From Requirements to Models: Feedback Generation as a Result of Formalization*

Leonid Kof and Birgit Penzenstadler

Fakultät für Informatik, Technische Universität München,
Boltzmannstr. 3, D-85748, Garching bei München, Germany
{kof,penzenst}@informatik.tu-muenchen.de

Abstract. Natural language is the main presentation means in industrial requirements documents. In addition, communication between the different stakeholders is often insufficient, therefore requirements documents are frequently incomplete and inconsistent. This causes problems during modeling or programming.

The aim of the presented paper is to make deficiencies in behavior specifications apparent in the early project stage. The basic idea is to model the required system behavior and to generate feedback for human analysts, based on the deficiencies of the resulting models. The presented feedback generation was evaluated in an experiment. It was found that it can address genuine problems of requirements documents.

Keywords: Requirements Engineering, Model Extraction, Feedback Generation.

1 Requirements Documents Suffer from Missing Information

At the beginning of a software project, the requirements of different stakeholders are usually gathered in a document. The majority of these documents are written in natural language, as the survey by Mich et al. shows [1]. Diversity of stakeholders and insufficient communication results in imprecise, incomplete, and inconsistent requirements documents, because precision, completeness and consistency are extremely difficult to achieve using mere natural language as the main presentation means.

In software development, the later an error is found, the more expensive its correction. Thus, it is one of the goals of requirements analysis to find and to correct the deficiencies of requirements documents. A practical way to detect errors in requirements documents is to convert informal specifications to system models. In this case, errors in documents would lead to inconsistencies or omissions in models, and, due to the more formal nature of models, inconsistencies and omissions are easier to detect in models than in textual documents.

Although there exist a number of automatic approaches that analyze specifications written in natural language and provide a model, the existing approaches go in one direction only: they transform a textual specification into a formal model. However, in the case that the specification exhibits some deficiencies, they either heuristically compensate these deficiencies or fail silently.

* This work was supported by the German Research Council (DFG), Grant BR 887/26-1.

H. Mouratidis and C. Rolland (Eds.): CAiSE 2011, LNCS 6741, pp. 93–107, 2011.

Contribution: The goal of the presented paper is to show, how the deficiencies of the models resulting from the text can be used to generate feedback for human analysts. The feedback can be presented in two forms: (1) in natural language and (2) by special markings on the produced models. The effectiveness of the generated feedback was evaluated in an experiment and it was found that the generated feedback can address genuine problems of requirements specifications that would be overseen by human analysts.

Outline: The remainder of the paper is organized as follows: Section 2 presents our approaches to text-to-model translation, used as the basis for the presented work on feedback generation. Sections 3 and 4 are the technical core of the paper, they present the feedback generation and its evaluation. Finally, Section 5 gives an overview of related work and Section 6 summarizes the paper.

2 From Text to Models: Our Existing Approaches

In our survey of existing modeling techniques [2] it was shown that all existing industrially relevant formalisms are based either on interaction sequences or on finite automata. For this reason, the target model types for the behavior modeling are either finite automata or Message Sequence Charts (MSCs), serving as a representative for interaction-based modeling techniques. The translation from text to MSCs is presented in Section 2.1, and the translation to finite automata in Section 2.2.

2.1 From Scenarios to Message Sequence Charts

Translation of textual scenarios to message sequence charts was presented in [3,4]. For the translation we assume that every message sequence chart (MSC) consists of a set of *actors*, a sequence of *messages* sent and received by these actors, and a sequence of *conditions* (or *assertions*) interleaved with the message sequence. This terminology is illustrated in Figure 1.

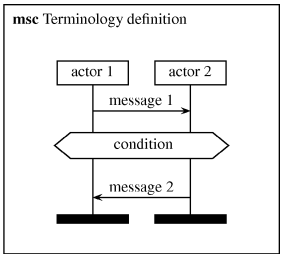

Fig. 1. MSCs, terminology

The basic idea of the scenario-to-MSC translation can be illustrated on the following scenario, taken from the Instrument Cluster Specification [5]:

1. The driver switches on the car (ignition key in position ignition on).
2. The instrument cluster is turned on and stays active.
3. After the trip the driver switches off the ignition.
4. The instrument cluster stays active for 30 seconds and then turns itself off.
5. The driver leaves the car.

