Synchronizing Domain Models with Natural Language Specifications

Mathias Landhäußer, Sven J. Körner, Walter F. Tichy
Karlsruhe Institute of Technology, IPD Tichy, Faculty of Computer Science,
{landhaeusser,sven.koerner,tichy}@kit.edu

***Abstract*—Textual specifications and domain models change during development and need to be kept consistent. However, in practice the cost of maintaining consistency is too high. Stakeholders need to be informed about model changes in natural language, software architects need to see the impact of specification changes on their models. Our *Requirements Engineering Feedback System* (REFS) automates the process of keeping specification and models consistent when the models change. Also, it can assess the impact of specification changes.**

I. Introduction

Most of the time, requirements documents and domain descriptions are provided in natural language [1]. The first steps of a software project comprise the transfer of natural language specifications into semi-structured documents. Eventually, models evolve that are used in the subsequent software development process. This transfer from natural language text to models (e.g. UML models) creates two separate representations – one in written text and one as models. Initially (and ideally) both representations of the future system are equivalent, but this is not always the case. A modification in one representation should be accompanied by a change in the other. If synchronization is not maintained in the development process, the representations diverge from another. Our *Requirements Engineering Feedback System* (REFS) records model changes and includes them into the specification text automatically; furthermore, we show, how the impact of specification changes in the domain model can be measured.

A. Text to Model Synchronization

Starting with a textual representation of the stakeholders' needs and wishes, an expert builds a model. Following model driven development, one starts with a domain model and refines it. This way, the domain model becomes a software model (first platform independent, then platform specific) and finally executable software. Gelhausen presented an approach to generate UML domain models directly from textual specifications [2], [3]. Following his approach, the first step of model driven development can be automated.

Often, stakeholders change the textual software specification whilst the software is already in the making [4]. The direct approach would be to generate a new model after modifications. This approach leads to information loss if work on the existing models has already begun. It is the requirements analyst's job to assess the impact of the changes on the existing domain model. Ideally, one calculates the necessary changes in the models and keeps the unchanged parts of the models as well as the other software artifacts that have been created already. Our aim is to provide a fast and accurate evaluation of the situation when changes occur. We help the analyst to decide if changes are worthwhile or if they imply an overhaul of the architecture and the implementation.

B. Model to Text Synchronization

However, models also change. Experts work with the domain model, refining, consolidating, and correcting it before transforming it into the platform independent model. The changes, additions, and deletions on the domain model should be transferred back to the textual specification enabling the stakeholders to evaluate the experts' work. Keeping track of changes manually is cumbersome and therefore often neglected. It is not until software projects reach the final phase when stakeholders realize and inform the manufacturer of shortcomings and misunderstandings of software features. Extending Gelhausen's work [3], we developed a system that tracks model changes and reflects them in the textual specification. Then stakeholders can evaluate the expert's model changes directly in the textual specification.

II. Related Work

In this section, we focus on related work that supports the connection between textual specifications and their model representation. Also we review some work on impact analysis.

A. Automatic Model Creation and Text Synthesis

Overmyer's Linguistic Assistant for Domain Analysis (LIDA) [5] is a tool for requirements engineers who want support in the iterative requirements engineering process. LIDA analyzes the lexical content of natural language specifications. It identifies and marks lexical items corresponding to candidate model elements. The analyst creates the UML model according to model elements proposed by LIDA. It supports the analyst with a well-arranged document that he or she can use to extract the system domain model. Comparing the feedback components, it generates a new specification text instead of maintaining the initial specification. In contrast to LIDA, our approach keeps the connection between the specification and the model representation without re-generating the specification.

