS tagger by Ratnaparkhi [Rat96] was used. This tagger calculates the most probable tag set for every sentence using the maximum likelihood method. The tagger considers each sentence as a word sequence and tries to find the best suitable tag sequence for the sentence.

The model used to calculate probabilities is based on word histories. A history of a certain word is information available when deciding about the tag of this word. Theoretically it can be the set of all the words in the sentence, as well as the tags generated for the words preceding the word under consideration. In order that the computational model remains tractable, Ratnaparkhi reduces the history to the word itself, its four adjacent words (two to the left, two to the right), and the already generated tags of the two adjacent words to the left. Let t_i be the tag of the word w_i. The history of the word w_i is defined in the following way:

$$h_i = \{w_i,\ w_{i+1},\ w_{i+2},\ w_{i-1},\ w_{i-2},\ t_{i-1},\ t_{i-2}\}. \tag{2.1}$$

To link the history of the word i, w_i with its POS tag, t_i, Ratnaparkhi introduces *features*. The feature measure is discrete, i.e., it equals 1, if certain condition holds for the pair (h_i, t_i) and 0 otherwise. For example, one of the features used by Ratnaparkhi is

$$f = \begin{cases} 1 & \text{if } \mathit{suffix}(w_i) = \text{“}\texttt{ing}\text{''} \land t_i = \text{“}\texttt{Verb Gerund}\text{''} \\ 0 & \text{otherwise} \end{cases}$$

The complete feature list can be found in the Ratnaparkhi's Ph.D. thesis [Rat98]

Word histories, together with the POS tags assigned to the respective words, are used to train the tagger. Training is necessary to estimate the parameters of the used probability model. A linguistic corpus of manually tagged texts is used for the training. The following probability model is used to train the tagger: if $f_1 \ldots f_k$ are the features taken into consideration, the probability $p(h, t)$, that a certain history h and a certain tag t co-occur, is defined as

$$p(h,t) = \pi\mu \prod_{j=1}^{k} \alpha_j^{f_j(h,t)}. \tag{2.2}$$

In Equation 2.2 π is just a normalization constant, whereas μ and $\alpha_1 \ldots \alpha_k$ are parameters to be trained. Training is based on the maximization of the likelihood L for a training corpus consisting of n (word, tag) pairs. For each word, the word history, as defined in equation 2.1, is taken into account:

$$L = \prod_{i=1}^{n} p(h_i, t_i) = \prod_{i=1}^{n} \pi\mu \prod_{j=1}^{k} \alpha_j^{f_j(h_i,t_i)}.$$

The likelihood is calculated as the product over the whole training set, i.e., for each tagged word in the training set the probability $p(h, t)$ is calculated.

After training (i.e., the calculation of α_j and μ maximizing the above likelihood) the probability model can be used for tagging. For tagging each sentence is considered separately. The probability that the word sequence $w_1 \dots w_i$ gets the tag sequence $t_1 \dots t_i$ is calculated as

$$P(t_1 \dots t_i | w_1 \dots w_i) = \prod_{k=1}^{i} p(t_k | h_k). \tag{2.3}$$

The single conditional probabilities $p(t|h)$, on their own, are computed as

$$p(t|h) = \frac{p(h,t)}{\sum_{t' \in T} p(h, t')}, \tag{2.4}$$

where T is the set of all the possible POS tags and $p(h, t)$ is defined as in Equation 2.2.

Let $\{w_1, \dots, w_n\}$ be the test sentence and let s_{ij} be the partial tag sequence, consisting of i tags, with j-highest probability among the sequences of the length i according to Equation 2.3. The final goal of the tagger is to find s_{n1} for the whole sentence. Let N be the number of most probable partial tag sequences stored for a partially tagged sentence. The tagging algorithm works on the basis of the above definitions in the following way:

1. Generate tags for w_1 and set s_{1j} for $1 \leq j \leq N$.

2.
```
for i:=2 to n do
begin
   for j:=1 to N do
   begin
      Generate tags for $w_i$ given $s_{(i-1)1} - s_{(i-1)N}$
      Append the generated possible tags $t_i$ to each sequence $s_{(i-1)j}$
         to get set of sequences $s_{ik}$
      Store all $s_{ik}$
   end
   Choose the $N$ most probable sequences of length $i$
   from the set generated above.
end
```

3. Return s_{n1} as the tagging result.

In order to train and test the Ratnaparkhi tagger the Penn Treebank Wall Street Journal corpus [BFKM95] was subdivided into two independent parts: the training and the test part. The tagger provided the tagging accuracy of 96,3% on the test set, which is one of the best results available at the moment.

2.1.2 Parsing

Parsing natural language is much more ambitious than tagging. The goal is to build a parse tree for every sentence. The commonly used basis for parsing are Chomsky-2 grammars[3]. The following simple grammar illustrates the idea.

$$\begin{array}{rcl}
S & \rightarrow & NPB \quad VP \\
NPB & \rightarrow & DT \quad NN \\
VP & \rightarrow & VBZ \quad VP\text{–}A \\
VP\text{–}A & \rightarrow & VBN \quad PP \\
PP & \rightarrow & IN \quad NPB \\
NPB & \rightarrow & DT \quad VBG \quad NNS \\
DT & \rightarrow & \text{the} \\
NN & \rightarrow & \text{steam–boiler} \\
VBZ & \rightarrow & \text{is} \\
VBN & \rightarrow & \text{characterized} \\
IN & \rightarrow & \text{by} \\
VBG & \rightarrow & \text{following} \\
NNS & \rightarrow & \text{elements}
\end{array}$$

In this grammar S denotes the whole sentence, NPB denotes a noun phrase, VP and $VP\text{–}A$ denote verbal phrases, PP is a prepositional phrase, IN is a preposition, DT is a determiner, NN is a noun and $VB\_$ (VBZ, VBN, and VBG) are different verb forms. A complete tag list can be found in [MTM+99].

Using this grammar, the sentence

The steam-boiler is characterized by the following elements

can be parsed in the following way to get the parse tree in Figure 2.4 (page 37):

$$\begin{array}{rcl}
S & \rightarrow & NPB \quad VP \\
 & \rightarrow & DT \quad NN \quad VP \\
 & \rightarrow & DT \quad NN \quad VBZ \quad VP\text{–}A \\
 & \rightarrow & DT \quad NN \quad VBZ \quad VBN \quad PP \\
 & \rightarrow & DT \quad NN \quad VBZ \quad VBN \quad IN \quad NPB \\
 & \rightarrow & DT \quad NN \quad VBZ \quad VBN \quad IN \quad DT \quad VBG \quad NNS \\
 & \rightarrow & \text{The steam–boiler is characterizd by the following elements.}
\end{array}$$

[3]See for example [Bro98a] for an introduction to the Chomsky language hierarchy

However, due to complexity of the natural language, feasible grammars are much larger. Furthermore, the grammars are ambiguous in the sense that several parse trees are possible for the same sentence. This problem is not grammar-specific but inherent to natural language.

The first attempt to overcome this difficulty was the introduction of probabilistic context-free grammars (PCFG) (see for example [Sch00]). A PCFG assigns a probability to every production rule. For the given sentence it calculates the parse tree with the highest probability:

$$P(T) = \prod_{rule\ r} P(r)^{F(r)},$$

where $F(r)$ is the number of times the rule r was used to generate the parse tree T and $P(r)$ is the probability of this rule. However, experiments showed that the performance of this type of grammars is rather poor. The reason is their insensitivity to lexical information.

Lexical information is necessary, for example, in the following situation: to cater for compound concepts, consisting of several nouns, the rule $NP \rightarrow NP\ NP$ is necessary (NP denotes a noun phrase, as in the above grammar). With this rule “postscript input file” can be parsed both as “postscript (input file)” and as “(postscript input) file”, but only the first parse is correct in the common computer science usage of the terms “postscript”, “input”, and “file”. This ambiguity cannot be resolved by assigning probabilities to grammar rules and computing the most probable parse tree: in the above example of “postscript input file” both parse trees are the results of applying the rule $NP \rightarrow NP\ NP$ twice and the rule $NP \rightarrow NN$ three times, so their probabilities would be equal in the case of probabilistic context-free grammar.

The above problem could be solved by a head-lexicalized probabilistic context free grammar [Sch00]. A lexicalized grammar introduces the additional concept of a *lexical head* and takes this concept into consideration when calculating probabilities. Intuitively, the lexical head is the most important word of the parse subtree. For tree leafs the most important word is obvious because only one word is assigned to the leaf. Each inner node of the parse tree inherits the lexical head from its most important child. Each grammar rule is extended to mark this most important child. For example, in the production rule

$$S \rightarrow NP \quad \mathbf{VP}$$

VP is the most important child of S. Thus, the most important word of S is the same as of its child VP, which is the main verb.

For the head-lexicalized grammar the probability model is more complicated than for probabilistic context-free grammar [Sch00]. Five probability distributions are important for this model:

1. $P_{start}(C)$ is the probability that C is the category of the root node of the parse tree. C can be any of the sentence constituents tags (like S, NP, VP, etc).
2. $P_{start}(h|C)$ is the probability that a root node of category C has the lexical head h.
3. $P_{rule}(r|C,h)$ is the probability that a node of category C with lexical head h is expanded with rule r. $P_{rule}(\langle term\rangle|C,h)$ is the probability that the node of category C with lexical head h is the terminal node.
4. $P_{choice}(h|C,C_p,h_p)$ is the probability that a (non-head) node of category C has the lexical head h, given that the parent node has the category C_p and the lexical head h_p.
5. $P_{lex}(w|C,h)$ equals 1 if w and h are identical and 0 otherwise.

The probability of a particular parse tree for the given sentence is calculated in the following way:

$$
\begin{aligned}
&P(T) = \\
&P_{start}(cat(root(T)))* \\
&P_{start}(head(root(T))|cat(root(T)))* \\
&\prod_{nonterminal\ n\in T} P_{rule}(r|cat(n),\ head(n))* \\
&\prod_{nonroot\ n\in T} P_{choice}(head(n)|cat(n),\ cat(parent(n)),\ head(parent(n)))* \\
&\prod_{terminal\ n\in T} P_{rule}(\langle term\rangle|cat(n),\ head(n)) * P_{lex}(word(n)|cat(n),\ head(n))
\end{aligned}
$$

To make this probability useful for parsing, it is necessary to estimate the above probability distributions first. Just as for the tagger, it is possible to train the parser on the Penn Treebank [MTM^{+}99]. The treebank provides sentences with manually constructed parse trees. To train the parser, the probability for the provided parse trees is calculated and then the above probability distributions are adjusted in such a way that the total likelihood for the training corpus is maximized.

The above parsing model seems sensible. Experiments showed, however, that more lexical information must be taken into account to achieve better results. Collins [Col99] introduced a probability model taking into consideration both grammatical and lexical information. The rationale for this probability model are not explained here because this would go far beyond the scope of the thesis. Collins uses probability distributions for the following parse tree features:

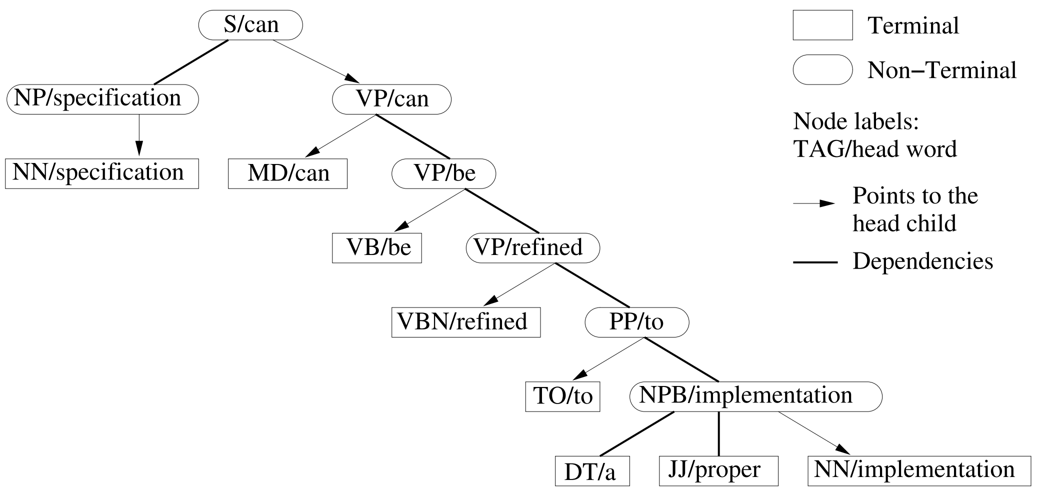


Figure 2.5: The parse tree for "Specification can be refined to a proper implementation"

Dependencies: Formally, a *dependency* is a rule of the form $w_i \to h_i$, where w_i is the i'th word in the sentence and h_i some other word in the sentence or the START symbol. Dependencies capture lexical information. A lexical dependency manifests itself in a parse tree as a link between an inner (non-leaf) tree node and its non-head child, w_i being the head word of the child and h_i being the head word of the parent. For example, Figure 2.5, showing the parse tree for the sentence "Specification can be refined to a proper implementation", contains the dependencies $be \to can$, $a \to implementation$, and $proper \to implementation$. [4] A parse tree for the sentence consisting of n words contains exactly n dependencies:

- For the head word of the root there is a dependency $w_{root} \to \mathrm{START}$.
- For the $n - 1$ non-root words there is exactly one parent in the tree where the parent head word is different from the word itself.

Directions: The direction shows for the dependency $w_i \to h_i$ whether w_i follows or precedes h_i in the sentence. For example, for the parse tree shown in Figure 2.5 $specification$ precedes can in the dependency $specification \to can$, but $implementation$ follows to in the dependency $implementation \to to$.

Relations: A relation is the grammatical representation of a dependency. For each dependency $headword(Y_i) \to headword(Y_h)$, derived from gram-

[4]The additional, not previously introduced tags in Figure 2.5 have following meanings: MD is a modal verb, JJ is an adjective, and TO is a special tag for "to".

mar rule $X \rightarrow Y_1 \dots Y_n$, the associated relation is the triple $\langle Y_i, X, Y_h \rangle$. For example, the node $NPB/implementation$ in Figure 2.5 was expanded with the grammar rule $NPB \rightarrow DT \;\; JJ \;\; NN$, which yields the relation $\langle JJ, NPB, NN \rangle$ for the dependency $proper \rightarrow implementation$.

Distances: The distance between the words of a certain dependency is defined on the basis of a *surface string*. For the dependency $headword(Y_i) \rightarrow headword(Y_h)$ derived from the grammar rule $X \rightarrow Y_1 \dots Y_n$ the surface string is defined in the following way:

- if $i < h$, the surface string is the string spanned by the non-terminals $Y_{i+1} \dots Y_{h-1}$ (the string is empty if $i + 1 = h$)
- if $i > h$, the surface string is the string spanned by the non-terminals $Y_{h+1} \dots Y_{i-1}$ (the string is empty if $h + 1 = i$)

For example, for the parse tree shown in Figure 2.5 the surface string for the dependency $a \rightarrow implementation$ is "*proper*" and for the dependency $proper \rightarrow implementation$ the surface string is empty. The distance measure consists of two bits:

Bit 1 equals 1 if the surface string is empty and 0 otherwise.

Bit 2 equals 1 if the surface string contains a verb and 0 otherwise.

Part-of-Speech: POS tags refine the dependencies and therefore provide more information for parsing. For example, the dependency $implementation \rightarrow to$ in Figure 2.5 becomes $implementation/NN \rightarrow to/TO$ when enriched with POS tags.

Subcategorization frames: A subcategorization frame is a tuple

$$\langle parent, head\text{–}child, head\text{–}word, direction, frame \rangle,$$

where $parent$ and $head\text{–}child$ are tags of the parse tree nodes, one of the nodes being the head child of the other, $head\text{–}word$ is their common head word, direction is either L or R (will be defined later) and $frame$ is a multi-set of non-terminals. The $frame$ contains only complement non-terminals, i.e. those whose tag ends with "-C" (e.g. NP-C for an NP complement).

Every grammar rule $X \rightarrow Y_1 \dots Y_n$ with the head non-terminal Y_h produces two subcategorization frames:

- $\langle X, Y_h, headword(X), L, frame_l \rangle$, where $frame_l$ is a multi-set containing all complement non-terminals in the sequence $Y_1 \dots Y_{h-1}$

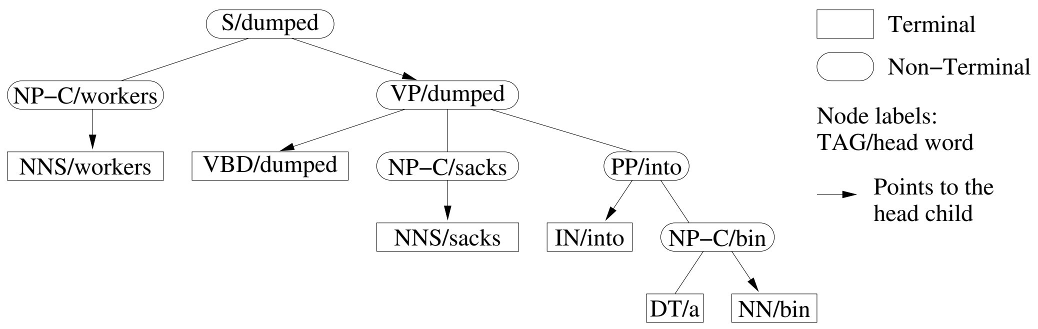


Figure 2.6: The parse tree for "Workers dumped sacks into a bin" [Col99]

- $\langle X, Y_h, \mathit{headword}(X), R, \mathit{frame}_r\rangle$, where frame_r is a multi-set containing all complement non-terminals in the sequence $Y_{h+1} \ldots Y_n$

For example, the sentence "Workers dumped sacks into a bin" has the parse tree shown in Figure 2.6. To expand the root node of this parse tree, the rule $S \rightarrow \mathit{NP}{-}C\ \ \mathit{VP}$ is used, VP being the head child of S. This rule produces for this parse tree two subcategorization frames: $\langle S,\ \mathit{VP},\ \mathit{dumped}, L, \{NP{-}C\}\rangle$ and $\langle S,\ \mathit{VP},\ \mathit{dumped}, R, \{\}\rangle$.

The rationale for the introduction of this parsing model is explained in detail in Collins' Ph.D. thesis [Col99].

The probability of a parse tree for a sentence consisting of n words is the product of $n + m$ *event probabilities*:

- n events are tuples of the form

$$\langle w_i/\mathit{tag}(w_i) \rightarrow h_i/\mathit{tag}(h_i), \mathit{direction}_i, \mathit{relation}_i, \mathit{distance}_i\rangle.$$

- m events are subcategorization frames. m depends on the grammar rules used to expand the tree nodes.

When the event probabilities are known, the parsing task is reduced to finding the most probable tree for the given sentence. The event probabilities are estimated on the basis of a training corpus, providing sentences together with their corresponding parse trees. In the case of Collins' parser [Col99] a part of the Penn Treebank [MTM$^+$99] was used as the training corpus.

To evaluate the parser quality, following quality measures are used:

Labeled Precision = $\frac{\mathit{number\ of\ correct\ constituents\ in\ the\ proposed\ parse}}{\mathit{number\ of\ constituents\ in\ the\ proposed\ parse}}$

Labeled Recall = $\frac{\mathit{number\ of\ correct\ constituents\ in\ the\ proposed\ parse}}{\mathit{number\ of\ constituents\ in\ the\ treebank\ parse}}$

Crossing Brackets = number of constituents which violate constituent boundaries with a constituent in the treebank parse.

A constituent (parse subtree) is correct if it spans the same set of words and has the same label as the corresponding constituent in the treebank.

Sentence length	**LR**	**LP**	**CBs**	**0 CBs**	≤ 2 **CBs**
$\leq$ 40 words (2245 sentences)	88.1%	88.6%	0.91	66.4%	86.9%
$\leq$ 100 words (2416 sentences)	87.5%	88.1%	1.07	63.9%	84.6%

Table 2.1: Test results for Collins' parser

The Collins' parser [Col99] was tested on a section of the Penn Treebank [MTM+99] not used for training and produced results shown in Table 2.1 (taken from [Col97]). The columns **LR** and **LP** show the labeled recall and precision respectively, the column **CBs** shows the mean number of crossing brackets per sentence, and the columns **0 CBs** and ≤ 2 **CBs** show the number of sentences with zero or ≤ 2 crossing brackets respectively. With these parsing results, Collins' parser is one of the best available at the moment.

2.1.3 Extraction of Predicates and their Arguments

To build a domain taxonomy on the basis of a requirements document, it is necessary to extract and classify the domain terms. To classify the terms, the following idea is used:

- all the subjects of the same predicate (verb) are assumed to be related,
- all the direct objects of the same predicate are assumed to be related,
- all the prepositional objects of the same predicate, used with the same preposition, are related as well.

This idea of concept clustering will be explained in more detail in Section 2.2. To make concept clustering possible, it is necessary to extract predicates and their arguments (subjects and objects with their prepositions) from the parse tree, which is the goal of the procedure described in the current section.

Predicate and term extraction consists of the following steps:

- Cutting the parse tree into smaller trees, each tree representing either the main or a subordinate sentence. For example, "This text constitutes an informal specification of a program which serves to control the water level in a steam-boiler" is cut into "This text constitutes an informal specification of a program ...” and “serves to control the water level in a steam-boiler”. The subsequent predicate and term extraction steps are performed separately for each subsentence.

- Extraction of the main verb (e.g. “can” in “Specification can be refined to a proper implementation.”)
- Extraction of the semantically relevant verb, suitable for the term classification (e.g. “refined” in “Specification can be refined to a proper implementation.”)
- Extraction of the subject, consisting of a single word (e.g. “specification” in “Specification can be refined to a proper implementation.”)
- Extraction of the object, consisting of a single word, and its preposition (when applicable) (e.g. “implementation” and “to” in “Specification can be refined to a proper implementation.”)
- Expansion of single-word subjects and objects to compound terms
- Unification of spelling: reduction of the verbs to their stems.

Splitting the Parse Tree

The first step, cutting the parse trees into smaller trees representing subordinate sentences, is necessary to extract as much information as possible from each sentence. This step transforms a parse tree into a forest, each subtree but one representing a subordinate sentence.

The tree splitting algorithm is quite simple: it starts with the parse tree produced by the parser and with an empty result set. Then, it checks the tag of the root node. If the root node carries one of the sentence tags, the tree is appended to the result set. Then the algorithm recursively descends to the children and checks if they are sentences on their own. If a subordinate sentence is found, it is pruned from its parent node. Subtrees representing subordinate sentences are appended to the result set. The algorithm descends recursively down to the tree leaves.

The following tags count as sentence tags:

$$S,\ SBAR,\ SINV,\ SBARQ,\ SQ,\ SG^{5}$$

The meaning of the tags is explained in [MTM^{+}99]. SG is not actually a sentence but an infinitive construction, like “`problem to be solved by ...`”, but

[5] There are also other sentence tags treated by the extraction algorithm. They are derived from the mentioned ones and not listed here just not to over-complicate the matters.

experiments showed that it makes sense to treat such constructions as if they were sentences.

Tree pruning is illustrated in Figures 2.7 and 2.8. The forest shown in Figure 2.8 is the result of pruning the tree from Figure 2.7. Figure 2.8 also shows why it is important to extract the infinitive construction separately: infinitive construction contains itself a predicate and its arguments. Dots in Figure 2.8 denote places where subtrees were cut.

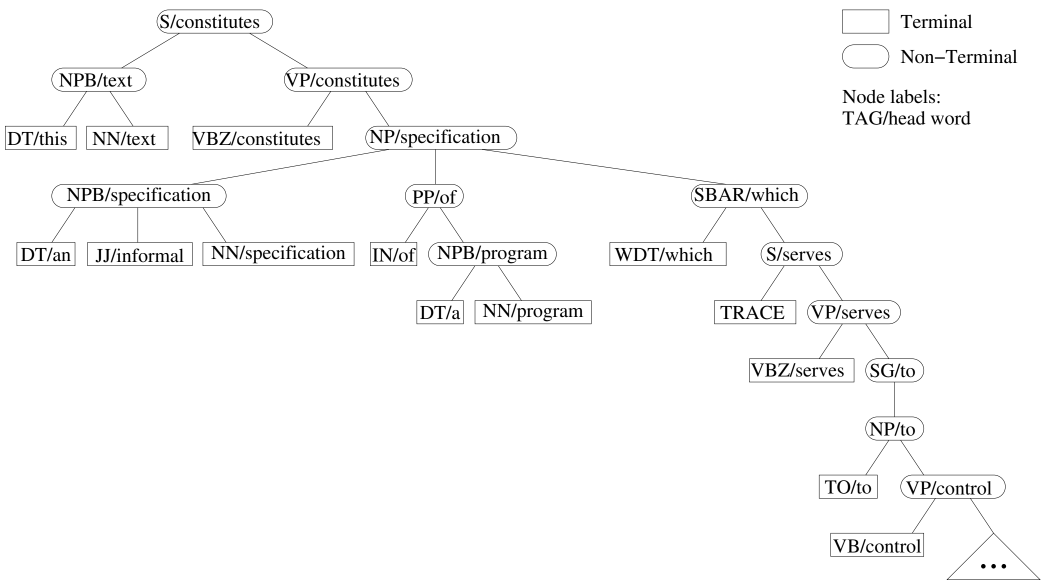


Figure 2.7: Parse tree for "This text constitutes an informal specification of a program which serves to control the water level in a steam-boiler"

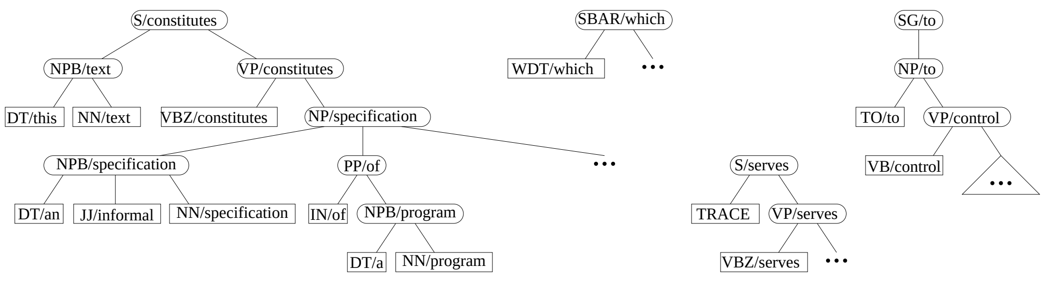


Figure 2.8: Split parse tree for "This text constitutes an informal specification of a program which serves to control the water level in a steam-boiler"

After the parse tree has been split into subtrees representing elementary sentences, the extraction heuristics is applied to each subordinate sentence separately.

Sentences with normal word order must be treated in a different way than sentences with inverse word order, such as questions and constructions like “Also present will be ...”.

The presented term extraction algorithm is currently implemented for sentences with normal word order only because the overall goal of the thesis is to analyze requirements documents. Sentences with inverse word order are extremely seldom there and therefore it was difficult just to obtain enough test data to implement the term extraction algorithm for sentences with inverse word order.

Predicate Extraction

Predicate extraction is the most important part of the whole extraction heuristics because predicates are used for term classification. Therefore, it is extremely important to extract the “right” predicate. It is not enough just to extract the sentence’s head word. For example, if the sentence contains a modal verb, this modal verb is usually the head word, like in “`Specification can be refined ...`” (see also Figure 2.5), but for further term classification it is necessary to extract “`refined`”, not “`can`”.

When extracting the predicate, it is necessary to differ three cases:

1. The sentence contains just an ordinary verb (no modal).
2. The sentence contains a modal, followed by an ordinary verb.
3. The sentence contains an infinitive construction.

It is important to consider the infinitive construction as an extra case because its head word is not a verb. On the other hand, there is no need to consider passive constructions separately: auxiliary verbs building passive can be filtered out in the same way as modal verbs.

At the beginning, the predicate extraction algorithm descends to the head terminal of the parse tree. It means, it descends to the head child of the root. If this child is a terminal, the descend process stops. Otherwise, it descends to the head child again, and so on. For example, for the parse tree shown in Figure 2.5, the head terminal is “`MD/can`”.

Then, the algorithm checks whether the found head terminal is a verb (either a modal or an ordinary one). If this is *not* the case, the parse tree under consideration is most probably an infinitive construction, like “to control the level”. As “`to`” has a special tag, it is enough to check if the head terminal carries the `TO`–tag. If this is the case, to extract the predicate, the extraction heuristics looks for the next verb or verbal phrase at the same tree level as “`to`”. For the parse tree shown in Figure 2.9, it goes from “`TO/to`” to “`VP/control`”. When the verbal phrase is

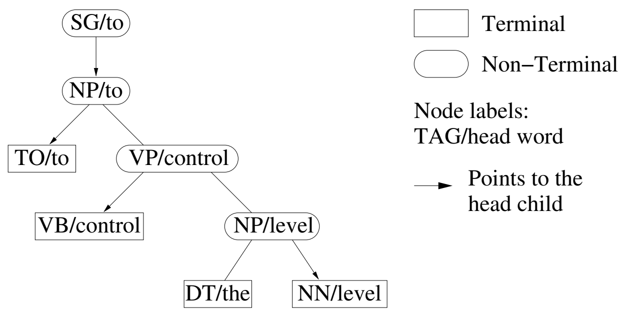


Figure 2.9: Parse tree for "`to control the level`"

found, it descends to the head terminal again. In the case of Figure 2.9 it descends from "`VP/control`" to "`VB/control`".

When the proper potential verb terminal is found, the extraction algorithm checks again whether it is really a verb. If it is not the case, the extraction just stops. In this case the analyzed sentence is most probably grammatically incorrect.

