Towards Generation of Textual Requirements Descriptions from UML Models

Petr Kroha, Philipp Gerber, Lars Rosenhainer *
{kroha, pge, lro}@informatik.tu-chemnitz.de

Abstract: In this paper, we describe our novel approach to requirements validation by paraphrasing a requirements model expressed in UML by natural language (NL). After an analyst has specified a UML model based on a requirements description, a text may be generated that describes this model. Thus users and domain experts are enabled to validate the UML model, which would generally not be possible as most of them do not understand (semi-)formal languages such as UML. A corresponding text generator has been implemented which makes use of such NL generation approaches as text schemata and rhetorical relations.

Keywords: requirements validation, UML, natural language generation

1 Introduction

Before the modelling phase of the software development can start we have to acquire and collect requirements. The obtained interviews and studies result in text documents that describe requirements. Using natural language is necessary because a customer would not sign a contract that contains a requirements definition written in some formal notation (e.g. the Z notation).

After discussions, a requirements specification must be written that describes the functionality and constraints of the new system in a more detailed way. It is usually written as a combination of text and some semi-formal graphical representation given by the used CASE tool. Since software engineers are not specialists in the problem domain, their understanding of the problem is immensely difficult, especially if routine experience cannot be used. It is a known fact [12] that projects completed by the largest software companies implement only about 42 % of the originally proposed features and functions.

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

In this contribution, we offer a textual refinement of the requirements definition which can be called requirements description. Working with it, the analyst is forced by our supporting tool TESSI to complete and explain requirements and to specify the roles of words in the text in the sense of object-oriented analysis. During this process, a UML model will be built by TESSI driven by the analyst's decisions.

This model is used for the synthesis of a text that describes the analyst's understanding of the problem, i.e. a new, model-derived requirements description will automatically be generated. Now, the user has a good chance to read it, understand it and validate it. His/her clarifying comments will be used by the analyst for a new version of the requirements description. The

* Dept. of Computer Science, Chemnitz University of Technology, 09107 Chemnitz, Germany

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

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

The rest of the paper is organized as follows. In Section 2 we discuss related work. In Section 3 we briefly introduce the architecture of the new system. In Section 4 the used method is described. A case study illustrating the method is presented in Section 5. The implementation of the system is described in Section 6. Finally, achieved results are discussed and an outlook is given in Section 7.

2 Related work

In this contribution we describe the further development of the text generator in the TESSI tool which has been introduced in [5]. Its improved version has been described in [4]. Text generation in this version of TESSI was based on templates corresponding to several types of UML model elements. These templates were selected and instantiated according to the type and contents of an element of the model. We have found that the generated texts were very large and boring. Another disadvantage was that the texts we generated were often not right in the sense of grammar. Building all possible grammatical forms for all possible instantiations had been too complex. Further, the terminology used in the generated texts also included specific terms from the software engineering domain, which reduces text understandability for users and domain experts.

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

The system proposed by Dalianis [2] accepts a conceptual model as its input. A query interface enables a user to ask questions about the model, which are answered by a text generator. User rules are used for building a dynamic user model in order to select the needed information. A discourse grammar is used for creating a discourse structure from the chosen information. Further, a surface grammar is used for surface realization, i.e. for the realization of syntactic structures and lexical items.

ModelExplainer [7] is a web-based system that uses object-oriented data modeling (OODM) diagrams as starting point, from which it generates texts and tables in hypertext which may contain additional parts not contained in the model. The outgoing text is corrected and adjusted by the RealPro system [6, 1], which produces sentences in English. However, since NL generation is based on OODM diagrams alone, it is confined to static models.

3 Architecture and dataflow of the text generation system

The component for text generation is a part of the CASE tool TESSI. As mentioned above, in the first phase of requirements acquisition, a text containing knowledge about the features of the system to be developed, is written in cooperation between the analysts, domain experts, and users. The analyst processes this text and, by using the MODEL component, decides which parts of the text can be associated to which parts of the UML model. Then the GENERATE component generates a text corresponding to the UML model. Thus the text also corresponds

