Traceability-Driven Model Refinement for Test Case Generation

Matthias Riebisch, Michael Hübner
Ilmenau Technical University, Ilmenau, Germany
{matthias.riebisch|michael.huebner}@tu-ilmenau.de

Abstract

Testing complex Computer-Based Systems is not only a demanding but a very critical task. Therefore the use of models for generating test data is an important goal. Tool support during the generation of test cases can considerably reduce the effort and the risk of errors of this task. While model the understanding of model transformation got better in the past, the analysis of the input – requirement specifications mostly consisting of natural language texts – still represents a bottleneck. In this paper a method for closing the gaps between manual techniques for structuring texts and automatic techniques based on linguistics is explained. By providing suggestions for missing or ambiguous terms the method supports the formalization within large projects. The suggestions are derived from a glossary and from an analysis of their integration in use case descriptions. Additionally the traceability links via feature models and other model elements are analyzed. Thus, the advantages of both techniques are joined to increase the level of tool support, resulting in a higher efficiency of the test case generation.

1 Introduction

Contemporary Computer-Based Systems have to satisfy a variety of requirements. Due to their complexity the usage of models during the design constitutes an essential necessity. Models have to support the developers in achieving comprehension and in communicating ideas, enabling the early assessment of a solution. However, there is another important need for the usage of models. If requirements for correctness and reliability play an important role, a lot of effort for system verification and validation has to be invested. Model-based test case generation addresses this issue to reduce this effort.

Depending on the domain, different degrees of model formality are accepted. If behavior, i.e. in terms of state models, plays the most important role and rigorous approaches are accepted for its description, a number of approaches provide considerable support.

If non-functional properties and quality requirements together with complex functional requirements have to be fulfilled, requirements are usually described by natural language texts. In such cases, rigorous approaches are not applicable for modeling requirements and their implementation. However, manual methods for the analysis of textual descriptions and the generation of test cases are very effort consuming and error-prone. The benefit in terms of efficiency and productivity is the justification to invest more effort in formalization and model-based refinement, in contrast to formalization of requirements specifications for traditional software development, where the benefit would be minimal. However, the generation of models for model-based development is supported as a side effect.

Especially the issues of data transformation, behavioral constraints and features at a higher level of abstraction are frequently expressed by natural language terms, often structured within semi-formal descriptions. In these types of descriptions a term carries a meaning for a reader because it is associated with ideas in his/her mind. Missing semantic information hinders the use of tools for analysis and generation.

In this contribution, automatic techniques for lexical and linguistic analyses for formalizing text are extended to reduce the effort for the manual completion and replacement of missing and ambiguous parts. Therefore, ideas of manual techniques like text templates are integrated with the concepts of features and traceability links to provide suggestions for the human activities. The efficiency of the usage of large glossaries is improved.

The main benefit of this combination is the improved degree of automation of the analysis and restructuring activities. Furthermore, the applicability of the method in large projects is improved by providing tool support for cross-referencing and replacing the terms based on a glossary. In addition to these effects, the resulting traceability links support later design comprehension and reengineering.

2 State of the Art

There are already good methods and techniques contributing to the goals of model-driven test case generation and model-driven software development.

2.1 Use Case Refinement

Semi-formal descriptions represent one of the approaches aiming at a reduction of inconsistency and ambiguity of natural language documents. In the case of requirements specifications, text documents are structured using text templates, i.e. by the Volere schema [RR00]. In many domains, use cases are widely accepted for specifying requirements, especially for object-oriented development methods [OMG]. To express behavioral information of use cases, attached text templates are established as a de facto standard [Co00]. These templates provide a semi-formal description of the procedure of a scenario. The template sections are described by keywords. As a result, the average description is more complete and less ambiguous than without a template. Later, Tab 1 will give an example.

In our case, the behavioral specification is of special interest for the generation of test cases. By its structure the sections precondition, main success scenario, postcondition and failure, contain information about the behavior. The information is expressed in natural language. The overall comprehension for the reader is improved in comparison to plain text, because the structure guides the writer of a specification. However, for the information within the sections no further structure or formalization is provided. Therefore the sections cannot be evaluated by automatic tools – i.e. for verification or for test case generation purposes. Semantics are expressed by the use of natural language terms only. Further formalization was out of the scope of the method, because the resulting specification is intended to be implemented in a conventional way.