If the found terminal is a verb, the extraction algorithm continues to extract the *sensible* predicate, suitable for term classification. It starts with the verb node extracted initially, i.e. "MD/can" in the case of Figure 2.5, and looks for sibling verb or verbal phrase nodes. In the case of Figure 2.5 the algorithm finds "VP/be". It descends from "VP/be" to its head child node "VB/be" and looks for the sibling verb or verbal phrase nodes again. In such fashion it reaches "VBN/refined". This node does not have any sibling verb or verbal phrase nodes, so "VBN/refined" is the verb node that is interesting for term classification.

Extraction of the Subject and Objects

Subjects and objects of every sentence are the actual concepts used to build an ontology. Their extraction relies on the predicate extraction described above. To extract the subject, the extraction algorithm starts with the main predicate node ("VP/can" in Figure 2.5 or "VP/sends" in Figure 2.10) and traverses the parse tree to the left until it finds a noun phrase. In Figure 2.10 it finds "NPB/program". Then it descends to the head child of the noun phrase, which is "NN/program".

The direct object is extracted in a similar way: the extraction algorithm starts with the significant predicate node, i.e., "VBN/refined" in Figure 2.5 or "VBZ/sends" in Figure 2.10 and traverses the parse tree to the right, looking for the *last* noun phrase. (See also "Bracketing Guidelines for Treebank II Style Penn Treebank Project" [BFKM95], page 12.) In the case of Figure 2.5 there is no direct object. The object extraction can be better illustrated on Figure 2.10. The direct object is the last noun phrase to the right from "VBZ/sends", which is

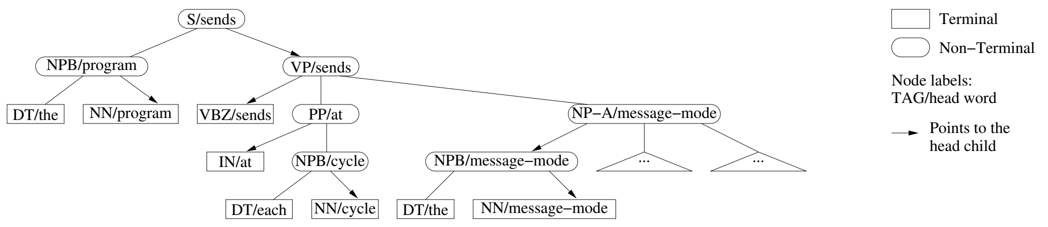


Figure 2.10: Simplified parse tree for "`The program sends at each cycle the message-mode ...`"

"NP-A/message-mode". The indirect object is the noun or prepositional phrase situated between the verb and the direct object. If the algorithm finds a direct object, it looks for such a phrase between the significant verb node "VBZ/sends" and the direct object, otherwise it looks for a prepositional phrase closest to the significant verb node. In the case of Figure 2.10 it extracts the prepositional phrase node "PP/at". The prepositional phrase is then split into the preposition "IN/at" and the noun phrase "NPB/cycle". Then the extraction algorithm descends to the head child of the noun phrase, as in the case of subject extraction.

Extraction of Compound Terms

The procedure described above extracts just concepts consisting of a single word. As many examples in case studies show, most concepts are compound ones. To extract compound concepts the extraction algorithm ascends from the leaf noun to the next noun phrase. For example, in the case of the parse tree shown in Figure 2.11 it ascends from "NN/failure" to "NPB/failure". For the example in Figure 2.11 it results in extracting "a critical failure" instead of "failure".

During the experiments with this extraction heuristics it was found out that there are many compound concepts of the form $\langle PROPERTY \rangle\ of\ \langle OBJECT \rangle$, like "failure of control unit". To extract concepts of this type, the extraction algorithm checks the node directly to the right from the noun phrase node extracted in the previous step. If this node contains "of" as its head word, the algorithm ascends one level up. For example, for the parse tree in Figure 2.11 it starts with the node "NPB/failure" and checks the node "PP/of". The head word of "PP/of" is "of", so the extraction algorithm ascends to "NP-A/failure" and extracts the whole subtree "a critical failure of the water level detection unit". At the end the words irrelevant for term classification (like determiners, conjunctions, numerals, etc.) are removed from the extracted term. Thus, "a critical failure of the water level detection unit" becomes just "critical failure of water level detection unit".

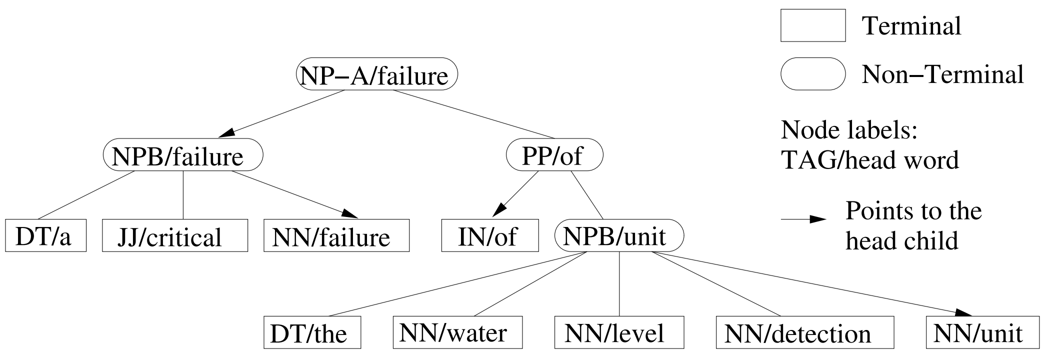


Figure 2.11: Parse subtree for "failure of ..."–construction

This reduction is necessary to identify different forms of actually the same term, like "a critical failure", "the critical failure", "second critical failure", etc.

Stemming

To properly classify the concepts, it is also necessary to make them grammatically uniform, i.e., different forms of the same word, like "use", "uses" and "using" should be identified. Porter's stemmer [Por80] was used for this purpose. Porter introduces an elaborate set of rules determining suffixes and reducing words to their stems. The stemmer is applied after the predicate extraction to active verbs only. For example, the stemmer is applied to "completes" and "complete" in the sentences "The user completes the copy operation" and "The user should complete the copy operation before ...". However, the stemmer is not applied to the passive form "completed" in the sentence "The operation must be completed", because this would put the "operation" and the "user" in the same cluster. Verbs are identified as active or passive on the basis of their position in the parse tree.

In principle it is possible to use the stemmer for parts of speech other than verbs as well. The stemmer was not applied to other parts of speech to ease the reading of the resulting terms. To unify the spelling of concepts, the ASIUM's[6] orthography correction feature was used.

2.1.4 Term Extraction, Summary

The goal of term extraction as the first step of text analysis is to identify concepts relevant to domain modelling. Term identification is built on the basis of parse tree analysis: subjects and objects of every sentence are extracted as subtrees

[6]ASIUM [FN98] is the tool used for concept classification

from the sentence parse tree. The predicates extracted from each sentence along with subjects and objects are the basis for later term classification.

The other steps of term extraction, part-of-speech tagging and parsing, are auxiliary ones and serve just to build the parse trees. For parsing and tagging standard linguistic tools are used. These tools were evaluated on broad-domain texts and were proven to provide good results on these texts.

2.2 Taxonomy Building

The main idea of taxonomy (concept hierarchy) building as the second step of ontology construction is to find related terms and to find concepts describing the sets of related concepts. Taxonomy building consists of two steps: finding related terms (building initial term clusters) and building a hierarchy by analyzing similarities in the clusters. In terms of the global process (Figure 1.3, page 26) taxonomy building starts with the extracted terms and predicates and provides a term hierarchy (a taxonomy). This process is presented in Figure 2.12: the steps relevant to taxonomy building are “term clustering” and “building cluster hierarchy”. The feedback loop, “ elimination of inconsistencies”, is not a part of taxonomy building but a potential consequence: taxonomy building includes an interactive step, where the analyst can identify terminological inconsistencies. Detected inconsistencies make the feedback loop (return to the textual specification) necessary.

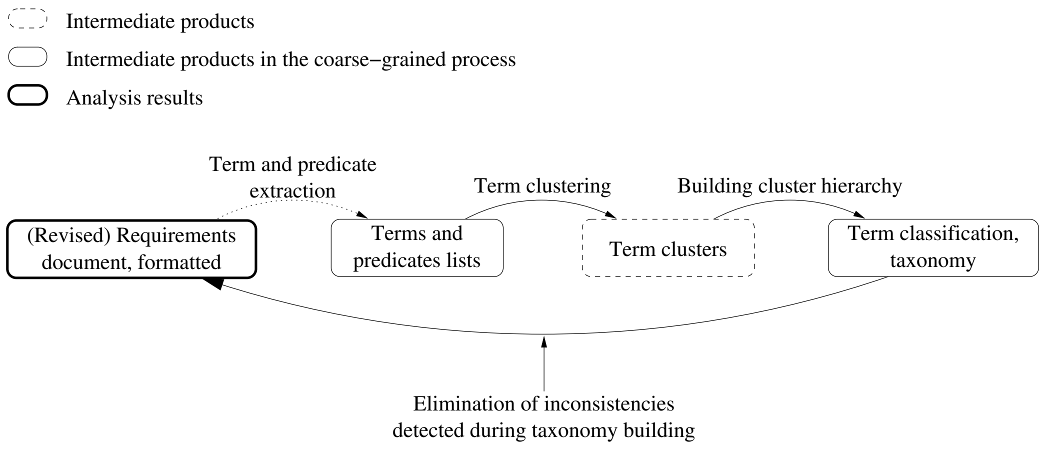

Intermediate products
Intermediate products in the coarse−grained process
Analysis results
Term and predicate extraction
Term clustering
Building cluster hierarchy
(Revised) Requirements document, formatted
Terms and predicates lists
Term clusters
Term classification, taxonomy
Elimination of inconsistencies detected during taxonomy building

Figure 2.12: Steps of the document analysis process, taxonomy building

The tool ASIUM by Faure and Nédellec [FN98] implements both taxonomy building steps (term clustering and building of a cluster hierarchy). Initial clusters based on grammatical contexts are built according to the ASIUM authors in the following way:

- All the subjects of the same verb are considered as related and are put into the same cluster.
- All the direct objects of the same verb are considered as related and are put into the same cluster.
- All the indirect objects of the same verb with the same preposition are considered as related and are put into the same cluster.

For example, if the analyzed document contains sentences like

> "Component X sends message A to controller 1"

and

> "Component Y sends message B to controller 2",

ASIUM, on the basis of the term extraction described above, builds three initial clusters:

- $\{component\ X,\ component\ Y\}$,
- $\{message\ A,\ message\ B\}$, and
- $\{controller\ 1,\ controller\ 2\}$.

This idea of term clustering explains the necessity to extract not only terms but also verbs and prepositions from the parse trees (see also Section 2.1.3).

The second step of taxonomy construction is the search for related clusters to build a cluster hierarchy. In the definition of ASIUM [FN98] two clusters are related if they contain a common concept. Related clusters may be joined to form a larger cluster. For example, in one of the case studies following clusters were extracted:

$$\{waiting\ state,\ emergency\ stop\ mode,\ normal\ mode,\ degraded\ mode\ \}$$

(direct objects of "enter")
and

$$\{initialization\ mode,\ rescue\ mode,\ emergency\ stop\ mode\}$$

(prepositional objects of "goes into"). The intersection of these two clusters is not empty, so they may be joined to a larger cluster. The user may freely choose the name for the new cluster. In the above example it would be "operation modes". This new larger cluster can be also used in further analysis of cluster intersections. Successive analysis of cluster intersections and joining of related clusters result in a concept hierarchy (domain taxonomy).

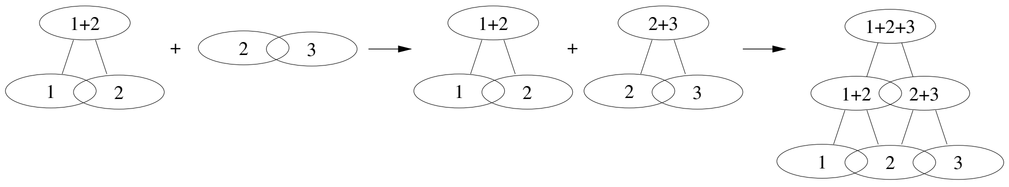


Figure 2.13: Default ASIUM tree building

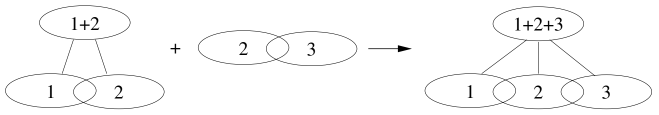


Figure 2.14: Building a less deep taxonomy tree

It is important that during the analysis each cluster and each cluster intersection, potentially leading to a larger concept cluster, is manually examined by the requirements analyst. In this way terminology inconsistencies present in the text can be detected. An inconsistency is indicated either by unrelated terms put in the same cluster or when clusters intersect that actually should be unrelated. For example, in one of the case studies (presented in Chapter 4) the following cluster was extracted:

$$\{program,\ steam\ boiler,\ water\ level\ measuring\ unit,\ pump\}.$$

This cluster contains both the term "program" and hardware components, which is a sign for either inconsistent naming or inconsistent formulation in the analyzed document.

The sentences using inconsistent terminology can be found by simple text search: for the inconsistent cluster it is known which terms, used in which context, caused the cluster inconsistency. Therefore, it is sufficient to look for the sentences containing the term in the corresponding context. The detected naming or formulation inconsistencies should be eliminated in the document before the analysis continues.

Improvement of ASIUM Tree Building Algorithm

During the case studies it turned out that an improvement to the basic tree building algorithm implemented in ASIUM was necessary. The standard ASIUM tree building algorithm produces taxonomy trees that are deeper than really needed. This is due to the fact that ASIUM can join clusters pairwise only. Pairwise joining of clusters is illustrated in Figure 2.13. Sometimes several clusters belong to the same general concept. In this case it is better to join these clusters in the way shown in Figure 2.14, without producing intermediate tree levels. For example, in the steam boiler case study (see also Section 4.2) following clusters were extracted:

1. direct objects of "receive":
$$\{message\text{—}physical\text{—}units\text{—}ready,\ message\}$$
2. subjects of "is sent":
$$\{message,\ message\text{—}valve,\ message\text{—}level,\ message\text{—}steam,\ \ldots\}$$
3. subjects of "indicate":
$$\{message\text{—}level\text{—}repaired,\ message\text{—}level,\ message\text{—}steam,\ \ldots\}$$

The second cluster intersects with the other two. The aggregate cluster representing messages should be a union of all the three clusters, intermediate taxonomy levels are neither necessary nor sensible in this case.

2.3 Association Mining

The goal of association mining as a part of the ontology extraction technique is to augment the constructed concept taxonomy with general associations. "General association" means in this context any association other than the "is-a"–relation. To find concepts that are somehow related, the same techniques can be used as for data mining. In the case of data mining the goal is to find co-occurring items in a set of database transactions. A similar idea can be applied to texts. Application of data mining to text consists of two steps, as shown in Figure 2.15: generation of potential associations and their validation. Generation of associations is an automatic process, whereas validation is interactive. This interactivity enables the analyst to detect inconsistencies. Correction of these inconsistencies makes a revision of the requirements document necessary (feedback loop in Figure 2.15).

Two questions should be answered to make the basic data mining idea applicable to ontology construction:

- How can taxonomy be taken into consideration? How can relations between more general (not leaf) terms be established?
- What is a transaction in the case of text analysis?

The subsequent sections discuss these questions: Section 2.3.1 gives a short introduction to data mining, Section 2.3.2 describes generalized data mining and inclusion of the taxonomy into data mining and Section 2.3.3 explains concrete application of the previously introduced ideas to text analysis.

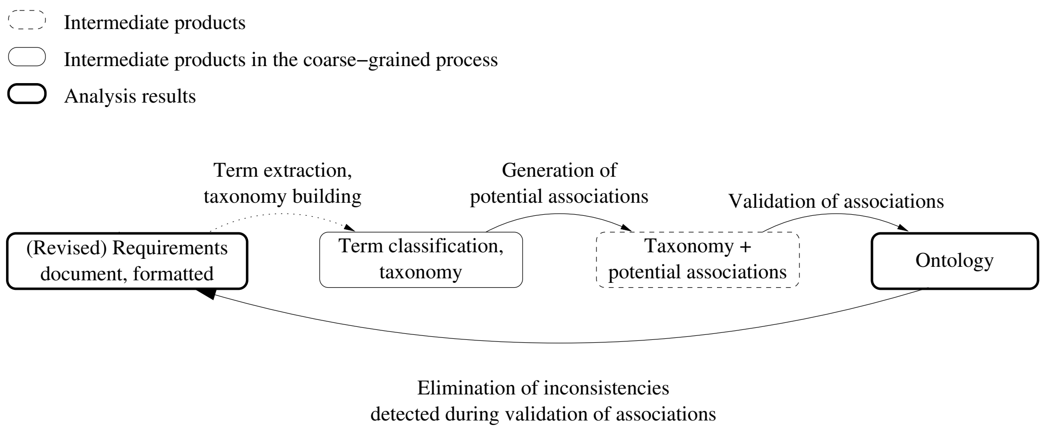

Intermediate products
Intermediate products in the coarse−grained process
Analysis results
Term extraction, taxonomy building
Generation of potential associations
Validation of associations
(Revised) Requirements document, formatted
Term classification, taxonomy
Taxonomy + potential associations
Ontology
Elimination of inconsistencies detected during validation of associations

Figure 2.15: Steps of the document analysis process, association mining

2.3.1 Very Short Introduction to Data Mining

The basic task of data mining, as applied to databases, is to find sensible correlations in data sets. For example, in the case of a supermarket sales database, it is interesting to know which items are often sold together (as for example beer and chips) in order to optimize the shelf arrangement. The basic idea for finding correlation is to analyze transaction statistic. Each transaction is considered as a set of items. For example, a supermarket transaction is a set of items bought by a single customer.

Two statistical measures are calculated to find correlating items: *confidence* and *support*. Let X be a set of items and let $trans(X)$ be the set of transactions containing X. Let A and B be sets of items whose correlation is to be determined and let N be the overall number of transactions in the database. The confidence of the association $A \Rightarrow B$ is defined as $\frac{|trans(A \cup B)|}{|trans(A)|}$. The support of the association $A \Rightarrow B$ is defined as $\frac{|trans(A \cup B)|}{N}$.

When looking for relevant associations, it is up to the data mining user to choose A, B and confidence and support thresholds above that the association $A \Rightarrow B$ is considered significant. To facilitate the choice of A and B, it is possible to generate all the thinkable item sets and then to calculate the support and confidence measures for all the pairs. Although this strategy is problematic for real databases (just due to the number of item sets to be generated), it is perfectly sensible for text analysis because the number of items (concepts) is not as high.

The idea described above finds relations between single items (e.g., $beer \Rightarrow chips$). It does not find aggregated relations, as for example $beer \Rightarrow snacks$ or $beverages \Rightarrow snacks$. The next section presents an algorithm improvement finding such generalized relations.

2.3.2 Generalized Association Mining

The goal of generalized association mining is to find associations not only between basic items, but also between item classes. In the case of the supermarket transactions it would be interesting, for example, to see whether the generalized association $beverages \Rightarrow snacks$ has higher confidence and support measures than the original association $beer \Rightarrow chips$. The prerequisite for such generalization is the existence of a taxonomy. In the particular case of document analysis the taxonomy is available as the result of the previous analysis step (see Section 2.2).

The problem of generalized association mining is that the generalized terms do not occur in the transactions, i.e., there is no supermarket transaction containing "beverages" or "snacks" as sold items. Srikant and Agrawal [SA97] present an approach solving this problem. The basic idea is to calculate confidence and support for extended associations: a potential association $A \Rightarrow B$ can be extended either by replacing some $a \in A$ or some $b \in B$ by its ancestor. This replacement can also be performed for several elements of A or B. This replacement makes slight redefinition of confidence and support necessary because, as stated above, more general terms do not directly occur in transactions.

Srikant and Agrawal introduce an additional notion of a transaction *supporting* a certain item or item set: a transaction is said to support the item x if it contains x or one of its children. A transaction supports the set X if it supports all its elements. Obviously, if some transaction supports the item set X, it also supports the generalized item set $\hat{X}$ where some items were replaced by their ancestors. Furthermore, in the case that $x \in X$ the sets $X \cup \{\hat{x}\}$ and $(X \backslash \{x\}) \cup \{\hat{x}\}$ are supported by exactly the same transactions. Thus, to generalize the association $A \Rightarrow B$, it is sufficient to replace some of the items $a \in A$ or $b \in B$ by their ancestors and it is unnecessary to keep the children in the item set.

The above definitions for association support and confidence (introduced in Section 2.3.1) can be reused for generalized association mining just by replacing the definition of $trans(X)$ as the set of transactions containing X by the set of transactions *supporting* X. Generating all the possible potential associations $A \Rightarrow B$ and generalizing them results in large number of potential associations. Srikant and Agrawal [SA97] introduce efficient algorithms coping with this problem. The efficient algorithms are not presented here because the volume of data in the case of text analysis is not as large as in data mining and the confidence and support measures can be calculated just by simple counting the supporting transactions for every potential association.

2.3.3 Application of Generalized Association Mining to Text Analysis

The idea to apply the data mining ideas introduced above to finding generalized associations between concepts is implemented by Maedche and Staab [MS00] in the tool KAON [KAO05]. The original version of the tool defines a transaction as a pair of adjacent terms. The term pairs taken into account as transactions do not overlap: as soon as a term pair is extracted from the text, the extracted terms are not considered any more as potential constituents of further transactions. For example, if the text contains the two sentences

> Every message sent by the control unit must be acknowledged in due time.
>
> If any component fails to send the acknowledgement message in due time, the control unit assumes a failure of the corresponding component.

the transactions would be:

- $\{message, control\ unit\}$
- $\{due\ time, component\}$ (last term of the first sentence and first term of the second sentence)
- $\{acknowledgement\ message, due\ time\}$
- $\{control\ unit, failure\}$

This method of mapping sentences to transactions ignores information about co-occurrence of terms in the same sentence. Thus, the original KAON definition was replaced in the presented work by the following: a transaction is any pair of terms occurring in the same sentence. With this definition the two above sentences generate following transactions:

- $\{message, control\ unit\}$
- $\{message, due\ time\}$
- $\{control\ unit, due\ time\}$
- $\{component, acknowledgement\ message\}$
- $\{component, due\ time\}$
- $\{component, control\ unit\}$

- $\{component, failure\}$
- $\{acknowledgement\ message, due\ time\}$
- $\{acknowledgement\ message, control\ unit\}$
- $\{acknowledgement\ message, failure\}$
- $\{due\ time, control\ unit\}$
- $\{due\ time, failure\}$
- $\{control\ unit, failure\}$

Potential association rules are generated as all the possible pairs of previously extracted terms. KAON [MS00] does not generate associations consisting of term sets. Generalization of these potential associations is performed on the basis of the taxonomy constructed with ASIUM (see Section 2.2). For every transaction the support and confidence values are calculated. The associations whose confidence and support values exceed previously set thresholds are presented to the user for validation. The associations that are validated by the analyst are absorbed into the ontology. Manual validation of the associations makes sure that the resulting model is sensible.

As in the case of taxonomy building, senseless extracted associations are a sign of some inconsistency in text. Just as in the case of taxonomy building, it is possible to find the corresponding text passages via simple text search because the terms that caused the association are known and it is known that they occur in the same sentence. The detected inconsistencies should be corrected before the analysis continues.

2.4 Ontology Extraction, Summary

The ontology extraction approach presented in this chapter consists of several steps. Some of these steps are automatic, whereas some of them are interactive. The following list recapitulates all the analysis steps, originally introduced in Figure 2.2, and shows which steps are performed completely automatically and which ones require human interaction.

1.	Format the text (one sentence per line)	partially automatic
2.	Tag each word (Part-of-Speech)	automatic
3.	Parse the tagged text	automatic
4.	Extract predicates and their arguments	automatic
5.	Build concept clusters	automatic
6.	Look for cluster intersections and build a taxonomy	**interactive**
7.	Look for potential associations, generalize them	automatic
8.	Decide which associations are sensible	**interactive**

These steps correspond to the principal approach, they do not show detection and correction of inconsistencies. Inconsistencies are detected in interactive steps: taxonomy building (Step 6) and decision about sensible associations (Step 8). After the correction of inconsistencies (paraphrasing) it is necessary to restart with the tagging (Step 2).

It is easy to see that one step is marked as partially automatic, while others are interactive. The difference is fundamental: the partially automatic step is not completely automatic yet because of some technical problems: there are problems with formatting incomplete or grammatically incorrect sentences that are often present as bullet points in specification texts.

For the steps that are marked as interactive, complete automation is senseless. As Goldin and Berry state [GB97], complete automation is not desirable if it could lead to information loss or wrong results. In the case of taxonomy building (Step 6) and association ascription (Step 8) inconsistencies can be found. They often manifest themselves in senseless term clusters or senseless associations. It is impossible for an automatic tool to decide which clusters/associations are sensible. Even after elimination of inconsistencies not every cluster intersection leads to a *sensible* larger cluster defining a more general concept and not every potential association is a sensible one. Thus, even for a perfectly consistent text a completely automatic tool would not be feasible. This tool interactivity achieves one of the most important goals of document analysis and validation: detection of terminology inconsistencies.

To summarize, although the approach *does require* manual intervention, this cannot be seen as its weakness: manual intervention results in better document validation, which is itself as important as ontology extraction.

Chapter 3

Ontology Extraction in the Requirements Engineering Process

Misunderstanding between project partners is a key factor potentially causing a project failure. In early project stages it is vital to be aware of the potential difference between the expressed and the real customer wishes. This potential difference can touch all the requirements aspects: use cases, scenarios, domain vocabulary, etc. To eliminate this source of project problems, it is necessary to validate the customer specification as early as possible.

Validation means in this context, that it is necessary to ensure that the requirements, as stated by the customer, really comply with the customer needs. The only way of proper validation is to present a working system to the customer and to get his feedback. Obviously, this way of validation is far too expensive. Furthermore, if the complete system fails to be validated by the customer, it requires rollback of the whole project back to the early stages, which is mostly unacceptable just due to budget and deadline constraints.

A possible solution to this problem, proposed in the presented work, is confrontation of the customer in the early project stages with preliminary system models. For the validation purpose the customer must be able to understand the models. This puts certain limitations on the type of models that can be used for this purpose. It is senseless to build a specification in some formal notation, for example VDM or Z, just because formal specification languages are neither common knowledge nor intuitively understandable.

The solution proposed in the presented thesis is to give the customer the ontology extracted from the requirements document and an initial system model, based on this ontology. In the case the customer finds errors either in the ontology or in the system model, it is most probably a sign of errors in the requirements document. In the case that the extracted ontology or the system model is found incomplete, it is a sign of omissions in the requirements document.

The remainder of this chapter presents the validation process in detail: Section 3.1 introduces the requirements engineering process, Section 3.2 shows the place of validation in this process, and Section 3.3 describes validation on the basis of ontology extraction.

3.1 Requirements Engineering Process

The goal of the requirements engineering phase of every software project is to provide a specification serving later as the basis for the whole project. The requirements engineering process, according to Robertson and Robertson [RR99], consists of the following steps (see also Figure 1.1, page 19):

Initiate Project: In this stage the project stakeholders are identified. The stakeholders then have to gather some basic objectives for the prospective system. In this phase the people responsible for later requirements writing are defined.

Trawl for Knowledge: In this phase the requirements engineers try to understand the customer's business goals, business logic and processes. This is reached for example by the means of interviews or by observing of key people at customer's site at work.

Reuse Requirements: Specifications of the products related to the product under development or previous versions of the same product are valuable sources of requirements. For this reason requirements engineers consider reuse of existing requirements. However, they should be aware that any of this requirements may be outdated or irrelevant for the new product.

Prototype the Requirements: On the basis of the observations done in the previous phase the requirements engineers formulate the use cases and usage scenarios for the prospective system. Use cases and scenarios result in "potential requirements".

Write the Specification: The experiences gained in the trawling and prototyping phases must be written down. Robertson and Robertson [RR99] introduce a special format for requirements, containing for each requirement fields like "description", "rationale", "requirement source", etc. Filling all these fields makes, according to Robertson and Robertson, "potential requirement" to "formalized potential requirement".

Assure Requirements Quality: The quality assurance step serves to test whether each requirement fulfills certain quality criteria, like relevance to the overall project goal, traceability, terminology consistence, acceptance criteria, etc. It also prevents irrelevant requirements from finding their way into specification. A “formalized potential requirement” that passes through the quality assurance becomes an “accepted requirement”.

Take Stock of the Specification: In this phase the requirements engineers check whether the specification is complete and whether the requirements are consistent with each other. The original process does not foresee any feedback, but in the case of inconsistency detection it would be sensible to go back to the phase “trawl for knowledge” or “write the specification”, depending on the type of the detected inconsistencies.

Analyze, Design, Build: Analysis and later phases are not constituents of the requirements engineering process according to Robertson and Robertson [RR99]. In these phases the requirements specification is taken “as is” and is not changed any more.

Two phases of the process introduced above are relevant from the validation point of view: “assure requirements quality” and “take stock of the specification”. Robertson and Robertson prescribe for the step “assure requirements quality” that each requirement must be checked for its relevance, completeness, traceability, etc. Furthermore, it is required that the terminology be consistent. This requirement is easy to pose but difficult to satisfy: as the example in the introduction chapter shows (see page 21), it is not sufficient just to list the domain terms used in the specification. Another drawback of the quality assurance step is that every requirements is analyzed independently, without any reference to other requirements. This is a striking contradiction to the requirement of consistent terminology: when each requirement is checked on its own, terminology consistency can be enforced only at the level of single requirements. Terminology consistency at the document level cannot be achieved by quality assurance working on single requirements. The only phase where Robertson and Robertson consider the requirements document as a whole is “take stock of the specification”. However, in this phase they check the requirement set only for the absence of conflicting and ambiguous requirements, but not for consistent terminology. Establishing consistent terminology remains out of scope of the requirements engineering process, as presented by Robertson and Robertson, but, surely, terminology consistency is as important as requirements consistency. The remainder of this chapter proposes modifications to the original process in order to include ontology extraction and validation into the process.