A possible manual translation of this scenario to an MSC is shown in Figure 2. There are two challenge/response interactions in this MSC: The instrument cluster replies to the requests of the main controller ("car").

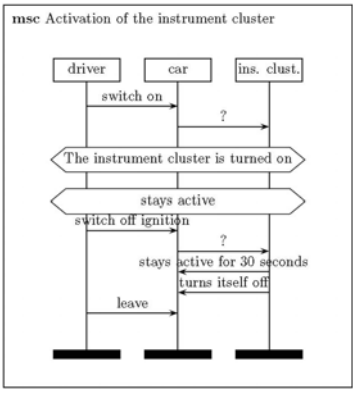

Fig. 2. Possible interpretation of the example scenario

To model challenge/response patterns in MSCs translated from textual scenarios, we organize messages in a stack: If a new message m represents an answer to some previously pushed message m', m' and the messages above it are popped from the stack. Otherwise, the new message m is pushed onto the stack.

We assume that the actors involved in the MSC are provided before the actual text-to-MSC translation. They can be extracted from the requirements document (cf. [4]) or listed manually. The list of actors allows us to decide, which sentences should be translated to messages, and which to assertions: A sentence is translated to a message, if its subject is contained in the set of actors, the sentence is in active voice and contains a grammatical object. All other sentences are translated to assertions (cf. [4]).

For the sentences translated to messages, we assume that the sentence subject is the message sender. For the message receiver there are two possibilities: if the sentence object is contained in the set of actors, the sentence object becomes the message receiver. If the sentence object is not contained in the set of actors, we have to infer the message receiver from the message stack: Let m_{top} be the message on the top of the stack and m_{new} the message under analysis. Then, we assume that m_{new} is the response to m_{top} and, therefore, the receiver of m_{new} is the sender of m_{top}. In a similar way, we can infer missing messages: if the sender of m_{new} is not equal to the receiver of m_{top}, we assume that there is a missing message m' from the receiver of m_{top} to the sender of m_{new}. We put this missing message m' on the stack too. The details of stack management are presented in [3].

2.2 From Automata Descriptions to Automata

Similarly to the text-to-MSC translation, it is possible to translate text pieces describing automata to automata themselves, as presented in [6]. The difference lies in the relation between sentences that we have to model. This can be illustrated on the specification excerpt in Table 1 (from [7]). The header and the first sentence of this excerpt set the context ("normal mode"), and further sentences refer to this context. Thus, instead of a message stack, we have to model the context setting and the usage of the set context.

Table 1. The Steam Boiler, specification excerpt (copied from [7])

Normal mode
1. The normal mode is the standard operating mode in which the program tries to maintain the water level in the steam-boiler between N1 and N2 with all physical units operating correctly. 2. As soon as the water level is below N1 or above N2 the level can be adjusted by the program by switching the pumps on or off. 3. The corresponding decision is taken on the basis of the information which has been received from the physical units.

We model the context by assigning every (sub)sentence to one of the four categories: "state transition", "transition condition", "context setting", or "irrelevant". The assignment of sentence segments to categories takes place in the following steps: (1) splitting of every sentence to segments, (2) assignment of segments to categories on the basis of grammatical information only, and (3) re-assignment of segments to categories, by using context information. Each of these steps is described below.

Sentence splitting: Punctuation symbols, the words "if" and "when" as well as the conjunctions "and" and "or" are used as splitting marks, unless they directly follow an adjective or a number.[1] A splitting example is shown in Table 2.

Table 2. Splitting example

Original sentence
As soon as this signal has been received, the program enters either the mode normal if all the physical units operate correctly or the mode degraded if any physical unit is defective
Splitting
1. As soon as this signal has been received 2. the program enters either the mode normal 3. all the physical units operate correctly 4. the mode degraded 5. any physical unit is defective

[1] These heuristics prevent splitting of, e.g., "if the water level lies between N1 and N2, …".

Assignment of segments to categories on the basis of grammatical information: Identification of the four segment classes is possible on the basis of the Part-of-Speech (POS) tags: A POS tagger decides, for every word, if this word is a noun, verb, adjective, ... The applied tagger has the precision of about 97% [8] which makes it unlikely to become an error source. Furthermore, we assume that the names of the automaton states are extracted from the specification before the actual text-to-automaton translation, cf. [6]. The assignment of the sentence segment to one of the four classes takes place in the following way:

- If the sentence segment does not contain any reference to a state, it is marked as "irrelevant". This holds, for example, for the first segment in Table 2.
- If the sentence segment contains a reference to a state, but first occurrence of the state is not preceded by a verb, this segment is marked as "context setting". For example, in Table 1, the header ("normal mode") and the first sentence set the context for the translation of the following sentences.
- Otherwise, the sentence segment is marked as "state transition".