The SOPHIST method [Ru01] performs a refinement and a formalization of structured texts by introducing text templates with a defined syntactical structure instead of free natural language text. Therefore, the inner structure provides some formalization, leading to less ambiguous and more complete descriptions. However, the method is carried out manually, and the templates reduce the flexibility strongly. Another drawback is the low support for formalizing behavioral descriptions.

Information Retrieval methods [SG83] contribute to the exploration of the content of natural language texts. There is a broad variety of methods and techniques that have shown their applicability in practice, e.g. in patent research. However, the usage of a glossary is mostly limited to simple lists of terms, and the semantics of texts cannot be analyzed clearly enough to derive test case specifications.

Linguistic approaches, as developed in the CREWS project [RBA98], search for linguistic structures within structured texts of requirements descriptions. The linguistic structures are replaced by templates to increase the degree of formalization. This approach supplies behavioral information as a part of specific conceptual models. The methods of CREWS provide good preconditions for automatic test case generation by providing automatic text analysis and rule based evaluation. Verification and validation is supported by a model-based generation of a text and its analysis after a revision. The flexibility of the natural language is less limited than by the SOPHIST method. However, 10 .. 30 percent of information is not recognized during the automatic analysis. There is no substantial support for the manual completion of this missing information. Furthermore, the CREWS output format does not enable a direct generation of behavioral UML models.

2.2 Test Case Generation

For a transformation of behavioral models to test case specifications, a large set of methods is available. E.g. the papers by Krueger et al. [KGSB99][BGK02] show possibilities for transforming sequence diagrams to state charts. The resulting behavioral models can serve as input for the test case generation and for a model-driven development.

The integration of text analysis concepts to test case generation approaches [RRC00] enables a construction of test cases to some extend, but there is a high effort for human interaction, especially for larger systems with larger glossaries and complex interactions.

2.3 Traceability Links

Traceability links describe the dependencies between the artifacts of different stages of development. They support the verification of solutions by linking rationale, decisions and solution elements. In this way they structure complex information and ease the developers comprehension.

Traceability links are usually applied to four major tasks within software systems development:

- compliance verification,
- requirements (elaboration and) refinement,
- design allocation, and
- rationale determination and decision-making.

The CREWS project explored the usage of Traceability Links to software development activities [RJ01]. Different categories of links have been analyzed.

For applying the traceability links to generate test cases they have to connect model elements and expressions at a detailed level. As a result, their number is high. The issue of maintaining and checking them is crucial. There are some approaches supporting the elaboration and the maintenance of the traceability links, but they demand for further development [Rie04].

2.4 Feature models

A feature model represents the relations between the features of a system. A feature stands for "a logical unit of behaviour that is specified by a set of functional and quality requirements representing an aspect valuable to the customer and system architect" following the definitions in [RSP03]. The feature model provides an abstract view on the requirements by grouping them. Later, Fig 5 will show the relations between rationale, decisions and solution elements by traceability links grouped by features.

Features are used in this approach for providing a context by the use cases belonging to one feature.

3 Method overview

The test case generation method as a whole consists of four major steps; its first step is described in this paper. As initial input it works on a requirements specification in a structured form of use cases description templates as defined by [Co00]. Furthermore, a feature model is used for structuring these use cases. Additionally, an initial glossary is helpful to provide important terms of the domain, extended by a short explanation. Fig 1 shows an overview over the four steps.

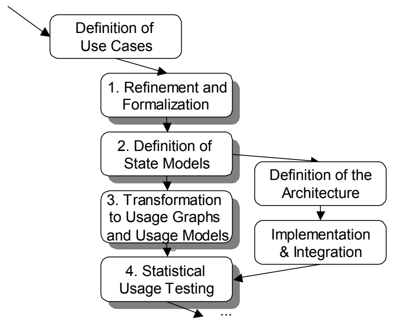
Definition of Use Cases
1. Refinement and Formalization
2. Definition of State Models
Definition of the Architecture
3. Transformation to Usage Graphs and Usage Models
Implementation & Integration
4. Statistical Usage Testing
...