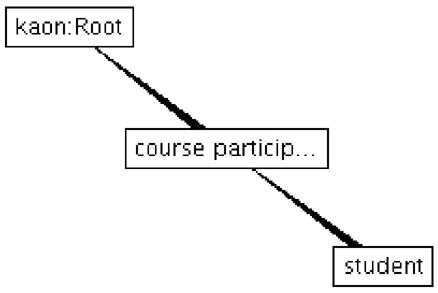


Figure 3.1: Example by Zave and Jackson: “student” is a special kind of “course participant”.

3.2 Document Analysis and Validation in the Requirements Engineering Process

The ontology extraction method proposed in this thesis supports quality assurance at the document level. There is a certain difference between quality assurance at the level of single requirements and at the level of the whole document: for quality assurance at the requirement level, it is sufficient to check whether each requirement has certain quality attributes, listed in [RR99]. For quality assurance at the document level it is necessary in addition to ensure terminology consistency over the whole document and absence of requirements conflicts.

The first of these goals, namely terminology consistency, can be achieved with the approach presented in this thesis. The proposed document analysis approach is iterative (see Figure 1.3, page 26). Two of the analysis steps (term classification and relation extraction) are interactive. In the interactive steps the analyst identifies terminology inconsistencies which should be corrected. The result of this interactive analysis is a domain ontology and a cleaned up document, free from terminological inconsistencies.

The process of cleaning up provides consistent terminology for the whole document and in this way contributes to document validation. However, if the ontology is to be used further in the development process, the ontology itself must be validated as well. For example, consider the specification of a university information system by Zave and Jackson [ZJ97], introduced in Section 1.1.3, page 21. Either by the means of text analysis or even manually, Able and Baker can build an ontology where “student” is a subordinate concept of “course participant”, as in Figure 3.1.[1] This design decision must be validated by a domain specialist by answering, at least, the following questions:

[1]See comments to Figure 1.2, page 24, for arrow meanings.

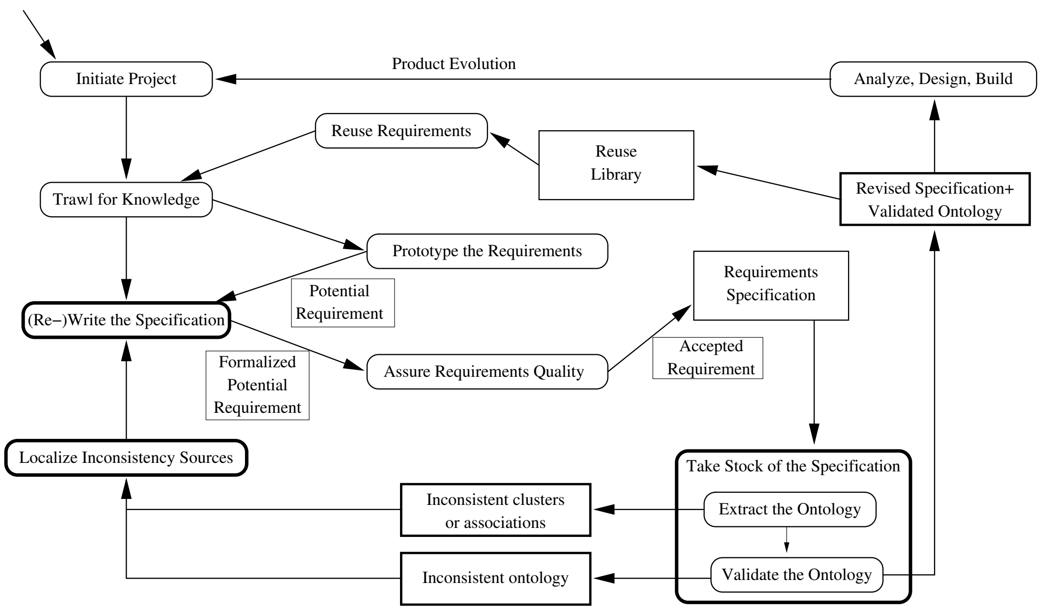

Initiate Project
Product Evolution
Analyze, Design, Build
Reuse Requirements
Reuse Library
Revised Specification+ Validated Ontology
Trawl for Knowledge
Prototype the Requirements
Requirements Specification
Potential Requirement
(Re−)Write the Specification
Formalized Potential Requirement
Assure Requirements Quality
Accepted Requirement
Localize Inconsistency Sources
Take Stock of the Specification
Inconsistent clusters or associations
Extract the Ontology
Inconsistent ontology
Validate the Ontology

Figure 3.2: Requirements engineering process, augmented with explicit specification validation step

- Is every student a course participant? If not, what is the proper superordinate concept for student? Or are there several superordinate concepts?
- Are there other course participants than students? If yes, what are the other participant types (for example, guest researchers)? If no, is there any reason to distinguish between students and course participants?
- Is course participant really a direct ancestor of student? Are there any intermediate concepts?
- ...

This sanity check is necessary for every concept and every relation in the ontology and requires profound domain knowledge. Surely such a test is time-consuming, but there is no reason to abandon it: errors in the domain ontology signalize that the requirements analysts misunderstood the domain experts. These errors, not corrected in the requirements phase, propagate to later stages and may cause high correction cost.

The ontology extraction and validation process can be integrated into the requirements engineering process without major changes to the latter. The modified process is presented in Figure 3.2. Bold boxes in Figure 3.2 denote changed steps and new analysis products. The ontology extraction steps, shown in Figure 1.3 (page 26), become a part of the step "take stock of the specification". When inconsistencies are detected, the text passages that caused the inconsistencies should be revised. Localization of these text sections is easy:

- In the case of inconsistent term clusters, it is known, which term, used with which verbs, caused the inconsistency.
- In the case of inconsistent associations, it is known, co-occurrence of which terms gave rise to the inconsistency.

In both cases a text search is sufficient to localize problematic text passages.

When, finally, the ontology itself is validated, it can happen that it still does not present well the domain-specific concepts and relations between them. In this case there is no particular recipe for error correction. The domain expert should revise the requirements document and the process of ontology extraction in order to determine where errors come from.

The result of the iterative ontology extraction and validation process is both a revised requirements document and a domain ontology, which are used in the further development process. The last change necessary to the original requirements engineering process (Figure 1.1) is the input to the reuse library: the *revised* specification and the *validated* ontology should be reused, but not the raw requirements specification, as in the original process.

3.3 Validation via Modelling

Domain ontology, extracted form the requirements document, is a communication basis for all the project stakeholders, but, obviously, there are other sensible applications of an ontology in a software project. An ontology can be used to model the domain-specific part of the application, and, furthermore, this modelling can be used for validation purposes as well: a senseless model is most probably a result of a senseless ontology, which, in turn, indicates defects of the requirements document.

Translation of the ontology into a model usable in the future project phases depends greatly on the used modelling language. For example, for object-oriented modelling, translation into UML class diagrams is straightforward:

- The "is-a"–relations in the ontology can be directly translated into class inheritance.
- General relations in the ontology are all binary. Thus, they can be directly translated into associations between classes.

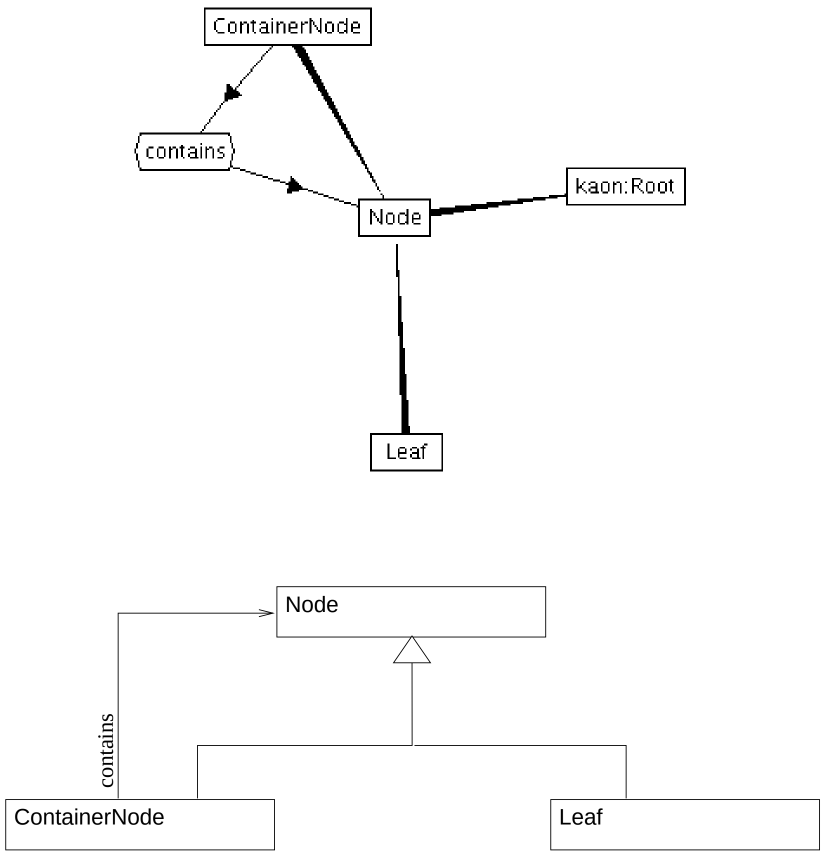


Figure 3.3: Ontology vs. UML class diagram

Figure 3.3 shows an example of such a transformation. The ontology in the upper part of Figure 3.3 represents a set of concepts for building trees of arbitrary depth and branching. There are two types of nodes (“Leaf” and “ContainerNode”), with the common superconcept “Node” and a relation indicating that “ContainerNode” contains general “Nodes”. The lower part of the figure shows the same set of concepts in the UML class diagram notation.

The above translation of ontologies into class diagrams makes sense for object-oriented modelling. In the cases where the modelling technique concepts do not fit into the ontology scheme of concepts, “is-a”-relation and general relations, the translation becomes trickier. For example, embedded systems models are mostly based on communicating components and not on classes, associations between classes and objects as runtime instances of classes. In this case a special translation heuristic becomes necessary: it is necessary to map ontology concepts and relations onto components, messages, and other native concepts of the modelling technique. Following sections introduce an example heuristic for embedded

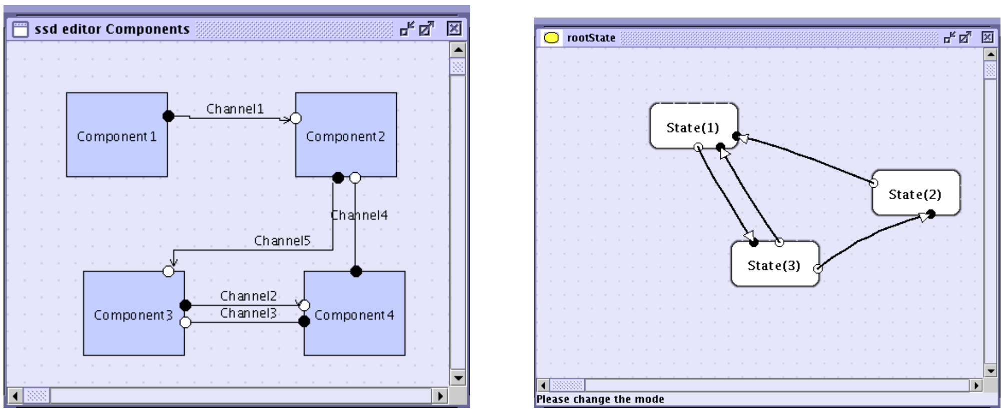


Figure 3.4: AutoFOCUS: components and automata

system models: Section 3.3.1 gives a short introduction to AutoFOCUS, a modelling tool for embedded systems, and Section 3.3.2 presents a heuristic translating an ontology into an AutoFOCUS model. As before, the resulting AutoFOCUS model should be validated and then used in the further development process.

3.3.1 Short Introduction to AutoFOCUS

AutoFOCUS [AF-04] is a CASE tool for modelling distributed systems. It models the system as a network of components, communicating via channels. Figure 3.4 (left) shows an example of such a component network. The circles on the channel ends are component ports. The component communicates with its environment by the means of these ports. Black circles are input ports, white circles are output ports. The channels between the components are typed: for each channel it is necessary to define the type of messages it can transmit.

The components can be either refined to another component network or directly implemented as a finite automaton (Figure 3.4, right). An automaton implementing a component reads the input ports of this component, performs a state transition, and writes the outputs to the output ports of the component. Each automaton transition can be provided with a precondition. The output depends both on the automaton state and on the input (Mealy-automata).

When an automaton is defined for every component, AutoFOCUS can simulate the modelled system. The execution model for the automata is synchronous: the system is simulated in rounds, every component is activated exactly once in the simulation round. If the activated component is refined to a component network, a component activation means that each component of the refining network is activated exactly once.

This sketch of main AutoFOCUS ideas is rather short, but it is sufficient for the translation of the extracted ontologies into AutoFOCUS models. Huber, Schätz and Einert give a more in-depth introduction to AutoFOCUS in [HSE97].

3.3.2 Translation of Ontology into AutoFOCUS

A possible way of ontology validation is through transforming the ontology into a domain–specific model. The concepts that can be used for this model strongly depend on the type of the system to be modelled. In the case of distributed embedded systems the typical modelling concepts are components, their states and state transitions. The components communicate via channels. The tool AutoFOCUS [AF-04] uses these modelling concepts, which makes it especially suitable for modelling of distributed systems. Thus, an ontology from the embedded domain can be validated via its translation into an AutoFOCUS model.

The goal of translation of an ontology extracted from the requirements document into an AutoFOCUS model is twofold:

- The translated model serves to validate the ontology and the requirements document: if the resulting AutoFOCUS model is incomplete or inconsistent, it is a sign of omission or inconsistency in the ontology and, most probably, in the requirements document.
- If the translated model is validated, it can be used in further development process.

The method of ontology translation into AutoFOCUS models, proposed in this thesis, works in two steps:

1. Mapping of ontology concepts onto components, states and messages (concepts understood by AutoFOCUS)
2. Mapping of relations between ontology concepts onto links between components, states and messages

The proposed translation method can be most simply explained on the small example ontology shown in Figure 3.5.[2] When mapping the ontology concepts onto AutoFOCUS, it is sufficient to instantiate the leafs of the taxonomy: more general concepts serve just to classify the really existing objects. For example, when translating the ontology in Figure 3.5 into an AutoFOCUS model, it is sufficient to instantiate "message 1–message 3" as messages, "mode 1–mode 4" as states and "actuator 1, actuator 2" and "central controller" as components.

[2]See comments to Figure 1.2, page 24, for arrow meanings.

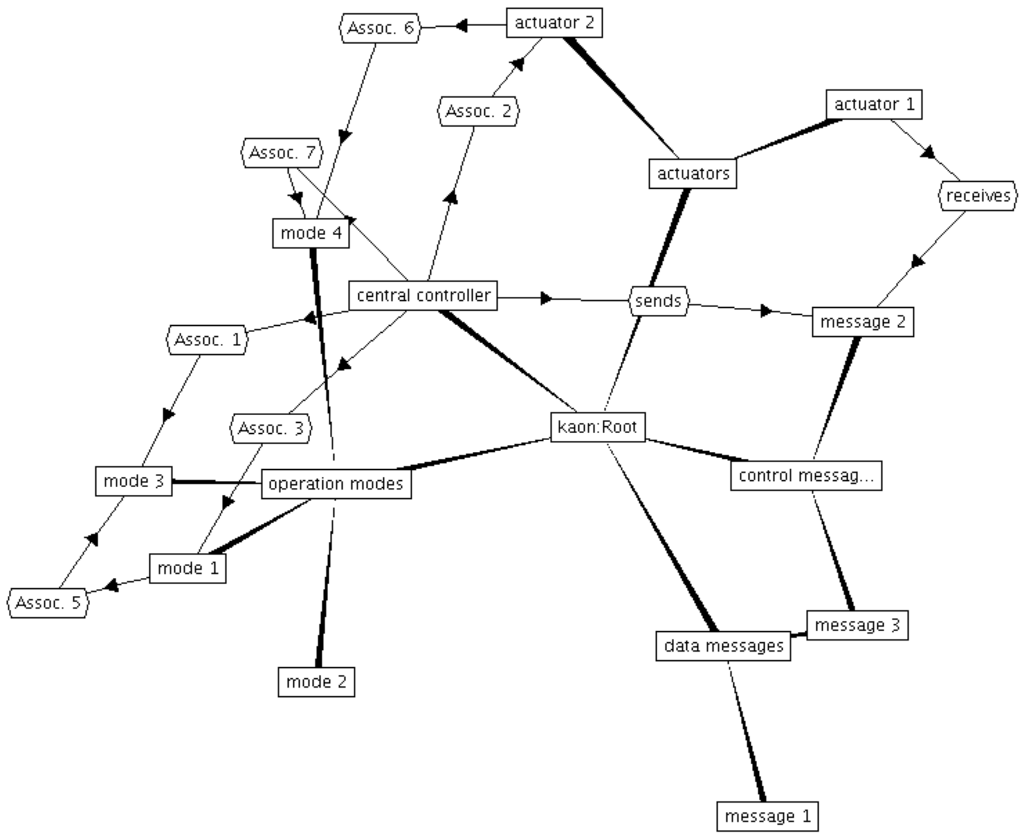


Figure 3.5: An example embedded system ontology

Although the superior concepts are not directly translated into the AutoFOCUS concepts, they are used to ease the translation. For example, for the ontology shown in Figure 3.5, the requirements analyst can map all the "actuators" and the "central controller" onto components, all the "operation modes" onto states and all the "data messages" and "control messages" onto AutoFOCUS messages. This mapping idea works fine as long as the taxonomy subtrees do not overlap. For example, in Figure 3.5 "message 2" is both a "data message" and a "control message". When the overlapping subtrees are mapped onto the same AutoFOCUS concept (in this case "message"), the overlapping is not a problem. However, if "data messages" were mapped onto AutoFOCUS components and "control messages" were still mapped to AutoFOCUS messages, this would cause a problem with the mapping of "message 2".

The solution of this problem is rather simple: such a mapping conflict indicates an inconsistency in the ontology itself. Thus, such an ontology should be revised to eliminate the inconsistency. The requirements analyst should reconsider the ontology construction. If no errors were made in the ontology construction itself, this implies an error in the analyzed requirements document. In terms of the requirements engineering process in Figure 3.2 (page 67), it means the return from the step "take stock of the specification" back to "write the specification".

When all the ontology concepts are successfully mapped onto AutoFOCUS concepts, it is necessary to connect them with each other:

- Components should be connected to each other via channels,
- states should be attributed to components,
- states should be connected to each other via state transitions.

To establish these connections, the associations present in the ontology are used.

Channels: Channels are connections between components. There are two cases when two components are considered as related and connected via channels in the generated AutoFOCUS model:

- If the ontology contains a direct association between two concepts translated into components, they are assumed to be related and a channel connecting these components is created in the resulting AutoFOCUS model. For example, the ontology in Figure 3.5 contains the association "Assoc. 2", connecting "central controller" with "actuator 2". Due to this association the components "central controller" and "actuator 2" are connected via channel in the generated AutoFOCUS model.

- Another means to generate communication channels is to look for messages that are associated with several components. The intuition behind this translation heuristics is that the message associated with several components is most probably sent and received by these components, which implies communication channels. For example, in Figure 3.5 “message 2” is sent by the “central controller” and received by “actuator 1”. This implies a channel between these two components. In Figure 3.5 the associations are explicitly named “sends” and “receives” to ease understanding, but the association names are actually not taken into account. The existence of the associations between components and messages is sufficient to create communication channels.

Assignment of states to components: A state is assigned to a component in the AutoFOCUS model if the ontology contains an association between this state and this component. The intuition is that associations reflect co-occurrence of terms in the same sentence. Associations between states and components could originate from sentences like “state X of component Y denotes . . . ” or “component Y goes into state X when . . . ”. For this reason an association between a state and a component leads to ascription of this state to the particular component in the generated AutoFOCUS model. For example, for the ontology in Figure 3.5, “mode 3” is a state of the “central controller”.

If a state is associated with several components, as for example “mode 4” in Figure 3.5, this state is created several times in AutoFOCUS model, one time for each component. Thus, “mode 4” becomes both a state of “central controller” and a state of “actuator 2”. If a state is not associated with any component, as for example “mode 2” in Figure 3.5, it is not instantiated in the AutoFOCUS model, because AutoFOCUS does not accept states not belonging to any component.

State transitions: When creating state transitions in the AutoFOCUS model, additional care is necessary: transitions between states belonging to different components make no sense. For this reason two conditions are necessary to create a state transition:

- The states belong to the same component in the AutoFOCUS model. This condition is possible to prove, as the states has already been assigned to component in the previous step of the AutoFOCUS model generation.
- There is an association between the states. The intuition behind this heuristics is the same as above: an association between two states im-

plies that they occur in the same sentence somewhere in the specification text. When two states are involved, this is probably a sentence of the form “The system goes from state X into state Y when . . . ”.

These conditions hold, for example, for “mode 1” and “mode 3” in Figure 3.5. Thus, a transition between “mode 1” and “mode 3” is created in the AutoFOCUS model. The ontology does not contain enough automatically analyzable information to make a sensible decision about transition direction, thus the transition direction in the AutoFOCUS model is chosen arbitrarily.

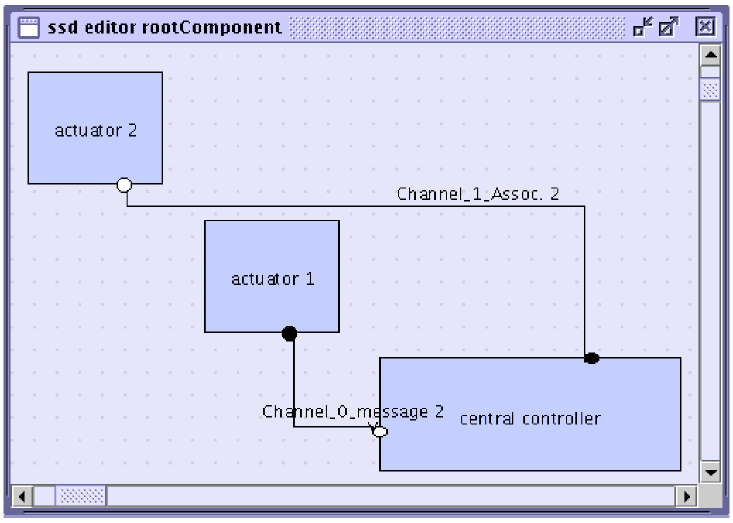


Figure 3.6: Component network, converted from the example ontology in Figure 3.5

Figures 3.6 and 3.7 show the component network and the automaton for the “central controller” produced with the above translation heuristic for the ontology in Figure 3.5. Both the AutoFOCUS model itself and a comparison of the model with the ontology give hints about quality of the ontology and of the analyzed requirements document. For example, the state “mode 2” is present in the ontology but absent in the AutoFOCUS model. This implies that the role of “mode 2” is not further specified in the ontology. If the same is true for the requirements document, i.e., the state is just mentioned but not further described, it is an obvious omission in the requirements, that should be corrected. Another omission is visible in the automaton (Figure 3.7): “mode 4” is not involved in any state transition. The requirements engineer should analyze where this problem comes from: as above, it can be either an ontology construction problem or an omission in the requirements document.

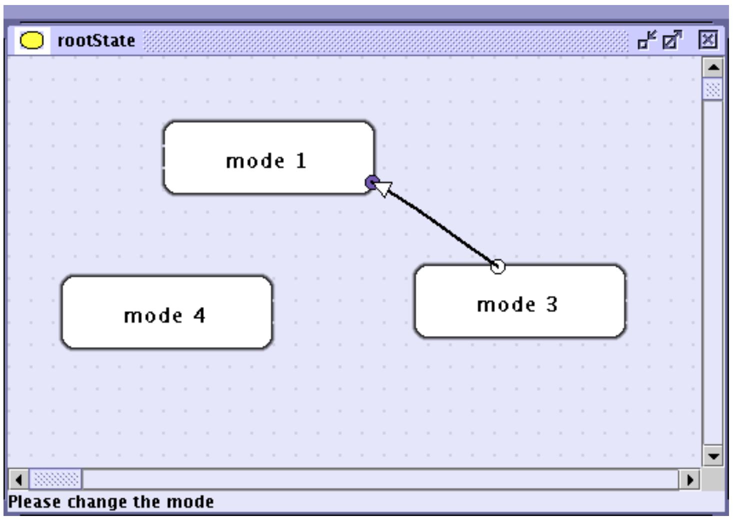


Figure 3.7: The automaton for "central controller", converted from the example ontology in Figure 3.5

The above translation ideas were implemented by Klitni [Kli04]. In [Kli04] he gives also more details on the translation algorithm and on the features of the translation tool.

3.4 Summary: Validation and Ontology Extraction in Requirements Engineering

A very common problem in requirements engineering are misunderstandings between the customer and the software engineers. These misunderstandings can come both from software engineers' lack of domain knowledge and from requirements documents that do not really correspond to the customer's wishes. Lack of domain knowledge is surely a difficult problem at the project beginning. However, potential discrepancy between the customer's wishes and the requirements document is even a more dangerous one: Unless preventively taken into account, this potential discrepancy is not perceived up to a certain point and it jeopardizes the whole project.

The problem of lack of domain knowledge is solved by ontology extraction. Surely the domain ontology does not represent the whole domain knowledge, but it provides a good basis. Furthermore, terminology inconsistencies are detected in the process of ontology extraction, which contributes to quality assurance of the requirements document.

Potential discrepancy between the customer wishes and the requirements document cannot be seen solely on the basis of the requirements document analysis. Even the mere discrepancy of the terminology really used by the customer and the terminology used in the requirements document cannot be seen on the basis of the document alone. For this reason it is necessary that the results of the document analysis are validated by a domain specialist. The validation process proposed in this thesis consists of two steps:

- Validation of the ontology itself, i.e. examining whether the produced concept classification and the relations are sensible.
- Translation of the ontology into a domain-specific model and validation of the resulting model. Problems in the translation process also uncover ontology errors.

If both the ontology and a domain-specific model are validated, they can be used in further project phases. Otherwise, it is necessary to go back to the requirements elicitation and document writing steps. This feedback loop is not a drawback but a strength of the proposed approach: it is better to detect and correct requirements errors in the early phases than in the later ones.

Chapter 4

Case Studies

Case studies are indispensable to demonstrate the feasibility and to explore the limitations of any method. To evaluate the results of a case study, it is possible either to compare the extracted ontology to a reference ontology or, if a reference ontology is not available, to check whether all the concepts contained in the requirements document were extracted. These evaluation criteria are introduced in Section 4.1.

Three case studies were performed to evaluate the approach presented in this thesis. For the first case study the steam boiler specification [ABL96b] was chosen. This specification was written for a formal method contest and is very precise from the point of view of the human reader. The goal of the first case study was to see the applicability of the method. The first case study showed that the ontology extraction works and that even a seemingly precise document can contain a lot of terminology inconsistencies. A by-product of this case study was a set of writing rules. A document following these rules does not contain terminology inconsistencies and is easier to analyze. The first case study is presented in Section 4.2. The writing rules are presented in Section 4.5, as a part of the lessons learned in all case studies.

The first case study gave rise to the question whether the ontology extraction approach is applicable to larger documents, due to manual effort necessary to eliminate inconsistencies. To answer this question, a second case study, on a much larger document [BHH^{+}04], was performed. This document specifies a part of a car dashboard, providing information about the current speed, motor temperature, outside temperature, etc. to the driver. As expected, the effort spent on ontology extraction was larger than in the first case study, but it was still affordable. The details of this case study can be found in Section 4.3.

Although the first two case studies showed actual applicability of the ontology extraction method, their explanatory power was restricted, due to the fact that the analyzed documents originated from academic projects. Thus, the third case study was performed to evaluate the applicability of the text analysis approach to industrial documents. In the industrial case study two documents specifying a business information system for a car repair shop were analyzed. This last case study is presented in Section 4.4. Finally, Section 4.5 summarizes lessons learned in the case studies.

4.1 Evaluation Criteria for Case Studies

To evaluate the ontologies extracted in the case studies, several methods are possible:

- when a reference ontology is available, it is possible to compare the extracted and the reference ontologies,
- when a reference glossary (term list) is available, it is possible to compare the extracted terms to the glossary terms,
- when neither a reference ontology nor a reference glossary are available, it is solely the manual examination by a domain expert that can evaluate the extracted ontology.

Due to insufficient knowledge of the applications domains, only the first two evaluation methods, namely ontology comparison and glossary comparison, were used in the presented work.

To measure ontology similarity, Maedche and Staab [MS02] introduced several similarity measures. The following list is a simplified presentation of similarity measures introduced in [MS02].