Kroha's TESSI [6] is another iterative-capable automatic model generator. TESSI helps the analyst to complete requirements. The analyst needs to specify the roles of words in the text. The problem is that the analyst needs to know every role of every word during the modeling process, because incomplete role arguments lead to incomplete UML models. The model is then used to synthesize a new specification from the model, i.e. to provide a model-derived requirements description. Again, the original specification is not updated but discarded. With SUGAR [7] Deeptimahanti et al. offer a tool to extract models from text. Unfortunately, changes in the specification text require a rerun of the process and models from the previous run are discarded. Model changes are not fed back to the text.

Mala's system [8] uses a NLP pipeline to generate a model without the help of a domain expert. Mala states that the yielded results are at least as good as or exceeding human made class diagrams. Other tools that extract models from natural language with the phrase pattern approach come from Fliedl [9] and Li [10]. Bajwa's UMLG [11] extracts nouns and verb combinations from input texts and maps the nouns and verbs to UML elements and relations respectively. Unfortunately, none of these tools support iteration or impact analysis.

Adding new information to the model needs to be expressed in the textual specification, too. An example would be a new class element that has been added to the UML domain model. In this case, natural language would have to be generated from the model. Research projects from Reiter, Meziane, and Kroha focus on this [12], [13], [14]. But still, this cannot be considered as synchronization between model and textual specification rather than document generation from models. There is no direct connection to the initial specification.

B. Impact Assessment

Being able to determine the impact of changes on a software specification is a well-known problem that has existed ever since software development became an industry. Bohner and Arnold [15] define impact analysis as "identifying the potential consequences of a change, or estimating what needs to be modified to accomplish a change". To be able to do that, one has to maintain traceability among various entities of the software development process. Also one has to detect possible side and ripple effects of the changes.

Kung et al. use impact analysis to focus testing efforts on hot spots [16]. They describe a formal model to identify changes and their impact on an object-oriented software library. Chaumun et al. use impact analysis methods to assess maintainability [17].

Most impact analysis approaches focus on changes of the program code whereas Han used dependencies defined between software artifacts to identify the impact of a change [18]. Briand et al. [19] propose a tool that uses a set of OCL constraints to detect the differences between two versions of an UML model and their impact on unchanged model elements. To the best of our knowledge, there are no systems, that deal with impact analysis on models for changes in a textual specification.

III. Implementation

Our implementation uses Gelhausen's [2] approach to derive a UML model from textual specifications. He uses a graph as an intermediate representation of the text. The graph contains typed nodes for sentences and words. Edges stand for thematic roles and are the heart of his approach because they represent the semantic information given in the text. The roles are based on the work of Fillmore [20] and were adapted for the purpose of model extraction. Graph transformation rules are then used to build a UML representation [3]. Fig. 1 illustrates the graph representation. The left part of Fig. 1 shows the text subgraph for the phrase `The WHOIS client makes a text request to the WHOIS server, then the WHOIS server replies with text content`. The right part shows an excerpt of the corresponding UML subgraph; typed edges connect the class and method nodes.

To implement REFS, we extended Gelhausen's model extraction process. We added tracking edges that connect text nodes with UML nodes; the tracking edges are added to the graph during the UML generation and are shown as dashed lines in Fig. 1. This way, the UML representation is tightly linked with the underlying text. To account for possible repetitions in the text, we allow a given UML element to be linked with multiple text nodes. We store the original specification and the connected model together to reflect model changes back to text later on.

A. Transferring Model Changes to the Specification

After the first UML models have been created from the textual specification, software architects make design decisions, rearrange UML model elements, group parts of the models, create superclasses, and so on. These changes are carried out with UML tools. Essentially, updates, creations, and deletions occur. Updates on relations are being treated as a combination of deletions and creations. Simple name updates are reflected directly into the text.

