Scenario Analysis: Generation of Possible Scenario Interpretations and their Visualization*

Leonid Kof
Fakultät für Informatik, Technische Universität München
Boltzmannstr. 3, D-85748, Garching bei München, Germany

E-mail: kof@informatik.tu-muenchen.de

Abstract

Natural language is the main presentation means in industrial requirements documents. In such documents, system behavior is mostly specified in the form of scenarios, with every scenario written as a sequence of sentences in natural language. The scenarios are often incomplete: For the authors of requirements documents some facts are so obvious that they forget to mention them; this surely causes problems for the requirements analyst.

In our previous work we developed an approach to translate textual scenarios to message sequence charts (MSCs). In order that the produced MSCs can be used for further development, they must be validated: i.e., for each MSC we have to say whether it really represents a possible system behavior, and whether the textual scenario was correctly interpreted. In the presented paper we suggest an approach to visualize different interpretations for the same scenario. For visualized scenarios, the user can decide, which of them represent allowed system behavior. This allows, in turn, to generalize exemplary scenarios to universal specifications. Applicability of the presented approach was confirmed in a case study.

1. A Scenario Can Have Several Interpretations

At the beginning of every software project, some kind of requirements document is usually written. The majority of these documents are written in natural language, as the survey by Mich et al. shows [22]. This results in requirements documents are imprecise, incomplete, and inconsistent. From the linguistic point of view, document authors may introduce three defect types, without perceiving them as defects, cf. Rupp [24]:[1]

Deletion: "... is the process of selective focusing of our attention on some dimensions of our experiences while excluding other dimensions. Deletion reduces the world to the extent that we can handle."

Generalization: "... is the process of detachment of the elements of the personal model from the original experience and the transfer of the original exemplary experience to the whole category of objects."

Distortion: "... is the process of reorganization of our sensory experience."

The authors of requirements documents are mostly unaware of these document defects. According to Boehm [2], the later an error is found, the more expensive its correction. Correction of an error found in the design phase is much more expensive than the correction of the same error found already in the requirements engineering phase. Thus, it is one of the goals of requirements analysis, to find and to correct the defects of requirements documents.

The presented paper focuses on the "deletion"-defects in scenarios. Deletion can manifest itself in exemplary specifications of some general rules: For example, the specification can look like "The system receives input a and produces output b" instead of more general "*Every time when* the system receives input a, it *immediately* produces output b". In order to find out, which interpretation of such sequences of actions is intended by the author of the requirements document, we visualize different possible interpretations, in the form of MSCs. Then, the analyst should decide which of them really represent allowed system behavior.

Terminology: For the remainder of the paper we use the following terminology: A *scenario* is a sequence of natural language sentences. A *Message Sequence Chart* (*MSC*) consists of a set of *actors*, a sequence of *messages* sent and

*This work was partially funded by the Deutsche Forschungsgemeinschaft (German Research Foundation), grant "Inserve III, BR 887/19-3"

[1] The following definitions are translations of the definitions from [24], in German

received by these actors, and a sequence of *conditions* interleaved with the message sequence. Figure 1 illustrates the introduced terminology.

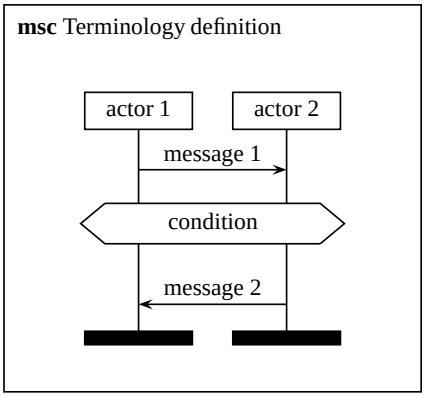

Figure 1. MSCs: Terminology Definition

The remainder of the paper is organized as follows: Section 2 introduces the case study used to evaluate the presented approach. Section 3 is the technical core of the paper. It presents and evaluates the approach translating scenarios to MSCs and then visualizing and validating the resulting MSCs. Then, Sections 4, 5, and 6 present an overview of related work, the summary of the paper, and possible directions for future work, respectively.

2. Case Study: The Instrument Cluster

Authors of requirements documents tend to forget to write down facts that seem obvious to them. Even in a relatively precise requirements document, as for example the Instrument Cluster Specification [6], used as a case study in the presented paper, missing facts can be identified. The Instrument Cluster Specification describes the optical design of the instrument cluster (a part of a car dashboard), its hardware, and, most importantly, its behavior. The behavior is specified as a set of scenarios, like the below example, taken from [6]:

1. The driver switches on the car.
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.

If we translate this scenario to an MSC, we get the MSC in Figure 2[2] as a possible interpretation. It is easy to see that some implicit assumptions had to be made to construct the MSC: it was assumed that the instrument cluster is turned off by special messages sent by the car. Furthermore, it was assumed that the message "turn off ignition" from the driver directly follows turning the instrument cluster on and that there are no messages in between. It is the goal of the presented work, to make such assumptions explicit and to validate them with the analyst.