Lexical comparison: Lexical comparison is based on the edit distance [Lev66], measuring the minimal number of insertions, deletions and substitutions necessary to convert one string into another. For example, the edit distance between “toy example” and “toy-examples” is 2. On the basis of the edit distance Maedche and Staab define string similarity of the two strings L_i and L_j. Let $|L|$ denote the length of the string L. The lexical similarity is defined as follows:

$$StrSim = \max\left(0, \frac{\min(|L_i|, |L_j|) - EditDistance(L_i, L_j)}{\min(|L_i|, |L_j|)}\right)$$

The string similarity measure $StrSim$ returns values between 0 and 1; 0 meaning completely different, 1 for match. Lexical ontology similarity of two ontologies built on lexicons $\mathcal{L}_1$ and $\mathcal{L}_2$ is defined as

$$LexSim(\mathcal{L}_1, \mathcal{L}_2) = \frac{1}{|\mathcal{L}_1|} \sum_{L_i \in \mathcal{L}_1} \max_{L_j \in \mathcal{L}_2} (StrSim(L_i, L_j))$$

The lexical similarity measure LexSim is asymmetric. The tool TextTo-Onto [Tex05], implementing the similarity measures, differs between lexical recall and lexical precision. Lexical precision is the LexSim measure defined above. Lexical recall is symmetric to LexSim:

$$\mathit{LexRecall}(\mathcal{L}_1, \mathcal{L}_2) = \frac{1}{|\mathcal{L}_2|} \sum_{L_i \in \mathcal{L}_2} \max_{L_j \in \mathcal{L}_1} (\mathit{StrSim}(L_i, L_j))$$

Taxonomy comparison: Taxonomy similarity measures whether the sub- and superconcepts of a certain concept coincide in two ontologies. Let $\mathcal{H}$ denote the taxonomic (hierarchical) part of the ontology and let $\mathit{SupSub}(L, \mathcal{H})$ be the set of sub- and superconcepts of L in the hierarchy $\mathcal{H}$. Taxonomic overlap of two hierarchies with respect to the term L is defined as follows:

$$TO(L, \mathcal{H}_1, \mathcal{H}_2) = \frac{|\mathit{SupSub}(L, \mathcal{H}_1) \cap \mathit{SupSub}(L, \mathcal{H}_2)|}{|\mathit{SupSub}(L, \mathcal{H}_1) \cup \mathit{SupSub}(L, \mathcal{H}_2)|}$$

The average value measures the extent to that the hierarchies agree.

$$\overline{TO}(\mathcal{H}_1, \mathcal{H}_2) = \frac{1}{|\mathcal{L}_1|} \sum_{L \in \mathcal{L}_1} TO(L, \mathcal{H}_1, \mathcal{H}_2)$$

This metric is asymmetric, just like the lexical similarity metric.

Relation comparison: Relational overlap measures the number of relations common to two ontologies. It is measured on the basis of common domain and range concepts for the relations. For the graphical ontology representation used in this thesis (see for example Figure 1.2), relations domains are the concepts that associations arrows stem from and relations ranges are the concepts that associations arrows point to. To measure the similarity of relations domains and ranges, Maedche and Staab introduce the notion of concept match. Let $\mathit{Sup}(L, \mathcal{H})$ be the set of superconcepts of L in the hierarchy $\mathcal{H}$. Then, concept match CM for concepts $\mathcal{C}_1$ and $\mathcal{C}_2$, from hierarchies $\mathcal{H}_1$ and $\mathcal{H}_2$ respectively, is defined as follows:

$$CM(\mathcal{C}_1, \mathcal{H}_1, \mathcal{C}_2, \mathcal{H}_2) = \frac{|\mathit{Sup}(\mathcal{C}_1, \mathcal{H}_1) \cap \mathit{Sup}(\mathcal{C}_2, \mathcal{H}_2)|}{|\mathit{Sup}(\mathcal{C}_1, \mathcal{H}_1) \cup \mathit{Sup}(\mathcal{C}_2, \mathcal{H}_2)|}$$

For two given relations, characterized by their domain and range, it is possible to measure their similarity using the above definition of concept match. Let $d(R)$ and $r(R)$ denote the domain and range of the relation respectively.

Furthermore, let $\mathcal{O}$ denote an ontology and $\mathcal{H}$ its hierarchical part (taxonomy). Then, similarity of two relations is defined as follows:

$$RO'(R_1, \mathcal{O}_1, R_2, \mathcal{O}_2) \\ = \sqrt{CM(d(R_1), \mathcal{H}_1, d(R_2), \mathcal{H}_2) \times CM(r(R_1), \mathcal{H}_1, r(R_2), \mathcal{H}_2)}$$

To measure the relational overlap of two ontologies, it is necessary to introduce a function mapping relations names to relations themselves. Let $\mathcal{L}^r$ denote the set of relations names in the ontology $\mathcal{O}$ and let $\mathcal{G}$ be the function mapping relation names to relations themselves. Then, for $L \in \mathcal{L}_1^r$, $L \in \mathcal{L}_2^r$, the relation similarity with respect to the name L is defined as follows:

$$RO''(L, \mathcal{O}_1, \mathcal{O}_2) = \frac{1}{|\mathcal{G}_1(L)|} \sum_{R_1 \in \mathcal{G}_1(L)} \max_{R_2 \in \mathcal{G}_2(L)} (RO'(R_1, \mathcal{O}_1, R_2, \mathcal{O}_2))$$

The mean relational overlap is the mean value over all relation names:

$$\overline{RO}(\mathcal{O}_1, \mathcal{O}_2) = \frac{1}{|\mathcal{L}_1^r|} \sum_{L \in \mathcal{L}_1^r} RO''(L, \mathcal{O}_1, \mathcal{O}_2)$$

The similarity measures introduced above are implemented in the tool Text-ToOnto [Tex05]. In the absence of a reference ontology these measures are not applicable to evaluate the ontology extracted from text. In this case there are four thinkable criteria to evaluate the quality of the extracted ontology. An ontology consist of concepts and associations. To evaluate the extraction quality, it makes sense to check:

- whether the extracted concepts/associations are correct, i.e. contained in the text,
- whether all the relevant concepts/associations were extracted.

As the objective criterion for the evaluation it is sensible to take *completeness for concepts* ("Were all the concepts extracted?"). It does not make sense to consider completeness for associations because associations are not explicitly defined in text. It does not make sense to consider correctness either (in the sense "are all the extracted concepts/associations relevant?"), neither as applied to concepts nor as applied to associations. Correctness evaluation makes no sense for concepts, as long as the analyst does not invent terms, but only extracts concepts from text. It makes no sense to evaluate correctness of the extracted associations because every single proposed association is checked manually before it is included into the ontology. Thus, the associations that are present in the final model are per definition correct from the human analyst's point of view. Both evaluation criteria (similarity measures and completeness of concepts extraction) were used in the case studies presented below.

4.2 Steam Boiler Case Study

The steam boiler specification [ABL96b] describes a control application whose aim is to support required water level in a steam boiler. The steam boiler system consists of the following units (see also Figure 4.1, taken from [ABL96b]):

- the steam-boiler
- a device to measure the quantity of water in the steam-boiler ("water level display" in Figure 4.1)
- four pumps to provide the steam-boiler with water
- four devices to supervise the pumps (one controller for each pump) ("ctrl" in Figure 4.1)
- a device to measure the quantity of steam which comes out of the steam-boiler ("steam measurement" in Figure 4.1)
- an operator desk (missing[1] in Figure 4.1.)
- a message transmission system (missing in Figure 4.1)

The system should provide the required water level even despite failures of some components. Depending on the functioning components the system works in different operation modes.

Figure 4.2 shows, for later comparison, a manually constructed ontology for the steam boiler system.[2] It shows the concepts introduced in the specification and a classification of these concepts. The classification is explicitly introduced in the requirements document as well. The concept classes are:

- sent messages (messages sent by the control program)
- received messages (messages received by the control program)
- failures
- operation modes
- physical units
- physical parameters

[1]Missing components will be addressed later, see also Section 4.5

[2]See comments to Figure 1.2, page 24, for arrow meanings.

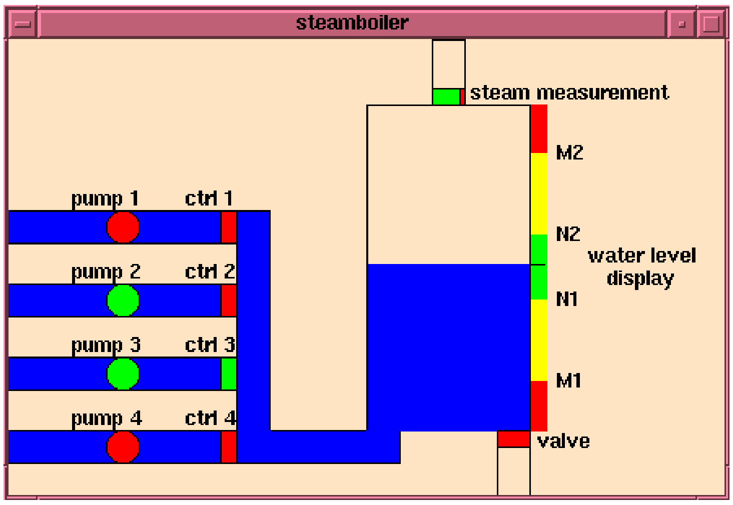


Figure 4.1: The steam boiler system [ABL96a]

Only few associations are explicitly stated in the requirements document. Figure 4.2 shows three classes of them:

"signalizes" is an association between a hardware failure and a message signalizing this failure.

"causes" is an association between a hardware failure and the operation mode caused by this failure.

"opens/closes" are associations between messages controlling the pumps and the pumps themselves.

This manually constructed ontology will be used later to evaluate the results of automated ontology extraction.

4.2.1 Overview of the Case Study

In the first run of the case study the text was analyzed as it was, without eliminating inconsistencies. The results of the first analysis run did not allow to build a *sensible* domain model. Unrelated concepts were put into the same cluster during the taxonomy building. For example, one of the clusters contained both the term "program" and hardware components:

$$\{program,\ steam\ boiler,\ water\ level\ measuring\ unit,\ pump\},$$

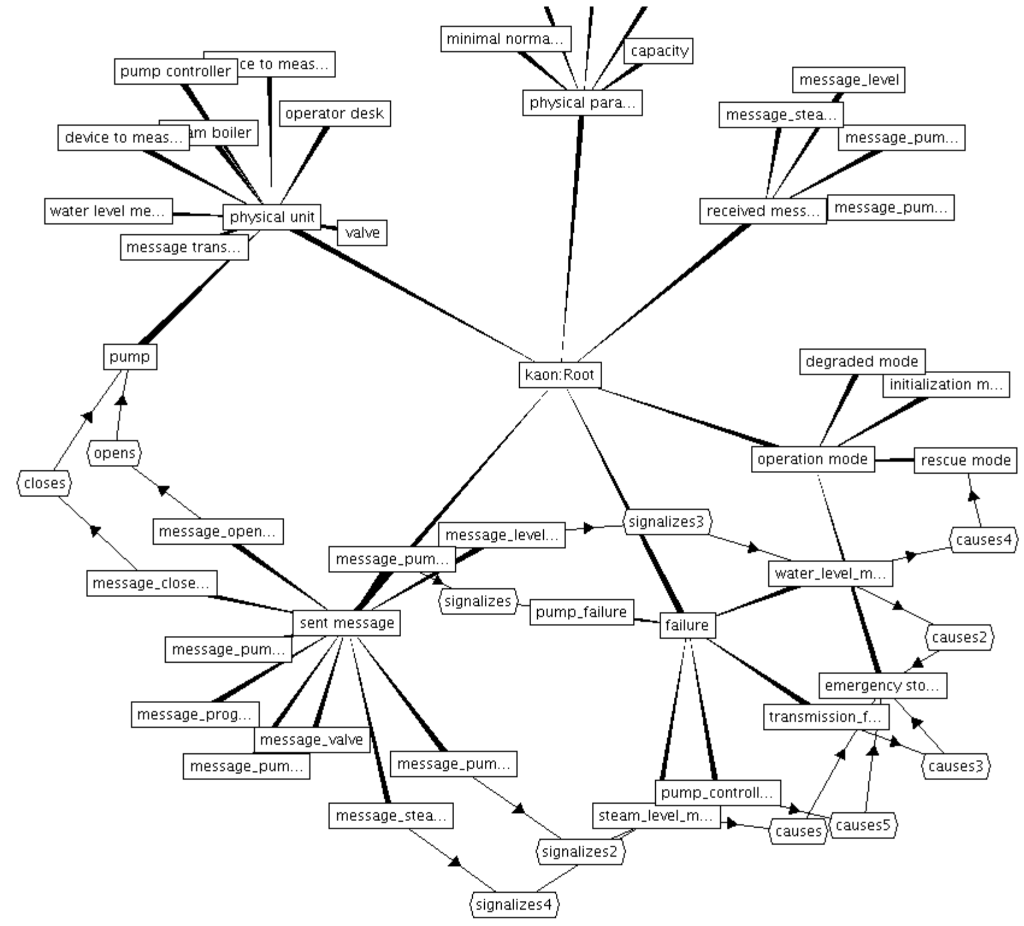


Figure 4.2: Steam boiler ontology, manually constructed

(subjects of “work”)
and another contained completely unrelated terms:

$$\{level,\ mode,\ program\}$$

(subjects of “reach”).

Association mining was not performed with this original text, as it was necessary to build a sensible taxonomy first. Manual analysis of the input text, driven by the clustering results, showed that it contained many inconsistencies preventing from direct building of domain ontology. Recognition of terminology inconsistencies made document revision necessary. The case study followed the idea of iterative process of ontology extraction and document validation, introduced above (see also Figure 1.3, page 26). In the end, two ontology building iterations were necessary. The results of these two iterations were both a domain ontology and a revised document, free from terminology inconsistencies. The remainder of this section presents each ontology building iteration in detail.

4.2.2 First Case Study Iteration: Detection and Elimination of Inconsistencies

The first manual analysis of the extracted predicates and terms showed that some parts of the specification text are unsuitable for sentence-based analysis, i.e., they lose their meaning if considered outside of their original context. For example, the text part describing possible failures looks like this:

> Detection of equipment failures
>
> TRANSMISSION: (1) The program receives a message whose presence is aberrant. (2) The program does not receive a message whose presence is indispensable.

The first sentence of this example causes a wrong parse, as the parser considers “TRANSMISSION” to be a part of the sentence. From the second sentence “program” is extracted as the subject, “receive” as the predicate and “message” as the object. Although this extraction is absolutely correct, it is not what is really necessary: it is necessary to relate failure detection and message reception or non-reception.

Wendt developed in his diploma thesis [Wen04] an approach to extraction and clustering of terms occurring in such constructions, on the basis of the term extraction technique presented in Section 2.1. See Section 5.4.2 for the details of the clustering algorithm. This approach can potentially be integrated with the clustering method used in this thesis. The integration was not performed within the scope of the thesis due to inaccessibility of the source code of the tool ASIUM, used for taxonomy building. See Section 6.3.2 for a sketch of the possible integration.

For the case study, the consequence of the need for grammatically correct sentences was to replace the above constructions by full-fledged sentences like

> The program detects transmission failure if it receives a message whose presence is aberrant. The program detects transmission failure if it does not receive a message whose presence is indispensable.

This first correction step replaced all the enumeration-like constructions by full-fledged sentences. This was mostly necessary in constructions like

> Message X: This message is sent ...
>
> Failure Y: This failure is detected when ...

The text without enumeration-like constructions was suitable for analysis with ASIUM. Clustering of the extracted concepts using ASIUM discovered further problems: for example, one of the clusters (direct objects of "enter") consisted of

$$\{state, emergency\ stop\ mode,\ mode,\ mode\ emergency\ stop\}$$

First of all, this cluster showed that at least one operation mode had several names. Replacing the different names by "emergency stop mode" was easy. The second problem was more interesting: neither "state" nor "mode" is suitable for classification without further specification, which state or mode is meant. Text search showed that "state" comes from the sentence

> The program enters a state in which it waits for the message STEAM-BOILER-WAITING to come from the physical units.

To make the specification more precise, "a state" was replaced by "the waiting state".

The origin of the orphan (not further defined) "mode" in the above term cluster was even more interesting: it arose from the sentence

> As soon as this signal has been received, the program enters either the mode normal if all the physical units operate correctly or the mode degraded if any physical unit is defective.

The two mode names mentioned in this sentence are grammatically incorrect. The parser cannot recognize that "normal" and "degraded" are mode names and parses them as ordinary adjectives. This puts the words "mode" and "normal" into disjoint subtrees and makes the extraction of compound concepts "normal mode" and "degraded mode" impossible. An additional difficulty arises from the "either

... or"–construction in this sentence. This difficulty is a deficiency of the current heuristics for predicate and term extraction, but not an inherent problem. The current extraction heuristics just ignores the conjunctions like "and", "or", "either ... or" altogether. Although the conjunctions are vital for semantics capturing, they are not that important for term classification. To overcome all these difficulties, the original sentence was replaced by

> As soon as this signal has been received, the program enters either the normal mode or the degraded mode. If all the physical units operate correctly it enters the normal mode. If any physical unit is defective it enters the degraded mode.

A similar problem was detected in other clusters: there were orphan "unit" and "device" terms. Text search discovered "unit which measures the quantity of steam", "unit which measures the outcome of steam", "physical unit which measures the outcome of steam", "device to measure the quantity of steam" and "steam measurement device". All these constructions were replaced by "steam level measurement unit". The same name unification was necessary for "pump controller" and "water level measuring unit".

The last curiosity discovered with ASIUM in the first iteration was the cluster

$$\{program,\ physical\ unit\}$$

containing prepositional objects of "emitted by" and "received by". This is an example of metonymy, whereby one object is used to stand for another. The program itself does not send or receive messages, whereas the control unit running the program does. Every human reader understands this substitution, but it provokes senseless clusters. For this reason "program" was replaced by "central control unit" everywhere in the sending or receiving context.

The purified text allowed for building of this simple taxonomy:

- Message sources (prepositional objects of "comes from"):

$$\{water\ level\ measuring\ unit,\ steam\ measurement\ unit,\\ pump\ controller\}$$

- Potentially failing hardware (subjects of "is repaired" (passive form), subjects of "working")

$$\{water\ level\ measuring\ unit,\ steam\ measurement\ unit,\ pump,\\ physical\ control\ unit,\ pump\ controller\}$$

- Operation modes (direct objects of "enter", prepositional objects of "goes into")

$$\{\textit{waiting state},\ \textit{emergency stop mode},\ \textit{normal mode},\ \textit{degraded mode},\ \textit{initialization mode},\ \textit{rescue mode}\}$$

- Messages (direct objects of "receive", subjects of "indicate", subjects of "is received" (passive form), subjects of "is sent" (passive form))[3]

$$\{\textit{message},\ \textit{message—pump—state},\ \textit{message—steam—boiler—waiting},\ \textit{message—stop},\ \textit{message—valve},\ \textit{message—open—pump},\ \ldots\}$$

- Actuators (direct objects of "activate")

$$\{\textit{valve},\ \textit{pump}\}$$

- Failures (subjects of "is detected" (passive form), direct objects of "detect", subjects of "put" (In the context "... puts the program into mode XY"))[4]

$$\{\textit{failure},\ \textit{pump failure},\ \textit{transmission failure},\ \textit{pump controller failure},\ \textit{water—level—measuring—unit failure},\ \textit{steam—level—measuring—unit failure}\}$$

This taxonomy was transferred to an association mining tool KAON [KAO05]. When extracting the terms with the KAON concept extraction facility, it was discovered that the concept extraction by ASIUM was incomplete. For example, "message level", "message mode" and many more other messages were not discovered by ASIUM. Recognition of this problem made a second iteration necessary.

4.2.3 Inconsistency Elimination and Ontology Building: Second Iteration

First of all, all the new messages discovered in KAON were marked in the second iteration as compound concepts. It is sufficient to write "message-level" instead of

[3]For this cluster it was necessary to use the improved version of the clustering algorithm, as described in Section 2.2

[4]For this cluster it was necessary to use the improved version of the clustering algorithm as well.

"message level" for the parser to consider it as a single compound concept. During this marking the expression "start or stop message" was discovered , which was replaced by "message-start or message-stop".

In a similar way "acknowledgement message" and "detection message" were discovered. As neither "acknowledgement message" nor "detection message" is a real message used for communication, the corresponding sentences were rephrased more precisely, in order that they specify *which* acknowledgement or detection message is used in every particular case.

Analysis of term clusters extracted from the corrected text showed the following problems:

- there were large clusters produced by the verbs "be" and "have", containing different unrelated concepts
- there were orphan "mode" and "failure" concepts.

The first problem was solved by rephrasing all the sentences containing "be" or "have". For example, "has a failure" and "is defective" were replaced by "fails"; "is really zero" was replaced by "really equals zero", etc.

The orphan "mode" arose from "this mode", where the actual mode was specified in the previous sentence. The current text analysis approach cannot establish relations between sentences, so the only solution was to replace "this mode" by the actual mode name that is meant. After the purification of the text it was possible to build a *sensible* ontology using the text analysis techniques.

The steam boiler case study showed that the amount of manual work necessary to process the document is not negligible. However, this manual work is not in vain: detection and correction of inconsistencies is a part of both document validation and quality assurance. This part of validation is eased by the tool that fails to extract a consistent ontology from an inconsistent document. The amount of manual work was not measured during the first case study because the goal of this case study was to evaluate the feasibility of the approach itself. Applicability to larger documents was addressed in the second case study, presented in Section 4.3.

4.2.4 Results of the Steam Boiler Case Study

The Domain Ontology

Figure 4.3 shows a part of the produced ontology. The diagram shows the ontology root (`kaon:Root`), four top-level concepts (`operation mode`, `failure`, `physical unit`, and `message`) with some of their subordinate concepts and relations between them.[5] For example, there are associations "Transmission_failure causes emergency_stop_mode" and "Rescue_mode is_caused_by water_level_measuring_unit_failure".

When compared to the manually constructed ontology in Figure 4.2 (page 85), the extracted ontology contains all the concept classes but "physical parameters". The names of physical parameters were not extracted as they occur solely in incomplete phrases (enumerations). Extraction of concepts from incomplete phrases is not possible yet. For the same reason two of the physical units were not extracted: "operator desk" and "message transmission system". These concepts are mentioned only once in the document and their role is not further specified. A human reader would extract these two concepts, but would have to guess how they interact with other components. This point can be seen both as a weakness of the extraction technique and as an omission in the document: these two components are also missing in the steam boiler simulator (Figure 4.1, page 84), programmed for the participants of the formal methods contest, whose goal was to provide a formal steam boiler specification. As for other concept classes (messages, operation modes and failures), the approach succeeded in extracting all the concepts belonging to these classes.

Additionally to the concepts present in the original requirements document, operation mode "waiting state" was extracted from the revised document version. This concept was added during document revision, as the original document contained some abstract "state", which was treated exactly in the same way as operation modes. The extracted ontology also differs from the manually constructed one in the classification of "physical units" (see also Figure 4.4). This classification contains additional information about message senders and receivers, about potentially faulty hardware (abbreviated as "faulty"), etc. This more detailed classification is made possible due to additional information, available in the specification text. The manually constructed ontology was based on the explicit list of hardware elements contained in the document and did not take additional information into account. The extracted ontology contains also additional relations, like "message_program_ready isSentIn initialization_mode" and "message_steam_boiler_waiting triggersProgramStart_in initialization_mode". These relations are sensible, but missing in the manually constructed ontology.

When compared to the manually constructed ontology using ontology similarity measures introduced in Section 4.1, the extracted ontology still shows high degree of lexical similarity. Table 4.1 shows similarity measures between the extracted ontology (Figure 4.3) and the manually constructed ontology (Figure 4.2). The ontologies similarity measures are asymmetric. For this reason Table 4.1 shows two lines of comparison values for every ontology pair. In order to give

[5]See comments to Figure 1.2, page 24, for arrow meanings.

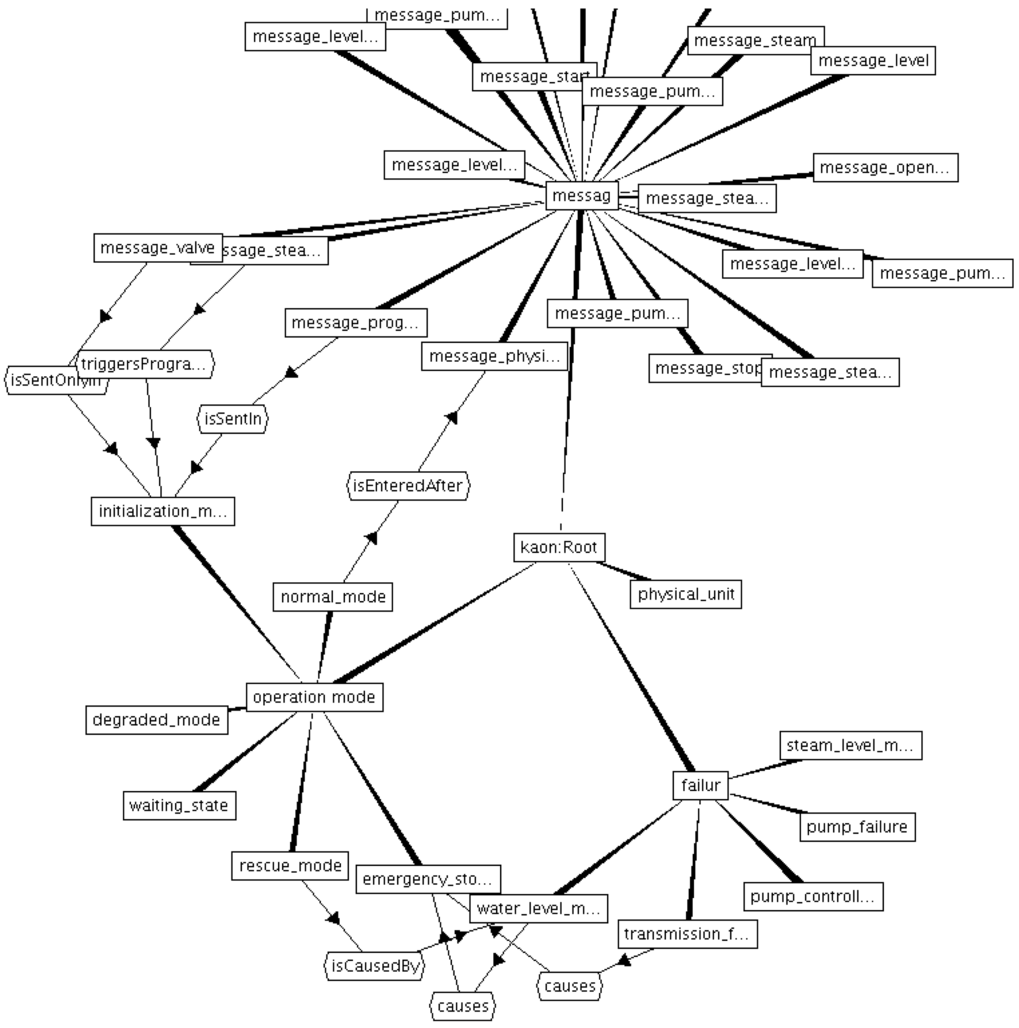


Figure 4.3: Steam Boiler: part of the produced ontology

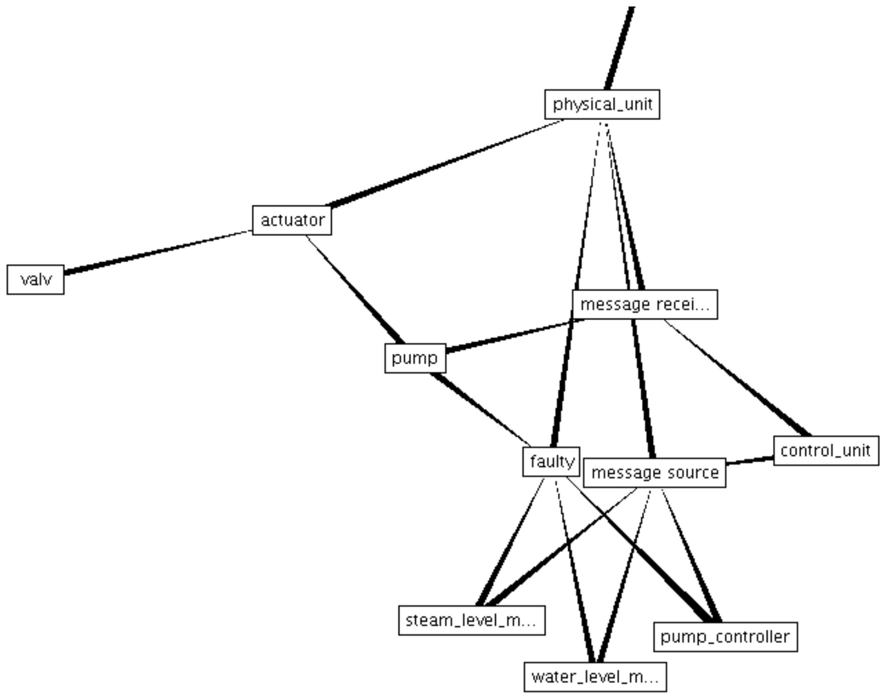


Figure 4.4: Steam Boiler: part of the produced ontology, subtree "physical units"

compared ontologies	lexical recall	lexical precision	taxonomic overlap	relational overlap
manually constructed vs. extracted via text analysis (final)	0.6041	0.4393	0.1896	0.2549
extracted via text analysis (final) vs. manually constructed	0.4393	0.6041	0.2731	0.0622
extracted ontologies: 1st iteration vs. 2nd iteration	0.5833	0.8235	0.6565	0.4945
extracted ontologies: 2nd iteration vs. 1st iteration	0.8235	0.5833	0.5547	0.2927

Table 4.1: Ontologies similarities for the steam boiler case study

some reference to compare the similarities values, Table 4.1 shows also the results of the comparison of two extracted ontologies: the result of the first extraction iteration and the final result of inconsistency elimination and subsequent ontology extraction. It is easy to see that the similarity values for the two extracted ontologies are higher, but the similarity of the manually constructed and the extracted ontologies is still of the same order of magnitude.

The AutoFOCUS model

In order that the extracted ontology can be validated, it was translated into an AutoFOCUS model, using the algorithm described in Section 3.3.2. Figures 4.5 and 4.6 show the generated component network and the state transition diagram for "control_unit" respectively. Both diagrams are correct in the sense that the generated components, communication channels, states and state transitions really exist in the specification. But, obviously, at least the state transition diagram is incomplete. As the translation algorithm produces state transitions from associations between states, missing state transitions imply missing associations. An association between two terms is extracted when these two terms occur in the same sentences. Thus, missing state transitions mean that the corresponding states never occur in the same sentence. In this way the model gives information about specification incompleteness.

