Scenarios: Identifying Missing Objects and Actions by Means of Computational Linguistics

Leonid Kof

Fakultät für Informatik, Technische Universität München,
Boltzmannstr. 3, D-85748 Garching bei München, Germany
E-mail: kof@informatik.tu-muenchen.de

Abstract

In industrial requirements documents natural language is the main presentation means. In such documents, system behavior is specified in the form of scenarios, 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.

This paper presents an approach that analyzes textual scenarios with the means of computational linguistics, identifies where communicating objects or whole actions are missing in the text, completes the missing information, and creates a message sequence chart (MSC) including the information missing in the textual scenario. Finally, this MSC is presented to the requirements analyst for validation. The paper presents also a case study in which scenarios from a requirement document based on industrial specifications were translated to MSCs. The case study shows the feasibility of the approach.

1. Document Authors are not Aware that some Information is Missing

Some kind of requirements document is usually written at the beginning of every software project. The majority of these documents are written in natural language, as the survey by Mich et al. shows [13]. This results in the fact that the requirements documents are imprecise, incomplete, and inconsistent. The authors of requirements documents are not always aware of these document defects. From the linguistic point of view, document authors introduce three defect types, without perceiving them as defects, cf. Rupp [15]:[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."

It is one of the goals of requirements analysis, to find and to correct the defects of requirements documents. This paper focuses on the "deletion"-defects in scenarios. Deletion manifests itself in scenarios in the form of missing action subjects or objects or even in whole missing actions. One of the reasons for the deletion may be the fact that some information is too obvious for the author of the requirements document, so that she finds it unnecessary to write down this information. It is the goal of the approach presented in this paper, to identify missing parts of the scenarios written in natural language and to produce message sequence charts (MSCs) containing the reconstructed information.

For the remainder of the paper we use the following terminology: A *scenario* is a sequence of natural language sentences, each sentence representing some *action*. An *MSC* is a set of *communicating objects* and a sequence of *messages* sent and received by these objects.

The remainder of the paper is organized as follows: Section 2 introduces the case study used to evaluate the presented approach. Section 3 explains ontology extraction, a technology necessary for the analysis of scenarios. Section 4 is the technical core of the paper, it presents the algorithms transforming scenarios to MSCs. Section 5 presents the evaluation of the approach on a case study. Finally, Sections 6, 7, and 8 present an overview of related work, the summary of the paper, and possible directions for future work, respectively.

[1] The following definitions are translations of the definition from [15], in German

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 [5], some missing facts can be identified. The aim of the approach presented in this paper is to identify missing parts of scenarios and complete them.

The instrument cluster specification describes the optical design of the instrument cluster as a part of the car dashboard, its hardware, and, most importantly, its behavior. The behavior is specified as a set of scenarios, like this:

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.

If we translate this scenario to an MSC containing solely a message sequence, we get the MSC in Figure 1. This translation is surely incomplete, in the sense that timing, "...stays active for **30 seconds**", is not taken into account. Theoretically, MSCs allow also to set timers and for every timer to perform some actions when the timer expires. Such MSCs can be constructed by a human analyst. However, extraction of MSCs with timers with the means of computational linguistics requires semantic analysis, going far beyond the capabilities of state-of-the-art linguistic tools. Thus, in the presented work we focus on MSCs consisting of message sequences only.

There are apparent problems even for the construction of MSCs without timers: For example, in the second sentence of the above example, "The instrument cluster is turned on and stays active", either the message sender or receiver is missing, and the connection between the first and the second message is not specified. Furthermore, the MSC contains missing messages from the car to the instrument cluster, most probably, something like "turn on" and "turn off in 30 seconds". These messages were simply forgotten by the scenario author. It is one of the goals of the approach presented in this paper, to identify such missing messages.

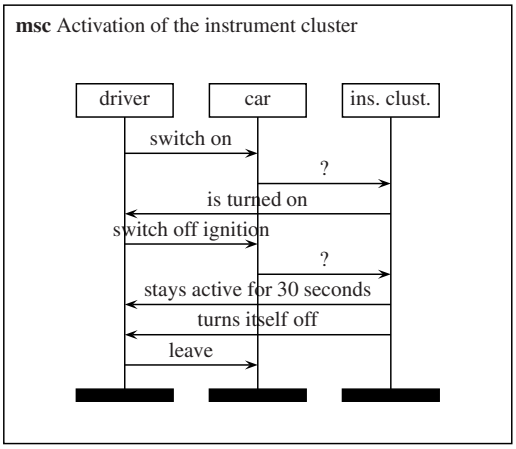

Figure 1. Scenario "Activation of the instrument cluster", manual translation to MSC

3. Identifying Communicating Objects in Scenarios

To identify communicating objects in scenarios, to distinguish them from other nouns, it is useful to have a list of application specific names of potential communicating objects. This idea is similar to the glossary in CICO [8], where application specific concepts are manually annotated either as potential message senders or receivers, or sent data.