Fig 1: Overview over the test case generation process

In the first step, the requirements specification is formalized and refined in an incremental way. It results in behavioral description by formalized semantic patterns, that is transformed into a activity model as defined by UML 2.0 [OMG]. The terms describing the model elements provide semantic information via associations to the glossary and to other models. Only these activities of the method are described in this paper

If still unavailable, a data model and a model of the graphical user interface GUI has been built in parallel to the incremental formalization. Their elements e.g. a button of a GUI or a data field are required as targets of the behavioral descriptions, and as objects within the grammatical structure of the text templates.

As step 2 the activity diagrams of the behavioral model are transformed to sequence diagrams and to state models, based on the definitions of UML 2.0. from the formalized behavioral description. For these transformations, a set of methods is applied as mentioned in section 2.2. The resulting behavioral models can serve as the input for the test case generation as well as for a model-driven development.

In step 3 the behavioral model consisting of state charts is transformed into usage graphs by adding usage information (e.g. probabilities). Subsequently, these usage graphs are transformed to usage models applying the methods [WP00] and [HPR03]. As result, sequences of state transitions describe test cases. These activities are performed to prepare a statistical usage testing. If other types of tests - e.g. coverage tests - have to be performed, different activities have to be done in this step.

As step 4 the test cases are processed automatically. For this activity a broad variety of powerful tools is available and ready for industrial use.

4 Incremental Refinement and Formalization of Use Case Descriptions

This paper concentrates on step 1 of the method as described in the previous section and in Fig 1. In this step, a description of natural language has to be transformed into an expression with formally defined syntax and semantics. This transformation is performed in an incremental process. In conjunction with the formalization, the text parts are refined by adding more detailed information or by replacing general terms by more concrete ones.

This step of the method is mostly based on the powerful text analysis techniques of the CREWS project. They provide tool support in the transformation of 70 .. 90 percent of the text parts into syntax graphs. The main contribution of the new method consists in the support for the remaining parts, leading to a strong reduction of the human effort for the transformation, especially in the case of large systems. Suggestions for completion and replacement are provided by a comparison of semantic patterns and by a traceability-driven analysis of the roles of glossary terms as described by section 4.4. Fig 2 gives an overview about this step of the method.

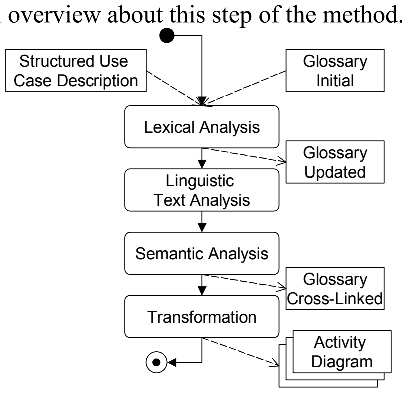

Fig 2: Formalization of a use case description

The method is explained using the following case study to provide details about the activities and to demonstrate its applicability in a practical setting. A mobile device – in this case the Blackberry handheld 7230 – is extended by a plugin component to support emergency services. The device provides information services to emergency and rescue teams in addition to their usual means of communication. The provided information includes an access to medical knowledge bases, geographic data and building information as well as a background information e.g. dangers of the particular situation.

In parallel, data about the current operation are collected both to reduce bureaucracy and to provide information to others earlier: data about injured persons such as name, social security data, status, current place; information about the operation such as orderer, time and costs.

If a GPRS connection is available, data is transmitted immediately to a backoffice service. Otherwise it is uploaded via the Blackberry's docking station after an operation. Fig 3 shows the use case diagram.

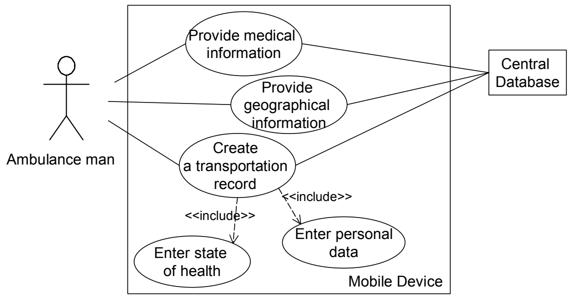

Fig 3: Use Case Diagram (partly)

The use cases are structured by assigning them to features or a feature model. For our case study, among others, the use cases "Create a transportation record", "Enter state of health" and "Enter personal data" are assigned to a feature "Collect patient transportation data".

Further on, the use case "Enter personal data" is applied as an example. Tab 1 shows its use case description in the initial form, according to the template of [Co00]. The sections Main success scenario and Alternative scenarios are of special interest for behavioral modeling. The Social Security Number SSN plays the role of an identifier of the patient.