Here it is important to emphasize that in the first phase no sentence segment is marked as "transition condition".

Re-assignment of segments to categories, by using context information: To take context into account, it is necessary to revise the "context setting"-marks first. Here, the following heuristics is applied: If, for a given sentence, any of its segments is marked as "state transition", then all segments marked as "context setting" are relabeled to "state transition". This compensates for potentially missing verbs in some sentence segments. In the case of the example shown in Table 2, it marks the fourth segment as "state transition" and leaves the other marks unchanged.

When the marking of segments as "state transition" is finished, it is possible to identify transition conditions:

- If a sentence segment is marked as "irrelevant" and directly precedes a segment marked as "state transition", then the former segment is relabeled to "transition condition" (e.g., the first segment of the example in Table 2).
- If a sentence segment is marked as "irrelevant" and directly follows a segment marked as "state transition", then the former segment is relabeled to "transition condition". This allows to treat conditions like "⟨`some transition`⟩ if ⟨`some condition`⟩".

When this relabeling process is finished, transitions are created from the sentence segments marked as "state transition". Transition conditions, as well as source and target state of transitions, are inferred from the adjacent "context setting" and "transition condition" segments.

The presented approach is domain-independent and only relies on a special writing style with guidelines for industrial requirements specifications: The complete set of system states is known, sentences describing state transitions contain a reference to the target state, and context setting is stated explicitly (for details, see [6]).

3 Feedback Generation Instead of Inference Assumptions

The translation approaches presented in Section 2 use different inference rules in order to complete information not explicitly stated in the text. The main idea of feedback generation is to turn off the inference, and, whenever the inference would become necessary, to generate feedback questions addressing the missing information. For MSCs, this missing information can be the unspecified message receiver or an unspecified intermediate message that is inferred by the means of the stack model. Feedback generation for MSCs is presented in Section 3.1. For automata, the missing information is always the source state of a state transition (due to peculiarities of the algorithm presented in Section 2.2). Feedback generation for automata is presented in Section 3.2.

In general, the generated questions can refer either to the specification text or to the extracted model. Reference to the model means that, in addition to the actual models, special markers are generated that pinpoint the problematic model elements. Reference to the text means that hints in natural language are generated, in the following form:

- **Problematic sentence:** ⟨sentence cited⟩
- **Detected problem:** ⟨problem description⟩

Both for automata and for MSCs, we implemented the generation of both representations.

3.1 Feedback Generation for MSCs

The algorithm for MSC generation presented in Section 2.1 can infer unspecified message receivers and whole missing messages. Correspondingly, the feedback questions that are generated for MSCs, address (1) unspecified message receivers or (2) messages that are necessary from the point of view of a continuous message flow, but not explicitly specified in the document.

Every type of the feedback question (addressing the missing receiver or the missing messages) can refer both to the model and to the specification text. Reference to the text means that we generate questions or hints in natural language, addressing the detected problem. Reference to the model means that we use special markers in the generated MSCs in order to address the detected problems.

Missing message receivers result from active sentences only, as passive sentences are translated to MSC assertions, cf. [4]. Furthermore, an active sentence needs a grammatical object to be a candidate for an MSC message. Nevertheless, if the sentence object is not contained in the set of potential MSC actors, inference rules sketched in Section 2.1 must be applied. Thus, in order to address missing message receivers, we generate schematic hints referring to the specification text, structured as follows:

> ⟨sentence cited⟩
> **Problem:** Problematic active sentence: object ⟨sentence object⟩ is not contained in the list of actors. Message receiver cannot be identified.

The inference of missing messages becomes necessary when the sender (basically, grammatical subject) of the sentence under consideration does not coincide with the receiver of the last message. In this case we generate coherence questions:

⟨sentence cited⟩
Problem: COHERENCE: Which component / who controls the ⟨sentence subject⟩ to perform this action?