4.3 Instrument Cluster Case Study

The goal of the first case study was to test the actual applicability of the approach and to experiment with available tools. This case study showed that the approach works, but a certain amount of manual work is necessary. Although this manual

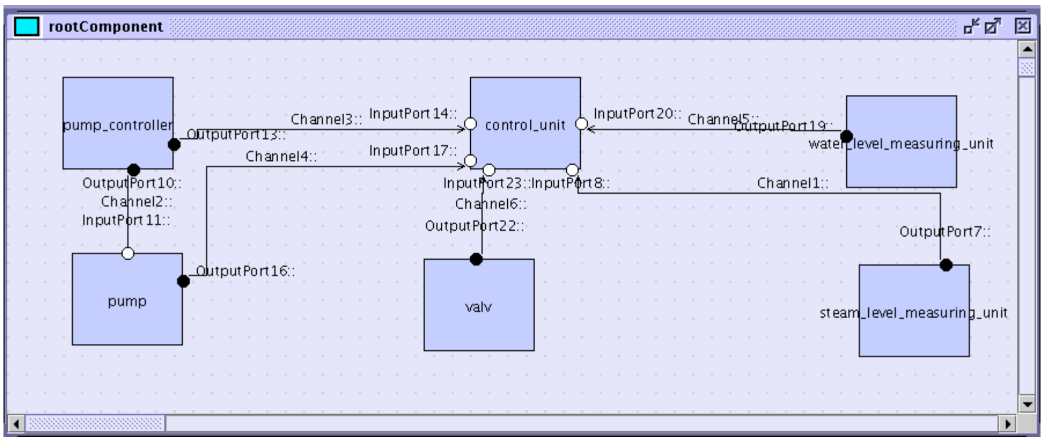


Figure 4.5: Component network, converted from the steam boiler ontology in Figure 4.3

work may be perceived as bothersome, this work is necessary to validate the document: apart from allowing the tool to extract the ontology, it produces a consistent document.

Necessity of manual work gives rise to the question whether the approach scales. The second case study, based on the DaimlerChrysler Demonstrator [BHH$^+$04], was conducted to prove the scalability. This document is much larger than the steam boiler specification (approx. 80 pages vs. 6 pages for the steam boiler), what makes it suitable for a scalability case study.

The document [BHH$^+$04] describes a car instrument cluster, showing the current speed, RPM (motor revolutions per minute), outside temperature and so on. The instrument cluster communicates via CAN bus with other ECUs (electronic control units).

As in the first case study, the goal was to extract the application domain ontology from the document. In the scalability case study the time that was necessary for different process steps was also documented. This way it was possible to identify time consuming steps that potentially do not scale.

The rest of this section describes the single steps of the case study. Section 4.3.1 describes document preparation, which was necessary for a large document, Section 4.3.2 introduces the results of the first parsing and Section 4.3.3 explains why rephrasing of some text parts were necessary. Sections 4.3.4 and 4.3.5 describe the results of taxonomy building and association mining respectively. Section 4.3.6 summarizes the lessons learned from this case study.

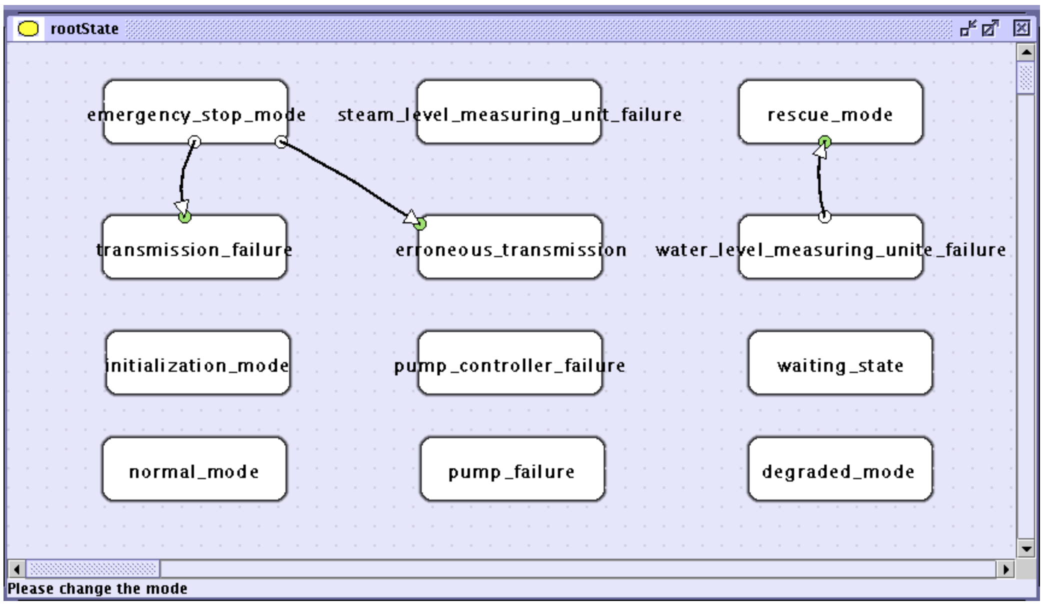


Figure 4.6: The automaton for “control_unit”, converted from the steam boiler ontology in Figure 4.3

4.3.1 Document Preparation

Text analysis starts with document preparation. There is a set of purely technical issues that are unimportant for smaller documents, but can become time consuming for larger ones. For the analysis it is necessary to convert the text into a one-sentence-per-line format. There are tools that recognize sentence boundaries, as for example the one by Ratnaparkhi [Rat98]. However, it turned out that this approach does not work well if the text contains also incomplete sentences.

In the first step of text preparation, the text was manually transformed into a one-sentence-per-line format. The formatting and the first reading of the specification text took one working day.

At this stage, grammatically wrong sentences were not reformulated and item lists and tables were not converted to full-fledged sentences. Although the predicate and term extraction in its current form (see Section 2.1.3) works for grammatically correct sentences only, the goal was to see how much “noise data” is produced in such a way and whether it is really necessary to rephrase incorrect sentences manually.

4.3.2 Parsing and Information Extraction

After reformatting the text it was possible to parse it and to extract syntax information. The predicate, the subject and objects were extracted from each sentence. Extraction results showed that rephrasing of incorrect sentences was necessary.

By analyzing the extracted predicates and their arguments, a lot of wrong verbs and objects were discovered. For example, the operations "=", "$<$" and "$>$" were classified as verbs, as they often occurred in the specification text in the verb position:

- If Ig-Lock = 1 then the ignition key is in position ignition on.
- If Current-Speed-V $<$ 30 km/h and the Internal-Temp values are sinking, then Outside-Temp-Shown = Internal-Temp.
- If Current-Speed-V $>=$ 50 km/h the rising Internal-Temp values are ignored for 1,5 minutes.

There was an additional problem with the text containing incomplete and grammatically incorrect sentences: the term extraction looks for the sentence predicate and then extracts predicate's arguments (terms). For grammatically incorrect sentences this is not always possible, so incorrect sentences are just ignored during term extraction. If the requirements document contains incorrect sentences, it is not possible to guarantee that all the relevant concepts are extracted. It could happen that some concepts occur in incomplete sentences only, so that they are completely ignored.

For these reasons the next step was to rewrite incomplete sentences into grammatically correct ones.

4.3.3 Lists and Tables: Proper Phrasing

It turned out that lists and tables were the main source of incomplete sentences. For example, input signals of the instrument cluster were described like this:

Ig-Lock: Describes the position of the ignition key. If Ig-Lock = 1 then the ignition key is in position ignition on. Sent by the ignition lock control unit. Scope: $\{0, 1\}$. Received every 100 ms. Transferred by the CAN bus.

Ig-LockR: Describes the position of the ignition key. If Ig-LockR = 1 then the ignition key is in position radio. Sent by the ignition lock control unit. Scope: $\{0, 1\}$. Received every 100 ms. Transferred by the CAN bus.

Status-Door-dd: Describes the status of the driver's door. Scope: $\{$ open (= 1), closed (= 0)$\}$. Sent by the door control unit. Received every 100 ms. Transferred by the CAN bus.

Each phrase of such constructions was completed so that it became a grammatically correct sentence. In most cases it could be done schematically, but the rephrasing still required manual work. For example, the above list was transformed into

- Ig-Lock describes the position of the ignition key. If Ig-Lock equals 1 then the ignition key is in ignition-on-position. Ig-Lock is sent by the ignition lock control unit. Ig-Lock can equal 0 or 1. Ig-Lock is received every 100 ms. Ig-Lock is transferred by the CAN bus.
- Ig-LockR describes the position of the ignition key. If Ig-LockR equals 1 then the ignition key is in radio-position. Ig-LockR is sent by the ignition lock control unit. Ig-LockR can equal 0 or 1 Ig-LockR is received every 100 ms. Ig-LockR is transferred by the CAN bus.
- Status-Door-dd describes the status of the driver's door. Status-Door-dd can equal 0 or 1. If Status-Door-dd equals 1, the driver's door is open. If Status-Door-dd equals 0, the driver's door is closed. Status-Door-dd is sent by the door control unit. Status-Door-dd is received every 100 ms. Status-Door-dd is transferred by the CAN bus.

Some transformations according to the writing rules, learned in the steam boiler case study, were necessary as well. These writing rules include:

- always use the same name for the same concept. (The original text obeyed this rule, so no correction was necessary.)
- In the case of compound names, either use names that, put in the sentence, remain grammatically correct (e.g., "normal mode" instead of "mode normal") or mark the compound names as such (i.e., "mode-normal"). In the instrument cluster specification, "position radio" was replaced with "radio position", "switched off position" with "switched-off-position", etc.

(See Section 4.5 for the complete list of writing rules.)

Such transformations made syntax-based analysis possible. All these transformations took 1.5 working days, which is justifiable for a 80-page document. The overall time cost for document preparation up to this point amounted to 2.5 working days.

4.3.4 Taxonomy Extraction

Taxonomy extraction is based on the analysis of cluster intersections. The first ASIUM run showed that there were more than 600 cluster intersections produced by the text. To build a taxonomy it is necessary to analyze cluster intersections, so this step could become time consuming.

During taxonomy building single clusters were analyzed as well to detect wrong usage of terms: every time a cluster containing unrelated concepts was encountered, it was possible to detect the textual source of this inconsistency and eliminate it.

In the instrument cluster case study a relatively small number of inconsistencies was detected:

- The verb "denote" produced a huge concept cluster containing unrelated concepts. This was due to the fact that the verb "denote" occurred both in constructions like "`⟨some-signal⟩ denotes` ..." and in "`⟨some-parameter⟩ denotes` ..." This problem could be corrected for example by replacing "denote" by "influence" when talking about system parameters. In the case study this correction was not done because both signals and system parameters could be clustered using other verbs. The "denotes"-cluster was just ignored.
- During the clustering it was discovered that some concept names had to be replaced. The replacement was necessary because several different names were used for the same concept. The following concept names were corrected:
  - engine-warning → engine-warning-signal
  - indicator-left → indicator-left-signal
  - indicator-right → indicator-right-signal
  - turn-signal-left signal → turn-signal-left
  - turn-signal-right signal → turn-signal-right
  - the pointer of the engine speed indicator → rev-meter-display-pointer

With the corrections described above the following taxonomy was built:

- users (subjects of "adjust", subjects of "enter", subjects of "press", subjects of "release"):

$$\{driver,\ service\ man,\ user\}$$

- hardware (subjects of "determine", prepositional objects of "seen as", prepositional objects of "sent to", prepositional objects of "send to", prepositional objects of "transmitted by", subjects of "turned on")

$$\{system,\ engine\ control\ unit,\ message\ receivers,\ message\ transmitters\}$$

Message receivers and *message transmitters* are clusters on their own, so there are the following sub-clusters:

- message receivers (prepositional objects of "sent to", prepositional objects of "send to")

  $$\{\mathit{engine\ control},\ \mathit{indicator},\ \mathit{digital\ display},\ \mathit{radio},\ \mathit{display}\}$$

- message transmitters (prepositional objects of "transmitted by")

  $$\{\mathit{can\ bus},\ \mathit{instrument\ cluster}\}$$

- displays (direct objects of "watch")

  $$\{\mathit{rev\ meter},\ \mathit{speedometer},\ \mathit{outside\ temperature\ display}\}$$

- signal (subjects of "equal", subjects of "sent" (passive form), direct objects of "sending", subjects of "transferred", subjects of "describes", subjects of "sent by" (passive form), direct objects of "sending", subjects of "processed", subjects of "received", subjects of "transmitted", direct objects of "equal", subjects of "describe"). There are too many signals to present all of them, so just a subset is presented here.

  $$\begin{gathered}\{\mathit{actual–number–of–revolutions},\\ \mathit{actual–number–of–wheel–revolutions–sensor1},\ldots,\\ \mathit{actual–number–of–wheel–revolutions–sensor4},\\ \mathit{but–down},\ \mathit{but–left},\ \mathit{but–minus},\ \mathit{but–plus},\ \mathit{but–right},\\ \mathit{command},\ \mathit{computed–second},\ \ldots\}\end{gathered}$$

- errors (subjects of "determined")

  $$\{\mathit{error},\ \mathit{problem}\}$$

- values

  - adjusted values (subjects of "adjusted", direct objects of "decrease", direct objects of "increase")

    $$\{\mathit{time},\ \mathit{minutes/hours}\}$$

  - computed values (subjects of "computed", subjects of "calculated")

    $$\{time,\ speed,\ car\ speed\}$$

- pointer (prepositional objects of "goes to", subjects of "steered" (passive form))

  $$\{rev\text{—}meter\text{—}display-pointer,\ the\ pointer\ of\ the\ engine\ speed\ indicator\}$$

- temperature (direct objects of "falling", subjects of "sinking")

  $$\{temperature\ values,\ internal-temp\ values\}$$

- scale position (prepositional objects of "is below", direct objects of "remain at")

  $$\{horizontals,\ minsv,\ right\ scale\ end\}$$

- warning (subjects of "appear")

  $$\{warnings\ of\ level\ 2,\ other\ warnings,\ warning\}$$

- actuator (direct objects of "activate", prepositional objects of "turn off", subjects of "turned on" (passive form), subjects of "deactivated", subjects of "activated" (passive form)).

  $$\{stepping\ motor,\ automatic\ door\ lock,\ both\ arrows\ of\ the\ indicator\ lights,\ indicator\ lights,\ lights,\ turn\ signal,\ hazard\ warning,\ display,\ attribute,\ the\ left\ arrow\ of\ the\ indicator\ lights,\ the\ right\ arrow\ of\ the\ indicator\ lights,\ the\ display\ of\ the\ engine\ warning\ light,\ the\ indication\ of\ the\ outside\ temperature,\ radio,\ instrument\ cluster,\ ic,\ ignition,\ engine\}$$

- indication (direct objects of "stop")

  $$\{visible\ and\ audible\ indication,\ hazard-warning\ signal\ flasher,\ blinking\}$$

- suppressed information (subjects of “suppressed” (passive form), subjects of “ignored”)

$$\{\mathit{numbers\ of\ revolutions\ below\ 320\ min^{-1}},\ \mathit{warnings},\ \mathit{the\ warnings\ of\ level\ 3},\ \mathit{messages\ of\ level\ 2},\ \mathit{rising\ internal-temp\ values},\ \mathit{r-ic-stat\ messages},\ \mathit{r-stat\ messages}\}$$

- settings (subjects of “stored” (passive form), subjects of “damping”)

$$\{\mathit{blink-frequency-adj},\ \mathit{blink-frequency-colon},\ \mathit{ice-threshold},\ \mathit{parameter-value},\ \mathit{release-bit},\ \mathit{damping},\ \mathit{variant-car},\ \mathit{adjustment-speed-minutes},\ \mathit{12-24-time-format},\ \mathit{variant-specific-bit-temp},\ \mathit{adjustment-speed-hours}\}$$

  - $\mathit{Damping}$ is itself a cluster, consisting of subjects of “damping”:

$$\mathit{damping} = \{\mathit{damping-pt1},\ \mathit{damping-pt2}\}$$

Analyzing the whole plethora of cluster intersections and building a taxonomy (concept and cluster hierarchy) took approximately 1.5 working days. The overall time cost up to this point amounted to 4 working days.

4.3.5 Association Mining

To explain scalability problems potentially posed by association mining, it is sensible to start by repeating some definitions from Section 2.3: for an item set A, let $\mathit{trans}(A)$ be the set of transactions containing A and let N be the total number of transactions. The *support* of the association $A \Rightarrow B$ is defined as $\frac{|\mathit{trans}(A \cup B)|}{N}$. The *confidence* of the association $A \Rightarrow B$ is defined as $\frac{|\mathit{trans}(A \cup B)|}{|\mathit{trans}(A)|}$.

In the case studies the analysis was performed on the per-sentence basis and a transaction was defined as a pair of concepts occurring in the same sentence. For the instrument cluster case study this definition led to more than 1000 potential associations. In order that this plethora of potential associations became manageable, the associations were sorted lexicographically by $(\mathit{absolute\ frequency}, \mathit{confidence})$. Absolute frequency of the association $A \Rightarrow B$ is defined as $|\mathit{trans}(A \cup B)|$. *Formally*, two associations with the same support have also the same absolute frequency, so it is possible to use the standard measure $\mathit{support}$. Due to rounded support values presented by KAON to the user, $\mathit{absolute\ frequency}$ gives more information. Lexicographical sorting means that the associations were sorted by $absolute\ frequency$ and in the case of equal $absolute\ frequency$ they were sorted by $confidence$.

For the ontology building the associations with $absolute\ frequency \geq 5$ were used, which corresponded approximately to the most frequent 25% of associations. It took about one working day to manually validate these associations and to include the relevant ones into the ontology. The overall time cost up to this point amounted to 5 working days.

4.3.6 Results of the Instrument Cluster Case Study

The goal of the instrument cluster case study was to see whether the ontology extraction approach presented in Chapter 2 still works for large documents and whether the amount of manual work necessary for the extraction is still justifiable.

Figure 4.7 illustrates that ontology extraction worked for this case study as well: it shows an excerpt of the extracted ontology.[6] Figure 4.7 shows the top ontology class (`kaon:Root`), its subclasses (actual ontology classes) and relations between them. It shows also some typical associations:

- can_bus transfers input_signals
- display contains pointer
- display displays warnings
- display displays errors
- ...

The other goal of the instrument cluster case study was testing the scalability of the approach. During this case study were extracted:

- 123 concepts and concept classes, organized in 13 top-level classes and further subclasses
- 61 associations between different concepts

Additionally to the extraction of concepts and associations inconsistencies in term usage were discovered and corrected. The time cost of 5 working days seems justifiable for an 80-page document, given that inconsistencies were detected and corrected and a domain ontology was constructed.

[6]See comments to Figure 1.2, page 24, for arrow meanings.

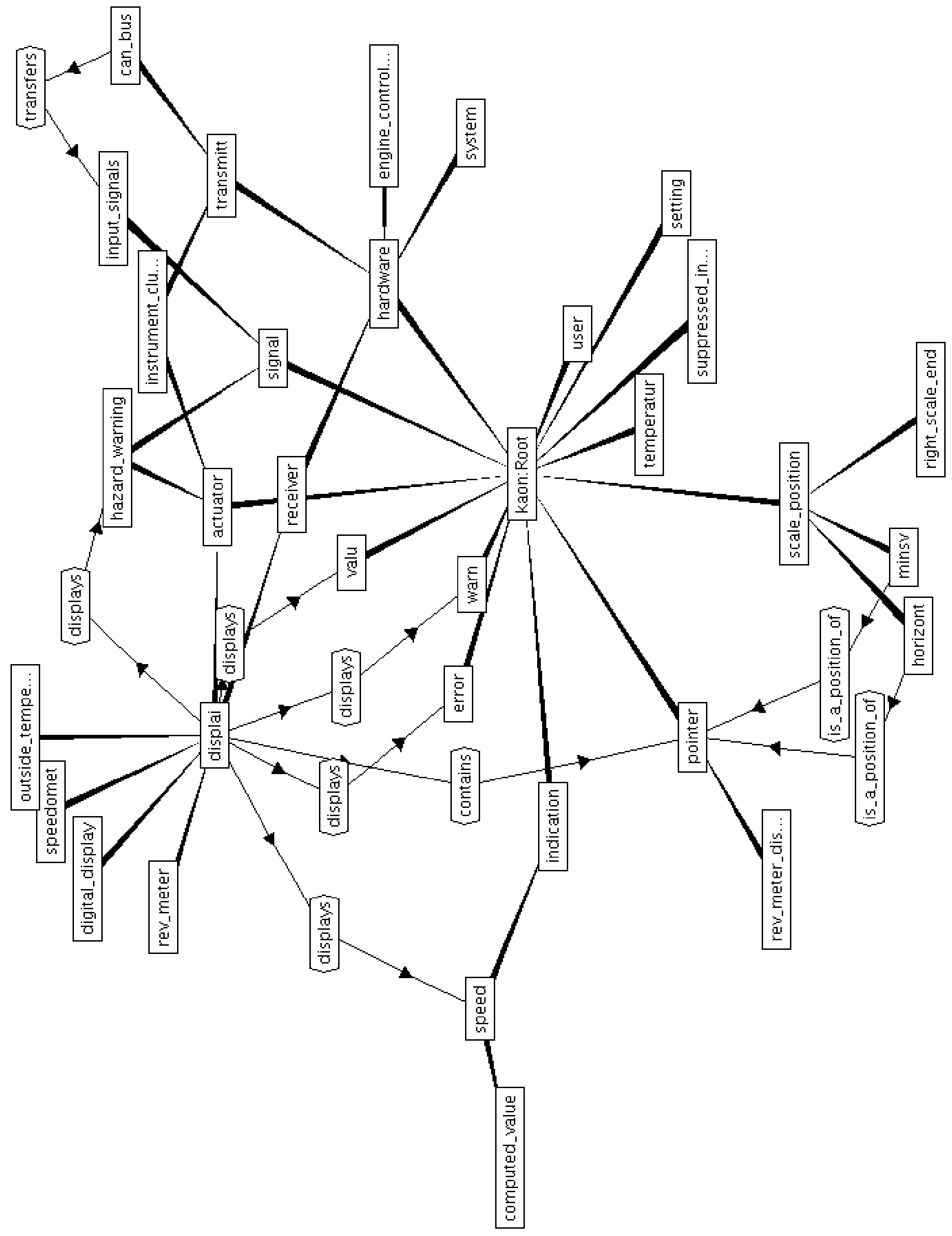


Figure 4.7: Instrument Cluster: part of the extracted ontology

4.4 Industrial Case Study

The first two case studies, conducted on academic requirements documents, demonstrated the applicability of the approach. In the first (steam boiler) case study the extracted term list was almost complete, as compared to the explicit glossary provided in the document. The second (instrument cluster) case study showed scalability of the ontology extraction method. However, evaluation of the extraction results was impossible due to a missing reference ontology, as well as an explicit glossary.

To better evaluate the ontology extraction approach, a case study handling real industrial documents was conducted. As in the instrument cluster case study, it was not possible to evaluate completeness of the extracted ontology directly due to missing explicit glossary. However, the industrial requirements documentation consisted of several documents, which made cross-validation possible. The ontologies were separately extracted from different documents and then compared. Due to insufficient domain knowledge no inconsistency detection and/or elimination was performed on these documents.

The analyzed requirements documentation describes an information system for a large car repair shop.[7] The repair shop is specialized on old precious cars, which makes the exact documentation of every restoration step necessary. The repair shop employes a large team of motorcar mechanics with different skills and experience levels. Due to the complexity of certain maintenance operations, they may be performed by mechanics with special skills only. The analyzed requirements documentation describes the interplay of restoration steps, mechanics and documentation.

Figures 4.8 and 4.9 show excerpts of ontologies extracted from two different requirements documents, both documents describing the car repair shop. Due to the size of the ontologies it is impossible to show the complete structure: the ontologies consist of more than 150 concepts and more than 70 relations each. The extracted ontologies contain both common and differing concepts and associations. The presented excerpts show for example:

Some system functions: “verbal_order_entry”, “free_text_note_entry”, “electronic_communication”, and “quick_access”.

Different actors: “reception_clerk”, “testing_junior_mechanic”, “assisting_mechanic”, etc.

Repair shop areas: “holding_area” and “registration_department”.

[7]The analyzed documents are obtained under non-disclosure agreement. For this reason the document origin cannot be named. The examples given further in the text are obfuscated, solely the grammatical structure is preserved.

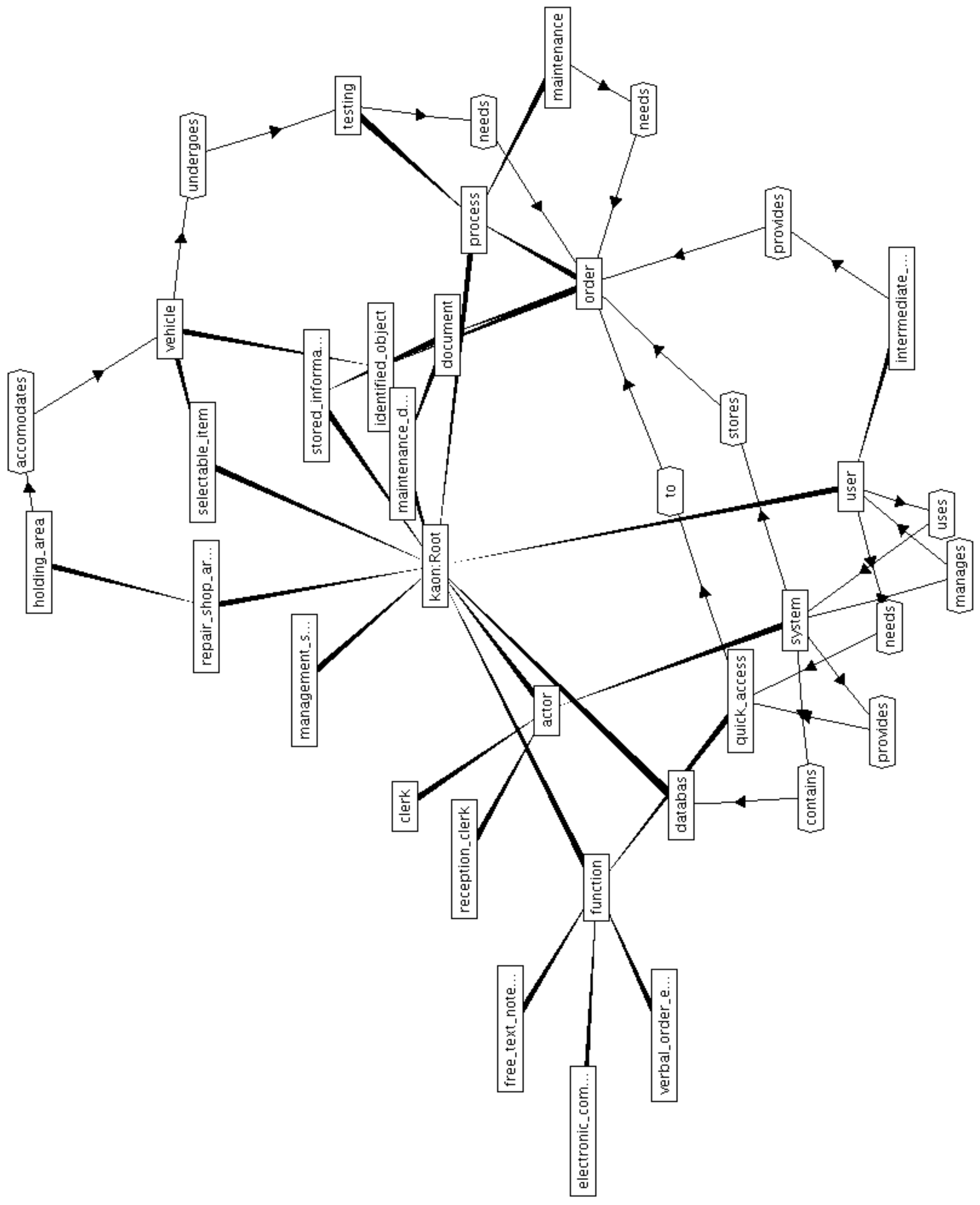


Figure 4.8: Car Repair Workshop, first document

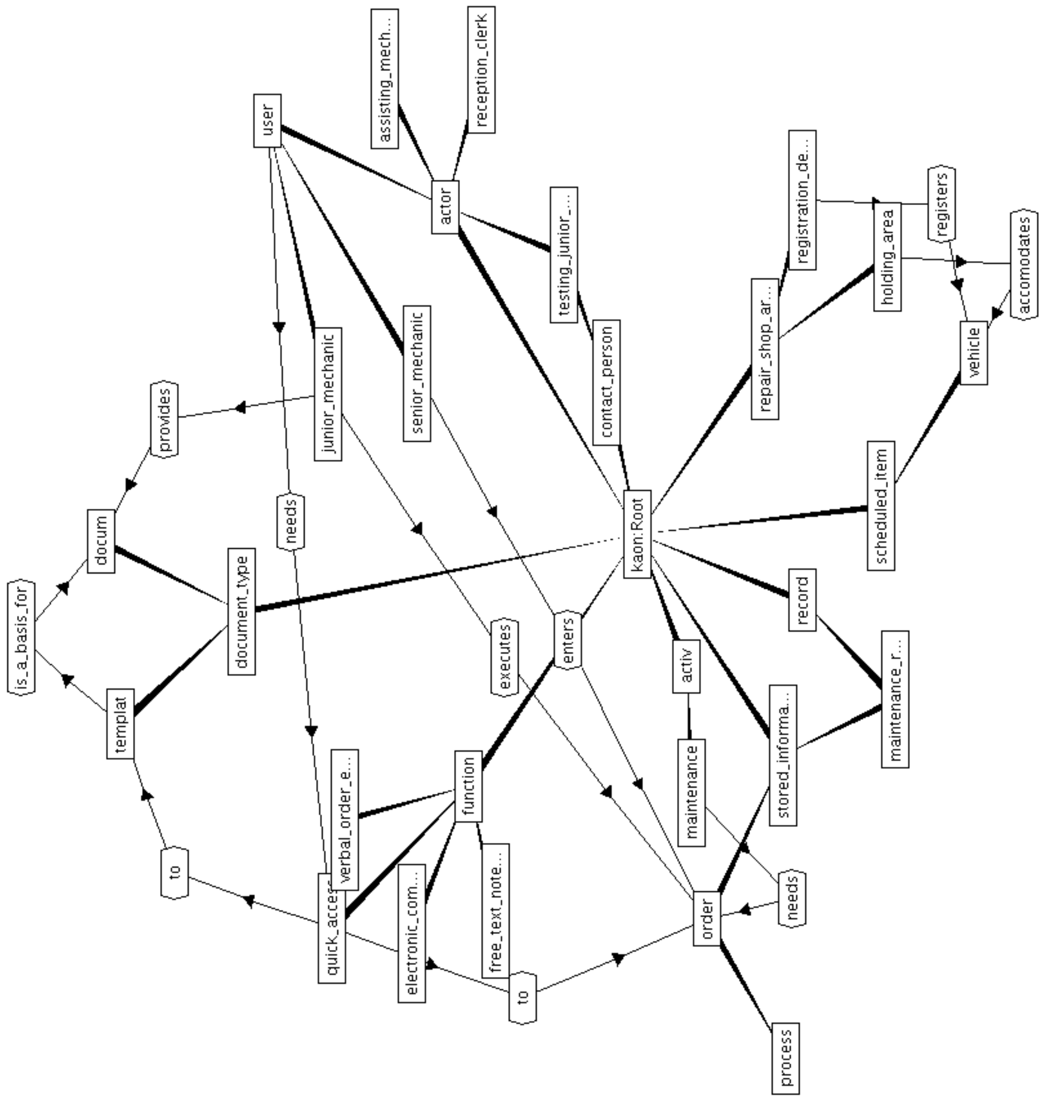


Figure 4.9: Car Repair Workshop, second document

	lexical recall	lexical precision	taxonomic overlap	relational overlap
1st document vs. 2nd document	0.6030	0.4846	0.3480	0.2565
2nd document vs. 1st document	0.4846	0.6030	0.4113	0.2333

Table 4.2: Ontologies similarities for the industrial case study

The two ontologies extracted from different documents overlap to a large part, but they are still different. For example, in the ontology in Figure 4.9 "order" is both a "process" (some repair shop workflow) and "stored_information" (some object stored in the database), whereas in the other ontology "order" is additionally a subclass of "document". In this case it is not necessarily an inconsistency, but it is at least a sign that the requirements documents are different and one of the documents does not explicitly introduce the document classes managed by the system. Further examples of discrepancies are the class of document templates[8] ("template", present in Figure 4.9, absent in 4.8) and the experience level of mechanic allowed to create maintenance orders: in the case of Figure 4.8 it is the "intermediate_mechanic" but in Figure 4.9 it is the "senior_mechanic". Manual analysis of the requirements documents showed that these discrepancies are due to differences in documents. Table 4.2 shows similarity measures, introduced in Section 4.1, for the extracted ontologies. It is easy to see that the ontologies are different, but still have high degree of similarity.