To reflect model changes to the specification, the initial model (M_i) and the changed model (M_c) need to be compared and matched. We use EMFCompare of the Eclipse Modeling Framework to compute the difference between M_i and M_c. EMFCompare identifies model elements of M_i that also occur in M_c by hierarchically (type-aware) matching model elements: At first, the name of the model element is considered; then, all references to other elements are being examined. After that, the attributes are analyzed

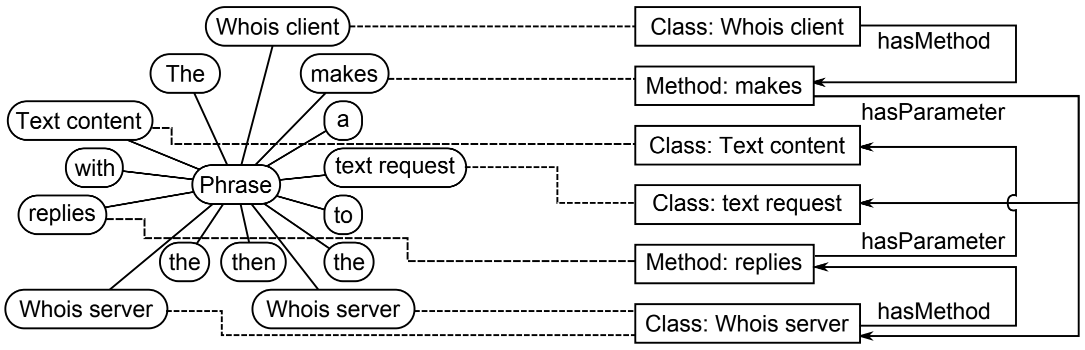


Figure 1. Tracking Edges Connect the Textual Nodes With their UML Counterparts.

and finally the type of the element (i.e. the meta-model type) is considered. Every comparison gives a similarity value between 0 and 1. After comparing names, EMFCompare sums up the weighted combination of the values. If the sum is above a certain threshold, the elements are considered equal.

All elements that are being matched are either unchanged or can be identified for modification. All other elements of M_i have been removed in M_c; all unmatched elements of M_c are considered new. This way, we create a list of creations, updates, and deletions and successively integrate them into the original specification. Changes are processed in a create/update/read sequence to assure the correct order of changes without ripple effects. Changed and deleted elements can be identified in the text graph using the tracking edges for updating the text or removing the text elements. New elements are appended to the specification using simple templates; readability could be increased using more sophisticated approaches. At the end of the process, a modified specification text can be generated; parts not affected by the model changes remain untouched during the text modification.

The updated specification can be handed over to the stakeholders, who can also use text comparison to review the changes. A list of changes can be used for a quick overview. Fig. 3 shows an updated specification after some modifications (see Sec. IV).

B. Impact Assessment of Requirements Changes

The idea of keeping the interconnection of textual specifications and the corresponding models also works from text to model. With the introduction of the bidirectional connection of text and model, we are able to detect the results of textual changes in the corresponding model. To retrieve this information, we create an UML model of the initial specification and of the altered specification. Then we compute the differences between the initial and the altered model. Inspired by the function point method, we attach a weight-factor to every UML change. Adding these factors, we assess the impact of specification changes to models.

```
A WHOIS server listens on TCP port 43 for
  requests from WHOIS clients.
The WHOIS client makes a text request to the
  WHOIS server, then the WHOIS server replies
  with text content.
All requests are terminated with ASCII CR and
  then ASCII LF.
The response might contain more than one line of
  text, so the presence of ASCII CR or ASCII LF
  characters does not indicate the end of the
  response.
The WHOIS server closes its connection as soon as
  the output is finished.
The closed TCP connection is the indication to
  the client that the response has been received.
```

Listing 1. The Whois Protocol Specification (IETF RFC 3912)

This technique is straight forward for additional text, but also works for text changes and deletions. This can mean anything from changing a class name to altering the parameter list of a class' method. Deletions are simpler – we check if the deleted text element appears anywhere else in the text. If not, the corresponding model element is deleted.

Reordering the sentences of a specification has no effect on the model an thus no impact at all.