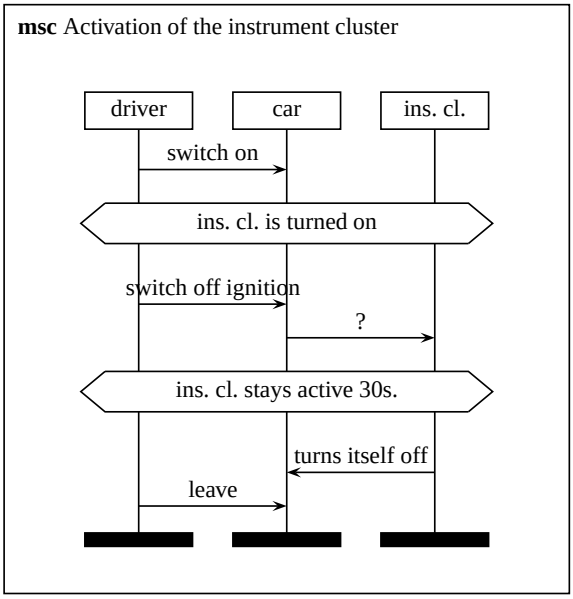

Figure 2. Scenario "Activation of the instrument cluster", manual translation to MSC

3. Interactive Validation of Scenarios

The starting point for our behavior analysis approach is textual behavior specification, represented as a set of scenarios. In our previous work [17, 18, 19, 20] we developed an approach translating textual scenarios to message sequence charts (MSCs). MSCs were chosen because they represent a rather natural formalization of scenarios: in a textual scenario, every sentence typically represents some interaction with the system, and in an MSC every message represents some atomic interaction between two actors. The formalization approach, sketched in Section 3.1, results in one MSC for every scenario.

To become useful for further system development, the MSCs should be validated. To validate them, we visualize every MSC and generate and visualize further MSCs, representing further potential system behaviors. For every generated MSC, the analyst has to decide whether the MSC represents an allowed system behavior. The theoretical basis

[2] "instrument cluster" is abbreviated as "ins. cl.".

for the generation of further MSCs is the PROPEL ("PROPerty ELucidation") approach [26], presented in Section 3.2. We generate additional MSCs in two modes: batch, presented in Section 3.3, and interactive mode, presented in Section 3.4. The case study has shown that the batch mode produces too many MSCs, imposing high workload for the analyst, whereas the interactive mode produces much less MSCs without major impact on the informative value of the produced MSCs. Both in the batch and in the interactive mode, every generated additional MSC addresses the deletion defect in scenarios, as it represents a potential system behavior, possibly forgotten by the author of the requirements document.

3.1. From Text to MSCs

Translation of textual scenarios to MSCs is the prerequisite for our visualization and validation approach, for two reasons:

- Firstly, requirements documents typically contain textual scenarios, and not MSCs.
- Secondly, special structure of the MSCs extracted from the text can be explicitly used for validation, cf. Sections 3.3 and 3.4.

In our previous work [17, 18, 19] we developed an approach for fully automated generation of MSCs from textual scenarios. This approach takes two inputs: a set of actors, with every actor represented as a word sequence, and a set of scenarios, with every scenario represented as a sentence sequence. First, our approach determines the passive sentences (like "instrument cluster is turned on") and translates them to MSC conditions. Active sentences are translated to messages, by identifying the message sender/receiver as the longest word sequence occurring both before/after the main verb and in the set of actors.

When translating scenarios to MSCs, our approach deals with some typical deficiencies of natural language texts: It can happen that either the message sender or the receiver are not explicitly mentioned, or the whole message is just omitted. For example, the sentence "The instrument cluster turns on", does not specify the message receiver.

The problem of unspecified message senders/receivers and missing messages was solved by the organization of MSC messages in a stack. Organization of messages in a stack is motivated by the idea of situation stack by Grosz et al. [14]. Grosz et al. introduce a situation stack to explain how the human attention focuses on different objects during a discourse. The focus depends on the sequence of sentence heard so far. By default, a sentence defines some situation and is pushed onto the stack. If a sentence reverts the effect of some previous sentence, the corresponding stack element is popped:

John enters the shop //push "enter"
— Some actions in the shop —
John leaves the shop //pop "enter" and the above //stack elements

The idea of the situation stack can be easily transferred to MSCs: We define an active actor as an actor that has sent a message but has not received an answer yet. If the receiver of the message under analysis (msg) is an active actor, then it is possible to find the topmost message of the stack sent by this actor (msg'). Then, msg' and the messages contained in the stack above it are popped. If the receiver is not an active actor, the message under analysis is pushed onto the stack.

The organization of messages in a stack makes also the identification of missing messages possible: If the sender of the message under analysis ($sender_{new}$) differs from the receiver of the message on the top of the stack (rec_{top}), then the message from rec_{top} to $sender_{new}$ is missing. For example, for the MSC in Figure 2, missing message from "car" to "instrument cluster" just after the message "switch off ignition" can be identified in this way. The message stack enables the identification of missing message receivers as well: The default message receiver equals to the sender of the message on the top of the stack. For the example MSC shown in Figure 2, this allows to state that the message "turns itself off" goes from the instrument cluster to the car.

The above approach can visualize textual scenarios, but it does not provide any means for validation of the generated MSCs. Generation and visualization of further MSCs for validation purposes are presented in Sections 3.3 and 3.4.