The following list enumerates the terms extracted from one of the documents[9].

- actors:
  - senior_mechanic
  - assisting_mechanic
  - junior_mechanic
  - ...
- workflow
  - maintenance
  - testing
  - order (in the sense of maintenance order)

[8]The concrete document templates are not shown for the reason of non-disclosure
[9]The list is shortened both for the sake of brevity and non-disclosure

- departments:
  - testing_department
  - registration_department
  - ...
- documents
  - testing results
  - maintenance results
  - ...
- ...

The term list extracted from the other document was different. Though, this difference was due to differences in the documents. When reduced to concepts occurring in both documents, the extracted term lists became the same.

The two extracted ontologies, in spite of (or perhaps even due to) their differences, illustrate the core ideas of the whole thesis:

- The extracted ontology depends on the analyzed document. To construct an ontology correctly representing the application domain, documents properly representing the domain are necessary.
- Inconsistencies in the document can be detected during/as a result of ontology extraction.
- The extraction results must be validated by a domain expert for further usage in the development process.

The industrial case study was successful in the sense of confirming these three core ideas.

4.5 Case Studies: Lessons Learned

The goal of the case studies was to evaluate the feasibility and to explore the limitations of the ontology extraction approach. The first case study resulted in an extraction of an ontology that differed from the manually constructed one. Direct ontology comparison was difficult: the ontologies overlap, but none of them contains the other. However, the two ontologies show high similarity according to the metrics introduced in Section 4.1.

Another evaluation criterion is the completeness of the extracted term list. Completeness evaluation for the steam boiler case study was easy: the steam boiler specification explicitly defines the following concept classes: hardware components, messages, operation modes, physical parameters and failures. As for hardware concepts, all but two were extracted. The approach did not extract "operator desk" and "message transmission system". These concepts are mentioned only once in the document and their role is not further specified. A human reader would extract these two concepts, but would have to guess how they interact with other components. This point can be seen both as a weakness of the extraction technique and as an omission in the document: these two components are also missing in the steam boiler simulator (Figure 4.1, page 84), programmed for the participants of the formal methods contest, whose goal was to provide a formal steam boiler specification. The approach did not extract the physical parameters either: the parameter names (like "capacity", "minimal limit", "maximal limit") were mentioned solely in incomplete phrases. An integration with the Wendt's list analysis approach [Wen04] could eliminate this drawback[10]. As for other concept classes (messages, operation modes and failures), the approach succeeded in extracting all the concepts belonging to these classes.

Evaluation of the instrument cluster case study was more difficult because the analyzed document does not provide an explicit glossary. Due to lack of domain knowledge no reference ontology was built, either. Ad hoc, by skimming the document, one can identify following concepts: instrument cluster, rev meter, speedometer, indicator lights, engine control light, display, ignition key, radio, ..., that are all present in the extracted ontology. One can also easily identify some messages and technical parameters, like default pointer positions for dials. Nevertheless, for proper evaluation of completeness it is unwise to rely on such a comparison. Either a domain expert that could evaluate completeness of the extracted model directly or an extraction tool that *guarantees* that all the concepts are extracted is necessary for proper evaluation. For this reason completeness of term extraction was not evaluated for this case study.

The explanatory power of these two case studies was limited to academic documents. To investigate the applicability of the ontology extraction approach to industrial documents, a third case study was performed. The third case study was conducted on two industrial documents, describing different aspects of the same system. As in the instrument cluster document, there was no explicit term list in the documents, which made direct completeness evaluation of the extracted term list impossible. However, the availability of two documents allowed for cross-evaluation of completeness: it was possible to evaluate whether all the terms extracted from the first document *and* present in the second document were com-

[10]See Sections 5.4.2 and 6.3.2 for details

pletely extracted from the second document and vice versa. The comparison of extracted term lists showed their completeness. Furthermore, the two ontologies show high similarity when compared as described in Section 4.1.

Writing Rules

The case studies showed that ontology extraction works for properly written documents. "Properly written" means in this context that the documents are grammatically correct and do not contain terminology inconsistencies. The first case study resulted in a number of writing rules, helping to produce properly written documents:

- "Common sense" rules, sensible to adhere to even without automated document analysis:
  - Always use the same name for the same concept (avoid synonyms).
  - In the case of compound names, either use names that, put in the sentence, remain grammatically correct (e.g., "normal mode" instead of "mode normal") or mark the compound names as such (i.e., "mode-normal").
  - Always use the complete form in the case of compound names: i.e., "stop message or start message" instead of "stop or start message".
  - Do not use the verbs "be" and "have". They do not provide much information even for the human reader. For the computer-based analysis they produce large clusters of unrelated concepts. Nevertheless, it is allowed to use these verbs to build passive form or perfect tenses. In those cases they are easy to filter out.
- Rule due to technical deficiency:
  - Do not use cross-sentence references like "Message X is sent by unit Y. This message indicates ..."

To test whether specification texts written according to these rules are still human-readable, the following small poll was performed: 14 people of similar background (computer science Ph. D. candidates) were given two versions of the steam boiler specification: the original one and the version obtained after the elimination of all inconsistencies. The respondents were not told which one of the texts was the original version of the specification. They had to recognize which one of the texts was the original version and which one was prepared for automated analysis. The poll resulted in 7 correct answers, 5 wrong answers, and 2 abstentions from voting. This balance of the correct and wrong answers shows that texts written according to the above writing rules are still human-readable.

Another case study experience is that the documents in their initial form barely follow the above writing rules. The most often (but not sole) inconsistency is the usage of several different names for the same concept. The inevitable presence of inconsistencies makes ontology extraction an iterative process. The inconsistencies are detected in interactive analysis steps, namely analysis of term clusters and their intersections and validation of the associations proposed by the data mining tool. This iterative process may seem time consuming. However, as one of the case studies showed, the effort to eliminate inconsistencies is affordable. Furthermore, it is illusory to save time by non-elimination of requirements documents inconsistencies: it just shifts the problem to later project phases, where elimination costs even more time.

Limitations of the Approach

The approach in its current form has an inherent limitation that further constrain the writing rules. This limitation is due to the clustering algorithm. The algorithm clusters the concepts used with the same verb and looks for cluster intersections. It cannot relate disjoint clusters containing similar concepts. It is possible to define similarity of concepts by means of their main noun. This definition would relate for example “stop message” with “start message” and “pump state message” and so on. This kind of similarity (lexical similarity) is taken into account in the ATRACT approach [MAN01]. See Section 5.4.2 for the details of similarity measures used by Mima et al. in ATRACT.

At a first glance, building basic clusters solely on the basis of grammatical contexts does not look like a limitation, but it caused, for example, that “message-pump-control-state” was completely ignored in the first run of the steam boiler case study. “Message-pump-control-state” occurs only in the sentence

> Message pump-control-state(n; b) gives the information which comes from the pump controller of pump n (there is flow of water or there is no flow of water).

The verb “give” is also used solely in this sentence, which causes a stand alone concept.

This problem could be solved in two ways:

- Another principle to construct basic clusters can be used. Additionally to similarity of grammatical contexts, which is used now for clustering, it is possible to use measures introduced in Section 5.4.2: lexical and syntactical similarity. Lexical similarity would solve the above problem because "message" is the head word of both "Message pump-control-state(n; b)" and other message names.

- The same verbs must be systematically used with related concepts. For example, "indicate" is used with other messages, so "Message pump-control-state(n; b) indicates . . . " would solve the problem. In this case it is sufficient to stick to basic clusters built on the basis of grammatical contexts.

To summarize the results of the case studies, they demonstrated applicability of the proposed ontology extraction approach, showed its limitations and produced a set of rules improving the quality of requirements documents and leading to better analysis results.

Chapter 5

Related Work

The amount of research work on requirements engineering is enormous. To keep the following overview of the related work manageable, solely approaches that are sufficiently close in their ideas to the presented thesis are considered. Related work in a narrower sense includes work on requirements documents analysis and on ontology construction. The work on ontology construction is the most simple, as most homogeneous, part of the related work. It is presented in Section 5.1.

The presentation of the related work on document analysis is subdivided into the following three groups:

- general guidelines for quality requirements documents, quality assurance and summarization of requirements documents,
- systematic detection of inconsistencies in the document,
- information extraction from the document.

The above list is sorted according to the degree of similarity to the presented thesis. The boundaries between these categories are blurred: information extraction can also detect inconsistencies and inconsistency detection gives hints about document quality.

The first type of the RE approaches is necessary due to the fact that requirements documents are mostly very complex and of poor quality. Some quality assurance can be achieved when requirements documents follow certain writing guidelines. However, even perfectly written requirements can still be contradicting. To find contradicting requirements, it is necessary, first, to make related requirements somehow visible. The approaches measuring potential requirements obscurity and finding related requirements are presented in Section 5.2.

The next group of related work, namely inconsistency detection, goes further than pure evaluation of the document quality: it makes the phrases visible, that are responsible for poor document rating. These phrases are mostly troublesome, as they allow for several interpretations. The approaches presented in Section 5.3 introduce criteria for detection of such misinterpretable phrases. The approaches listed in Section 5.3 are comparable to only a limited extent to the work presented in this thesis: they mostly address semantical inconsistencies, but no terminology inconsistencies, as in the presented thesis.

The most thorough analysis of the related work is devoted to the information extraction approaches, as they are the closest to the presented thesis. Section 5.4 introduces several approach types (lexical, syntactical and semantical) in detail in order to show the difference to the work presented in this thesis. Last but not least, Section 5.5 shows where the presented thesis goes beyond state of the art.

5.1 Semantic Web and Related Work on Ontology Building

The idea to use an ontology as a communication basis is not new to computer-related fields. In artificial intelligence ontologies were proposed as a communication means for intelligent agents. An ontology, as defined in artificial intelligence, is a set of concepts and relations between these concepts. Such a concept network serves as a common world model for communicating intelligent agents. Agents sharing a common world model can be sure, at least, that they talk about the same concept when they use the same word. As a continuation of the agent communication idea, it was proposed to use ontologies as an explicit definition of subjects and relations for the semantic web. Berners-Lee, Hendler, and Lassila [BLHL01] define semantic web as

> "...an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation."

The core idea of semantic web is to replace the state of the art keyword-based search with semantic search. For example, keyword-based search for "testing" results in web sites devoted both to testing as software engineering activity (e.g., `http://www.junit.org/`), and to language testing (e.g., `http://www.ets.org/`).[1] If the user is interested in software engineering only, the restriction to the testing as a software engineering activity would yield more relevant results. Such a restriction is not always possible by the means of keywords. The solution to this problem, as proposed for semantic web, is to introduce an

[1]The cited URLs are among top results in Google search for "testing", `http://www.google.de/search?hl=de&q=testing&btnG=Google-Suche&meta=`

ontology, defining relevant terms. This ontology must be shared by the search engine, searching user and the web site to be found: the user should be able to say "I am looking for testing as SE activity" by pinpointing the ontology concept he is looking for. The search engine should be able to understand this query and the site to be found should be explicitly indexed as "relevant to testing as SE activity". See also [BL98] for more details on semantic web.

The necessity to construct such search ontologies for semantic web gave rise to a number of approaches. Breitman and Sampaio do Prado Leite [BS03] list several ontology construction methodologies. The listed methodologies all share the same basic steps, shown in Figure 2.1, page 32. These steps, apart from validation and verification, include:

1. identify information sources
2. identify list of terms
3. classify terms
4. describe terms

The approach proposed by Noy and McGuinness [NM01] is more explicit and defines more steps of ontology construction:

1. Determine the domain and scope of the ontology
2. Consider reusing existing ontologies
3. Enumerate important terms in the ontology
4. Define the classes and the class hierarchy
5. Define the properties of classes – slots
6. Define the facets of the slots
7. Create instances

Obviously, this list is not really different from the first list of ontology constructions steps: steps 1 and 2 correspond to "identify information sources", step 3 to "identify list of terms", step 4 to "classify terms" and steps 5–7 to "describe terms".

These steps seem natural to any ontology building approach. However, these ontology building methodologies are rather abstract in the sense that they do not specify *how* to identify information sources, *how* to classify terms, and so on. They just give some not-as-easy-to-follow recipes, like "List each term that seems to have a special meaning" [BS03] or give some ontology construction examples for a particular domain [NM01]. It was the primary goal of the presented thesis, to make this ontology construction steps more concrete and to bridge the gap between the pure text analysis approaches (introduced in Section 5.4) and the aforementioned ontology building steps.

5.2 General Work on Requirements Engineering

One of the major problems in requirements engineering is the poor quality of requirements documents. This poor quality can have many facets: requirements come from different sources, use different terminology, have different priorities, etc. To manage this chaos and to enforce certain quality standards on requirements, Robertson and Robertson [RR99] introduced a requirements template. Filling all the template fields should ensure quality of each single requirement. The template consists of the following fields:

Requirement Number: The need for requirement number is straightforward: it serves to identify each requirement.

Requirement Type: There are certain very common requirements types that can be found in every project, like "usability requirement", "look and feel requirement", "functional requirement", etc. This template field refers to such basic category. Robertson and Robertson introduce a category list as well.

Event/Use Case Number: Each requirement is a product of some use case (business event). For the sake of traceability, it is necessary to note the relevant use case for each requirement, in order to update the requirement when the use case changes.

Description: Description is the actual statement of what is required.

Rationale: Rationale tells why the requirement is important. This field is necessary to manage requirements evolution: when business goals change, this changes the rationale and shows which requirements are affected by this change.

Source: Source shows where the requirement comes from. It is necessary to keep the source in the case of several user groups.

Fit Criteria: Fit criteria are the most important part of the requirement: they state how it can be tested whether a system satisfies this requirement.

Customer Satisfaction and Dissatisfaction: Customer satisfaction (in the case of requirement compliance) and dissatisfaction (for noncompliance) measure requirements priority: requirements with high customer dissatisfaction have the highest priority, requirements with low satisfaction have the lowest priority.

Dependencies and Conflicts: This field lists (for each requirement) related and perhaps conflicting requirements. The definition of related and conflicting requirements is up to the requirements engineer.

Filling all the above fields for each requirement provides some quality assurance at the requirements level. A flaw of the above template is the necessity to determine related and conflicting requirements manually. Due to huge number of requirements in any realistic project it can easily happen that some related requirements are overseen.

To address this problem, Natt och Dag et al. [NRC$^+$02] developed a tool finding related requirements. The similarity degree of two requirements is measured on the basis of common words occurring in these requirements. The core of the work by Natt och Dag et al. is the evaluation of different similarity measures.

The approaches sketched above are barely comparable to the presented thesis. The goal of the above approaches is to introduce some guidelines for documents and find related parts in them. The goal of the presented thesis, to the contrary, is to go further in document analysis and to extract a domain model from requirements documents.

5.3 Related Work on Inconsistency Detection

Requirements inconsistencies and ambiguities are a well-recognized problem, contributing to poor quality of requirements documents. Kamsties et al. [KBP01] define requirement ambiguity in the following way:

> “a requirement is ambiguous if it has multiple interpretations despite the reader’s knowledge of the RE context.”

The presence of several interpretations is not as easy to determine as it seems. It is necessary to differ between the perceived and non-perceived ambiguity. The perceived ambiguity is less problematic, in the sense that as soon as the requirements analyst detects such an ambiguity, he can ask the domain expert for a proper interpretation. The more dangerous situation is when several analysts sees just one meaning of a requirement, but in fact there exist several interpretations.

To overcome this problem, Kamsties et al. [KBP01] introduced a set of rules for detection of typical ambiguous sentences. They define several ambiguity categories, for example:

Polysemy: Polysemy occurs when a word has several meanings. For example, in the requirement

> When the user inserts the paper strip, the Tamagotchi is set to its defaults,

"Tamagotchi" can mean both the electronic device and the creature simulated by this device.

Scope Ambiguity: Scope ambiguity can occur when several quantifier, like "a", "every", "each" occur in the same sentence. For example, the sentence

> All sections have a hallway.

can be interpreted both as

$$\forall s : section(\exists h : hallway \;\; has\_hallway(s, h))$$

and as

$$\exists h : hallway(\forall s : section \;\; has\_hallway(s, h))$$

Referential Ambiguity: Referential ambiguity is caused by an anaphora in a requirement that refers to more than one element introduced earlier in the sentence or in a sentence before. For example, in the sentences

> The controller sends a message to the pump. It acknowledges correct initialization.

"it" can refer both to the pump and to the controller and to the message.

Discourse Ambiguity: Discourse ambiguity occurs when relation between two requirements is ambiguous, but not due to anaphora. For example, in the sentences

> When the user pulls the paper strip, the cyber chicken is born.
> ... After its first night, the cyber chicken becomes a Marutchi.
> ... After 2 days, the cyber chicken becomes a Tamatchi (friendly teen).

"after 2 days" can refer both to "2 days after pulling the paper strip" and to "2 days after becoming a Marutchi".

The above list is not complete, as the work by Kamsties et al. is really extensive. See [KBP01] for the whole list of potential ambiguities. On the basis of this list, Kamsties et al. developed a checklist and a set of reading rules for detection of particular ambiguity types.

Rupp [Rup02] performs inconsistency detection in a similar way: she identifies a set of common writing patterns causing a possible misinterpretation and introduces a checklist for detection of such ambiguous writing. She introduces also a set of writing templates that should lead to better requirements due to more explicitness and less omissions.

Such lists of typical ambiguity sources can also serve to evaluate general document quality. This idea was implemented by Fabbrini et al. [FFGL01]. To evaluate requirements documents quality with respect to ambiguous requirements, Fabbrini et al. developed a tool detecting some typical phrases leading to requirements defects. They subdivide requirements defects into classes, like optionality, ambiguity, under-specification, etc. For each class of requirements defects they identify typical expressions. For example, optionality is characterized by expression like "if possible", "if appropriate", "if needed". The requirements document quality is evaluated on the basis of presence (or absence) of such expressions.

The above ambiguity approaches focus on semantical ambiguities. They look for potentially dangerous sentences, having several interpretations. For this reason these approaches are barely comparable to inconsistency detection in the presented thesis, as it focuses on detection of terminology inconsistencies. Elimination of terminology inconsistencies is a prerequisite for sensible detection of semantical ambiguities.

5.4 Related Work on Requirements Documents Analysis

All the text analysis methods have a common goal: they take a natural language text and produce some model of the system described in this text. The key differences in the approaches lie in the type of the produced models and in the requirements to the input texts. Ben Achour [Ben97] classifies the text analysis techniques as either lexical or syntactical or semantical. This classification, shown in Table 5.1, is based on the type of results provided by the corresponding analysis technique.

Lexical techniques are the most simple ones. They consider each sentence as a character or word sequence, without taking further sentence structure into account. Due to this simplicity lexical techniques are extremely robust. The flip side of this robustness is that lexical methods are limited to pure term extraction.

Approach type	Analysis tasks	Analysis results
lexical	identify and validate the terms	set of terms used in the text
syntactical	identify and classify terms, build and validate a domain model	set of terms used in the text and a model of the system described in the text
semantical	build a semantic representation of the text	logical representation of the text, formulae

Table 5.1: Classification of text analysis techniques

Syntactical approaches, as opposed to lexical ones, take also sentence structure into consideration. Based on this sentence structure, they extract not only the terminology, but also some domain model. Semantical approaches achieve more than the other two classes: they produce a formal representation of the text. It is mostly a kind of first order predicate logic, but the concrete representation may differ. This task is surely very demanding, which poses severe limitations on the text for the approaches to work.

The remainder of this section describes different kinds of text analysis approaches in more detail: Section 5.4.1 introduces the lexical approaches, Section 5.4.2 the syntactical and Section 5.4.3 the semantical ones.

5.4.1 Lexical Approaches: Term Identification

The goal of the lexical approaches is to identify concepts used in the requirements document. They do not classify the identified terms or build a domain model. The common feature of these techniques is that they analyze the document just as a character or a word sequence. Berry [Ber01] lists several approaches applying information retrieving techniques to requirements engineering. to give the flavor of lexical approaches, the following will be considered here: AbstFinder [GB97], lexical affinities by Maarek [MB89] and documents comparison by Lecoeuche [Lec00].

AbstFinder [GB97] works in the following way: it considers each sentence simply as a character sequence. Such character sequences are compared pairwise to find common subsequences. These subsequences are assumed to be domain concepts. For example, consider two sentences taken from the steam boiler case study [ABL96b]:

The steam-boiler is characterized by the following elements:

and

> Above m2 the steam-boiler would be in danger after five seconds, if the pumps continued to supply the steam-boiler with water without possibility to evacuate the steam.

The first sentence is shorter and it is augmented with spaces before the start of the search for common character subsequences. Then one of the sentences is rotated character-wise and for each rotated position AbstFinder controls whether there are aligned common subsequences. Rotation of the sentences is necessary to identify character chunks placed differently, like "flight" and "book" from "The flights are booked" and "He is booking a flight". (This example is taken from the AbstFinder article [GB97].) Such analysis is performed for all sentence pairs.

For the steam boiler example introduced above, the aligned position would look like

```
         The steam-boiler is characterized by...
Above m2 the steam-boiler would be in danger...
```

In this case AbstFinder would identify "the steam-boiler" as a concept contained in the document.

However, when considering two other sentences from the steam boiler specification, like

> Below m1 the steam-boiler would be in danger after five seconds, if the steam continued to come out at its maximum quantity without supply of water from the pumps

and

> Above m2 the steam-boiler would be in danger after five seconds, if the pumps continued to supply the steam-boiler with water without possibility to evacuate the steam

AbstFinder would identify "the steam-boiler would be in danger after five seconds, if" as a common concept, which is definitely too much.

The approach by Maarek [MB89] identifies concepts as word pairs where the appearances of these two words in the same sentence correlate. For example, "steam" and "boiler" often co-occur in the steam boiler specification [ABL96b], so this approach would identify "steam boiler" as an application concept.

Both Goldin and Berry and Maarek assume that the most important terms can be identified as the most frequent ones. Thus, they would probably identify some common words like "or", "and", etc. as important concepts. The approach by Lecoeuche [Lec00] is free from this drawback. It compares the frequency of the concept in the analyzed document with the frequency of the same concept in some baseline document. Let F_a be the number of occurrences of some term in the analyzed document and F_b the number of occurrences of the same concept in the baseline document. Then, the importance measure of a concept is defined as $imp = \frac{F_a}{F_a+F_b}$. High importance measure can imply that the concept is mentioned just few times in the baseline document (for example in the definitions), but is mentioned many times in the analyzed document. Concepts with high importance measure are identified as application domain terms.

Summarizing the approaches presented in this section, it is possible to say that they extract concepts from documents, but they do not classify the concepts. They do not build a domain model either. They can help in building a domain glossary, but it is only a part of an ontology.

5.4.2 Syntactical Approaches: Identifying Terms and Relations

Syntactical approaches, presented in this section, promise more than pure concept identification. These approaches became widely known in the field of object-oriented analysis, as they allow for easy mapping of extracted concepts to classes, objects, attributes and methods. In their original versions these approaches do not offer any automation, but they could be partially automated using linguistic techniques available now.

One of the first approaches aiming at analysis of specification texts is the one by Abbott [Abb83]. The goal of the Abbot's approach is to

> "... identify the data types, objects, operators and control structures by looking at the English words and phrases in the informal strategy"

Abbott takes the following types of words and phrases into consideration during model building:

- common nouns
- proper nouns and other forms of direct reference
- verbs and attributes

These word types are used in the following way during model building:[2]

1. A common noun in the informal strategy suggests a data type.
2. A proper noun or a direct reference suggests an object.
3. A verb, predicate or descriptive expression suggests an operator.

[2]This list and the examples are taken from the Abbott's paper [Abb83]

4. The control structures are implied in a straightforward way by the English.

This strategy works in the following way: given the specification text like

> If the two given DATEs are in the same MONTH, the NUMBER_-OF_DAYS between them is the difference between their DAYs of MONTH,

Abbott identifies the common nouns (capitalized in the above example) as data types. A similar strategy is applicable to objects: in a phrase like

> Determine the number of days between THE_EARLIER_DATE to the end of its month. Keep track of this THAT_NUMBER in the variable called "DAY_COUNTER"

there are direct references "THE_EARLIER_DATE" and "THAT_NUMBER", marked by "the"/"that" and a proper noun "DAY_COUNTER". They are identified as program objects.

The third kind of concepts translated from text to program, the operators, are identified either as verbs or as attributes or descriptive expressions. For example, in the sentence

> If the two given dates ARE_IN_THE_SAME_MONTH, THE_NUMBER_OF_DAYS between them is the DIFFERENCE_BETWEEN their DAYS_OF_MONTH,

there is a predicate "ARE_IN_THE_SAME_MONTH" and descriptive expressions "THE_NUMBER_OF_DAYS", "DIFFERENCE_BETWEEN" and "DAYS_OF_-MONTH", which become program operators.

The Abbott's procedure gives some guidelines for translating the specification text into a program, but these guidelines are not automatable. Even given a part-of-speech (POS) tagger, attaching a POS-tag to every word, (not available at the time as Abbott wrote the paper but available now), it would be possible to identify nouns, verbs, etc., but it would still remain impossible for example to differentiate between a common and a proper noun.

Furthermore, as the above examples show, the text representations of concepts are not always disjoint: in the sentence "If the two given dates are in the same month, ..." "month", identified as a data type, is a part of "are in the same month", identified as an operator.

Chen [Che83] goes a similar way as Abbott in the sense that he tries to map natural language texts onto entity-relationship (ER) diagrams. He defines a set of rules translating English text to ER diagrams. The first two rules coincide with the Abbott's ones:

1. A common noun corresponds to an entity type.

2. A transitive verb corresponds to a relationship type.

Further rules are specific to the ER-representation:

3. An adjective in English corresponds to an attribute of an entity in the ER-diagram.

4. An adverb in English corresponds to an attribute of a relationship in an ER-diagram.

8. [3] The objects of algebraic or numeric operations can be considered as attributes.

9. A gerund in English corresponds to a relationship-converted entity type in ER-diagrams.

The remaining rules address firm expression patterns:

5. If the sentence has the form: “There are . . . X in Y”, we can convert it into the equivalent form “Y has . . . X”

6. If the English sentence has the form “The X of Y is Z” and if Z is a proper noun, we may treat X as a relationship between Y and Z. In this case, both Y and Z represent entities.

7. If the English sentence has the form “The X of Y is Z” and if Z is not a proper noun, we may treat X as an attribute of Y. In this case, Y represents an entity (or a group of entities), and Z represents a value.

It is easy to see that the rules 1–4 and 8–9 are very similar to the Abbott’s rules. They just target at another representation form as the Abbott’s rules (ER-diagrams instead of ADA programs). The rules 5–7 create additional relations by analyzing firm expression patterns.