Additionally to the two above question types, based completely on the inference rules presented in Section 2.1, we address a further problem, resulting from passive sentences: whenever we come across a passive sentence, we check if the subject of the sentence coincides with the receiver of the last message. If this is not the case, we generate the following question:

⟨sentence cited⟩
Problem: Problematic sentence: passive, actor unspecified (Who / which component controls the ⟨sentence subject⟩?)

As an example, the set of questions addressing the problems of the scenario on page 95 is presented in Table 3.

Table 3. Questions referring to the specification text, generated for the scenario on page 95

1. The instrument cluster is turned on **Problem:** Problematic sentence: passive, actor unspecified (Who / which component controls the instrument cluster?)
2. The instrument cluster stays active for 30 seconds (a) **Problem:** Problematic active sentence: object ”seconds” is not contained in the list of actors. Message receiver cannot be identified. (b) **Problem:** COHERENCE: Which component / who controls the instrument cluster to perform this action?

For the questions referring to the model, we further develop the idea shown in Figure 2: there, the inferred missing message was represented as a “?”-marked message. To generate the same types of questions as above, we introduce a special actor named “???” and send all the messages where the message receiver is not explicitly specified, to the “???”-actor. Similarly, every missing message is “?”-marked. Missing messages in the sense of Section 2.1 are split into two: a message from the inferred message sender to the “???”-actor, and a message from the “???”-actor to the inferred receiver.

The set of questions generated for the example on page 95 is shown in Figure 3. It is easy to see the correspondence to the questions referring to the specification: Question 1 from Table 3 is represented by two “?”-messages before the first assertions, Question 2a is represented by two messages between the assertions, and Question 2b by the “?”-messages after the second assertion.

3.2 Feedback Generation for Automata

The algorithm for automata generation presented in Section 2.2 infers, if necessary, the source state of the transition from the discourse context. It is easy to turn this inference

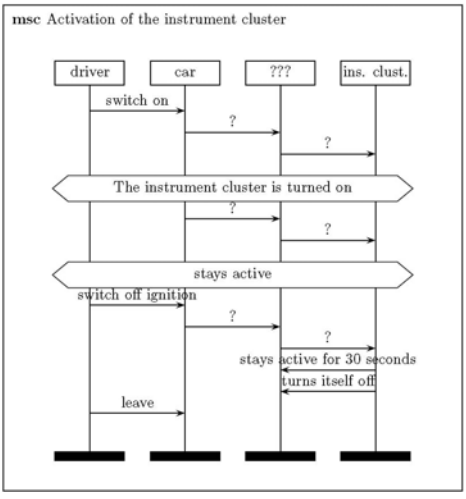

Fig. 3. Questions referring to the model, generated for the scenario on page 95

into feedback generation: whenever the source state of a transition is not explicitly specified, we can generate a corresponding question.

To generate questions referring to the specification text, we create schematic hints structured as follows:

- **Sentence:** ⟨sentence cited⟩
- **Unspecified source state:** transition ⟨transition condition⟩ to state ⟨target state⟩

For the specification excerpt shown in Table 1, this results in the set of questions presented in Table 4.

To generate questions referring to the model, we use the same representation as used in [6] for generated automata. In [6], the generated automata are represented as a three-column table: Every line of the table represents exactly one state transition: it contains the source and the target state, and the transition condition. The generated questions are represented in a similar table, with the only difference that the states that are not explicitly specified in the text are marked as "unknown". For the specification excerpt shown in Table 1, this results in the set of questions presented in Table 5.

4 Evaluation

The presented approach to question generation was evaluated in an experiment. The goal of the experiment was to see, if the generated questions address genuine specification problems that would be overseen by human analysts. The experiment setting is presented in Section 4.1. In the experiment, it was found that the generated questions can address genuine specification problems, not detected by human analysts. The results of the experiment are presented in Section 4.2. It was found, furthermore, that the problems not addressed by the generated questions can be detected by other techniques. The lessons learned in the experiment are presented in Section 4.3.