3.2. Behavior Visualization: Theoretical Basis

The PROPEL ("PROPerty ELucidation") approach by Avrunin et al. [26] introduces a set of rules for interactive validation of behavior models. The validation rules are based on the specification patterns by Dwyer et al. [9]. PROPEL states that two specification patterns, *response* and *precedence*, are especially important for interactive validation: Avrunin et al. identified several interpretation possibilities for them. The response pattern consists, in its basic form, of the message *action*, followed by the message *response*. An interpretation possibility for the response pattern consists, for example, in the question whether the message *response* must immediately follow the *action* or whether some other messages are allowed in between. Based on different possibilities to interpret the response and precedence patterns, Avrunin et al. introduced a set of questions to distinguish interpretation possibilities from each other. For example, one of the questions for the *response*-pattern is "Does *response* have to immediately follow *ac-*

tion?". They apply the same set of questions to the *precedence*-pattern too.

Avrunin et al. introduce six options to interpret the response- and precedence-patterns:

"Pre-arity, which determines whether *action* may occur one time or many times before *response* does." To determine pre-arity, we generate an additional MSC with two instances of *action* before the occurrence of *response* and let the analyst decide whether this behavior is allowed.

"Post-arity, which determines whether *response* may occur one time or many times after *action* does." To determine post-arity, we generate an additional MSC with two instances of *response* after the occurrence of *action* and let the analyst decide whether this behavior is allowed.

"Immediacy, which determines whether or not other intervening events may occur between *action* and *response*." To determine immediacy, we generate an additional MSC containing a new message between *action* and *response*.

"Precedency, which determines whether or not *response* is allowed to occur before the first occurrence of *action*." To determine precedency, we generate an additional MSC containing *response* before *action*.

"Nullity, which determines whether or not *action* must ever occur." To determine nullity, we generate an additional MSC containing no *action*, but in other respects coinciding with the original MSC.

"Repeatability, which determines whether or not occurrences of *action* after an occurrence of *response* are required to be followed by *response*." To determine repeatability, we generate two additional MSCs. Both of them contain *action* after *response*. In one of them, the second occurrence of *action* is again followed by *response*, and in the other, the second occurrence of *action* is not followed by *response*.

Altogether, seven additional MSCs are necessary to decide which variant of the response-pattern is the correct one for a given scenario. For every additional MSC, the analyst must decide whether this MSC represents a possible system behavior.

3.3. Generation and Visualization of Additional MSCs: Batch Mode

To apply the PROPEL ideas to MSCs extracted from the text, it is necessary to cut every MSC. Every cut divides the MSC in four parts:

1. prefix, not a constituent of the pattern,
2. sequence of messages/conditions declared to *action*,
3. sequence of messages/conditions declared to *response*,
4. and suffix, not a constituent of the pattern.

Let i be the length of the prefix and j the length of the suffix. After stipulating the prefix and the suffix, we can arbitrarily cut the remaining MSC elements (messages and conditions) into *action* and *response*. This implies that the number of cut possibilities of an MSC containing n elements is equal to $\sum_{i=0}^{n} \sum_{j=0}^{n-i} (n-i-j) \approx \frac{n^3}{6}$.

The longest MSC extracted from the text in our case study contains 22 elements. The above procedure would result in $22^3/6 \approx 1800$ possibilities to instantiate the response pattern. Given seven additional MSCs for every pattern instantiation, this would lead to approximately 12,000 additional MSCs. Obviously, the above, brute force, validation strategy is not feasible.

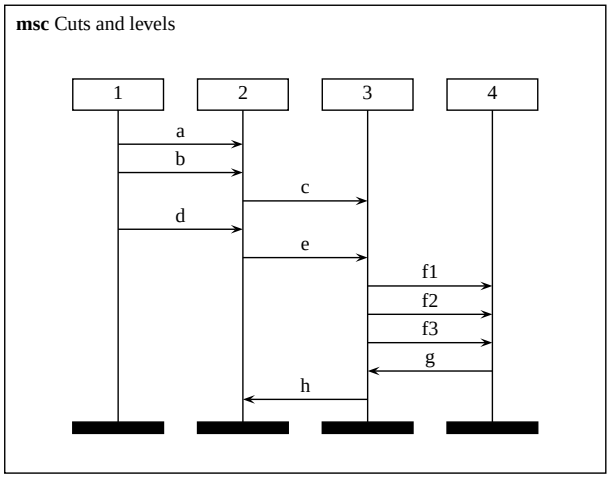

Figure 3. MSCs: levels to instantiate the *response* or *precedence* patterns

To circumvent the above problem of combinatorial explosion, we cut MSCs in blocks, on three levels:

Level 1: A block of Level 1 is a message sequence starting with a message from the actor that initiates the whole MSC. For example, in Figure 3 we have three such blocks: the first block consists of the message "a", the second one consists of the messages "b"-"c", and the third one of the messages "d"-"h".

The rationale for blocks of Level 1 is possible aggregation of several independent interactions in a single textual scenario. For example, in the example scenario presented in Section 2, sentence 1 and sentence 3 initiate independent interactions, but are present in the

same scenario. Cutting an MSC in blocks of Level 1 separates such independent interactions.