The approach presented in this paper differs from CICO in the way how the actions specified in single sentences are glued together, cf. Section 4, and in the way how the list of communicating objects is constructed. In this paper an *ontology* extracted from the requirements document is used instead of a glossary, whereas in CICO a glossary has to be handcrafted.

In computer science an ontology consists of three parts: a list of concepts, a classification of these concepts ("is-a" relation, taxonomy), and a set of non-taxonomic relations. An ontology can be extracted from a requirements documents in four steps, as described in [12]:

- Every sentence is parsed, the subject and the objects of every sentence are extracted.
- Each subject and each object is clustered according to grammatical context in which it occurs.
- Every pair of overlapping clusters is joined to a larger cluster, representing more general concepts. "is-a" relation (taxonomy) is derived from the cluster hierarchy.
- A non-taxonomic relation is assumed for every two concepts that often co-occur in the same sentence.

An ontology is vital when analyzing scenarios: for example, the sentence "The driver switches off the ignition" can be translated in two ways to an MSC message:

- The driver sends the message "switch off" to the ignition.

- The driver sends the message "switch off ignition" to some unspecified receiver. In this particular example the receiver is most probably the car.

Thus, identification of message senders and receivers is a non-trivial problem in the process of translation of scenarios to MSCs. Given an ontology, a possible heuristics to identify message senders and receivers would be the following: if the concept occurring in the sentence before/after the verb is also contained is the ontology, it is the message sender/receiver.

At the first glance, these heuristics has a obvious drawbacks: when the ontology contains concepts that are neither message senders nor receivers, wrong communicating objects are adopted for the MSC. Furthermore, if the ontology misses some concepts, correct message senders and receivers are not identified either. However, this seeming drawback is really an advantage: Before an ontology can be used as a common project vocabulary, it must be validated. If errors in the MSCs generated from scenarios arise from missing or redundant concepts in the ontology, they contribute to ontology validation. The case study (cf. Section 5.2) showed that the ontology correction induced by the analysis of scenarios is *not* time consuming.

4. From Scenarios to Message Sequence Charts

4.1. Translating Sentences to Messages: Linguistic Prerequisites

When analyzing scenarios, we want to translate every sentence of a textual scenario to an MSC message. Furthermore, we want to identify which messages are missing, as in the example in Figure 1. When translating a sentence to an MSC message, we decompose the sentence in three constituents: message sender, message content, message receiver. For this purpose we consider the sentence parts before and after the main verb:

- The message sender/receiver is the longest word sequence before/after the verb, contained in the ontology.
- The message content is the verb plus the word sequence between the verb and the first word of the message receiver, except for the determiners. If the receiver/sender cannot be identified, the message content is the whole part of the sentence after/before the sender/receiver.

For example, if the concepts "driver", "drunk driver", and "car" are all contained in the ontology, then, in the sentence "The drunk driver switches on the car", "drunk driver" is identified as the message sender, "car" as the receiver, and "switch on" as the message content. These heuristics work under the assumption that the sentence is in active voice and contains exactly one verb.

As the technical means for identifying the verb, a part-of-speech (POS) tagger is used. Such a tagger assigns a POS-tag (substantive, verb, adjective, ...) to every word. Currently available taggers, as for example the tagger by Ratnaparkhi [14], have the precision of about 97%, which makes them unlikely to become an extra error source.

In the process of sentence decomposition we have to take two linguistic error sources into consideration:

1. The sentence with POS-tags does not contain any verb tag, what can happen in two cases:
   - The sentence really contains no verb.
   - The sentence contains a verb, but no verb tag due to a tagger error. This situation is rather unlikely given the tagger precision of 97%.