Table 4. Questions referring to the text, generated for the specification in Table 1

Specification	Generated Question
Sentence: the unit for detection of the level of steam is defective – that is , when v is not equal to zero – the program enters the emergency stop mode.	**Unspecified source state:** transition “the unit for detection of the level of steam is defective – that is , when v is not equal to zero – the program enters” to state “emergency stop mode”.
Sentence: the program realizes a failure of the water level detection unit it enters the emergency stop mode.	**Unspecified source state:** transition “the program realizes a failure of the water level detection unit it enters” to state “emergency stop mode”.
Sentence: as soon as this signal has been received , the program enters either the mode normal all the physical units operate correctly the mode degraded any physical unit is defective.	**Unspecified source state:** transition “all the physical units operate correctly” to state “mode normal”.
Sentence: as soon as this signal has been received , the program enters either the mode normal all the physical units operate correctly the mode degraded any physical unit is defective.	**Unspecified source state:** transition “any physical unit is defective.” to state “mode degraded”.
Sentence: a transmission failure puts the program into the mode emergency stop.	**Unspecified source state:** transition “a transmission failure puts” to state “mode emergency stop”.

4.1 Experiment Setting

In order to evaluate the generated feedback questions, the following evaluation hypothesis was put forward:

H: The automatically generated questions address deficiencies of the specification that would be overseen by human analysts.

In total, 9 PhD candidates/postdocs in computer science participated in the evaluation. The evaluation took place in the following way:

1. First, the experiment subjects were given the chapter of the Steam Boiler Specification [7] that refers to the system behavior (2 pages). Each subject was asked to read

Table 5. Questions referring to the model, generated for the specification in Table 1

Source	Target	Condition
unknown state	emergency stop mode	the unit for detection of the level of steam is defective – that is , when v is not equal to zero – the program enters
unknown state	emergency stop mode	the program realizes a failure of the water level detection unit it enters
unknown state	mode normal	all the physical units operate correctly
unknown state	mode degraded	any physical unit is defective .
unknown state	mode emergency stop	a transmission failure puts

the specification and to mark words, word sequences or sentences that, in his/her opinion, would cause problems if we model the system behavior.
2. Then, every subject was given a set of questions generated as presented in Section 3.2, and asked, for every generated question, to evaluate if the question addresses a genuine problem of the specification.
3. Similarly to step 1, every subject was given 15 (out of 41) scenarios from the Instrument Cluster Specification [5]. The 15 scenarios were selected in such a way that they represent typical system behavior. Thus, the results obtained with the 15 scenarios can be extrapolated to all scenarios provided in the specification.
   Analogously to step 1, every subject was asked to read the specification and to mark words, word sequences or sentences that, in his/her opinion, would cause problems if we model the system behavior.
4. Analogously to step 2, every subject was given a set of generated questions and asked, for every generated question, to evaluate if the question addresses a genuine problem of the specification.

In order to address different representations of the generated questions, the subjects were separated in two groups: One group was given questions referring to the specification text for the Steam Boiler and questions referring to the model for the Instrument Cluster. The other group, inversely, was given questions referring to the model for the Steam Boiler and questions referring to the specification text for the Instrument Cluster. Due to this setting, we avoid two potential threats to validity of the results:

- Every subject was given one set of questions referring to the model and one set of questions referring to the specification. Thus, the differences in subjects' responses cannot be attributed to the differences in the representation of the questions.
- Every subject was given one set of questions about the Steam Boiler Specification, and one set of questions about the Instrument Cluster Specification. This way, we avoid learning effects that would become important if we would have given two sets of questions about the same specification to the same subject.

The evaluation was performed in groups, but the subjects were not allowed to communicate with each other. In total, every subject spent approximately two hours performing the above steps 1-4.

4.2 Experiment Results

The results of the experiment are presented in Table 6. The numbers in the cells represent the number of manually marked problems or generated/confirmed questions. The number of questions generated for the Steam Boiler Specification (automaton-based) coincide for the questions referring to the model and to the text, as the differences in the representations are not too big. For the Instrument Cluster (MSCs), however, differences in representations result in the different number of generated questions (cf. Section 3.1).

Questions generated for the Steam Boiler Specification (automaton-based) were considered by the most subjects as spurious. With one notable exception, most subjects

Table 6. Experiment results. Numbers of found/addressed deficiencies.