Level 2: Block of Level 2 is motivated by the structure of the MSCs extracted from the text (cf. Section 3.1): A block of Level 2 consists of a consecutive sequence of messages pushed onto the stack or popped from the stack. Formally, a block of Level 2 is a part of a block of Level 1, consisting of a sequence of consecutive messages where each message either involves a new actor in the communication or has the same sender and receiver as the preceding message.

In Figure 3 we have four such blocks: the first block consists of the message "a", the second one of the messages "b" and "c", the third one of the messages "d"-"f3", and the fourth one of the messages "g" and "h".

Level 3: A block of Level 3 is a part of a block of Level 2, consisting of a sequence of consecutive messages having the same sender and receiver. The reason is to bundle communication between two actors. In Figure 3 we have one such block, consisting of the messages "f1"-"f3".

If a condition occurs on the boundary between two blocks of the same level, we consider this condition as belonging to both blocks. If a condition occurs inside some block, we consider it as an ordinary element of the block. The rationale for this design decision is the assumption that a condition occurring between two blocks of the same level denotes an important system state, such that the messages occurring in the preceding block result in this state, and messages occurring in the subsequent block are initiated in this state. This idea is illustrated in Figure 4: the condition "instrument cluster is turned on" belongs both to the first and to the second block.

On every level we consider all cut possibilities. However, we consider blocks as atomic. Formally, all cuts on the level n are performed within the corresponding block of level $n-1$, other parts of the MSC remain untouched. Figures 4 and 5 illustrate this idea: Figure 4 shows blocks of Level 1, and Figure 5 shows blocks of Level 2, belonging to the same block of Level 1. When generating additional MSCs, as required by PROPEL, we consider the MSC in Figure 4 as consisting of just three elements, and the MSC in Figure 5 as consisting of just two.

For every level and every cut on this level, we generate all additional MSCs necessary to determine which instance of the response pattern represents the correct interpretation. This strategy, although generating less additional MSCs than the brute force strategy, can still result in so many MSCs that this way of behavior validation becomes infeasible. To measure the effort necessary to validate the MSCs, we generated additional MSCs for the MSCs ex-

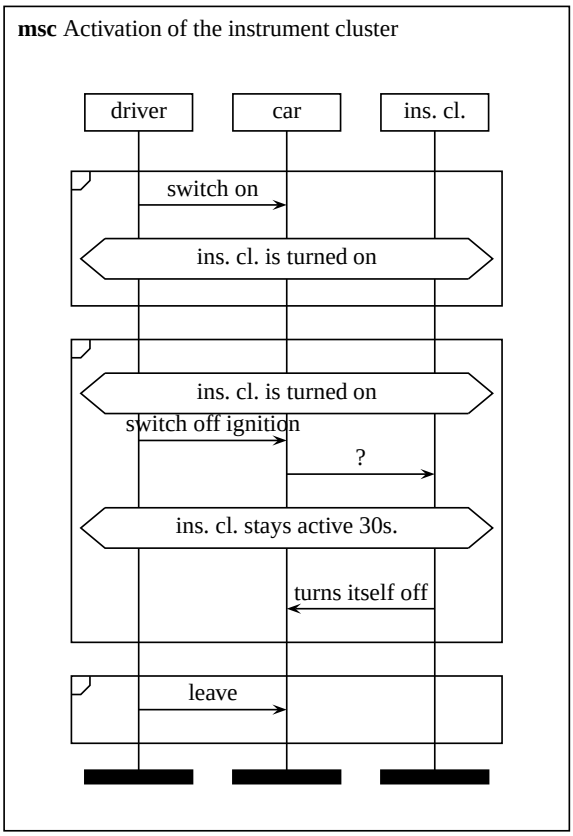

Figure 4. Cut of the MSC from Figure 2, Level 1

tracted from the case study presented in Section 2. To illustrate the generated MSCs, Figure 6 shows an additional MSC ("post-arity" in the PROPEL-terminology, cf. page 4) generated for the MSC from Figure 5.

To evaluate the number of additional MSCs, we generated them in two settings, proven to provide the highest number of correct MSCs [20]. Every setting is defined by the heuristics used to construct the set of actors. The set of actors influences the identification of senders and receivers in the sentences translated to messages, which, in turn, influences the structure of the extracted MSC. This implies that different MSCs can be extracted from the same scenarios, and thus different number of additional MSCs can be generated, depending on the heuristics used to construct the set of actors.

We used two different linguistic heuristics to extract actors: one based on the analysis of sentence structure, and one based on the recognition of named entities[3], cf. [20]. 41 out of 42 scenarios from the case study presented in Section 2 were used for evaluation. One scenario was not taken

[3] `http://www.cnts.ua.ac.be/conll2003/ner/`

heuristics to extract actors	number of generated additional MSCs, for 41 MSCs extracted from the text							
	maximum, for one MSC				total, for all MSCs			
	Level 1	Level 2	Level 3	max. sum, level 1-3	Level 1	Level 2	Level 3	sum, level 1-3
recognition of named entities	588	70	42	**658**	4935	455	511	**5901**
analysis of sentence structure	588	140	252	**686**	4970	469	1050	**6489**