Further information about the requirements of the system and about its environment e.g. the graphical user interface or the data structure is described in additional models. These models are frequently necessary for test cases because they are referenced by use cases or test activities, e.g. "Button XY is pressed". The names of model elements are related to glossary items. Fig 4 shows a part of the data model that is used in the examples.

Tab 1: Use Case Description "Enter personal data"

USE CASE	Enter personal data	
Description	Verify or enter the personal data of the patient.	
Actors	Ambulance man	
Scope	Software on the Blackberry device	
Level	User-level	
Precondition	The ambulance man is logged into the program	
Main success scenario	1	The ambulance man enters the SSN of the patient.
	2	The system retrieves the patient's records with the given SSN from the central database.
	3	The ambulance man verifies the last name and the address of the patient.
	4	If the personal data of the patient have been changed, the personal data of the patient is transmitted to the central data base.
Postcondition	The information about the transport is stored in the central data base.	
Alternative scenarios	1a	The SSN of the patient is not available.
	1a1	The ambulance man enters the first name, last name, birth name, sex, date of birth and place of birth.
	1a2	The system searches in the central database for a patient record with the given birth name, sex, date of birth and place of birth.
	1a3	There is a patient record with birth name, sex, date of birth and place of birth matching to the given data.
		Proceed with step 3
	1a3a	There is no patient record with the given birth name, sex, date of birth and place of birth.
	1a3a1	The ambulance man enters the address.
	1a3a2	A new patient record with the given personal data is created.
		Proceed with step 4
	4a	There is no connection to the central data base.
	4a1	The data is stored temporarily in the Blackberry device.
	4a2	When the connection is established, the data is transmitted to the central data base.

4.1 Lexical analysis

The lexical analysis is performed for every entry of the behavioral parts of the use case description. It determines the word class (part of speech) for every word. In the English language, there are many words with identical forms for verb and noun. At this stage, no further distinction is possible, therefore more than one word class can be assigned.

First, rules are applied to map derived words to a corresponding stem word, i.e. a plural noun is mapped to the corresponding singular noun, and a verb with a 3rd person singular "s" is mapped to the infinitive form.

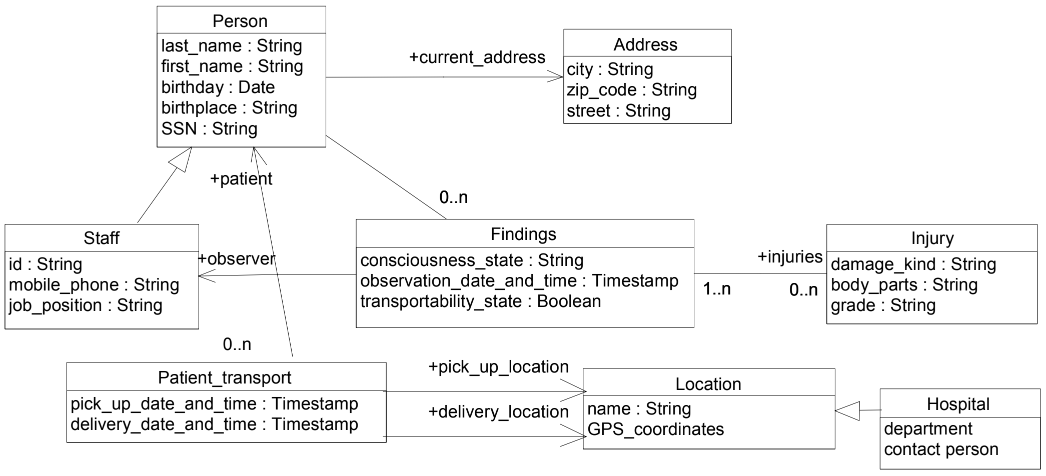

Fig 4: Data Model as an UML Class Diagram

The word class for every stem word is retrieved from a word list. In rare cases, unknown words have to be classified manually while the classification is recorded for a later update of the list and the rules. The lexical analysis is performed using well-established methods of linguistics, e.g. [Ch71][SG83].

Example: The$_{\text{definite-article}}$ ambulance$_{\text{noun}}$ man$_{\text{noun}}$ enters$_{\text{verb}}$ the$_{\text{definite-article}}$ SSN$_{\text{noun}}$ of$_{\text{pronoun}}$ the $_{\text{definite-article}}$ patient$_{\text{noun}}$.