	Generated	Subject	Manual	Generated$_{confirmed}$	Manual ∩ Generated	Manual ∩ Generated$_{confirmed}$
Automaton, questions referring to the text	21	1	32	1	1	1
		2	30	1	1	1
		3	13	0	0	0
		4	29	19	0	0
MSCs, questions referring to the model	87	1	38	73	0	0
		2	38	46	5	5
		3	25	43	0	0
		4	16	52	0	0
Automaton, questions referring to the model	21	1	9	1	0	0
		2	4	5	0	0
		3	11	0	0	0
		4	14	2	0	0
		5	12	1	0	0
MSCs, questions referring to the text	50	1	26	31	8	8
		2	18	7	1	1
		3	19	28	3	1
		4	11	27	0	0
		5	14	1	0	0

found that the generated questions do not address genuine problems of the specification. Thus, the experiment hypothesis has to be rejected for the automaton-based specification. For the Instrument Cluster Specification (MSCs), however, the subjects found that many more generated questions address genuine problems of the specification text. The fraction of questions addressing genuine problems (column "Generated$_{confirmed}$" of Table 6) varies from 2% (1 out of 50) to 84% (73 out of 87). It looks like the number of generated questions addressing genuine problems is higher for the questions referring to the model than for the questions referring to the text: For the questions referring to the text, at most 31 out 50 questions (62%) are found to address genuine problems. For the questions referring to the model, though, this rate varies from 49% (43 out of 87) to 84% (73 out of 87). The size of the sample, though, is too small to claim that questions referring to the model better help to detect specification problems.

The most interesting finding of the experiment is presented in the last two columns of Table 6: the manually found problems and the problems addressed by the automatically generated questions are almost always disjoint. The figures in both columns coincide, except for one line. This line results from the following situation: the subject marked two sentences as problematic. The tool marked the same two sentences as problematic too. However, the subject did not confirm the generated questions as addressing genuine problems. The most probable reason is that the subject saw different problems in these sentences than the tool. However, as the subject did not explicitly write down the identified problems, this remains solely our interpretation of the discrepancy.

Two last columns of Table 6, together with the number of generated questions that the subjects found to address genuine specification problems, allow us to claim that the hypothesis is confirmed for the Instrument Cluster Specification, namely, that the automatically generated questions address deficiencies of the specification that would be overseen by human analysts.

4.3 Lessons Learned

The specification problems that were detected by our subjects were highly different from the problems addressed by the automatically generated questions. The problems most often marked by the human analysts were pertinent to unclear phrasing. The automatically generated questions, to the contrary, addressed missing specification pieces. For example, our subjects marked following phrases as problematic:

- **For the Steam Boiler Specification:**
  - "Once all the units which were defective *have been repaired*, the program comes back to normal mode."
    Evaluator's comment: how is it detected that all the units have been repaired?
  - "... when either the *vital units* have a failure..."
    Evaluator's comment: which units count as "vital"?
  - "*As soon as* the water measuring unit is repaired..."
    Evaluator's comment: what does "as soon as" really mean?
- **For the Instrument Cluster Specification:**
  - "... and the instrument cluster *is activated temporarily*."
    Evaluator's comment: by whom is the instrument cluster activated? And what does "temporarily" really mean?
  - "The system displays the *calculated speed*"
    Evaluator's comment: how is the speed calculated?
  - "The input signals from the motor *are sent regularly*"
    Evaluator's comment: to which component are they sent? And what does "regularly" mean?

These phrases are indeed problematic if we want to build a system model, but they represent fuzzy specification, not completely missing facts.

In order to address such fuzzy phrasings, it makes sense to apply the previously developed tool that detects poor phrasings [9]. In its current version, the tool detects poor phrasings listed in the Ambiguity Handbook [10]. Adding further patterns for problematic phrases, however, is a matter of minor extension of the configuration files. An integrated tool, detecting both missing specification pieces (as in the presented paper) and poor phrasings, would provide a reliable means of specification analysis.

5 Related Work

Work related to the presented paper can be subdivided in two areas: work on text-based modeling and work on natural language processing (NLP) in requirements engineering. Both areas are presented below.

5.1 Text-Based Modeling

Saeki at al. [11], Overmyer et al. [12], and Ermagan et al. [13] introduced tools providing modeling approaches. The approach by Saeki et al. allows the user to mark words in the requirements documents, and then to assign the marked word to some noun or verb

type. Then, the approach maps nouns to classes (in the sense of object oriented design), and verbs to operations. The approach can handle four predefined verb classes.

Overmyer et al. developed a tool allowing the user to mark words or word sequences and map them to classes, roles, and operations. As opposed to the approach by Saeki et al., they do not assume that the verb must fall into one of the four predefined categories.

Ermagan et al. developed the tool SODA, allowing to link textual use cases to behavior models. However, SODA sticks to manual modeling and does not provide automatic extraction of model elements.

