# Refinement and Formalization of Semi-Formal Use Case Descriptions

Matthias Riebisch, Michael Hübner
*Ilmenau Technical University*
*Max-Planck-Ring 14; 98684 Ilmenau; Germany*
*{matthias.riebisch|michael.huebner}@tu-ilmenau.de*

**Abstract**

*Behavioral models of computer systems are required for their synthesis, for verification and validation. The system behavior is usually described in requirements specifications. However, most specifications are provided in natural language or in a semi-formal way. Incompleteness and ambiguity inhibit their successful exploitation by tools. In this paper an approach for stepwise refinement and formalization of natural-language or semi-formal descriptions is presented. Based on the structure of Use Case descriptions, the formalization of behavioral information is reached by a stepwise transformation of an input text to structured sentences with a well-defined syntax by performing linguistic analyses. The terms of the text are replaced by references to other items, i.e. of the glossary. By providing a tool-supported text output with hypertext-like navigation facilities, a verification of the result by the human experts is provided. The resulting behavioral description is suitable for a derivation of dynamic models, e.g. message-sequence charts and state diagrams.*

## 1. Introduction

Considering the risk for software development projects to fail, the highest risk is related to the requirements specification phase of the development process. The CHAOS study has shown that 20 % of the failing U.S. software projects suffer from problems with this phase [CR95]. Furthermore, many of the efforts in other stages of the development - i.e. the system test - depend on the correctness of the requirements specification. In most cases, requirements specifications are provided or verified by customers, and they consist of natural-language documents. Ambiguity and incompleteness are typical for these specifications. These properties effect the correctness of the development results. Furthermore, they influence the efficiency of the development process by preventing the use of automatic tools. Formal specifications would prevent these drawbacks. However, they are not well accepted in most domains.

This paper presents an approach for a stepwise refinement and formalization of natural-language expressions in use case descriptions. The information is structured by a formally defined syntax that keeps the similarities to natural language to enable a verification by customers. The approach is developed as part of a methodology for automatic test case generation based on use cases.

## 2. Semi-formal description in Use Case Templates

Semi-formal descriptions represent one of the approaches of reducing 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 skeleton [RR00]. If object-oriented development methods are applied, requirements are described by use cases [OMG]. For expressing behavioral information of use cases, text templates attached to them are widely accepted [Co00]. These templates provide a semi-formal description of the procedure of a scenario. The keywords give a structure to a natural language expression (see Tab 1). As a result, the average description is more complete and less ambiguous than without the structure.

*Tab 1: Use Case template*

| | | |
|---|---|---|
| **USE CASE** | Name of this Use Case | |
| **Service** | Benefit of this Use Case for the user | |
| **Brief description** | Optional description | |
| **Actors** | List of actors related to this use case | |
| **Scope** | Extent of the system of concern | |
| **Level** | Level of Abstraction | |
| **Precondition** | Conditions for the state of the actors and the system, required for the application of this Use Case | |
| **Main success scenario** | 1 | first step of the usual execution |
| | 2 | second step of the usual execution |
| **Postcondition** | (List of) conditions fulfilled after a successful execution | |
| **Alternative scenarios** | 1a | first step of an exceptional execution after step 1 |
| | 1b | second step of an exceptional execution after step 1 |
| **Failure** | Short description of a possible exceptional condition | |
| **Cause** | Situation, cause, condition for occurring | |
| **Postcondition** | Postcondition in the case of this failure | |

In our case, the behavioral specification is of special interest for the generation of test cases. By its structure in the sections precondition, main success scenario, postcondition and failure contain information about the behavior. The information is expressed in natural language. The comprehension to humans is improved, compared to plain text. However, the information cannot be evaluated by automatic tools – i.e. for verification or for test case generation purposes. 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.

## 3. Refinement and Formalization by Transformation into Expression Templates

Ambiguities in the structured texts mentioned above consist, e.g., in expressions without a reference to the particular context or in words with more than one meaning within the particular context. Such expressions have to be replaced by ones with a defined reference within the context. In our case, predefined keywords and glossary items provide possible targets for such references of expressions. In this way, a formalization can be performed.

Incompleteness in structured texts is frequently caused by missing objects, subjects, transitions or relations within expressions. Furthermore, missing parts represent another cause, i.e. missing refinements.