4.2 Linguistic analysis

There are rules for the linguistic analysis of clauses (i.e. of simple sentences). A slightly simplified version of these rules is given in the sequel.

The whole set of these rules describes a subset of the natural language. This subset is defined in a way that the resulting grammar still enables human-readable texts, because the formalized requirements specification has to be verified and validated by the customer. In contrast to the rather strict definitions by the templates of the SOPHIST method (see section 2.1) there is a higher degree of flexibility for such descriptions. Of course, the grammar subset should be as simple as possible to limit the effort of a tool-based evaluation.

The description of the rules for our method is very similar to the Backus Naur format of grammar specifications. The rules are mostly based on the CREWS rules. Some examples are given below, but a further discussion of the grammar would be out of the scope of this paper.

```
<clause> ?=<active-clause>
          | <passive-clause>

<active-clause> != agent:<noun-phrase >
          verb:<active-predicate>
          [object:<noun-phrase>]
          [complement:<preposition-phrase>]...

<noun-item> ?= <noun> | <noun-sequence>

<noun-phrase> ?= <unrelated-noun-phrase>
          | <related-noun-phrase>

<related-noun-phrase> != owned:<unrelated-noun-phrase>
          „of“ owner:<noun-phrase>
```

As an example, the following sentence from the use case description in Tab 1, activity 1:
“The ambulance man enters the SSN of the patient.”

is transformed by the linguistic analysis into the tree:

((the ambulance man)$_{\text{agent}}$ (enters)$_{\text{verb}}$ ((the SSN)$_{\text{owned: noun-sequence}}$ of (the patient)$_{\text{owner}}$)$_{\text{object}}$)$_{\text{active-clause}}$

4.3 Semantic analysis

At this stage the structure of the sentences is defined by the roles of the terms within them. For a transformation to formally defined expressions, the CREWS method proposes an assignment to typical structures, so-called semantic patterns. According to CREWS, a comprehensive set of semantic patterns has to be developed. For building behavioral models, the templates of the SOPHIST technique are being applied here.

For the later transformation into activity diagrams the semantic patterns refer to typical structures in activity diagrams. In this way, the patterns map a set of clauses to an activity diagram.

For every semantic pattern there are one or more rules that map a linguistic structure to that semantic pattern. The rules are applied to a node, e.g. to a *clause*. For every part of the semantic pattern (e.g. part *agent*) an expression specifies its contents in terms of the linguistic tree. For a semantic pattern to be applicable all mandatory parts must be present in the linguistic tree. Optional parts are indicated with a question mark before the equal sign.

The linguistic structure of the example sentence can be transformed by the following rule

```
<clause>{verb:<enter> } → <dataEntry>
        Agent:object!=agent
        Object:object!=object
        Source:object!=agent
        Destination:object{default : ‘the
        system’}?=complement[preposition=’into’].object
```

As a result it is instantiated and it builds a reference to a use case providing a refinement. The name of the use case was provided from the glossary:

```
<clause>{verb: <enter>, agent:object!=’the ambulance man’,
        object.owned=”the SSN”, object.owner = ”the
        patient“}
```

The resulting semantic pattern instance refers to glossary entries:

<dataEntry> (the ambulance man)$_{agent}$ (the ssn)$_{attribute}$ (the patient)$_{object}$

To provide the input for the completion in the next session, as another example the section 1a3a1 of Tab 1 is analyzed “The ambulance man enters the address.” The linguistic analysis leads to this result:

((The ambulance man)$_{agent}$ (enters)$_{verb}$ (address)$_{simple-unrelated-noun-phrase}$)$_{active-clause}$

The transformation rule is rather similar to the one above. However by providing more than one transformation rules a higher flexibility of the texts is allowed:

```
<clause>{verb:<enter> } → <dataEntry>
        Agent:object!=agent
        Object:object!=object
        Source:object!=agent
        Destination:object{default : ‘the
        system’}?=complement[preposition=’into’].object
```

The use case describes a patient transport. That’s why an instance of type *PatientTransport* is the main object of the use case and hence the starting point of the search for attribute and relation names in the data model. The resulting semantic pattern instance contains *Address* as an object because of its role in the sentence.