IV. Case Study

At first, we show how REFS transfers model changes back to a specification text. Then we explain, how model differencing can help determine the impact of specification changes on UML domain models. To illustrate our approach, we work with the WHOIS protocol specification (IETF's RFC 3912) as printed in Listing 1.

Model to Text Synchronization: To exemplify the model to text synchronization, we use a generated class diagram for the WHOIS protocol specification. The diagram (see Fig. 2 for an excerpt) has been generated from the unmodified specification as shown in Listing 1. Model elements can be updated, deleted, or created. For our example, we change the model as follows: We delete the class `ASCII_LF` and its members as well as the class `text_content` and the corresponding parameter of the method `replies`. Also,

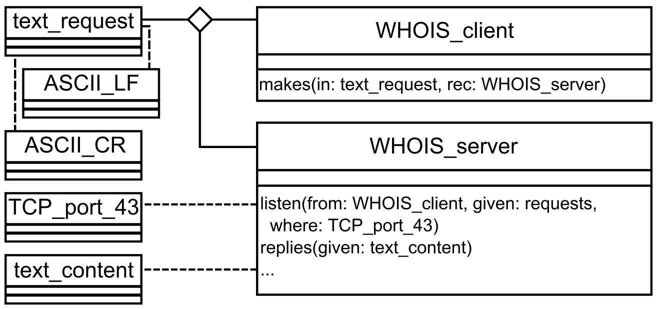
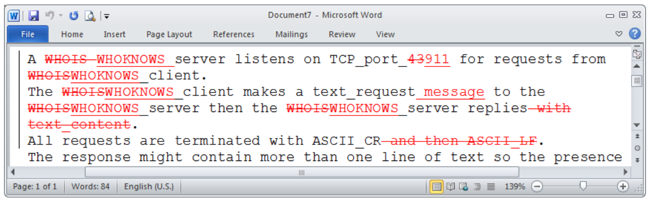

text_request
ASCII_LF
ASCII_CR
TCP_port_43
text_content
WHOIS_client
makes(in: text_request, rec: WHOIS_server)
WHOIS_server
listen(from: WHOIS_client, given: requests, where: TCP_port_43)
replies(given: text_content)
...

Figure 2. An Excerpt of the Generated UML Class Diagram for the WHOIS Protocol Specification.



A ~~WHOIS~~ WHOKNOWS_server listens on TCP_port_~~43~~911 for requests from ~~WHOIS~~WHOKNOWS_client.
The ~~WHOIS~~WHOKNOWS_client makes a text_request_message to the ~~WHOIS~~WHOKNOWS_server then the ~~WHOIS~~WHOKNOWS_server replies ~~with text_content~~.
All requests are terminated with ASCII_CR ~~and then ASCII_LF~~.
The response might contain more than one line of text so the presence

Figure 3. Using Microsoft Word to Present Model Changes in Textual Specifications

we rename the `WHOIS_server` and `WHOIS_client` to `WHOKNOWS_server` and `WHOKNOWS_client` respectively. Furthermore, we want the server to listen on `TCP_Port_911` instead of `43`. With this modified model, we run REFS to transfer the changes back to text. The resulting text and the comparison to the original text are shown in Fig. 3. We show the first three sentences only, but a deletion in the model can lead to multiple deletions in the textual specification; also the renaming of server and client is propagated to the entire specification.

Text to Model Synchronization: Assume the last two sentences of the WHOIS specification are missing in the initial specification. If a stakeholder now enters this additional information, the model has to be extended. Elements are not modeled repeatedly: if already existing model elements appear in new text, they are reused. Tab. I shows the detected natural language elements from the last two sentences. The right column shows the UML model elements that were detected and added if not already existing.

Treating Updates and Creations: Until now, we have described the mutual synchronization process when parts of the UML model are deleted or parts of text are added to the specification. Of course, our approach also allows modifications, deletions of text, and the creation of new model elements. For newly created model elements, we need natural language generators to add the changes to the specification. So far, we use only simple templates to create sentences [21]. Updates of model elements and text passages are handled in a similar manner. Updates are usually treated as deletions followed by creations. A

Table I
TEXT ADDITIONS CREATE NEW UML MODEL ELEMENTS UNLESS THEY ARE ALREADY EXISTING.