   In both these cases the sentence has to be rephrased in order to be translated to an MSC message.
2. The sentence with POS-tags contains more than one verb tag, what happens in the following cases:
   - The sentence contains a modal verb, like "must" or "should", together with a normal verb. In this case it is even for the human analyst unclear, how the sentence can be translated to an MSC message. Thus, the sentence should be rephrased anyway.
   - The sentence contains passive, like "X is sent a message" or "X is sent to Y by Z". In the first case a human analyst would make the grammatical subject "X" to the message receiver. In the second case a human analyst would make the grammatical subject "X" to the message content. Due to this ambiguity and resulting necessity for advanced linguistic analysis, we consider passive as an error case in this paper. Analysis of passive sentences is a part of future work, cf. Section 8.
   - The sentence contains several subordinate sentences, like "If X sends Y to Z, Z sends ...". In this case a human analyst would produce two MSCs for a single scenario: one MSC where X sends Y to Z and Z reacts, and one MSC where X does not send Y to Z. Our analysis software cannot generate several MSCs for one scenario yet. For this reason we consider subordinate sentences as an error case. Analysis of compound sentences is a part of future work.
   - The sentence contains several actions, like in "X sends Y to Z and replies something to T". In this

case a human analyst would produce two messages for a single sentence: X sends Y to Z, X sends something to T. Our analysis software cannot generate several MSC messages for one sentence yet. For this reason we consider several actions in one sentence as an error case. Analysis of sentences containing several actions is a part of future work.

From the linguistic point of view, all these error cases can be detected with the same technique: The sentences contain more than one verb, thus, the tagged sentence contains the regular expression VB.*VB[2]

4.2. Glueing Messages Together

From now on we consider sentences containing exactly one verb. Even in this case the identification of message senders and receivers is not always simple. In order to avoid the identification of wrong objects, e.g. sent messages, as communicating objects for an MSC , we accept only objects contained in the previously extracted ontology as message senders and receivers. This leads to the problem that in some sentences we cannot identify the message sender or receiver. The problem arises additionally from the fact that some sentences, like "instrument cluster turns on", do not explicitly mention the sender or the receiver. We solve this problem by introducing a default sender and receiver for every sentence under analysis. Obviously, the default sender and receiver depend on the messages previously sent. Management of this dependency is *the core* of the approach to MSC construction presented in this paper.

To identify default senders and receivers, we organize the messages in a stack. The idea of organization of messages in a stack is based on the following analogy: Grosz et al. [10] 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"

This idea can be transferred to MSCs in the following way: 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.

[2]VB is the verb tag.

The remainder of this section describes the transfer of the situation stack idea to MSCs in more detail: Section 4.2.1 introduces stack-based identification of the senders and receivers in incomplete sentences and Section 4.2.2 shows stack management after the identification of senders and receivers.

4.2.1. Identifying the Sender and Receiver in Incomplete Sentences

When translating a sentence to an MSC message, we say that the message sender/receiver is the longest word sequence before/after the verb[3] that is contained in the ontology. The problem arises when we cannot identify the sender or receiver. For example, if we translate the sentence "The instrument cluster turns on" to an MSC message, we know that "instrument cluster" is the message sender, but we do not know the receiver. To identify missing message senders and receivers, we analyze the sender and receiver of the message on the top of the stack. Here we assume that the message stack contains the sent messages, for which no corresponding answer message is available yet. The exact rules of stack management will be described below, see Section 4.2.2.

When identifying missing message receiver, we distinguish three cases, shown in Figure 2:

- The sender of the message under analysis equals to the receiver of the message on the top of the stack, as in Figure 2(a). In this case we assume that the message under analysis is an answer to the message on the top of the stack. The receiver of the message is then the sender of the message on the top of the stack. In the case of Figure 2(a) it is "Object 1".
- The sender of the message under analysis equals to the sender of the message on the top of the stack, as in Figure 2(b). In this case we assume that the message under analysis augments the message on the top of the stack and, thus, has the same receiver. In the case of Figure 2(b) it is "Object 2".
- The sender of the message under analysis does not equal to the sender nor to receiver of the message on the top of the stack, as in Figure 2(c). In this case we assume that some message is missing in the MSC, denoted by the dashed message in Figure 2(c), and that the message under analysis is an answer to the missing message. The sender of the missing message and the receiver of the message under analysis coincide with the receiver of the message on the top of the stack. In the case of Figure 2(c) it is "Object 2".

A missing message sender can be identified in a similar way.

[3]Please note that here we consider only sentences containing exactly one verb.