<dataEntry> (the ambulance man)$_{agent}$ (the address)$_{object}$ (the system)$_{destination}$

4.4 Tool-supported formalization

For a text of the use case description, the relations between the words and terms of the texts are at this stage described by formal means, because they are assigned to the linguistic variables of semantic patterns. However, the meaning of most terms is still related to the context. Therefore they can not be evaluated during a transformation into a model. A glossary is applied to provide a context and to relate terms to each other. Three typical cases have to be handled:

1. a term is part of the glossary
2. a term is not available in the glossary
3. a term is incomplete

Other cases, e.g. if a term does not match any of the semantic patterns, have to be handled manually. However, in our experience these cases are less than one third, others report 10 percent [RBA98].

If the term is part of the glossary then the term within a semantic pattern is replaced by a reference to the glossary item, and in this way references to other elements of the context can be drawn.

If a term is not available in the glossary, then it has to be replaced manually by a glossary item, or the glossary has to be extended. However, for large systems with a large glossary this is a very effort-consuming task. Moreover, any changes to the glossary are critical for the evaluation. The consistency of the references between glossary items and terms in requirement description as well as in model elements has to be maintained. As an experience from our case studies we had to draw the conclusion, that the effort for manual term replacement and the necessary consistency checks is nearly as high as that of manual techniques for formalizing natural language texts, e.g. the SOPHIST method [Ru01].

In our method, a subset of the glossary items is built, that is provided for a manual selection and replacement. This subset is derived by evaluating the references between glossary items and features. As mentioned earlier, a feature represents an abstraction of a set of use cases. Fig 5 shows the relations of the data model in an UML class diagram. Features map use cases to the solution, as usual for many approaches [RSP03]. This mapping is described by traceability links (full lines in Fig 5). In this formalization method, terms are related to each other via the glossary (shown as dashed lines in Fig 5).

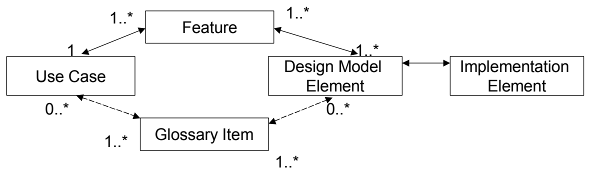

Fig 5: Relations Between Features and Glossary Items via Traceability Links

The context of a use case is expressed by its assignment to a feature. Therefore, all glossary items occurring in use cases (and model elements) of the same feature can be regarded as belonging to this context. This relation is exploited for building the subset of glossary items that is provided for the manual replacement. Furthermore, the linguistic information is included to provide only terms of the required type.

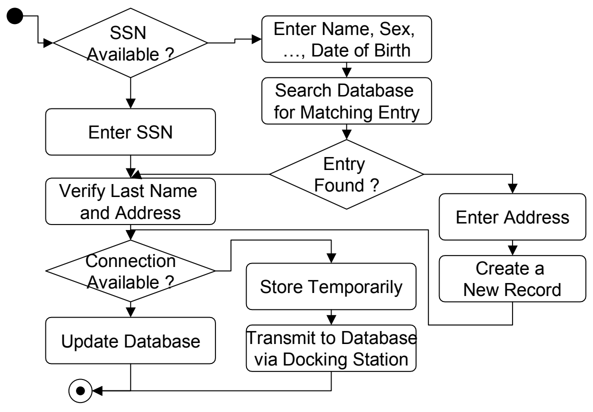
SSN Available ?
Enter Name, Sex, ..., Date of Birth
Search Database for Matching Entry
Enter SSN
Entry Found ?
Verify Last Name and Address
Enter Address
Connection Available ?
Store Temporarily
Create a New Record
Update Database
Transmit to Database via Docking Station

Fig 6: Resulting Activity Diagram

If none of the proposed terms is an appropriate replacement, the developer may enter a new term. However, this new term has to be integrated into the glossary, e.g. by building references via a thesaurus. By enabling new terms, a flexible way of working is not reduced by the method.