To remove ambiguity and incompleteness, in this approach a set of templates is used for structuring expressions. For each natural-language part of a semi-formal description (i.e. No. 1, 2, 1a, 1b of Tab 1) a template is applied to formalize it.

Every template is defined by a relation to a semantic pattern and a linguistic structure. Based on this relation, a linguistic analysis can be applied to derive suggestions for the application of a template based on a natural language expression. For the evaluation of such expressions a usual procedure of textual analyses is performed:

- In a *lexical analysis* the words of the natural language text are compared to a lexicon, known words are identified and their type is marked. Unknown words have to be classified interactively.
- In a *syntax analysis* for every sentence a syntax tree is built.
- A *linguistic analysis* is performed to identify linguistic structures within the syntax tree.
- The *semantic analysis* compares the linguistic structure to semantic patterns. Suggestions for suitable templates are derived.

The templates follow to a strict syntax. For each field within a template, a specified type of term can be placed (Tab 2). By defining the terms via a reference to the glossary of predefined terms, the semantics of such a description are formalized to some extent. In this way, the descriptions are transformed from a free vocabulary to a restricted one. As a result, the descriptions can be evaluated by tools, i.e. for verification purposes or in our case for a generation of test cases.

*Tab 2: Scheme of one of the templates with types (italics) and an example.*

| *actor* | *activity* | *object* | *Destination* |
|---|---|---|---|
| The librarian | selects | the book | in the list of available books. |

As a part of this approach, a set of templates and transformation rules is developed. One part of templates was developed after those of the Sophist method [Ru01], others form the completion rules of Rolland et al. [RBA98]. The templates are based on linguistic structures and on semantic patterns as defined by Chomsky [Ch71]. The template of Tab 2 is based on the communication pattern

*Communication (V) [Agent; Object; Source; Destination]*

that has been instantiated in the way

*Communication (*select*) [Agent: '*the librarian*'; Object:* the book*'; Source: '*the librarian*';*
*Destination: '*in the list of available books*']*

As a result of this transformation, the degree of formal definition of syntax and semantics is increased. The step between the syntax analysis and the semantic analysis is relatively large. For common natural language texts there is a wide variety of linguistic structures, demanding for a big maintenance effort. Semi-formal descriptions offer advantages because the number of linguistic structures can be limited.

## 4. Increments, Iterations and Tool Support

The goal of the transformation – an unambiguous, complete description – has to be constructed in a step-by-step procedure. Depending on the input, two basic cases can be distinguished: In the first case a domain expert is expressing the knowledge directly in templates. In the second case a written, natural-language requirements specification (i.e. by use case descriptions, in the form of Tab 1) is provided, that is transformed into the more formally-defined description mentioned above.

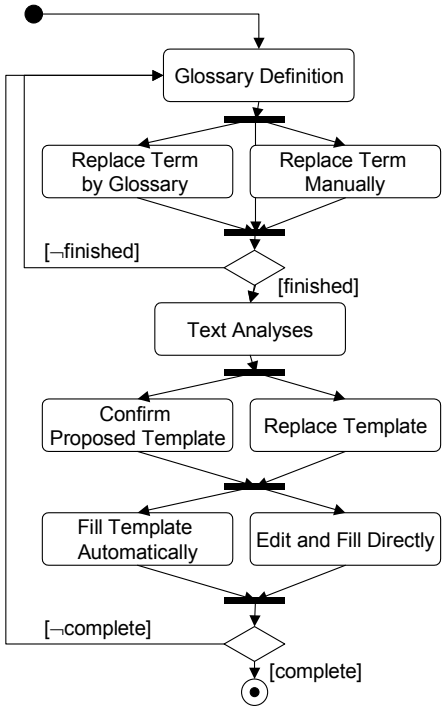


*Fig 1: Transformation of a semi-formal description*

In both cases, a base for references has to be provided by defining basic terms in a glossary as a prerequisite.

In the first case, a special editor is used. It offers template structures and sets of predefined term to fill them. Tab 2 has shown such a template. The term *the librarian* i.e. is contained in the glossary. The expert selects a proper template, then he selects a term for each part of the template. Frequently there are not enough predefined terms, or the issue requires further investigation. In such a case free terms without a reference to the glossary can be entered to complete a template. These parts have to be replaced in a next cycle. Analysis tools detect them to support their incremental formalization.