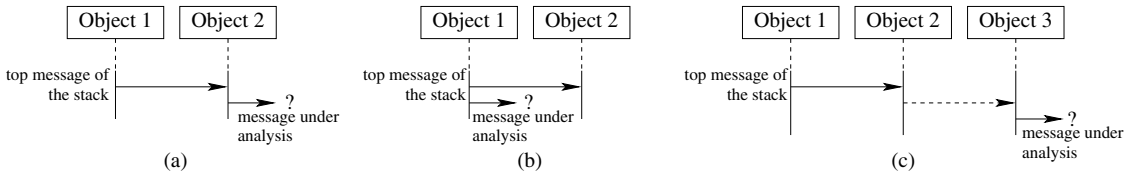

Figure 2. Missing message receiver, possible situations

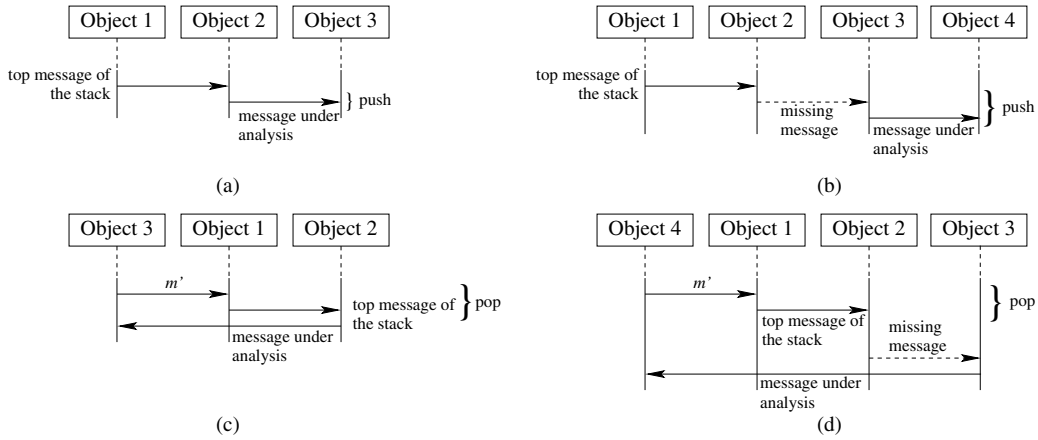

Figure 3. Identification of missing messages and stack management

4.2.2. Identifying Missing Messages

When every message has a defined sender and receiver, we can determine the necessary operations on the message stack. Following situations are taken into account:

1. The sender of the message under analysis equals to the sender of the first message, starting the whole MSC. In this case we assume that the message under analysis starts a completely new MSC segment. This is the case, for example, for the sentence "After the trip the driver switches off the ignition" in the introductory scenario in Section 2. Thus, we empty the stack and reinitialize it with the new message.

2. The sender and the receiver of the message under analysis are equal to the sender and the receiver of the message on the top of the stack, respectively. In this case we assume that the new message augments the message on the top of the stack. Thus, we just add the new message to the MSC but do not change the stack.

3. The sender of the message under analysis equals to the receiver of the message on the top of the stack, as in Figures 3(a) and 3(c). We call an object "active" if it is a sender of some message contained in the stack (for example, "Object 1" and "Object 3" in Figure 3(c)) or is the receiver of the top message on the stack.

   - If the receiver of the message under analysis is not an active object (Figure 3(a)), we add the new message to the MSC and push it onto the stack.
   - If the receiver of the message under analysis (m) is an active object ("Object 3" in Figure 3(c)), we assume that m is the reply to some message m' sent by this active object. We pop m' and all the messages contained in the stack above m'.

4. The sender of the message under analysis is *not* equal to the receiver of the message on the top of the stack, as in Figures 3(b) and 3(d). In this case some message is missing. We add a missing message to the MSC, as shown in Figures 3(b) and 3(d).

   - If the receiver of the message under analysis is not an active object (Figure 3(b)), we add a new message to the MSC and push it onto the stack.
   - If the receiver of the message under analysis (m) is an active object ("Object 4" in Figure 3(d)), we assume that m is the reply to some message m' sent by this active object. We pop m' and all the messages contained in the stack above m'.