In the third case if a term is incomplete, one or more suggestions for a replacement and completion are provided to the developer. In our example from section 1a3a1 of Tab 1 "The ambulance man enters the address." the object is missing, and the *address* occupies the role of the object. This incompleteness is not discovered during the lexical, the linguistic or the semantic analysis. However, when the terms are replaced by a comparison between other models and the glossary, *Address* is not available in the data model as an object but as an attribute of a *Person* (Fig 4). Therefore, the expression has to be completed by the developer. This is supported by proposing possible candidate terms from the context. The suggestions are based on a subset of the glossary items. Similar to the former case, the subset is derived by an evaluation of the glossary items from the context as determined by a feature and the use cases related to it via traceability links. All glossary items of the word class *noun* and with a reference to *Person* in the data model are included in the subset.

The selection demands only low effort, because only the two terms "the ambulance man" and "the patient" are proposed.

If a proposed subset is incomplete, e.g. because it is based on incomplete model information, the developer can add items to the glossary or he can extend the models. In this way the flexibility is maintained.

For refinement there is the additional option of entering information directly into templates, as proposed by the SOPHIST technique. This way is fairly quick, but the flexibility is less than by analysing natural language texts.

4.5 Transformation to activity diagrams

The input of the last step is provided as expressions following a formally defined syntax and semantics. In our case, the expressions are provided in XML. The rules for transforming the expressions to activity diagrams have been defined by defining the semantic patterns for section 4.3. The rules are implemented in an XSLT-based prototype tool. Fig 6 shows the resulting UML activity diagram that represents the use case description of Tab 1.

5 Conclusion

The method presented in this contribution integrates forward engineering and verification by providing refined model information to both processes. To enable this integration, models have to express additional information. This integration is inspired by the idea of model-based development. Further work is necessary to prepare this approach to additional domains.

6 Acknowledgements

This work was partly funded by the German Research Association DFG under grant PH49/5-1.
We would like to thank our colleague Detlef Streitferdt for his feedback to an earlier version of this paper.

7 References

[BA99] C. Ben Achour: Extraction des Besoins par Analyse de Scénarios Textuels, Université Paris 6, 1999.

[Bo00] J. Bosch: Design & Use of Software Architectures - Adopting and Evolving a Product Line Approach. Addison-Wesley, 2000.

[BGK02] M. Broy, R. Grosu, I. Krüger: Automatically Generating A Program, US Patent No.: 06405361, 2002.

[CA05] CARE Requirements Engineering Tool, Version 3.x. Sophist Group, 2005. Online available at http://www.sophist.de

[Ch71] N. Chomsky, Deep Structure, Surface Structure and Semantic Interpretation. Steinberg & Jacobovits, 1971.

[Co00] A. Cockburn: Writing Effective Use Cases, Addison-Wesley, 2000.

[CR95] The Standish Group: CHAOS report, 1995.

[FL00] P. Fröhlich, J. Link: Automated test case generation from dynamic models. In: Proc. ECOOP 2000, LNCS 1850, Springer (2000) 472-491.

[Gal99] L. Gallagher. Conformance testing of object-oriented components specified by state/transition classes, 1999.

[Gö01] M. Götze: Statistical Usage Testing Based on UML Diagrams. Studies project report, Dept. Process

Informatics, Ilmenau Technical University, Ilmenau, Germany, 2001.

[HPR03] M. Hübner, I. Philippow, M. Riebisch: Statistical Usage Testing Based on UML. Proc. 7th World Multiconferences on Systemics, Cybernetics and Informatics. July 27-30, 2003, Orlando, FL, USA.

[Ja03] How to Write Doc Comments for the Javadoc Tool. Sun, 2003. Online available at http://java.sun.com/j2se/javadoc/writingdoccomments

[KGSB99] I. Krüger, R. Grosu, P. Scholz, M. Broy: From MSCs to Statecharts. In: Distributed and Parallel Embedded Systems, Kluwer Academic Publishers. 1999. http://www4.informatik.tu-muenchen.de/papers/KGSB99.html

[RJ01] B. Ramesh, M. Jarke: Toward reference models for requirements traceability. IEEE Transactions on Software Engineering, Vol. 27, No. 1, pp. 58-93, 2001

[RRC00] B. Regnell, P. Runeson, C. Wohlin: Towards Integration of Use Case Modelling and Usage-Based Testing, Journal of Systems and Software, 50(2):117-130, 2000.

[RSP03] M. Riebisch, D. Streitferdt, I. Pashov: Modeling Variability for Object-Oriented Product Lines. In: F. Buschmann, A.P. Buchmann, M. Cilia (Eds.): Object-Oriented Technology. ECOOP 2003 Workshop Reader. Springer, Lecture Notes in Computer Science , Vol. 3013, 2004, pp. 165 - 178.