The procedure for the second case is shown by Fig 1. In a first cycle a tool compares the sections of a use case description to the glossary. For each term in a text section glossary items are proposed to replace them. The developer either confirms a replacement, selects another glossary item for replacement, or skips to a next proposal. During this first cycle, the developer collects words and phrases as candidates for a definition in the

glossary. In a next cycle, more terms can be replaced by applying the enhanced glossary. After this replacements a first fraction of the description is defined by references.

In the next step, a textual analysis is performed as mentioned above. Large lexicons and powerful sets of linguistic structures are provided by analysis tools. As a result of the analyses, templates are proposed to replace sentences without a defined structure. Some parts of a template can be filled by the tool, others have to be filled interactively by the developer. The special editor used in the first case is applied for filling a template by predefined terms.

In about 10 % of the cases, a text part cannot be understood, i.e. because the grammatical structures cannot be detected [RBA98]. In these cases, the text part is replaced by using the special editor to select and to fill a template directly.

After these steps, the description of the use cases is following a formally defined syntax, and all terms are defined by references to others. The parts *precondition, main success scenario, postcondition* and *alternative scenarios* of the use case description (Tab 1) provide at this stage the information about the dynamic behavior in a machine-exploitable form.

## 5. Application of the Results

The resulting behavioral description of the use cases serves as the input for building behavioral models. A use case description - especially its parts main success scenario and alternative scenarios - is transformed to a sequence diagram as defined by UML 2.0 [OMG]. The set of sequence diagrams of a specification is then transformed to state diagrams by applying the method of Broy et al. [BGK02]. The state diagrams describe the system behavior in a way that a tool-supported software construction is possible (Fig 2). In our case, information about the typical usage of a system is added to build usage graphs. The usage graphs are then input for the generation of test cases for statistical usage testing [Gö 2001].

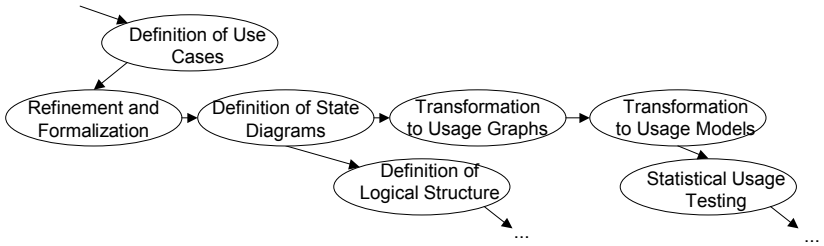


*Fig 2: Subsequent development activities after refinement and formalization of the use cases*

## 6. Related Work

A broad variety of semiformal expressions in different domains is used for structuring natural-language texts. Examples are requirements specifications i.e. in the mentioned structure of Volere [RR00] or structured comments as part of source code documentation i.e. as proposed for javadoc [ja03]. Different authors apply structured texts for use case descriptions, i.e. Regnell [RRC00].

Templates as a form of structured sentences are applied for structuring requirements specifications by the Sophist method [Ru01]. Their aim consists in reducing ambiguity and in giving structure to natural-language texts. Their special adaptation for use case descriptions and a tool-supported editing is not concerned by this method.

Powerful methods for text analyses have been developed. Information retrieval techniques are widely used in practice [SG83]. Linguistic analysis techniques for transforming natural language text are presented by various works, from Chomsky [Ch71] to the completion rules of [RBA98].

There are different methods for test case generation form object-oriented models, i.e. by Gallagher [Gal99] or Regnell [RRC00]. An approach for test case generation from structured use case descriptions via statecharts is described by Götze [Gö01].

## 7. Conclusion

Structured texts constitute the first step in the improvement of natural language descriptions towards a more precisely defined syntax and semantics. This task can be supported by information retrieval tools and techniques to some extent. To provide a base for more powerful tool support and automation, semantic patterns can be introduced. Linguistic analysis as a part of a textual analysis enables a transformation towards a formally defined syntax. The amount of interaction with the author is fairly low, and from our projects and case studies we expect a good applicability of the method in practice.

## 8. References


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

[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.

[ja03] How to Write Doc Comments for the Javadoc Tool. Sun, 2003. Online available at http://java.sun.com/j2se/javadoc/writingdoccomments

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

[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/.


Last changed May 25, 2004.