Summarizing the syntactical approaches, it is possible to say that they only give some guidelines for concept identification. They do not classify the extracted concepts. Even when automated (using part-of-speech (POS) tagger) they still require manual post-processing to become applicable: there is no POS tagger able to differentiate between common and proper nouns, and this difference is essential for the existing syntactical approaches. The syntactical approaches do not

[3]Rule numbers are not continuous because the rules are rearranged as compared to Chen’s paper [Che83]

classify the identified concepts either, which makes them inapplicable to ontology building.

The only rule introducing some structure to the extracted concept list are rules 5–7 of Chen's approach, handling firm expression patterns. Practice shows, however, that it is almost impossible to make requirements writer to stick to certain expression patterns, so the rules on the basis of fixed patterns (rules 5–7 of the Chen's approach) are barely applicable.

Saeki et al. [SHE89] designed a tool aiming at automation of the approaches introduced above. They extract nouns and verbs from the text and build a noun table and a verb table. Then they select actions and action relations from the verb table. Although they aim at constructing an object-oriented model from a specification text, they do not perform any concept classification (which would yield a class hierarchy) but produce a flat model.

Alternative Grammar–Based Term Clustering Techniques

Term clustering approach used in this thesis relies on sentence structure to classify the extracted terms. The definition of a cluster, used in the tool ASIUM [FN98], is rather simple: a cluster is built by all the subjects or all the objects of some verb. It is also possible to use other sentence information for the classification purpose. Nenadić et al. [NSA02] introduce following definitions of related terms:

Contextual Similarity of two terms measures the number of common and different contexts for the two terms whose similarity should be determined. For this measure the context is defined as a sequence of particular words with their Part–of–Speech (POS) tags (noun, verb, etc.) occurring in the sentence before and after the term. It is up to the analyst to use all the context words and tags or to define some words or word classes (adjectives, conjunctions, ...) as irrelevant and filter them out. It depends on the text domain which contexts (POS sequences, lexica, etc.) provide better term clustering. For this similarity measure to work, the requirements analyst has to decide which contexts to use. This decision can rely on the quality measure for contexts, also introduced by Nenadić et al. [NSA02].

Lexical Similarity of two terms measures the presence of common lexical heads (e.g., "message" in "start message" and "stop message") and the number of common modifiers. For example, "first start message" and "second start message" are more similar according to this measure than "start message" and "stop message". Lexical heads are provided by the parser, as presented in Section 2.1.2. Thus, lexical similarity can be measured on the basis of parse subtrees for each term, extracted as described in Section 2.1.3.

Syntactical Similarity checks for the presence of certain standard constructions. For example, in the construction "Xs, such as A, B, and C", X, A, B and C are seen as similar. The syntactical similarity measure is discrete: it can be either 0, if terms are not similar, or 1, if terms are similar.

To decide whether two terms are similar, a linear combination of the three above measures is calculated. Terms with high net similarity can be grouped to clusters.

The above definitions of term similarities are implemented in the tool ATRACT [MAN01]. Unfortunately, ATRACT is a result of an industrial project and it is not possible to use this tool for research. Syntactical similarity is also implemented in an open source tool TextToOnto [Tex05]. When augmented by contextual and lexical similarity, TextToOnto could be integrated with the approach presented in this thesis.

Clustering on the Basis of Lists and Tables

The ontology extraction approach, as presented in this thesis, extracts information from grammatically correct sentences. Lists and tables, occurring in requirements documents, often contain just sentence snippets. Although the incomplete sentences can be parsed by Collins' parser, used in this thesis (see also Section 2.1.2), the resulting parse tree cannot be properly analyzed.

To overcome this problem, Wendt [Wen04] developed a heuristic for clustering terms occurring in lists. Wendt defines in his thesis two list types:

Bullet point list may be introduced by a headline, but then it contains solely bullet points, as in the following example, taken from the steam boiler specification [ABL96b]:

> The steam boiler is characterized by the following elements:
>
> - A valve for evacuation of water
> - Its total capacity C (indicated in liters)
> - The minimal limit quantity M1 of water (in liters)
> - ...

Explaining list introduces concepts and at the same time provides definitions of these concepts, as in the following example, also taken from the steam boiler specification.

> The following messages can be received by the program:
>
> - STOP: when the message has been received three times in a row by the program, the program must go into emergency stop.

- STEAM-BOILER-WAITING: when this message is received in initialization mode it triggers the effective start of the program.
- PHYSICAL-UNITS-READY: this message when received in initialization mode acknowledges the message program-ready which has been sent previously by the program.
- ...

The goal of the Wendt's approach is to extract concepts from the lists and cluster them. The terms defined in the same list are put into the same cluster. For example, the first of the above lists would yield the cluster

$$steam\ boiler\ characteristics = \{valve,\ total\ capacity,\ minimal\ limit,\ \ldots\},$$

the second one would result in

$$messages = \{STOP,\ STEAM\text{–}BOILER\text{–}WAITING,\ \ldots\}.$$

The Wendt's list processing tool works in the following way:

1. First of all, list candidates (text passages between two empty lines) are identified.
2. For each list candidate the tool checks whether it is a potential bullet point list or explaining list or not a list at all. The tool user may revise this decision.
3. From explaining lists the terms just before the colon are extracted. For bullet point lists the procedure is a bit more complicated: each list item is parsed and then terms are extracted from parse trees, as described in Section 2.1.3.
4. The set of terms extracted from one list builds a cluster.

A similar clustering algorithm is thinkable for tables as well. Theoretically, the clustering approach for lists and tables could be integrated with the approach presented in the thesis. However, the tool ASIUM [FN98], used for taxonomy building, can solely build clusters based on grammatical contexts, and does not accept other clustering methods.

5.4.3 Interpreting Sentences: Semantical Approaches to Text Analysis

Semantical approaches are the most demanding on the formulation. In return they extract the most information from text. As the name says, these approaches build a semantic representation as analysis results. All these approaches use two kinds of semantic representations: discourse representation structures or mapping of verbs to predicates with their arguments.

Discourse representation structure (DRS) is a kind of first order predicate logic with explicit introduction of variables and definitions of variable scopes and accessibility. An example DRS (taken from [BBKdN98]) is shown in Figure 5.1. This DRS consists of one large box (scope) with two subordinate scope boxes. Each of the subordinate scopes contains some object references and statements about these objects. For example, the left box introduces the object x and states $woman(x)$. The right box introduces a new object y and states $boxer(y)$ and $loves(x, y)$. The whole DRS represents the sentence "Every woman loves a boxer" and is equivalent to the formula

$$\forall x.woman(x) \Rightarrow \exists y.boxer(y) \wedge loves(x, y).$$

(See the technical report by Blackburn et al. [BBKdN98] for the translation rules between DRSs and formulas and for other details.)

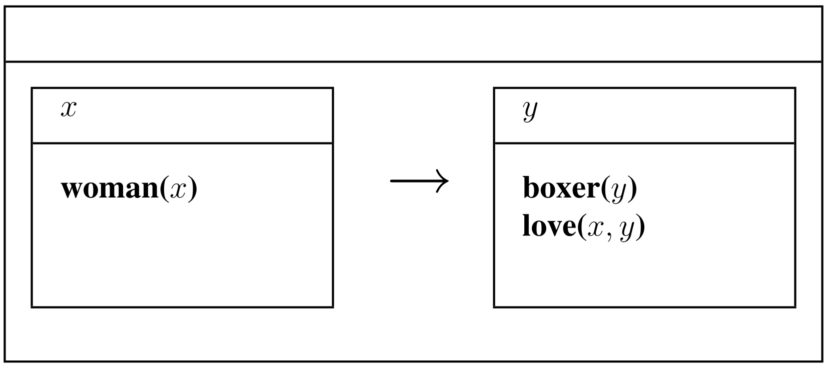


Figure 5.1: Discourse Representation Structure (DRS) for "Every woman loves a boxer"

To compute the semantics-DRS, Blackburn et al. [BBKdN98] define a calculus for such structures. This calculus defines operations on DRSs, like merging, conjunction, negation, and so on. In order to translate a sentence to the representing DRS, a DRS–λ–expression[4] is associated with every word, all the word–$lambda$–expressions are chained to one sentence–λ–expression and then this large λ–expression is evaluated according to the reduction rules of the λ–calculus.

The following example shows semantics calculation with ordinary first order formulae, but a very similar calculation can be done with discourse representation structures. The example uses ordinary first order formulae just not to overcomplicate the matters.

[4]an introduction to λ–calculus can be found, for example, in [Bro98b]

First of all, λ–expressions are introduced for every word class. In the following table and following examples λ–function application is made explicit using the "@" sign to improve readability.

Proper names:	Alice	=	$\lambda P.(P@Alice)$
Common names:	woman	=	$\lambda y.(woman(y))$
Intransitive verbs:	walks	=	$\lambda x.(walk(x))$
Transitive verbs:	loves	=	$\lambda X.(\lambda z.(X@(\lambda x.love(z,\ x))))$
"every":	every	=	$\lambda P.(\lambda Q.(\forall x.((P@x) \rightarrow (Q@x))))$
"a":	a	=	$\lambda P.(\lambda Q.(\exists y.((P@y) \wedge (Q@y))))$

Using this λ–expressions it is possible to calculate the sentence semantics just by replacing every word with its λ–expressions and performing standard reductions defined in the λ–calculus. For example, the semantics of "`Alice loves a man`" is calculated as follows:

$$\begin{aligned}
&Alice\ loves\ a\ man =\\
&= \lambda_{Alice}@(\lambda_{loves}@(\lambda_{a}@(\lambda_{man})))\\
&= \lambda_{Alice}@(\lambda_{loves}@((\lambda P.(\lambda Q.(\exists y.((P@y) \wedge (Q@y)))))@(\lambda y.(man(y)))))\\
&= \lambda_{Alice}@(\lambda_{loves}@(\lambda Q.(\exists y.(((\lambda y.man(y))@y) \wedge (Q@y)))))\\
&= \lambda_{Alice}@(\lambda_{loves}@(\lambda Q.(\exists y.((man(y)) \wedge (Q@y)))))\\
&= \lambda_{Alice}@((\lambda X.(\lambda z.(X@(\lambda x.love(z,\ x)))))@(\lambda Q.(\exists y.((man(y)) \wedge (Q@y)))))\\
&= \lambda_{Alice}@(\lambda z.((\lambda Q.(\exists y.((man(y)) \wedge (Q@y))))@(\lambda x.love(z,x))))\\
&= \lambda_{Alice}@(\lambda z.(\exists y.(man(y) \wedge (\lambda x.love(z,x))@y)))\\
&= \lambda_{Alice}@(\lambda z.(\exists y.(man(y) \wedge love(z,y))))\\
&= (\lambda P.(P@Alice))@(\lambda z.(\exists y.(man(y) \wedge love(z,y))))\\
&= (\lambda z.(\exists y.(man(y) \wedge love(z,y))))@Alice\\
&= \exists y.(man(y) \wedge love(Alice,y))
\end{aligned}$$

As the above example shows, the semantics calculation is quite complicated. Furthermore, introduction of additional words in the sentence would add additional λ–expressions to the computation and would disturb it. This makes approaches of this kind extremely fragile. They are applicable only to restricted specification languages with fixed grammars.

To make this approach applicable to document analysis, it is necessary to restrict the natural language. Fuchs et al. [FSS99], for example, introduced a controlled specification language (ACE, Attempto Controlled English). The language is restricted in the following way:

Vocabulary: The vocabulary of ACE comprises

- predefined function words (e.g. determiners, conjunctions, prepositions)
- user-defined, domain-specific content words (nouns, verbs, adjectives, adverbs)

Sentences: There are

- simple sentences,
- composite sentences,
- query sentences.

Simple sentences have the form

$$subject + verb + complements + adjuncts$$

Firm sentence structure and the necessity to explicitly define the vocabulary in advance restrict the applicability of ACE and other λ–calculus based approaches to real requirements documents.

The other group of semantical approaches uses verb subcategorization frames for semantics representation. A verb subcategorization frame is a verb with its arguments (subject and objects). For example, for the verb "`send`", possible arguments are: sender, receiver, sent object. When interpreting the sentence "`Component X sends message Y to component Z`", in the semantical representation "`component X`" becomes the sender, "`component Z`" the receiver and "`Component X sends message Y to component Z`" the sent object.

This idea is used by Hoppenbrouwers et al. [HvdVH97] to identify domain concepts and relations between them. Hoppenbrouwers et al. define a set of roles (semantical tags) like *agent*, *action*, *patient* etc. The analyst marks the relevant words with these tags. For example, the sentence "`Component X sends message Y to component Z`" can be (manually) tagged as

$$(\texttt{Component X})/agent\ \texttt{sends}/action\ (\texttt{message Y})/patient\ \texttt{to}\ (\texttt{component Z})/other.$$

Sentences marked in such a way are used to find *agents*, *actions*, and *patients*.

Ambriola and Gervasi [AG99] go further than Hoppenbrouwers et al. and build a semantic tree representation of a sentence. To build the semantic representation, they start with a list of terms, each term furnished with an associated list of tags. These tags are used to mark every word of a sentence. For example, the sentence

The terminal sends the password to the server is canonized as

`terminal`$/IN/OUT$ `sends` `password`$/INF$ `to` `server`$/IN/OUT/ELAB$

The applied tags are domain-specific.

After the tagging, a set of transformation rules is applied to marked sentences, translating the tagged sentence to a semantic tree. Figure 5.2 (taken from [AG99]) shows an example semantic tree. It shows the representation of the sentence

> When the server receives from the terminal the password, the server stores the signature of the password in the system log.

This tree shows dependencies between actions (left subtree depends on the right one) and the semantics of every action. This rich representation allow for extraction of abstract state machines, entity-relationship diagrams and other formalisms [AG03, Ger01].

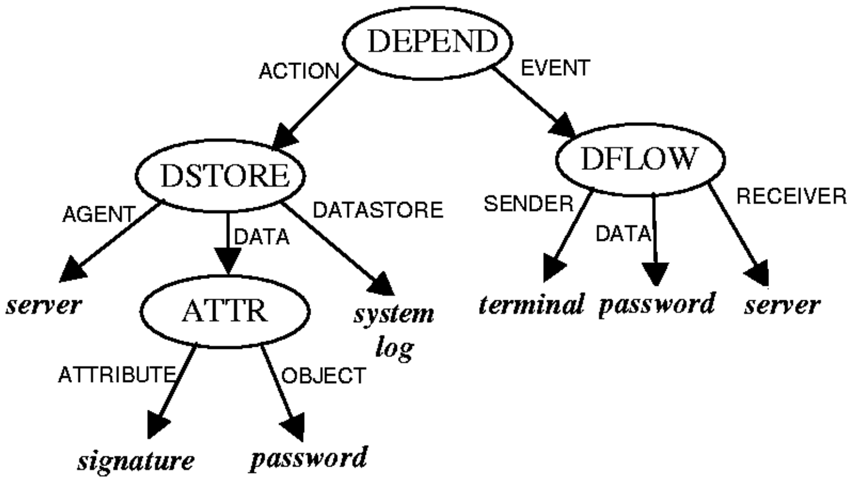


Figure 5.2: Semantic tree according to Ambriola and Gervasi [AG99]

The drawback of this approach is obvious: the approach is able to analyze only sentences that fit into the predefined templates (transformation rules). The templates are defined manually and it is almost impossible to cater for all the potential constructions that can occur in a real requirements document.

Colette Rolland and Camille Ben Achour [RB98] apply the idea of case frames, which is very similar to the approach by Ambriola and Gervasi, introduced above, to whole sentence sequences to build a semantics of a use case description. As in the case of previously mentioned approaches, only firm expression patterns are supported. They also define a set of expressions for temporal relations between individual sentences.

Although interesting in itself, semantics representation is not necessarily the final goal of document analysis. Vadeira and Meziane [VM94] use semantical text analysis and formulae representation to produce a VDM model. They start with a set of logical formulae and translate them to an entity/relationship model first. To build the entity/relationship model, they assume that the predicates that build up the formulae are the relationships and predicate arguments are the entities. Then they use a set of heuristics to determine multiplicity of the relations in the basis of formulae. The final step in their approach is the translation of the E/R-diagram to the formal specification language VDM.[5]

The drawback of this approach (and, actually, most semantical approaches) is an inherent contradiction: to translate sentences to formulae, it is necessary to explicitly introduce the vocabulary. But, when the set of domain concepts is known, it is possible to build a domain model without complex semantical analysis.

Although the idea of semantics analysis is very promising for the step from a requirements document to a system model, the approaches are not really mature yet. They are applicable solely to sentences with restricted grammar. What is missing is a semantical broad-domain parser, putting no restrictions on allowed expression forms and able to cope with sentences that are not completely grammatically correct.

5.4.4 Related Text Analysis Approaches, Summary

Requirements engineering is a non-trivial task, crucial for the whole project. The wish to extract as much information as possible from requirements documents initiated a lot of work on text analysis.

The whole body of text analysis work can be subdivided into three large classes: lexical, syntactical and semantical. The most promising approaches are the semantical ones, as they produce a formal representation of every sentence. Unfortunately, they all rely either on fixed grammar or on predefined expression patterns, which strongly limits their applicability to real life requirements documents, mostly poorly written. Moreover, some of the semantical approaches require explicit glossary definitions. Although the idea of a glossary is sensible in itself, such a glossary is seldom provided with a requirements document.

Syntactical and lexical approaches are less demanding to text quality and more robust. Existing syntactical approaches extract concepts from text and map them onto a predefined meta-model (either entity-relationship or ADA programming model). However, extraction going beyond pure term identification still relies on firm expression patterns. The lexical approaches are the most robust ones, as they do not rely on any expression patterns, but they are limited to pure term extraction.

[5]An introduction to VDM can be found, for example, in [Jon90]

They do not extract any relations between terms. The bottom line of this overview is obvious: there is no ontology extraction approach, based on text analysis and capable to deal with poorly written requirements documents.

5.5 Presented Thesis vs. Related Approaches

The plethora of the existing work on requirements engineering gives rise to the legitimate question about the novelty of the presented thesis. Related work on quality assurance of requirements documents, as presented in Section 5.2, is not quite comparable to the presented thesis: these approaches cater for document quality by defining quality features necessary for each requirement. They do not aim at terminology or ontology extraction.

Ontology construction methodologies, presented in Section 5.1, are better comparable to the presented thesis, as they pursue the same goal. However, the generic methodologies, as those listed in Section 5.1, just introduce the abstract ontology construction steps (like identification of information sources, term extraction, term classification, etc.), without any guidelines how to perform these steps. For this reason it makes little sense to compare the presented thesis with these ontology construction methodologies.

The existing approaches for detection of ambiguous requirements, introduced in Section 5.3, are barely comparable to the presented thesis as well: they focus on detection of semantical ambiguities (sentences with several interpretations), but they assume that the terminology is consistent throughout the requirements document. Inconsistency detection, as pursued in the presented thesis, addresses solely terminology inconsistence.

The best comparable approach class are text analysis approaches, whose overview is given in Section 5.4. In this overview all existing approaches were found insufficient for different reasons, which implies the question about the desired features of the feasible text analysis approach. The following requirements are quite intuitive for text analysis to become applicable to requirements documents:

1. The approach should not rely on any firm expression patterns. This is necessary due to extremely poor quality of real life requirements documents and practical impossibility to enforce any writing style.

2. The approach should extract not only terms relevant for the application domain, but also relations between these terms (i.e., ontology extraction instead of glossary extraction). Furthermore, when extracting terminology, the approach should extract not only single-word, but also compound terms.

3. The approach should be interactive and not completely automatic. This is necessary to detect inconsistencies in the analyzed document. As practice shows, inconsistencies are inevitable in requirements documents, which makes a completely automatic approach unfeasible. As Aussenac-Gilles [AG05] and Goldin and Berry [GB97] state, a completely automated technique is not desirable as it potentially results in wrong extraction or information loss.

4. The approach should not rely on any previous domain knowledge. It is mostly the case in requirements engineering, that at project beginning software engineers have only superficial knowledge about the application domain, which causes difficulties in understanding the customer.

The most important requirement is the independence from a particular writing style. For example, building the taxonomy (term hierarchy) solely on the basis of expression patterns like "every X is a Y" is insufficient. For this reason (restriction of allowed grammar) the semantical techniques (see Section 5.4.3) are barely applicable in practice. Syntactical approaches, presented in Section 5.4.2 satisfy the requirement independence from writing style to a certain part: they do not rely on fixed expression patterns for term identification. However, for the advanced step, namely for establishing relations between requirements, they do look for patterns like "every X is a Y". Lexical approaches, introduced in Section 5.4.1, are completely free from this drawback, but they provide sole term extraction but no ontology extraction. Some of the syntactical approaches perform term clustering and can be potentially integrated with the approach presented in this thesis.

The approach presented in this thesis was designed to satisfy the above requirements. First of all, it does not rely on fixed expression patterns: neither for term extraction, nor for term classification, nor for extraction of general relations. Term identification works on the basis of parse trees that are provided by a general purpose broad domain parser. Term classification relies solely on grammatical contexts the terms are used in. (Terms are considered as related if they are used as subjects or object of the same verb.) The detection of terminology inconsistencies works on the basis of term clusters as well. For relation extraction the approach relies solely on co-occurrence of the terms in the same sentence. These features make the approach presented in this thesis robust and thus applicable to real world requirements documents. It goes far beyond other existing approaches in satisfying the above requirements.

Chapter 6

Conclusions

Requirements engineering is a non-trivial task and the presented thesis does not claim to solve all its problems. However, the presented approach solves one of the most acute requirements engineering problems, namely establishing and validating a common language for the project stakeholders. This common language, a domain ontology, is extracted from requirements documents. The extraction is interactive, which results in detecting terminology inconsistencies. As practice shows, inconsistencies are widespread in requirements documents, which makes inconsistency detection itself an important task.

The proposed process of ontology extraction and inconsistency elimination is iterative. For this reason, integration of ontology extraction into the requirements engineering process requires some changes to the standard process. The standard requirements engineering process by Robertson and Robertson [RR99] (see also Figure 1.1, page 19) has two serious flaws preventing from direct integration of ontology extraction into requirements engineering:

- The process is linear and not iterative.
- Quality assurance and terminology unification is performed at the level of single requirements, not at the document level.

To overcome these difficulties, the presented thesis proposes following changes to the requirements engineering process (see also Figure 3.2, page 67):

- Ontology extraction becomes a part of one of the requirements engineering steps ("take stock of the specification" in terms of the standard process).
- Feedback loop necessary to eliminate inconsistency becomes a feedback loop in the requirements engineering process.
- The results of this iterative process are both an application domain ontology *and* a purified requirements document. This implies quality assurance not

only at the level of single requirements, but also at the level of the whole document.

The remainder of this chapter summarizes more technical aspects of the presented thesis. Section 6.1 presents the main ideas of ontology extraction, Section 6.2 discusses the case studies, the obtained results, and the identified limitations of the approach, Section 6.3 sketches the possibilities of further improvements and Section 6.4 shows how ontology extraction can be deployed in the enterprise context.

6.1 Ontology Extraction Summary

Ontology extraction is the technical core of the presented thesis. The following table summarizes the ontology extraction steps and shows which steps are performed completely automatically and which ones require human interaction.

1.	Format the text (one sentence per line)	partially automatic
2.	Tag each word (Part-of-Speech)	automatic
3.	Parse the tagged text	automatic
4.	Extract predicates and their arguments	automatic
5.	Build concept clusters	automatic
6.	Look for cluster intersections and build a taxonomy	**interactive**
7.	Look for potential associations, generalize them	automatic
8.	Decide which associations are sensible	**interactive**

These steps correspond to the principal approach, they do not show detection and correction of inconsistencies. Inconsistencies are detected in interactive steps: concept clustering (Step 6) and decision about sensible associations (Step 8). After the correction of inconsistencies (paraphrasing) it is necessary to restart with the tagging (Step 2).

It is easy to see that one step is marked as partially automatic, while others are interactive. The difference is fundamental: the partially automatic step is not completely automatic yet because of some technical problems: there are problems with formatting incomplete or grammatically incorrect sentences that are often present as bullet points in specification texts.

For the steps that are marked as interactive, complete automation is not desirable. As Goldin and Berry state [GB97], complete automation is not desirable if it could lead to information loss or wrong results. In the case of taxonomy building (Step 6) and association ascription (Step 8) inconsistencies can be found. They often manifest themselves in senseless term clusters or senseless associations. It is impossible for an automatic tool to decide which clusters/associations are sensible. Even after elimination of inconsistencies not every cluster intersection leads to a *sensible* larger cluster defining a more general concept and not every potential association is a sensible one. Thus, even for a perfectly consistent text a completely automatic tool would not be feasible. This tool interactivity achieves one of the most important goals of document analysis and validation: detection of terminology inconsistencies.

The above steps result in a revised requirements document and a domain ontology. To become really applicable in further development process, the ontology itself should be validated. This can be done either via manual examination of the ontology by a domain expert or via translation of the ontology into a domain-specific model and validation of the model. For distributed systems this thesis proposes a heuristic for translation of an ontology into an AutoFOCUS model.[1] This translation is interactive, which gives the analyst feedback about inconsistencies in the ontology itself. When the resulting model is successfully generated and validated, this also implies validation of the underlying ontology.

To summarize, although the approach presented in this thesis *does require* manual intervention, this cannot be seen as its weakness: manual intervention results in better document and ontology validation, which is itself as important as ontology extraction.

6.2 Discussion

The idea to establish a common language for all the project stakeholders is certainly sensible. Establishing a common language improves clarity of the communication and has only advantages for all the project participants. The presented thesis proposes a domain ontology as such a common language. However, the most crucial question in this context is whether it is realizable to extract an ontology from requirements documents with justifiable time cost.

It is the goal of the first case study to test the actual applicability of the ontology extraction approach and to investigate the prerequisites for the application. To evaluate the applicability of the method, the first case study was chosen relatively small. The advantage of the small case study was the understandability of the requirements document. Furthermore, the document used for the first case study was initially written for a formal methods contest and was extremely precise compared to real world documents.

The first case study showed that the proposed ontology extraction approach works and that even a seemingly precise document can still contain a lot of terminology inconsistencies. It is to expect that real world requirements documents, not designed for a formal method contest, are even less consistent. The first case study

[1]AutoFOCUS [AF-04] is a tool for modelling of distributed systems.

resulted in a set of writing rules, helping to write consistent documents. However, the question how time-consuming the rule enforcement is, remained open.

The goal of the second case study is to measure the time necessary to analyze a large document and simultaneously to enforce the writing rules. The document for the second case study is about 80 pages long, which is comparable to the size of industrial specifications. In this case study the time spent on different analysis stages was measured. The result of the measurement is that all the analysis stages took approximately the same time (see Table 6.1).

Analysis operation	**Time cost** (working days)
Skimming the text (taken as a comparison for other time measurements)	1
Elimination of terminology inconsistencies, detected in first analysis iteration	1.5
Taxonomy building	1.5
Association mining	1
Overall analysis time	5

Table 6.1: Time cost for an 80 pages document

The total time cost of document analysis amounted to 5 working days. A sensible benchmark for this time cost is the work necessary to write the analyzed document. According to one of the participants of the Empress project[2], where the specification was written, the effort necessary to produce the analyzed requirements document amounted to 4 man-months. Compared to the work necessary to produce the specification, the time cost of 5 man-days to analyze the document, to eliminate inconsistencies, and to produce an ontology, is negligible. The conclusion of the second case study is that the time cost of the text analysis approach proposed in this thesis is affordable even for larger documents. Finally, the third case study showed that the approach is applicable to industrial documents as well.

Limitations of the Text Analysis Approach

The case studies showed not only applicability of the proposed ontology extraction method, but also its limitations. Some of the identified limitations are due to deficiencies of the available tools, some of them are fundamental problems.

[2]`http://www.empress-itea.org/`

The most severe restriction is due to the fact that the analysis considers each sentence separately. For example, in the following part of the steam boiler specification, it is necessary to know that each sentence refers to the initialization mode:

> Initialization mode
> The initialization mode is the mode to start with.
> The program enters a state in which it waits for the message steam-boiler-waiting to come from the physical units.
> As soon as this message has been received the program checks whether the quantity of steam coming out of the steam-boiler is really zero.
> If the unit for detection of the level of steam is defective–that is, when v is not equal to zero–the program enters the emergency stop mode.
> ...

Understanding of such context information is far beyond the capabilities of the existing text analysis tools. A possible way to reach a kind of context awareness could be the idea of context stacks by Grosz and Sidner [GS86], but this idea is not implemented yet.

The other major limitation of the text analysis approach is due to the lack of tool integration: it is not possible yet to resolve cross-sentence references like "Message X is sent by unit Y. This message indicates ...". Integration of an anaphora resolution tool could help to solve this problem. See Section 6.3.2 for a detailed description of possible integration.

6.3 Outlook

6.3.1 Application of the Extraction Technique to German

The case studies presented in this thesis all analyzed English documents. The wish to apply the same technique to other languages is evident. The whole language dependency is capsuled in the early phases of term extraction, namely in the part-of-speech-tagging and parsing (see also Sections 2.1.1 and 2.1.2). Thus, to apply the extraction technique to German, it should be sufficient to replace the English parser by the German one. For this purpose a number of parsing experiments with the LoPar-parser [Sch00] were conducted. The results were rather disappointing: the available German parser is not good enough to be applied to requirements documents. For example, the parser failed to parse following grammatically correct sentences, taken from the requirements document for one of our former student projects [GKP+02].

> Das Tool APE stellt eine methodische Unterstützung für Softwareentwicklungsprojekte zur Verfügung - das Anwendungsspektrum von APE ist also sehr weit gefasst.
>
> Das Vorhandensein einer Vorgehensmodell- Wissensbasis ist natürlich Voraussetzung für die Anwendung von APE in einem Unternehmen.
>
> Diese Wissenbasis kann allerdings durchaus in dem Sinn partiell sein, dass es nicht zu jedem Dokumenttyp eine Vorgehensweise gibt.
>
> Im Rahmen der Einführung von APE wird sich die Wissenbasis erst nach und nach entwickeln.