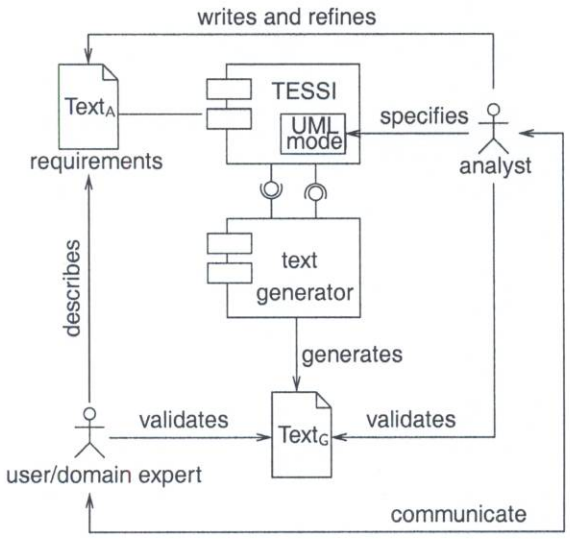

Fig. 1: Architecture and dataflow of the TESSI tool

to the analyst's understanding of the problem which is usually different from the domain expert's problem understanding. Comparing both texts, they can identify these differences and, consequently, the next development cycle can be started (see Fig. 1).

4 Text generator for use cases

A standard architecture for NL generation is the pipeline model [10]. Three modules are arranged in a pipeline where each module is responsible for one of the three typical NL generation subtasks, which include, in this order, *document planning*, *microplanning* and *surface realization*. The output of one module serves as input for the next one.

The input to the *document planner* is a *communicative goal* which is to be fulfilled by a text generation process. The communicative goal is basis for the selection of information (*content determination*) from a knowledge base. In our case, the goal is to validate a requirements model and the knowledge base is the model itself. Output of the document planner is a *document plan*, which structures the text with respect to the order in which the sentences must occur. The *microplanner* accepts a document plan as its input and determines the content of single sentences. *Sentence plans* are produced, which specify what propositions are to be aggregated (to avoid redundant information), what lexems (lexicalization), and what referring expressions (if two or more propostions refer to the same object) are to be used in each sentence. A sentence plan is transformed into the actual text (*surface text*) of a certain target language by a *surface realizer*. It determines word order, adds function words such as 'and' or 'or', and adapts the morphological features of lexems (e.g. endings).

As to the construction of the TESSI text generator, we have so far concentrated on the steps document planning and surface realization. Since the microplanning step is currently only implemented in a rudimentary way and surface realization is mainly based on the RealPro [1] tool, we will only describe our approach to document planning in this paper.

Two basic approaches to document planning have been used for TESSI, which are *text schemata* and *rhetorical relations*. Text schemata [9] lay down the structure of certain texts. Dependent on a given text type (e.g. weather forecast, UML model description) it is possible

to define templates (schemata) for single sections or a whole text. A schema definition may be recursive and may contain structuring elements such as alternatives, optional constituents, and repetitions [9, 11]. Document planning by means of rhetorical relations is based on the rhetorical structure theory, an approach to structuring texts independent of their text types.

Rhetorical structure theory (RST) [8] assumes that, in a coherent text, each text segment fulfills a certain function. A text segment may comprise a sentence part, several sentences or even whole paragraphs. The function results from the coherence to other text segments and is described by relations (*rhetorical relations*) between the segments. Two non-overlapping text segments connected by a relation form a new segment. This can be continued until the whole text is arranged into a hierarchical structure. Relations describe both semantic functions of text segments and intent behind text segments.

Examples for rhetorical relations include *Elaboration* and *Sequence*. *Elaboration* describes a relationship between a superordinate text segment (*nucleus*) and a subordinate text segment (*satellite*). Relations of this kind are denoted as mono-nuclear relations. While a nucleus contains the central information, a satellite contains supplemental information related to the nucleus. *Sequence* is an example for multi-nuclear relations where there is an equal relationship between text segments.

For the implementation of the document planner, a hybrid solution has been preferred, which consists of text schemata in combination with rhetorical relations. The usage of schemata is reasonable since text generation is limited to only one text type, namely UML model descriptions. Further, UML models only contain elements which are used in pre-defined diagram types. As the set of diagram types is limited, it is consequently possible to define a schema for each of them. Thus, a schema is responsible for content determination and the structure of a section of the document plan. For example, the schema **SchemaDocument** partitions the text into several chapters each of which describes a certain UML diagram type; the schema **SchemaStateMachine** produces a section for a state machine. A section of a document plan contains subsections and/or leaf nodes containing information on UML model elements, which are connected with rhetorical relations. Examples for relations include **SEQ_TEMPORAL** to represent a temporal order, **ELABORATION** for additional information, **ELAB_ASSOCIATION** for the description of relations, **ELAB_PART** for the description of an object's parts, and **ELAB_SPECIALIZATION** for the description of specialization relations.