Our approach has the important advantage that it does not assume the textual document to contain sufficient information to generate models. Thus, it can cope with incomplete documents and make the incompleteness apparent.

5.2 Natural Language Processing in Requirements Engineering

There are three areas where natural language processing is applied to requirements engineering: assessment of document quality, identification and classification of application specific concepts, and analysis of system behavior.

Approaches to the assessment of document quality were introduced, for example, by Rupp [14], Fabbrini et al. [15], Kamsties et al. [16], and Chantree et al. [17]. These approaches define writing guidelines and measure document quality by the degree to which the document satisfies the guidelines. These approaches have a different focus from our work: their aim is to detect poor phrasing and to improve it, they do not target system modeling, and do not generate any model-related feedback, as our approach does.

Another class of approaches, as, for example, those by Goldin and Berry [18], Abbott [19], or Sawyer et al. [20] analyzes the requirements documents, extracts application-specific concepts, and provides an initial static model of the application domain. However, these approaches do not generate feedback addressing specification incompleteness either.

The approaches analyzing system behavior translate requirements documents to executable models by analyzing linguistic patterns. Vadera and Meziane [21] propose a procedure to translate certain linguistic patterns into first order logic and then to the specification language VDM, but they do not provide automation for this procedure. Gervasi and Zowghi [22] go further and introduce a restricted language, a subset of English. They automatically translate textual requirements written in this restricted language to first order logic. Similarly, Breaux et al. [23] introduce a restricted language and translate this language to description logic. Avrunin et al. [24] translate natural language to temporal logic. Our work goes further than the above approaches, as we not only translate textual descriptions to models, but also use the resulting models to make the deficiencies of the textual specification apparent.

To summarize, to the best of our knowledge, there is no approach to documents analysis, yet, able not only to translate model descriptions to models themselves, but also to generate feedback concerning the deficiencies of the document.

6 Summary

Even though many formal and semi-formal specification techniques exist, requirements specifications in natural language remain a de-facto standard. Apart from being readable

by all stakeholders, specifications in natural language entail a lot of problems with poor phrasings, inconsistencies, and missing information.

The approach presented in our paper analyzes missing information in behavior specifications and automatically generates questions addressing these findings. As evaluated in our experiment, the generated questions can address specification deficiencies that would otherwise be overseen by human analysts. Thus, the presented method should be one of the means to ensure the quality of requirements specifications.

Acknowledgments

We want to thank the participants of the experiment: Mou Dongyue, Florian Hölzl, Maged Khalil, Alexander Krauss, Christian Leuxner, Daniel Méndez Fernández, David Trachtenherz, and Andreas Vogelsang.

References

1. Mich, L., Franch, M., Novi Inverardi, P.: Market research on requirements analysis using linguistic tools. Requirements Engineering 9(1), 40–56 (2004)
2. Kof, L., Schätz, B.: Combining aspects of reactive systems. In: Broy, M., Zamulin, A.V. (eds.) PSI 2003. LNCS, vol. 2890, pp. 344–349. Springer, Heidelberg (2004)
3. Kof, L.: Scenarios: Identifying missing objects and actions by means of computational linguistics. In: 15th IEEE International Requirements Engineering Conference, October 15–19, pp. 121–130. IEEE Computer Society Conference Publishing Services, New Delhi (2007)
4. Kof, L.: From Textual Scenarios to Message Sequence Charts: Inclusion of Condition Generation and Actor Extraction. In: 16th IEEE International Requirements Engineering Conference, September 10-12, pp. 331–332. IEEE Computer Society Conference Publishing Services, Barcelona (2008)
5. Buhr, K., Heumesser, N., Houdek, F., Omasreiter, H., Rothermehl, F., Tavakoli, R., Zink, T.: DaimlerChrysler demonstrator: System specification instrument cluster (2004), `http://www.empress-itea.org/deliverables/D5.1_Appendix_B_v1.0_Public_Version.pdf` (accessed 16.02.2010)
6. Kof, L.: Translation of Textual Specifications to Automata by Means of Discourse Context Modeling. In: Glinz, M., Heymans, P. (eds.) REFSQ 2009. LNCS, vol. 5512, pp. 197–211. Springer, Heidelberg (2009)
7. Abrial, J.-R., Börger, E., Langmaack, H.: The steam boiler case study: Competition of formal program specification and development methods. In: Abrial, J.-R., Börger, E., Langmaack, H. (eds.) Dagstuhl Seminar 1995. LNCS, vol. 1165. Springer, Heidelberg (1996), `citeseer.nj.nec.com/abrial96steam.html`
8. Clark, S., Curran, J.R.: Parsing the WSJ using CCG and log-linear models. In: ACL 2004: Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics, p. 103. Association for Computational Linguistics, Morristown (2004)
9. Gleich, B., Creighton, O., Kof, L.: Ambiguity detection: Towards a tool explaining ambiguity sources. In: Wieringa, R., Persson, A. (eds.) REFSQ 2010. LNCS, vol. 6182. Springer, Heidelberg (2010)
10. Berry, D.M., Kamsties, E., Krieger, M.M.: From contract drafting to software specification: Linguistic sources of ambiguity, `http://se.uwaterloo.ca/~dberry/handbook/ambiguityHandbook.pdf` (accessed 18.11.2004)