Text Addition	**UML Model Element**
WHOIS server	class (already existing)
closes	method of class WHOIS_server
connection	class
output	class
is finished	method
closed	attribute of class connection
indication	method of an indetermined class
WHOIS client	class (already existing)
response	class (already existing)
received	method of class response
closes	method of class WHOIS_server

few exceptions update the text or model elements directly thereby preserving contextual information. If applicable, we prefer deleting and creating objects to avoid orphans.

Evaluating Random Modifications: To assess, whether our approach provides a viable feedback loop between models and textual specifications, we conducted a small study. We used three specifications where we applied random modifications; one member of our team modified the texts, another independently modified the models using Altova UModel. Both randomly determined the elements to be modified or deleted. The complete specifications can be found on our website [22] with a detailed report.

Tab. II shows an excerpt of the results: Every entry a/b states how many random modifications (b) have been made and how many modifications have been correctly transferred in the opposite direction (a). The 3rd and 4th columns show the results of the analysis of text modifications on the UML models; columns 5 and 6 show the number of model changes correctly transferred to the specification. For example, in the Timbered House specification, we made three modifications to the text and deleted seven words from the it; all three updates have been correctly mapped to the corresponding model elements and two of the deletions had an effect on the model. As can be seen, the random text deletions (and updates) sometimes modified elements that were detected but irrelevant for a UML class diagram. Elements that are not used for the automatic model creation are omitted in the REFS feedback. Furthermore, we made three updates to the model and deleted seven model elements; all changes were correctly transferred to the text. The model to text feedback is not yet perfect: One update to the model was incorrectly identified as a deletion and a creation; this information was transferred to the specification, but the resulting text's readability was reduced. Also, Altova UModel creates extra tool specific packages. These packages are detected, but are irrelevant for the feedback loop. Therefore, they are removed before processing the model with REFS.

Table II
RESULTS OF THE RANDOM MODIFICATIONS EXPERIMENT

Case Study		Text to Model		Model to Text	
Text	Size	Updates	Deletions	Updates	Deletions
Cinema	153 Words	1/6 + 5 irrel.	6/7 + 1 irrel.	4/4	7/7
Timbered House	88 Words	3/3	2/7 + 5 irrel.	3/3	7/7
WHOIS Protocol	100 Words	8/8	2/2	5/6 + 1 incorr. crea/del	2/3

V. CONCLUSION

In this paper we presented a novel approach to synchronizing changes in UML model representations with textual specifications elicited from stakeholders. We explained how model changes can occur and how we transform these changes back into the textual specification as well as the other way around. We deliver the changed specification to the stakeholder for verification in an easy to read format, such as Microsoft Word. For the first implementation we concentrated on class diagrams only and the current implementation of our text generator is rudimentary. REFS also detects changes in activity diagrams, but we have not implemented the feedback loop yet. In future, we expect serious impact and true benefits if stakeholders can easily check if the new specification complies with their idea of the software to be implemented.

REFERENCES

[1] L. Mich, M. Franch, and P. Inverardi, "Market research for requirements analysis using linguistic tools," *Requir. Eng.*, vol. 9, pp. 40–56, 2004.

[2] T. Gelhausen and W. F. Tichy, "Thematic Role Based Generation of UML Models from Real World Requirements," in *Proc. of the ICSC 2007*, 2007, pp. 282–289.

[3] T. Gelhausen, B. Derre, and R. Geiß, "Customizing grgen.net for model transformation," in *Proc. of GRaMoT '08*. ACM, 2008, pp. 17–24.