Table 1. Statistics: generation of additional MSCs in batch mode

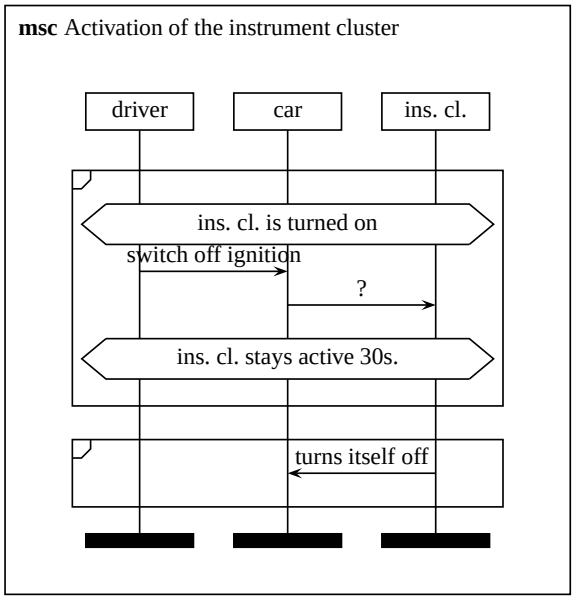

Figure 5. Cut of the MSC from Figure 2, Level 2

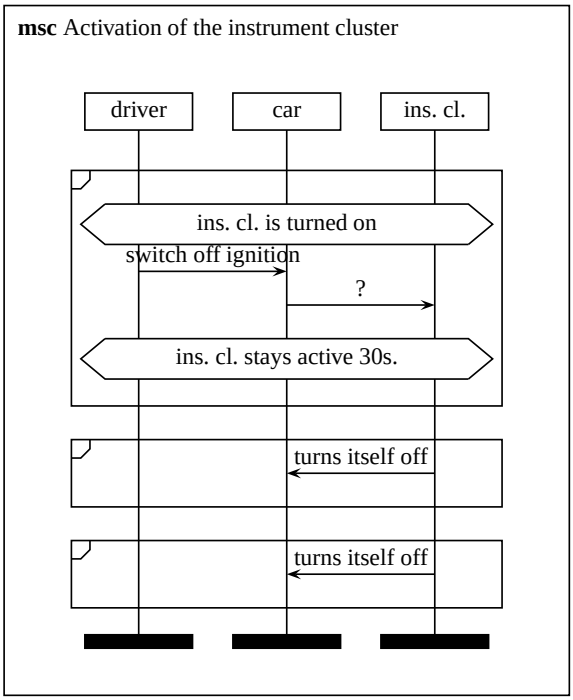

Figure 6. Generated additional MSC (postarity) for the MSC from Figure 5

into account due to technical difficulties of its batch processing. Table 1 shows the evaluation results: the number of generated additional MSCs for levels 1-3. In this table, columns "Level n" refer to the number of questions generated when cutting the blocks of the corresponding level.

In Table 1, it is easy to see that up to **686** additional MSCs can be generated for a single given MSC, and that the overall number of generated additional MSCs can amount to **6489**. In the brute force approach from page 4, approximately **19,000** additional MSCs would be generated in the setting with named entity recognition, and approximately **24,000** in the setting with sentence structure analysis. As every additional MSC has to be validated by the human analyst, the figures presented in Table 1, although significantly better than for the brute force approach, still imply high workload for the analyst. The interactive approach, presented in Section 3.4, reduces this workload at a price of slight reduction of completeness of the generated set of additional MSCs.

3.4. Generation and Visualization of Additional MSCs: Interactive Mode

Interactive generation and visualization of additional MSCs is based on the following key idea: in the batch version, the tool generates all possible cuts for an MSC, preserving the block structure. In the interactive version, the tool consecutively asks the user where the prefix ends, where the suffix starts, and where the boundary between action and response lies. These questions are asked by visualizing the MSC with a possible cut, and asking the user whether the visualized cut makes sense. An example of such visualized cut is shown in Figure 7.

The visualized cuts can be incomplete, as in Figure 7. This allows to determine the end of the prefix first, then the start of the suffix, and then the cut between the action and response. Theoretically, in a really interactive system it could be possible to let the user cut the MSC in prefix, suf-