5 Case study

To illustrate our requirements validation approach, we now apply it to a contrived specification of a library automation system. In the specification, we are able to identify two primary actors, which are *user* and *librarian*. They make use of the following use cases: *search medium*, *borrow medium*, *return medium*, *reserve medium*, *extend rental period of medium*, *manage medium*, and *manage user*. For the sake of the example, we consider the use case depicted in Tab. 1 (*extend rental period of medium*) in more detail.

5.1 Specify UML model based on requirements text

Supported by TESSI (see Fig. 2), the following model elements relevant for the *Extend rental period of medium* use case were manually derived from the complete requirements text:

- classes: *media administration*, *medium*, *medium instance*, *user account*, *user administration*
- attributes:

4.4 Use Case: Extend rental period of medium The user has to identify himself to the system and specifies a medium instance whose rental period he would like to extend. The return date is updated according to the new rental period if it does not exceed the maximum rental period and if it is not reserved by another user. Basic course: 1. User indicates his wish to extend the rental period of one or more medium's instances. 2. System prompts the user to identify himself. 3. User identifies himself to the system. 4. System validates that the user account is valid and is not locked. 5. System presents a view containing a list of medium instances the user has currently borrowed. 6. User selects an instance whose rental period he wants to extend. 7. System validates that the selected instance is not reserved and that its maximum rental time will not be exceeded. 8. System extends the instance' rental period and adds the extension time to the instance's total time for this rental.

Tab. 1: *Extend rental period of medium* use case

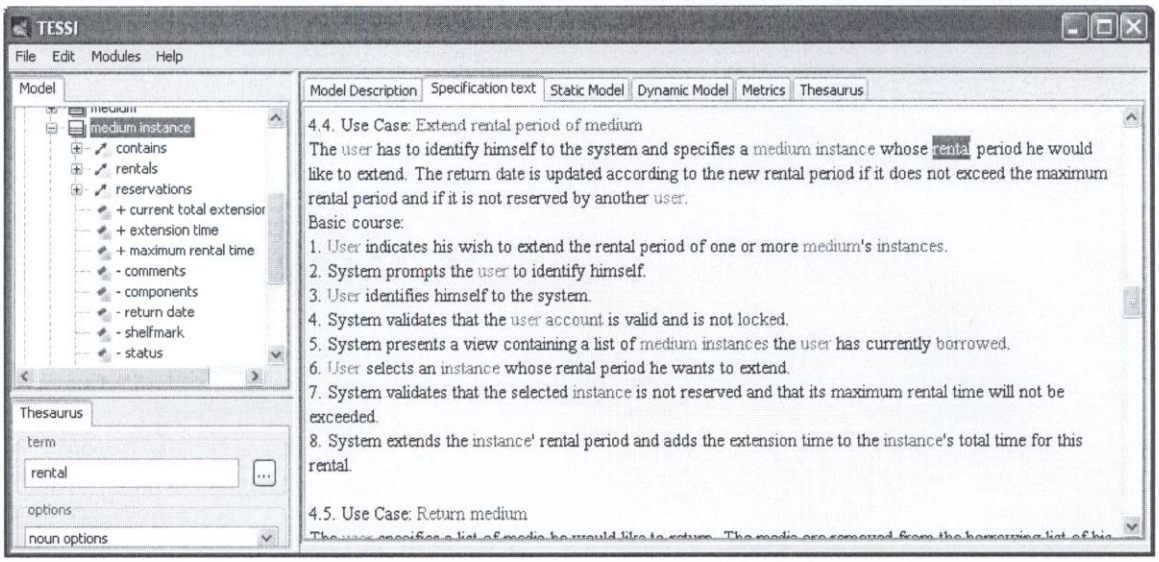

Fig. 2: Specification text in TESSI

  - *user account*: *user number*, *valid until*
  - *medium*: *title*, *authors*
  - *medium instance*: *return date*, *shelfmark*, *status*, *maximum rental time*, *extension time*, *current total extension time*
- operations:
  - *user account*: *islocked*
  - *medium instance*: *isRentalExtensionAllowed*
- associations:
  - between *user account* and *medium instance*: *rentals*, *reservations*
  - between *medium* and *medium instance*: *contains*

- between *user administration* and *user account*: *managesUsers*
- between *medium administration* and *medium*: *managesMedia*

– collaborations: interaction *basic course* of collaboration *extend rental period of medium* (assigned to the use case of the same name)

5.2 Generate natural language text from UML model

For the purpose of paraphrasing the specified UML model for users and domain experts, an NL text is generated from the UML model. For example, to provide an overview over the system to be developed, the partial section reproduced in Tab. 2 has been generated. In the corresponding document plan, an enumeration of use cases, being the satellites, are related to the nucleus *Library system*. The satellites' rhetorical relation is **ELAB_PART**, which represents the fact that the enumerated use cases are parts of the *Library system*.