11. Saeki, M., Horai, H., Enomoto, H.: Software development process from natural language specification. In: Proceedings of the 11th International Conference on Software Engineering, pp. 64–73. ACM Press, New York (1989)
12. Overmyer, S.P., Lavoie, B., Rambow, O.: Conceptual modeling through linguistic analysis using LIDA. In: ICSE 2001: Proceedings of the 23rd International Conference on Software Engineering, pp. 401–410. IEEE Computer Society, Washington, DC, USA (2001)
13. Ermagan, V., Huang, T.-J., Krüger, I., Meisinger, M., Menarini, M., Moorthy, P.: Towards Tool Support for Service-Oriented Development of Embedded Automotive Systems. In: Proceedings of the Dagstuhl Workshop on Model-Based Development of Embedded Systems (MBEES 2007), Informatik-Bericht 2007-01, Fakultät für Informatik, Technische Universität Braunschweig (2007)
14. Rupp, C.: Requirements-Engineering und -Management. Professionelle, iterative Anforderungsanalyse für die Praxis, 2nd edn. Hanser–Verlag (May 2002) ISBN 3-446-21960-9
15. Fabbrini, F., Fusani, M., Gnesi, S., Lami, G.: The linguistic approach to the natural language requirements quality: benefit of the use of an automatic tool. In: 26th Annual NASA Goddard Software Engineering Workshop, pp. 97–105. IEEE Computer Society, Greenbelt (2001), `http://fmt.isti.cnr.it/WEBPAPER/fabbrini_nlrquality.pdf` (accessed 08.02.2010)
16. Kamsties, E., Berry, D.M., Paech, B.: Detecting ambiguities in requirements documents using inspections. In: Workshop on Inspections in Software Engineering, Paris, France, pp. 68–80 (2001)
17. Chantree, F., Nuseibeh, B., de Roeck, A., Willis, A.: Identifying nocuous ambiguities in natural language requirements. In: RE 2006: Proceedings of the 14th IEEE International Requirements Engineering Conference (RE 2006), pp. 56–65. IEEE Computer Society, Washington, DC, USA (2006)
18. Goldin, L., Berry, D.M.: AbstFinder, a prototype natural language text abstraction finder for use in requirements elicitation. Automated Software Eng. 4(4), 375–412 (1997)
19. Abbott, R.J.: Program design by informal English descriptions. Communications of the ACM 26(11), 882–894 (1983)
20. Sawyer, P., Rayson, P., Cosh, K.: Shallow knowledge as an aid to deep understanding in early phase requirements engineering. IEEE Trans. Softw. Eng. 31(11), 969–981 (2005)
21. Vadera, S., Meziane, F.: From English to formal specifications. The Computer Journal 37(9), 753–763 (1994)
22. Gervasi, V., Zowghi, D.: Reasoning about inconsistencies in natural language requirements. ACM Trans. Softw. Eng. Methodol. 14(3), 277–330 (2005)
23. Breaux, T.D., Antón, A.I., Doyle, J.: Semantic parameterization: A process for modeling domain descriptions. ACM Trans. Softw. Eng. Methodol. 18(2), 1–27 (2008)
24. Smith, R.L., Avrunin, G.S., Clarke, L.A., Osterweil, L.J.: Propel: an approach supporting property elucidation. In: ICSE 2002: Proceedings of the 24th International Conference on Software Engineering, pp. 11–21. ACM, New York (2002)