[Rie04] M. Riebisch: Supporting Evolutionary Development by Feature Models and Traceability Links. In: Proceedings 11th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS2004), Brno, Czech Republic, May 24-26, 2004, pp. 370-377.

[RR00] J. Robertson, S. Robertson: Volere Requirements Specification Template, 2000.

[RBA98] C. Rolland, C. Ben Achour: Guiding the Construction of Textual Use Case Specifications, Data & Knowledge Engineering Journal, 25(1-2):125-160.

[Ru01] C. Rupp: Requirements-Engineering und -Management [in German]. Hanser, 2001.

[SG83] G. Salton, M. McGill: Introduction to Modem Information Retrieval. McGraw-Hill, 1983.

[OMG] UML 2.0 Specification. Online available at http://www.uml.org/.

[WP00] G.H. Walton, J.H. Poore: Generating transition probabilities to support model-based software testing. Software - Practice and Experience, 2000. 30:1095–1106.

7.1 Glossary

In the glossary there are entries for all semantic patterns

"enter the personal data of (person)"	→ use case
"enter the state of health of (patient)"	→ use case
"enter (attributes) of (object)"	→ Communication('enter')
"enter (attributes)"	→

7.2 Model Transformation for Test Case Generation

In this section approaches for the transformation of behavioral models are investigated for their applicability at the level of refined requirements and test case descriptions.

#+# Siehe HICSS04 S.12: non-deterministic behavior: wenn alternative Abläufe ohne klaren gegenseitigen Ausschluss (oder wegbekommen durch gleich verbieten ? oder durch Reihenfolge implizit vorgeben ?) Beispiel: welche Banknoten gibt ein Automat aus. Lesefehler von EC-Karten, fehlende Netzwerkverbindung,

Frage des Standpunkts, ob deterministisch oder nicht: Beobachtbarkeit

#+# global system states: nichts dazu in HICSS04 und Diss, also reduzieren auf Betrachtung realer Verhältnisse

The behavioral description as an important part of a requirements specification is provided in form of use case templates [Co00].

starting from use case descriptions e
incremental refinement process of use case templates represents an important part of the method described in this paper. In this process, the CREWS concept and the SOPHIST technique are integrated. In a stepwise refinement information is added interactively, and the degree of formalization is increased.

8 Traceability Links as a Bridge between Use Cases, Design Elements and Implementation Components

The aim is to transform the content into an expression with formally defined syntax and semantics, i.e. an UML state model or a MSC. In this section an approach for adapting the principles of traceability links is presented. Traceability links connect requirements, design elements and implementation components together with glossary terms. The links are extended by additional information of design decisions.

# The improvement consists in the additional information that is attached to the traceability link.

9 Categorization and Formal Definition of Traceability Links

Based on the results of the State of the Art analysis, a categorization of types of traceability links is developed to trigger different evaluation methods according to different elements. The categories match the different types of model elements and the categories of refinement activities. Furthermore, the concept of a semantic web for linking terms is added by introducing links between terms of a glossary with those (?) of template texts.

The syntax of traceability links is formally defined to enable their evaluation by tools. Based on this definition, links between elements of different UML models are introduced. Furthermore, glossary terms and templates of the use case description are established as targets of the links. Examples for the different types of links are derived from the case study.

10 Traceability Link Driven Development

Scoping decisions for the selection of requirements and features to be implemented are made by assessing both demand and effort. Effort estimation is performed based on traceability links.

During requirements refinement, traceability links are established to enable verification and assessments for completeness and conceptual integrity.

In the design process design alternatives are elaborated. Decisions lead to design elements, with traceability links between requirements and these elements. Information about the design decisions is attached.

Non-functional requirements are assigned to the appropriate elements or they are resolved by introducing functional solutions following to the method of Bosch [Bo00].

Changes and their verification heavily depend on traceability links. Therefore they have to be kept in a consistent state during changes.

structuring natural language texts are combined with to close the gaps and to increase the level of tool support as well as the degree of automation. A method is presented that joins the SOPHIST method for structuring texts by the usage of templates [Ru01][CA05] with linguistic and Information Retrieval methods developed in the CREWS project [RBA98]. Terms as the carrier of semantics are stored in a glossary, and they are linked to structured texts as well as model elements.