The Library system has seven tasks. • reserve medium • borrow medium • extend rental period of medium • search medium • return medium • manage user • manage medium

Tab. 2: Generated text containing the system's use cases

Tab. 3 contains a reproduction of the generated section for interaction *basic course* of collaboration *extend rental period of medium* (for the original see Fig. 3). In the document plan, the multi-nuclear relation **JOIN** has been used for list compilation, which can represent the viewpoint that all interaction step are nuclei occurring equally side by side.[1]

6 Implementation

TESSI is a Java/Swing application which provides a mechanism for easy integration of new modules. Thus the NL generation component has been developed as a TESSI module and integrated into TESSI. While the steps document planning and microplanning have been freshly implemented by respective sub-components, a third sub-component responsible for surface realization mainly uses RealPro [1] for this task.

7 Achieved results and outlook

An architecture has been designed and implemented which serves as an important basis for sophisticated NL text generation with the purpose of validating requirements analysis models. The text generator performs text generation in three consecutive steps: document planning,

[1] It can of course be argued whether the usage of, for example, **SEQ_TEMPORAL** might be a better choice so as to express temporal relationships between interaction steps.

Interaction basic course
1. The user sends 'extend rental period' to the user administration.
2. The user administration sends 'identify yourself by user number' to the user.
3. The user sends 'user number (= x)' to the user administration.
4. The user administration sends 'get account with user number = x' to the user administration.
5. The user administration sends 'user account (with user number=x)' to the user administration.
6. The user administration sends 'Are you valid?' to the user account.
7. The user account sends 'yes' to the user administration.
8. The user administration sends 'Are you locked?' to the user account.
9. The user account sends 'no' to the user administration.
10. The user administration sends 'get list of borrowed medium instances' to the user account.
11. The user account sends 'list of borrowed medium instances' to the user administration.
12. The user administration sends 'list of borrowed medium instances' to the user.
13. The user sends 'shelfmark of a medium instance (= y)' to the media administration.
14. The media administration sends 'get medium instance with shelfmark = y' to the media administration.
15. The media administration sends 'medium instance (with shelfmark=y)' to the media administration.
16. The media administration sends 'Are you reserved?' to the medium instance.
17. The medium instance sends 'no' to the media administration.
18. The media administration sends 'Is rental extension allowed?' to the medium instance.
19. The medium instance sends 'Is current total extension time + new extension time <= maximum rental time?' to the medium instance.
20. The medium instance sends 'yes' to the medium instance.
21. The medium instance sends 'No' to the media administration.
22. The media administration sends 'extend rental period' to the medium instance.
23. The medium instance sends 'new return date = old return date + extension time' to the medium instance.
24. The medium instance sends 'new current total extension time = old current total extension time + extension time' to the medium instance.

Tab. 3: Generated text corresponding to the *Extend rental period of medium* use case

microplanning (which has so far only been implemented in a rudimentary way) and surface realization. Compared to texts produced by the pre-existing template-based text generator of TESSI, texts generated by the new non-trivial text generator are definitely more structured, more clearly arranged and more readable. Further, the vocabulary used should be more understandable for people outside the software industry. Due to the usage of RealPro for surface realization, the grammar of generated sentences is also more correct than before.

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

A number of open issues may be addressed in the future: improvement of the microplanner, improvement of text generation for already considered model elements (e.g. inclusion of object names in collaborations), integration of text schemata for other model elements (such as static structures like classes). Further it is desirable to evaluate our proposed validation approach by application to real-world projects.