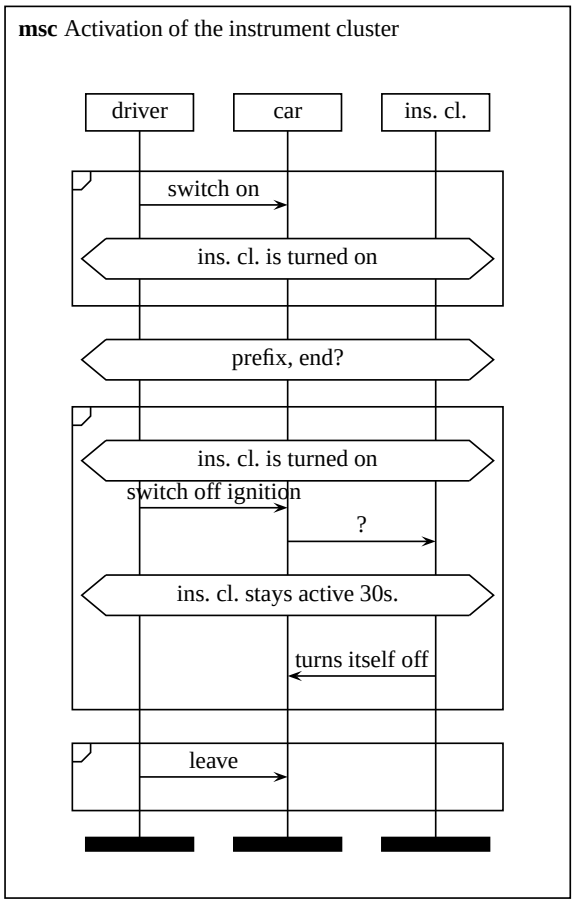

Figure 7. Possible partial cut of the MSC from Figure 4

fix, action, and response, for example just by placing special markers into the MSC to be cut. However, our current Tcl/Tk GUI[4] is limited to yes/no questions only. Thus, our system generates MSCs containing questions, like the one shown in Figure 7, visualizes such MSCs, and collects the yes/no answers from the user. If the user answers "yes" to the generated question-MSC, the corresponding cut is fixed. Otherwise, the position of the question-cut is moved one block down the time line, and the new question-MSC is presented to the user. The boundaries of the prefix and the suffix are determined consecutively, which implies that the number of additional MSCs generated for an MSC of length n is $\mathcal{O}(n)$, and not $\mathcal{O}(n^3)$, as for the batch mode.

First experiments with the interactive determining of MSC cuts have shown that there is one further possibility to reduce the number of questions that the user has to answer. It turned out that some blocks are repeatedly used in several MSCs. For example, the first block from Figure 4 is used in many MSCs dealing with different ways of turning the instrument cluster on or off. For every such block used in several MSCs, it makes sense to generate additional MSCs for it only once, and not every time it occurs in an MSC.

Implementation of this optimization resulted in the impossibility to evaluate the number of generated additional MSCs for a single given MSC. Thus, we evaluated the total number of generated additional MSCs only. As the total number of generated additional MSCs determines the actual workload for the analyst, the evaluation results are still meaningful.

Tables 2 and 3 show the number of generated additional MSCs in the interactive mode. As for Table 1, columns "Level n" refer to the number of questions generated when cutting the blocks of the corresponding level, and the columns "Whole MSCs" refer to the number of questions generated when cutting whole MSCs. Table 2 shows the results for the setting with actor extraction by means of named entity recognition, corresponding to the first line of Table 1. Table 3 shows the results for the setting with actor extraction by sentence structure analysis, corresponding to the second line of Table 1. It is easy to see that in both settings, the user has to answer significantly less questions than in the batch mode. Although the number of questions might seem high (on total, up to 915 questions), all the questions are yes/no-questions and can be answered by a single click. On total, in every setting it took us approximately three hours to answer all generated questions, which corresponds to approximately 10 seconds per question. In the long run, it should be empirically evaluated whether the question answering remains quick in an industrial setting.

When all questions are answered, the MSCs can be generalized, as proposed by Avrunin et al. [26], and used for further system development. Furthermore, the set of MSCs obtained after answering all questions can be used to generate an automaton for every actor, as sketched in Section 6.

4. Related Work

Work related to the presented paper can be subdivided in two areas: work on natural language processing (NLP) in requirements engineering, and work on validation of scenarios and MSCs. Both areas are presented below.

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 analysis of document quality were introduced, for example, by Rupp [24], Fabbrini et al. [11], Kamsties et al. [16], and Nuseibeh et al. [7]. These

[4] `http://www.tcl.tk/`

	Whole MSCs	Level 1	Level 2	Level 3	Total
"prefix end" questions	35	25	19	9	88
"suffix begin" questions	70	22	24	7	123
"action/response"-cut questions	35	22	21	8	86
additional MSCs as required by PROPEL	182	154	133	49	518
Total	322	223	197	73	**815**

Table 2. Statistics: generation of additional MSCs in interactive mode, setting with actors extracted by means of named entity recognition

	Whole MSCs	Level 1	Level 2	Level 3	Total
"prefix end" questions	43	29	24	9	105
"suffix begin" questions	53	27	28	12	120
"action/response"-cut questions	28	25	26	9	88
additional MSCs as required by PROPEL	196	175	168	63	602
Total	320	256	246	93	**915**

Table 3. Statistics: generation of additional MSCs in interactive mode, setting with actors extracted by means of sentence structure analysis

approaches have in common that they define writing guidelines and measure document quality by measuring the degree to which the document satisfies the guidelines. These approaches have a different focus from the approach presented in this paper: their aim is to detect poor phrasing and to improve it, they do not target at behavior analysis.

Another class of approaches, like for example those by Goldin and Berry [13], Abbott [1], and Sawyer et al. [25] analyze the requirements documents, extract application specific concepts, and provide an initial model of the application domain. These approaches do not perform any behavior analysis, either.