In general, the parser failed to parse approximately 30 to 40 percent of the grammatically correct sentences from the student project document, which is definitely too much for a sensible application in requirements documents analysis. For this reason no further experiments with German texts were performed. However, with an improvement of the German parser the application of text analysis to German may become interesting again.

6.3.2 Potential Improvement of the Extraction Technique

The ontology extraction techniques presented in this thesis produced good results on several case studies. However, there is still room for improvement. There are four major potential improvement directions:

1. The ontology extraction method works at the moment for grammatically correct sentences only. However, grammatically incorrect (incomplete) sentences often occur in requirements documents in the form of lists or tables. Surely, it would be advantageous to extract information from lists and tables as well. The Wendt's approach to list analysis [Wen04] can be applied for this purpose (see also Section 5.4.2).

2. The presented approach extracts all the terms occurring in grammatically correct sentences. It can be augmented by AbstFinder [GB97], a tool extracting all the terms occurring at least twice in the document, not necessarily in grammatically correct context (see also Section 5.4.1).

3. Term extraction analyzes at the moment each sentence separately. It cannot resolve cross-sentence references like "Message X is sent when ... This message implies ..." This problem could be solved by the means of pronominal anaphora resolution, introduced below.

4. The method of term clustering could be improved as well: in the presented approach the terms used as subjects or objects of the same verb are put in one cluster. However, other clustering principles are thinkable, as introduced by Nenadić et al. [NSA02] (see also Section 5.4.2).

The remainder of this section presents pronominal anaphora resolution in more detail, and then an integrated ontology extraction approach, incorporating the approach presented in this thesis, anaphora resolution, and some approaches presented in Section 5.4.

Pronominal Anaphora Resolution

The term extraction method introduced in this thesis assumes that the terms are explicitly present in the text. However, the usage of pronouns is frequent, which undermines this assumption. For example, in the following two sentences, taken from the steam boiler specification, the second sentence does not name the received message explicitly:

> The program enters a state in which it waits for the message steam-boiler-waiting to come from the physical units. As soon as this message has been received the program checks whether the quantity of steam coming out of the steam-boiler is really zero.

Usage of pronouns poses problems to the term extraction approach based on parse trees, introduced in Section 2.1.3: it would extract just "this message" as a subject of "received" from the second sentence.

This problem can be solved by the means of anaphora resolution [Pre02]. Resolution of pronominal anaphora would identify "`this message`" in the second sentence with "`message steam-boiler-waiting`" in the first one. An additional advantage of applying anaphora resolution would be detection of referential ambiguities. A referential ambiguity, according to the definition by Kamsties et al. [KBP01] "is caused by an anaphora in a requirement that refers to more than one element introduced earlier in the sentence or in a sentence before". For example, in the sentences

> The controller sends a message to the pump.
> It acknowledges correct initialization.

"it" can refer both to the pump and to the controller and to the message. Explicit anaphora resolution would disambiguate this reference. In the case of wrong resolution it would make the referential ambiguity visible.

Anaphora resolution, as presented by Preiss [Pre02], depends on the extraction of grammatical roles. The term extraction algorithm, presented in Section 2.1.3,

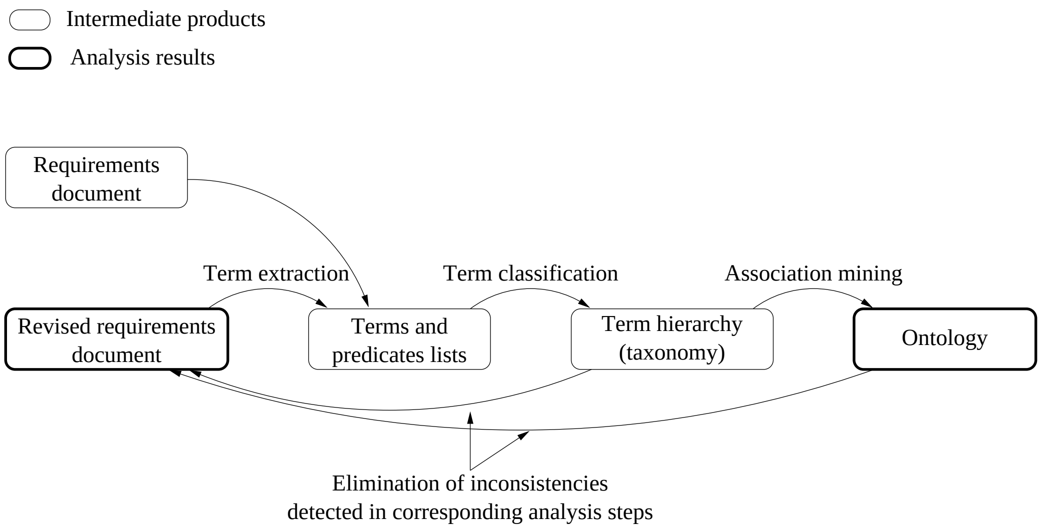

Intermediate products
Analysis results
Requirements document
Term extraction
Term classification
Association mining
Revised requirements document
Terms and predicates lists
Term hierarchy (taxonomy)
Ontology
Elimination of inconsistencies detected in corresponding analysis steps

Figure 6.1: Ontology Building Procedure, as presented in the thesis

extracts subjects and objects from each sentence, so it can be used as a preprocessor for anaphora resolution.

Integrated Ontology Extraction Approach

The discussion at the beginning of this section (page 142) shows that there are several methods potentially able to improve the original ontology extraction approach introduced in this thesis. The following sketch shows how these approaches can be integrated. The goal of the integration is to join the strengths and to hide the weaknesses of the isolated methods.

Figure 6.2 shows an overview of the proposed integrated approach. Just as the original approach shown in Figure 6.1, it starts with the specification text, written in natural language, and extracts an ontology. However, it consists of much more steps and extracts more information from the text. The remainder of this section presents each step in detail.

Parsing and anaphora resolution: The goal of this first step is to get rid of pronominal cross–sentence references. Anaphora resolution is necessary for the later steps, because it replaces pronouns by full-fledged terms, so that these terms instead of pronouns can be extracted.

The results of anaphora resolution should be examined by the domain expert. It is possible that some anaphora be resolved incorrectly, either due to referential ambiguity (several possibilities to resolve an anaphora) or due

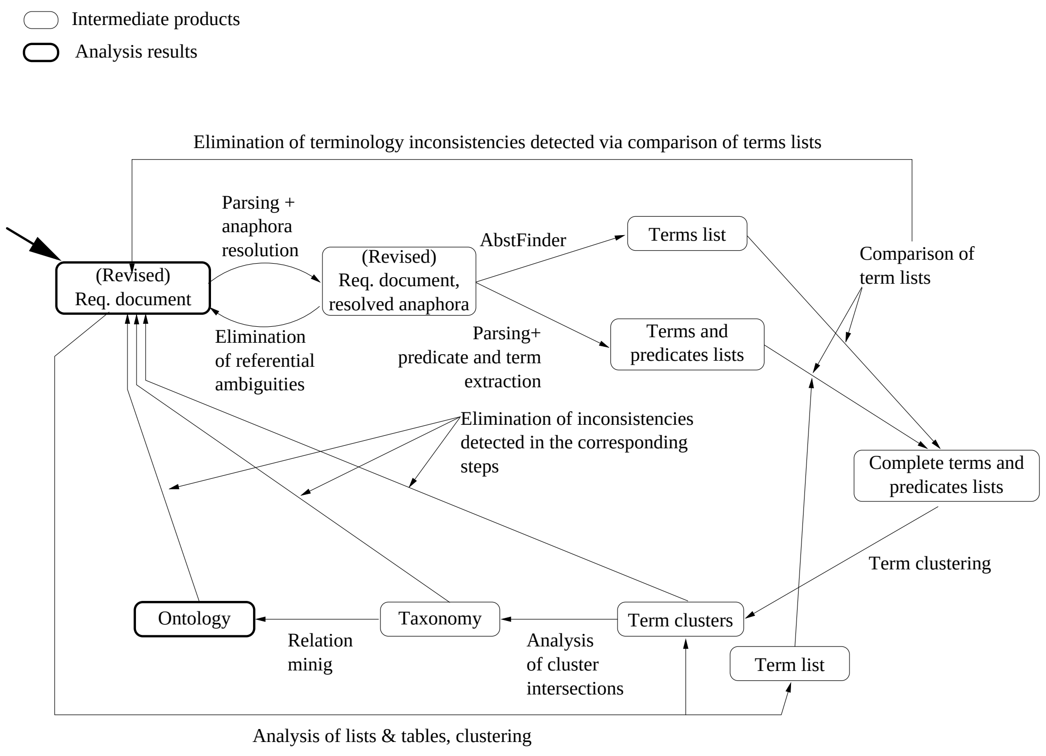

Intermediate products
Analysis results
Elimination of terminology inconsistencies detected via comparison of terms lists
Parsing + anaphora resolution
(Revised) Req. document
(Revised) Req. document, resolved anaphora
AbstFinder
Terms list
Comparison of term lists
Elimination of referential ambiguities
Parsing+ predicate and term extraction
Terms and predicates lists
Elimination of inconsistencies detected in the corresponding steps
Complete terms and predicates lists
Term clustering
Ontology
Taxonomy
Term clusters
Relation minig
Analysis of cluster intersections
Term list
Analysis of lists & tables, clustering

Figure 6.2: Integrated Ontology Extraction Approach

to deficiencies of the used parser or resolution algorithm. The manual revision of the resolution results would make sure that the terms are substituted correctly.

AbstFinder: AbstFinder considers the sentences just as character sequences and extracts character sequences that are common for at least two sentences. The requirements analyst may decide which sequences really represent a term. The extraction of *common* character sequences explains the necessity for anaphora resolution: the resolved pronouns become sensible character sequences.

Parsing, predicate and term extraction: The text with resolved anaphora is parsed anew. (Actually, it is necessary to parse only sentences that were changed because of anaphora resolution.) Then, the predicates and terms can be extracted using the technique described in Section 2.1.3. Verbs can be used later to cluster terms.

Analysis of lists and tables: The approach to list analysis, presented in Section 5.4.2, produces term clusters directly from lists. A similar approach can be used to produce term clusters directly from tables. As a side-effect, the approach produces a term list. This term list should be compared to the term lists extracted in previous steps.

Comparison of term lists: Neither AbstFinder nor the extraction of terms from parse trees nor the extraction of terms from lists and tables can *guarantee* the extraction of *all* the terms. AbstFinder extracts terms that occur at least twice in the text, and extraction of terms from parse trees works for grammatically correct sentences only.

Comparison of the extracted term lists can give important information about the text: It shows which terms are often (at least twice) used in the text, but solely in grammatically incorrect sentences. It shows also which terms are used just once and which terms are used solely in incomplete sentences (lists and tables). If this comparison of term lists discovers some omissions in the document, it is up to the requirements analyst to change the text to correct the flaws.

Term clustering: To build a taxonomy, it is necessary to find related terms first. Possible clustering criteria were introduced in Section 5.4.2: contextual, lexical and syntactic term similarity. The weights of each of the similarity measures can vary depending on the analyzed text. The clusters produced by the analysis of lists and tables, although composed according to other criteria, can be used in further analysis just in the same way as the other clusters, based on similarity measures.

The produced term clusters should be examined by the requirements analyst. Unrelated terms put in the same cluster usually signalize either a terminology inconsistency or inaccurate phrasing somewhere in the text. The sentences using inconsistent terminology can be found by simple text search: for the inconsistent cluster it is known which terms, used in which context, caused the cluster inconsistency. Therefore, it is sufficient to look for the sentences containing the term in the corresponding context. The detected inconsistencies should be corrected before the analysis continues.

Taxonomy building: To build a taxonomy, it is necessary to determine, which clusters are related. This can be done for example by analysis of cluster intersections and joining them to larger clusters, representing more general concepts. This step is the same as in the original ontology extraction approach presented in this thesis.

Relation mining: In the last step the taxonomy is augmented by more general relations. This step is exactly the same as in the original ontology extraction approach presented in this thesis. There is a potential association between two concepts if they occur in the same sentence. Each potential association again has to be validated by the requirements engineer. The validated associations are absorbed into the ontology.

The approach proposed here requires manual intervention, just as the original method presented in the thesis. However, manual intervention is necessary to detect inconsistencies. Thus, interactivity is not a weakness but an important feature of the proposed approach. As in the original approach, presented in this thesis, the extracted ontology should be validated. The result of the whole procedure is a validated application domain ontology *and* a corrected textual specification, free from terminology inconsistencies. The corrected textual specification is itself as important as ontology extraction.

6.4 Perspective: Enterprise Ontology

The approach presented in this thesis aims at ontology construction for a single project. This is surely advantageous in the project context. However, even more can be achieved by establishment of such a common ontology at the enterprise level.

The idea to establish a global ontology is not new: for example, the WordNet project [MFT$^+$05] introduces a lexical database for the English language. In this database the common term meanings are defined. Additionally to plain text definitions WordNet provides links between the terms. The overall ontology structure is the same as in the presented thesis: the ontology consists of a taxonomy (term hierarchy) and general associations.

Surely such a global ontology is too large and too abstract for usage in a software project. Furthermore, specific terms can have other meaning in certain domains. For example the meaning of “class” in object-oriented programming is quite different from the definitions given in WordNet. This gap between the available large abstract ontology and a project ontology can be bridged by successive concretion of the abstract ontology:

- The first possible concretion level is a general domain ontology, restraining the general ontology to a particular domain. For example, WordNet contains definitions of “hardware” both in the sense “computer hardware” and in the sense “ironware”. A restriction of WordNet to computer science would most probably neglect meanings other than “computer hardware”.
- Such a domain ontology is still too abstract to be used by a single enterprise. For example, two banks could have different notions about the concept of

an “account”. Thus, the enterprise should refine a general domain ontology to tailor it to own needs.

- The most fine grained ontology, namely the project ontology, would not be built from scratch any more, but as a concretion of the enterprise ontology. Definitions for project-specific terms would be necessary only for terms not yet defined in the enterprise ontology.

Surely, construction of each successive ontology level requires some effort, but the payoff would be a common language available not just in a single project but also at the enterprise level.

Bibliography

[Abb83] Russell J. Abbott. Program design by informal English descriptions. *Communications of the ACM*, 26(11):882–894, 1983.

[ABL96a] Jean-Raymond Abrial, Egon Börger, and Hans Langmaack. *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control*, volume 1165 of *LNCS*. Springer–Verlag, 1996.

[ABL96b] Jean-Raymond Abrial, Egon Börger, and Hans Langmaack. The steam boiler case study: Competition of formal program specification and development methods. In J.-R. Abrial, E. Borger, and H. Langmaack, editors, *Formal Methods for Industrial Applications*, volume 1165 of *LNCS*. Springer–Verlag, 1996. `http://www.informatik.uni-kiel.de/~procos/dag9523/dag9523.html`, accessed 01.05.2005.

[AF-04] The AutoFocus Homepage, 2004. `http://autofocus.in.tum.de/index-e.html`, accessed 21.02.2004.

[AG99] Vincenzo Ambriola and Vincenzo Gervasi. Experiences with domain-based parsing of natural language requirements. In G. Fliedl and H. C. Mayr, editors, *Proc. of the 4th International Conference on Applications of Natural Language to Information Systems*, number 129 in OCG Schriftenreihe (Lecture Notes), pages 145–148, June 1999.

[AG03] Vincenzo Ambriola and Vincenzo Gervasi. The Circe approach to the systematic analysis of NL requirements. Technical Report TR-03-05, University of Pisa, Dipartimento di Informatica, March 2003.

[AG05] Nathalie Aussenac-Gilles. Supervised Learning for Ontology and Terminology Engineering. In Nicholas Kushmerick, Fabio Ciravegna, AnHai Doan, and Craig Knoblock, editors, *Machine*

Learning for the Semantic Web, Dagstuhl seminar, Dagstuhl (Germany), 13-18 February 2005.

[BBKdN98] Patrick Blackburn, Johan Bos, Michael Kohlhase, and Hans de Nivelle. Inference and computational semantics. CLAUS-Report 106, Universität des Saarlandes, Saarbrücken, November 1998.

[Ben97] Camille Ben Achour. Linguistic instruments for the integration of scenarios in requirement engineering. In Philip R. Cohen and Wolfgang Wahlster, editors, *Proceedings of the Third International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ'97)*, Barcelona, Catalonia, June 16-17 1997.

[Ber01] Daniel Berry. Natural language and requirements engineering - nu?, 2001. `http://www.ifi.unizh.ch/groups/req/IWRE/papers&presentations/Berry.pdf`, accessed 09.01.2003.

[Ber03] Daniel Berry. Natural Language in Requirements Engineering, 2003. `http://se.uwaterloo.ca/~dberry/natural.language.html`, accessed 11.06.2005.

[BFKM95] Ann Bies, Mark Ferguson, Karen Katz, and Robert MacIntyre. Bracketing guidelines for treebank ii style penn treebank project, 1995. `http://www.cis.upenn.edu/~treebank/home.html`, accessed 01.05.2005.

[BHH$^+$04] K. Buhr, N. Heumesser, F. Houdek, H. Omasreiter, F. Rothermehl, R. Tavakoli, and T. Zink. DaimlerChrysler demonstrator: System specification instrument cluster, 2004. `http://www.empress-itea.org/deliverables/D5.1_Appendix_B_v1.0_Public_Version.pdf`, accessed 01.05.2005.

[BL98] Tim Berners-Lee. Semantic Web Road map, 1998. `http://www.w3.org/DesignIssues/Semantic.html`, accessed 11.06.2005.

[BLHL01] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web, 2001. `http://www.scientificamerican.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21&catID=2`, accessed 11.06.2005.

[Bro98a] Manfred Broy. *Informatik. Eine grundlegende Einführung*, volume 2. Springer–Verlag, 1998.

[Bro98b] Manfred Broy. *Informatik. Eine grundlegende Einführung*, volume 1. Springer–Verlag, 1998.

[BS03] Karin Koogan Breitman and Julio Cesar Sampaio do Prado Leite. Ontology as a requirements engineering product. In *Proceedings of the 11th IEEE International Requirements Engineering Conference*, pages 309–319. IEEE Computer Society Press, 2003.

[Che83] Peter Chen. English sentence structure and entity-relationship diagram. *Information Sciences*, 1(1):127–149, May 1983.

[Col97] Michael Collins. Three generative, lexicalized models for statistical parsing. In Philip R. Cohen and Wolfgang Wahlster, editors, *Proceedings of the Thirty-Fifth Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics*, pages 16–23, Somerset, New Jersey, 1997. Association for Computational Linguistics.

[Col99] Michael Collins. *Head-Driven Statistical Models for Natural Language Parsing*. PhD thesis, University of Pennsylvania, 1999.

[FFGL01] F. Fabbrini, M. Fusani, S. Gnesi, and G. Lami. The linguistic approach to the natural language requirements quality: benefit of the use of an automatic tool. In *26th Annual NASA Goddard Software Engineering Workshop*, pages 97–105, Greenbelt, Maryland, 2001. IEEE Computer Society.

[FN98] David Faure and Claire Nédellec. ASIUM: Learning subcategorization frames and restrictions of selection. In Yves Kodratoff, editor, *10th European Conference on Machine Learning (ECML 98) – Workshop on Text Mining*, Chemnitz Germany, April 1998 1998.

[FSS99] Norbert E. Fuchs, Uta Schwertel, and Rolf Schwitter. Attempto Controlled English (ACE) language manual, version 3.0. Technical Report 99.03, Department of Computer Science, University of Zurich, August 1999. `http://www.ifi.unizh.ch/attempto/publications/papers/ace3_manual.pdf`, accessed 21.05.2004.

[GB97] Leah Goldin and Daniel M. Berry. AbstFinder, a prototype natural language text abstraction finder for use in requirements elicitation. *Automated Software Eng.*, 4(4):375–412, 1997.

[GBB02] Paul Grünbacher, Barry W. Boehm, and Robert O. Briggs. EasyWinWin: A groupware-supported methodology for requirements negotiation, 2002. `http://sunset.usc.edu/research/WINWIN/EasyWinWin/index.html`, accessed 04.12.2004.

[Ger01] Vincenzo Gervasi. Synthesizing ASMs from natural language requirements. In *Proc. of the 8th EUROCAST Workshop on Abstract State Machines*, pages 212–215, February 2001.

[GKP$^+$02] Michael Gnatz, Leonid Kof, Franz Prilmeier, Andreas Rausch, and Tilman Seifert. APE, A Project Support Tool based on Process Patterns, 2002. `http://www4.in.tum.de/~ape/Dokumente/Lastenheft.pdf`, accessed 11.06.2005.

[Gru93] Thomas. R. Gruber. Towards Principles for the Design of Ontologies Used for Knowledge Sharing. In N. Guarino and R. Poli, editors, *Formal Ontology in Conceptual Analysis and Knowledge Representation*, Deventer, The Netherlands, 1993. Kluwer Academic Publishers.

[GS86] Barbara Grosz and Candace Sidner. Attention, intention and the structure of discourse. *Computational Linguistics*, 12(3):175–204, 1986.

[HSE97] Franz Huber, Bernhard Schätz, and Geralf Einert. Consistent graphical specification of distributed systems. In John Fitzgerald, Cliff B. Jones, and Peter Lucas, editors, *4th International Symposium of Formal Methods Europe*, volume 1313 of *LNCS*, pages 122–141. Springer, 1997.

[HvdVH97] J. Hoppenbrouwers, B. van der Vos, and S. Hoppenbrouwers. NL structures and conceptual modelling: grammalizing for KISS. *Data Knowl. Eng.*, 23(1):79–92, 1997.

[IEE05] What Is Software Engineering?, 2005. `http://www.sei.cmu.edu/about/overview/whatis.html`, accessed 11.06.2005.

[Jon90] Cliff B. Jones. *Systematic Software Development using VDM*. Prentice-Hall, Upper Saddle River, NJ 07458, USA, 1990.

[KAO05] Welcome to KAON, 2005. `http://kaon.semanticweb.org/`, accessed 11.06.2005.

[KBP01] Erik Kamsties, Daniel M. Berry, and Barbara Paech. Detecting ambiguities in requirements documents using inspections. In *Workshop on Inspections in Software Engineering*, pages 68 –80, Paris, France, 2001.

[Kli04] Alexander Klitni. Textanalyse für Requirements Engineering: Konvertierung der Analyseergebnisse nach AutoFOCUS, 2004. Technische Universität München, Fakultät für Informatik, Systementwicklungsprojekt.

[Lec00] Renaud Lecoeuche. Finding comparatively important concepts between texts. In *The Fifteenth IEEE International Conference on Automated Software Engineering*, pages 55–60, Grenoble, France, 2000. IEEE.

[Les05] Irene Leszkowicz. Detailed vision – Not all black or white, 2005. `http://www.reed-electronics.com/tmworld/article/CA489464.html#10x%20rule`, accessed 21.06.2005.

[Lev66] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Cybernetics and Control Theory*, 10(8):707–710, 1966.

[MAN01] Hideki Mima, Sophia Ananiadou, and Goran Nenadić. The ATRACT workbench: Automatic term recognition and clustering for terms. In *Text, Speech and Dialogue, 4th International Conference*, volume 2166 of *LNAI*, pages 126–133, Železná Ruda, Czech Republic, September 2001. Springer.

[MB89] Y. S. Maarek and D. M. Berry. The use of lexical affinities in requirements extraction. In *Proceedings of the 5th international workshop on Software specification and design*, pages 196–202. ACM Press, 1989.

[MFN04] Luisa Mich, Mariangela Franch, and Pierluigi Novi Inverardi. Market research on requirements analysis using linguistic tools. *Requirements Engineering*, 9(1):40–56, 2004.

[MFT$^+$05] George A. Miller, Christiane Fellbaum, Randee Tengi, Susanne Wolff, Pamela Wakefield, Helen Langone, and Benjamin Haskell.

WordNet, 2005. `http://wordnet.princeton.edu/`, accessed 11.06.2005.

[MS00] Alexander Maedche and Steffen Staab. Discovering conceptual relations from text. In W.Horn, editor, *ECAI 2000. Proceedings of the 14th European Conference on Artificial Intelligence*, pages 321–325, Berlin, 2000. IOS Press, Amsterdam.

[MS02] Alexander Maedche and Steffen Staab. Measuring similarity between ontologies. In *EKAW '02: Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web*, pages 251–263. Springer-Verlag, 2002.

[MTM$^+$99] Mitchell Marcus, Ann Taylor, Robert MacIntyre, Ann Bies, Constance Cooper, Mark Ferguson, and Alyson Littman. The Penn Treebank Project, 1999. `http://www.cis.upenn.edu/~treebank/`, accessed 02.01.2005.

[NM01] Natalya F. Noy and Deborah L. McGuinness. Ontology development 101: A guide to creating your first ontology, 2001. `http://protege.stanford.edu/publications/ontology_development/ontology101-noy-mcguinness.html`, accessed 22.11.2004.

[NRC$^+$02] Johan Natt och Dag, Bjorn Regnell, Par Carlshamre, Michael Andersson, and Joachim Karlsson. A feasibility study of automated natural language requirements analysis in market-driven development. *Requirements Engineering*, 7(1):20–33, 2002.

[NSA02] Goran Nenadić, Irena Spasić, and Sophia Ananiadou. Automatic discovery of term similarities using pattern mining. In *Proceedings of CompuTerm 2002*, pages 43–49, Taipei, Taiwan, 2002.

[Por80] Martin Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980. `http://www.tartarus.org/~martin/PorterStemmer/`, accessed 14.07.2003.

[Pre02] Judita Preiss. Choosing a parser for anaphora resolution. In Philip R. Cohen and Wolfgang Wahlster, editors, *DAARC 2002, 4th Discourse Anaphora and Anaphor Resolution Colloquium*, pages 175–180, Lisbon, 2002. Edições Colibri.

[Rat96] Adwait Ratnaparkhi. A maximum entropy model for part-of-speech tagging. In Eric Brill and Kenneth Church, editors, *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 133–142. Association for Computational Linguistics, Somerset, New Jersey, 1996.

[Rat98] Adwait Ratnaparkhi. *Maximum Entropy Models for Natural Language Ambiguity Resolution*. PhD thesis, Institute for Research in Cognitive Science, University of Pennsylvania, 1998.

[RB98] Colette Rolland and Camille Ben Achour. Guiding the construction of textual use case specifications. *Data & Knowledge Engineering Journal*, 25(1–2):125–160, March 1998.

[RR99] Suzanne Robertson and James Robertson. *Mastering the Requirements Process*. Addison–Wesley, 1999.

[Rup02] Chris Rupp. *Requirements-Engineering und -Management Professionelle, iterative Anforderungsanalyse für die Praxis*. Hanser–Verlag, second edition, 05 2002. ISBN 3-446-21960-9.

[SA97] Ramakrishnan Srikant and Rakesh Agrawal. Mining generalized association rules. *Future Generation Computer Systems*, 13(2–3):161–180, 1997.

[Sch00] Helmut Schmid. LoPar: Design and implementation, arbeitspapiere des sonderforschungsbereiches 340. Technical Report 149, IMS Stuttgart, July 2000. `http://www.ims.uni-stuttgart.de/~schmid/`, accessed 04.02.2005.

[SHE89] Motoshi Saeki, Hisayuki Horai, and Hajime Enomoto. Software development process from natural language specification. In *Proceedings of the 11th international conference on Software engineering*, pages 64–73. ACM Press, 1989.

[Tex05] TEXTTOONTO, 2005. `http://kaon.semanticweb.org/Members/rvo/Module.2002-08-22.4934`, accessed 22.06.2005.

[vL01] Axel van Lamsweerde. Goal-oriented requirements engineering: A guided tour. In *Proceedings of the 5th IEEE International Symposium on Requirements Engineering*, pages 249–263. IEEE Computer Society, 2001.

[VM94] Sunil Vadera and Farid Meziane. From English to formal specifications. *The Computer Journal*, 37(9):753–763, 1994.

[Wen04] Armand Wendt. Textanalyse für Requirements Engineering - Einbeziehung von Tabellen und Listen in die Analyse. Master's thesis, Technische Universität München, Fakultät für Informatik, November 2004.

[Wik05a] Ontology, 2005. `http://en.wikipedia.org/wiki/Ontology`, accessed 11.06.2005.

[Wik05b] Ontology (computer science), 2005. `http://en.wikipedia.org/wiki/Ontology_%28computer_science%29`, accessed 11.06.2005.

[Zav97] Pamela Zave. Classification of research efforts in requirements engineering. *ACM Comput. Surv.*, 29(4):315–321, 1997.

[ZJ97] Pamela Zave and Michael Jackson. Four dark corners of requirements engineering. *ACM Trans. Softw. Eng. Methodol.*, 6(1):1–30, 1997.

Own Publications

[Kof04a] Leonid Kof. Natural Language Procesing for Requirements Engineering: Applicability to Large Requirements Documents. In Alessandra Russo, Artur Garcez, and Tim Menzies, editors, *Automated Software Engineering, Proceedings of the Workshops*, Linz, Austria, September 21 2004.

[Kof04b] Leonid Kof. Using application domain ontology to construct an initial system model. In M. H. Hamza, editor, *Proceedings of the IASTED International Conference on Software Engineering*, Innsbruck, Austria, February 17-19 2004.

[Kof05a] Leonid Kof. An Application of Natural Language Processing to Domain Modelling – Two Case Studies. *International Journal on Computer Systems Science Engineering*, 20(1):37–52, 2005.

[Kof05b] Leonid Kof. Natural Language Processing: Mature Enough for Requirements Documents Analysis? In Andres Montoyo, Rafael Muñoz, and Elizabeth Methais, editors, *Application of Natural Language to Information Systems*, volume 3513 of *LNCS*, pages 91–102, Alicante, Spain, June 15–17 2005. Springer–Verlag.

[KP05] Leonid Kof and Markus Pizka. Validating Documentation with Domain Ontologies. In *The 4th International Conference on Software Methodologies, Tools and Techniques*, Tokyo, Japan, September 28–30 2005. accepted conference paper, to appear.