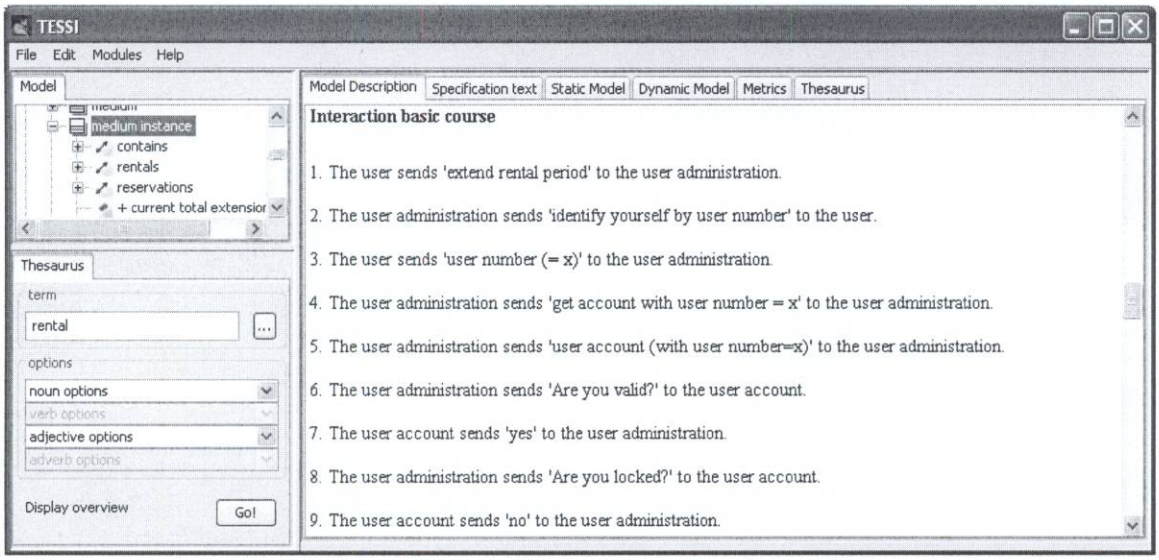

Fig. 3: Generated text in TESSI

Bibliography

1. CoGenTex: *REALPRO: General English Grammar, User Manual*. CoGenTex, Inc., 840 Hanshaw Road, Suite 2, Ithaca, NY, 14850, 2000.
2. Dalianis, H.: A Method for Validating a Conceptual Model by Natural Language Discourse Generation. In: Loucopoulos, P., ed., *Proc. Advanced Information Systems Engineering, CAiSE'92, Manchester, UK, May 12-15, 1992*, volume 593 of *Lecture Notes in Computer Science*, pp. 425–444, Springer, 1992.
3. Gerber, P.: Textgenerierung zur Beschreibung von UML-Modellen. Diplomarbeit, Fakultät für Informatik, Technische Universität Chemnitz, 2005, in German.
4. Kroha, P.: Preprocessing of Requirements Specification. In: Ibrahim, M., Küng, J., and Revell, N., eds., *Proc. 11th Int. Conf. on Database and Expert Systems Applications (DEXA'2000), London, UK, September 4-8, 2000*, volume 1873 of *Lecture Notes in Computer Science*, Springer, 2000.
5. Kroha, P. and Strauß, M.: Requirements Specification Iteratively Combined with Reverse Engineering. In: Plasil, F. and Jeffery, K. G., eds., *SOFSEM'97: Theory and Practice of Informatics*, volume 1338 of *Lecture Notes in Computer Science*, Springer Verlag, 1997.
6. Lavoie, B. and Rambow, O.: A Fast and Portable Realizer for Text Generation Systems. In: *Proc. Fifth Conference on Applied Natural Language Processing (ANLP97), Washington, DC*, pp. 265–268, 1997.
7. Lavoie, B., Rambow, O., and Reiter, E.: The ModelExplainer. In: *Demonstration Notes of International Natural Language Generation Workshop, INLG'96, Harmonceux Castle, Sussex, UK*, 1996.
8. Mann, W. C. and Thompson, S. A.: Rhetorical Structure Theory: Toward a Functional Theory of Text Organization. *Text*, 8(3):243–281, 1988.
9. McKeown, K. R.: *Text Generation: Using Discourse Strategies and Focus Constraints to Generate Natural Language Text*. Cambridge University Press, Cambridge, 1985.
10. Reiter, E. and Dale, R.: *Building Natural Language Generation Systems*. Studies in Natural Language Processing, Cambridge University Press, 2000.
11. Stede, M.: Textgenerierung. In: Lobin, H. and Lemnitzer, L., eds., *Texttechnologie - Perspektiven und Anwendungen*, Stauffenburg, Tübingen, 2004, in German.
12. The Standish Group, ed.: *The CHAOS Report*. The Standish Group, 1995.