The approaches analyzing system behavior, as, for example, those by Vadera and Meziane [28], Rolland and Ben Achour [23], Gervasi and Zowghi [12], Breaux et al. [4], Avrunin et al. [26], and and Díaz et al. [10] translate requirements documents to executable models by analyzing linguistic patterns. In this sense they are similar to our work. Vadera and Meziane 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. Rolland and Ben Achour state that system behavior can consist of pairs of service request and provision, which is comparable to the *response* pattern used in this paper. However, although they introduce event pairs, they do not use event pairs for validation. Gervasi and Zowghi 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. introduce a restricted language and translate this language to description logic. The approach by Avrunin et al. is similar to the approach by Gervasi and Zowghi in the sense that it introduces a restricted natural language. The difference lies in the formal representation means: Gervasi and Zowghi stick to first order logic, Avrunin et al. translate natural language to temporal logic. Our work goes further than the above three approaches, as we do not assume or enforce language restrictions. Díaz et al. introduce a transformation technique producing UML sequence diagrams. However, the input to this transformation technique is a semantical representation of the sentences and not plain text as in our work.

Validation of MSCs: The play in/play out approach by Harel and Marelly [15] is in its spirit most close to the presented work: It allows to specify system behavior as a set of Live Sequence Charts (an extention of the MSC notation), and then to simulate the system by sending stimuli and observing the system reaction. When simulating the system, the play in/play out approach assumes that the specification is complete and thus produces exactly the specified reactions. In our work we challenge this assumption and generate new additional MSCs representing possible system behaviors, in order to see whether the specification is really complete. In the long run, both approaches can be integrated: new generated MSCs that were approved as possible system behaviors can serve as additional input to the play in/play out approach.

There also exist a whole class of approaches dealing with generation of additional MSCs and MSC validation [3, 8, 21, 27]. These approaches are based on the in-

teractive procedure to learn finite automata from a set of examples. When necessary, the procedure to learn automata can generate new MSCs and ask the user whether the generated MSC represents a possible system behavior. The main drawback of these approaches is their complexity: For example, Smyle [3] is *PSPACE*-complete, and the approach by Letier et al. [21] requires exponential memory (exponential in the number of actors). Our approach presented in this paper is much cheaper in the sense of resource consumption, at the price of heuristic validation instead of precise automata learning.

To summarize, to the best of our knowledge, there is no approach to requirements documents analysis, that is able to analyze scenarios written in natural language, to identify missing information, and to visualize and validate resulting MSCs, yet.

5. Summary

Requirements Engineering is a non-trivial task and the presented approach does not claim to solve all its problems. However, it solves an important problem of requirements analysis, namely visualization and validation of the behavior extracted from textual documents, as well as generalization of explicitly specified system behavior. Visualization and validation ensure that the textual specification is correctly interpreted. Generalized behavior builds the basis for the formalization of system behavior in automata, which, in turn, can be used both for system simulation and code generation. In a nutshell, the presented approach provides a bridge between textual specifications and executable models.

6. Future Work

The presented approach provides a first step in the transition from textual behavior specifications to executable models. In the long run, the presented approach can be used as the basis of refinement-based development, as well as for automata generation. Broy et al. [5] proposed stepwise refinement of system functionality. In their specification process, MSCs build the basis for the first step, namely partial specification of the system behavior. To obtain complete behavior specification, partial specifications should be completed and refined. The process to generate possible system runs, as presented in Section 3, can be used to complete and validate partial specifications.

To go further in specification completion, the presented approach can be integrated with the approach by van Lamsweerde et al. [8] and the tool Smyle [3]. These approaches translate a set of MSCs to automata (one automaton for every actor). When additional information is necessary to produce automata, they generate new MSCs and ask the user, which of the new MSCs represent possible system behavior.

Experiments with the tool Smyle have shown that the number of generated MSCs is extremely high. This means high workload for the user of the tool. Integration with the approach presented in this paper can reduce the number of the questions to the user in the following way:

1. For MSCs generated as described in Section 3, the user decides which of them represent possible system runs.
2. When Smyle generates a new MSC, the integrated tool tries to decide on the basis of Step 1, whether the new MSC represents a possible system run.
3. If such decision is not possible (i.e., if the information obtained in Step 1 is not sufficient), the integrated tool forwards the question to the user.

Integration sketched above would combine the advantages of both approaches: it would imply complete check of possible system behaviors, provided by Smyle, and would imply lower workload than pure Smyle, due to heuristic validation developed in our work.

References

[1] R. J. Abbott. Program design by informal English descriptions. *Communications of the ACM*, 26(11):882–894, 1983.

[2] B. W. Boehm. *Software Engineering Economics*. Prentice-Hall, 1981.

[3] B. Bollig, J.-P. Katoen, C. Kern, and M. Leucker. Replaying Play in and Play out: Synthesis of Design Models from Scenarios by Learning. Technical Report AIB-2006-12, RWTH Aachen, Germany, October 2006. `http://www.smyle-tool.org/wordpress/wp-content/uploads/2008/02/2006-12.pdf`, accessed 10.04.2009.

[4] T. D. Breaux, A. I. Antón, and J. Doyle. Semantic parameterization: A process for modeling domain descriptions. *ACM Trans. Softw. Eng. Methodol.*, 18(2):1–27, 2008.

[5] M. Broy, I. Krüger, and M. Meisinger. A formal model of services. *ACM Transactions on Software Engineering Methodology (TOSEM)*, 16(1), 2007. available at `http://doi.acm.org/10.1145/1189748.1189753`.