[4] K. E. Wiegers, *Software requirements : practical techniques for gathering and managing requirements throughout the product development cycle*, 2nd ed. Redmond, WA: Microsoft Press, 2003.

[5] S. P. Overmyer, B. Lavoie, and O. Rambow, "Conceptual modeling through linguistic analysis using LIDA," in *Proc. of the ICSE '01*. Washington, DC, USA: IEEE Computer Society, 2001, pp. 401–410.

[6] P. Kroha, "Preprocessing of requirements specification," vol. 1873, pp. 675–684, 2000.

[7] D. K. Deeptimahanti and R. Sanyal, "An innovative approach for generating static UML models from natural language requirements," in *Advances in Software Engineering*, ser. Communications in Computer and Information Science, vol. 30. Springer, 2009, pp. 147–163.

[8] G. S. A. Mala and G. V. Uma, "Automatic construction of object oriented design models [UML diagrams] from natural language requirements specification," in *PRICAI*, 2006, pp. 1155–1159.

[9] G. Fliedl, C. Kop, and H. C. Mayr, "Recent results of the NLRE (natural language based requirements engineering) project," *EMISA Forum*, vol. 24, no. 1, pp. 24–25, 2004.

[10] K. Li, R.G.Dewar, and R.J.Pooley, "Towards Semi-automation in Requirements Elicitation: mapping natural language and object-oriented concepts," in *RE05*, 2005, pp. 5–7.

[11] I. S. Bajwa and M. A. Choudhary, "Natural language processing based automated system for uml diagrams generation," in *The 18th Saudi National Computer Conf. on computer science (NCC18)*. Riyadh, Saudi Arabia: The Saudi Computer Society (SCS), Mar. 2006.

[12] F. Meziane, N. Athanasakis, and S. Ananiadou, "Generating natural language specifications from UML class diagrams," *Requir. Eng.*, vol. 13, no. 1, pp. 1–18, Jan. 2008.

[13] P. Kroha, P. Gerber, and L. Rosenhainer, "Towards generation of textual requirements descriptions from UML models." pp. 31 – 38, Apr. 2006.

[14] E. Reiter and R. Dale, *Building Natural Language Generation Systems*, ser. Natural Language Processing. Cambridge University Press, 2000.

[15] S. A. Bohner and R. S. Arnold, "An introduction to software change impact analysis," in *Software Change Impact Analysis*. IEEE Computer Soc. Press, 1996, pp. 1–26.

[16] D. Kung, J. Gao, P. Hsia, F. Wen, Y. Toyoshima, and C. Chen, "Change impact identification in object oriented software maintenance," in *Proc. of the Int. Conf. on Software Maintenance*, Sep. 1994, pp. 202–211.

[17] M. A. Chaumun, H. Kabaili, R. K. Keller, and F. Lustman, "A change impact model for changeability assessment in object-oriented software systems," *Science of Computer Programming*, vol. 45, no. 2-3, pp. 155 – 174, 2002.

[18] J. Han, "Supporting impact analysis and change propagation in software engineering environments," in *Proc. of the 8th IEEE Int. Workshop on Software Technology and Engineering Practice*, Jul. 1997, pp. 172–182.

[19] L. C. Briand, Y. Labiche, and L. O'Sullivan, "Impact analysis and change management of uml models," Carleton University, Technical Report SCE-03-01, Feb. 2003.

[20] C. J. Fillmore, "Toward a modern theory of case," in *Modern Studies in English*, D. A. Reibel and S. A. Schane, Eds. Prentice Hall, 1969, pp. 361–375.

[21] B. Derre, "Rückkopplung von Softwaremodelländerungen in textuelle Spezifikationen," Master's thesis, Karlsruhe Institute of Technology, May 2010.

[22] S. J. Körner, M. Landhäußer, T. Gelhausen, and B. Derre, "RECAA – the Requirements Engineering Complete Automation Approach." [Online]. Available: https://svn.ipd.uni-karlsruhe.de/trac/mx