[6] K. Buhr, N. Heumesser, F. Houdek, H. Omasreiter, F. Rothermehl, R. Tavakoli, and T. Zink. Daimler-Chrysler demonstrator: System specification instrument cluster, 2004. `http://www.empress-itea.org/deliverables/D5.1_Appendix_B_v1.0_Public_Version.pdf`, accessed 11.01.2007.

[7] F. Chantree, B. Nuseibeh, A. de Roeck, and A. Willis. Identifying nocuous ambiguities in natural language requirements. In *RE '06: Proceedings of the 14th IEEE International Requirements Engineering Conference (RE'06)*, pages 56–65, Washington, DC, USA, 2006. IEEE Computer Society.

[8] C. Damas, B. Lambeau, and A. van Lamsweerde. Scenarios, goals, and state machines: a win-win partnership for model synthesis. In *SIGSOFT FSE*, pages 197–207, 2006.

[9] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Patterns in property specifications for finite-state verification. In *ICSE'99*, pages 411–420, 1999.

[10] I. Díaz, O. Pastor, and A. Matteo. Modeling interactions using role-driven patterns. In *RE'05: Proceedings of the 13th IEEE International Conference on Requirements Engineering (RE'05)*, pages 209–220, Washington, DC, USA, 2005. IEEE Computer Society.

[11] F. Fabbrini, M. Fusani, S. Gnesi, and G. Lami. 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*, pages 97–105, Greenbelt, Maryland, 2001. IEEE Computer Society. `http://fmt.isti.cnr.it/WEBPAPER/fabbrini_nlrquality.pdf`.

[12] V. Gervasi and D. Zowghi. Reasoning about inconsistencies in natural language requirements. *ACM Trans. Softw. Eng. Methodol.*, 14(3):277–330, 2005.

[13] L. Goldin and D. M. Berry. AbstFinder, a prototype natural language text abstraction finder for use in requirements elicitation. *Automated Software Eng.*, 4(4):375–412, 1997.

[14] B. J. Grosz, A. K. Joshi, and S. Weinstein. Centering: A framework for modeling the local coherence of discourse. *Computational Linguistics*, 21(2):203–225, 1995.

[15] D. Harel and R. Marelly. *Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine*. Springer, 2003.

[16] E. Kamsties, D. M. Berry, and B. Paech. Detecting ambiguities in requirements documents using inspections. In *Workshop on Inspections in Software Engineering*, pages 68 –80, Paris, France, 2001.

[17] L. Kof. Scenarios: Identifying missing objects and actions by means of computational linguistics. In *15th IEEE International Requirements Engineering Conference*, pages 121–130, New Delhi, India, October 15–19 2007. IEEE Computer Society Conference Publishing Services.

[18] L. Kof. Treatment of Passive Voice and Conjunctions in Use Case Documents. In Z. Kedad, N. Lammari, E. Méthais, F. Meziane, and Y. Rezgui, editors, *Application of Natural Language to Information Systems*, volume 4592 of *LNCS*, pages 181–192, Paris, France, June 27–29 2007. Springer.

[19] L. Kof. From Textual Scenarios to Message Sequence Charts: Inclusion of Condition Generation and Actor Extraction. In *16th IEEE International Requirements Engineering Conference*, pages 331–332, Barcelona, Spain, September 10-12 2008. IEEE Computer Society Conference Publishing Services.

[20] L. Kof. Requirements Analysis: Concept Extraction and Translation of Textual Specifications to Executable Models. In H. Horacek, R. Mu ̄noz, and E. Méthais, editors, *Application of Natural Language to Information Systems*, LNCS, Saarbrücken, Germany, June 24-26 2009. Springer, to appear.

[21] E. Letier, J. Kramer, J. Magee, and S. Uchitel. Monitoring and control in scenario-based requirements analysis. In *ICSE'05: Proceedings of the 27th international conference on Software engineering*, pages 382–391, New York, NY, USA, 2005. ACM.

[22] L. Mich, M. Franch, and P. Novi Inverardi. Market research on requirements analysis using linguistic tools. *Requirements Engineering*, 9(1):40–56, 2004.

[23] C. Rolland and C. Ben Achour. Guiding the construction of textual use case specifications. *Data & Knowledge Engineering Journal*, 25(1–2):125–160, March 1998.

[24] C. Rupp. *Requirements-Engineering und -Management. Professionelle, iterative Anforderungsanalyse für die Praxis*. Hanser–Verlag, second edition, 05. 2002. ISBN 3-446-21960-9.

[25] P. Sawyer, P. Rayson, and K. Cosh. Shallow knowledge as an aid to deep understanding in early phase requirements engineering. *IEEE Trans. Softw. Eng.*, 31(11):969–981, 2005.

[26] R. L. Smith, G. S. Avrunin, L. A. Clarke, and L. J. Osterweil. Propel: an approach supporting property elucidation. In *ICSE'02: Proceedings of the 24th International Conference on Software Engineering*, pages 11–21, New York, NY, USA, 2002. ACM.

[27] S. Uchitel, G. Brunet, and M. Chechik. Behaviour Model Synthesis from Properties and Scenarios. In *ICSE '07: Proceedings of the 29th international conference on Software Engineering*, pages 34–43, Washington, DC, USA, 2007. IEEE Computer Society.

[28] S. Vadera and F. Meziane. From English to formal specifications. *The Computer Journal*, 37(9):753–763, 1994.