Application of the procedure described above can be easily exemplified on the scenario presented in Section 2. Table 1 shows a slightly changed scenario, so that each sentence contains exactly one verb.[4] For the analysis we as-

[4]To make the table compacter, "instrument cluster" is abbreviated in Table 1 as "ins. clust.".

sentence	extracted sender	extracted receiver	sender assumed for MSC	receiver assumed for MSC	stack action (push/pop)	pushed/popped messages
The driver switches on the car.	driver	car	driver	car	push	driver → car
The instrument cluster turns on.	ins. clust.	—	ins. clust.	car	2x push 2x pop	car → ins. clust. ins. clust. → car
The instrument cluster stays active.	ins. clust.	—	ins. clust.	car	—	—
After the trip the driver switches off the ignition.	driver	—	driver	car	empty, push	driver → car
The instrument cluster stays active for 30 seconds.	ins. clust.	—	ins. clust.	car	2x push 2x pop	car → ins. clust. ins. clust. → car
The instrument cluster turns itself off.	ins. clust.	—	ins. clust.	car	—	—
The driver leaves the car.	driver	—	driver	car	empty, push	driver → car

Table 1. Organization of MSC messages in a stack

sume that the previously extracted ontology contains "instrument cluster", "driver", and "car", but no other nouns used in the scenario.

We initialize the analysis with an empty stack. In the first sentence the driver is identified as the message sender and the car as the message receiver. The message "driver → car" is pushed onto the stack. In the second sentence the instrument cluster is identified as the message sender, but the sentence does not contain any message receiver. The situation corresponds to Figure 2(c). Thus, "car" is identified as the receiver. Then, the missing message "ins. clust. → car" and the message "car → ins. clust." are added to the MSC, as in Figure 3(d). The next sentence, "The instrument cluster stays active", results in a message that augments the previous message. Thus, the stack remains unchanged. Then, the next sentence causes the emptying of the stack and the same sequence of push and pop operations is performed again. The resulting MSC is shown in Figure 5.

This example shows that the presented idea of message stack works for this simple scenario. The approach was also evaluated on more complicated examples. The results of the evaluation are presented in the next section.

5. Case Study and Evaluation

The presented approach was evaluated on the instrument cluster specification [5]. Although not used in an industrial development project, this specification was derived from real industrial documents. This specification was also intended to serve as the contract basis between the car manufacturer and the supplier of the instrument cluster. The ontology, necessary for the translation of scenarios to MSCs, was extracted in our previous work [12].

The case study consisted of three steps: First, the scenarios had to be rewritten. Section 5.1 documents the changes that were necessary and the time spent. Then, the scenarios were translated to MSCs using the ontology extracted in [12]. It turned out that the ontology contains some unnecessary concepts and, at the same time, does not contain some necessary ones. Section 5.2 presents the necessary ontology changes in detail and documents the time spent on these changes. Finally, Section 5.3 presents the MSCs extracted from the corrected scenarios with the corrected ontology.

5.1. Case Study: Correction of Linguistic Deficiencies

The original scenarios from [5] had to be rephrased in order to be translated to MSCs. The necessity to rewrite the scenarios was caused by the extensive usage of passive in the original version. Furthermore, some sentences contained several actions, like "The instrument cluster stays active for 30 seconds and then turns itself off".

On the total, 42 scenarios were analyzed in the case study. None of the scenarios satisfied the linguistic requirements, stated in Section 4.1, out of the box. 37 scenarios were rephrased in less than two hours. For the 5 remaining scenarios the necessary phrasing corrections were too large due to extensive usage of passive and combination of several actions in one sentence. These 5 scenarios were not rephrased and not translated to MSCs either. Table 2 shows the corrections performed on three of the scenarios. Sentences in bold font are the new versions of the corresponding sentences on the left hand side. It is easy to see that the corrections are minimal.

Use Case: activation of the instrument cluster	*Use Case: activation of the instrument cluster*
The driver switches on the car (ignition key in position ignition on).	The driver switches on the car (ignition key in position ignition on).
The instrument cluster is turned on and stays active.	**The instrument cluster turns on.**
	The instrument cluster stays active.
After the trip the driver switches off the ignition.	After the trip the driver switches off the ignition.
The instrument cluster stays active for 30 seconds and then turns itself off.	**The instrument cluster stays active for 30 seconds.**
	The instrument cluster turns itself off.
The driver leaves the car.	The driver leaves the car.
Use Case: temporary activation of the instrument cluster	*Use Case: temporary activation of the instrument cluster*
The driver opens the door.	The driver opens the door.
The instrument cluster is activated temporarily.	**The instrument cluster turns on temporarily.**
The instrument cluster turns itself off after 30 seconds.	The instrument cluster turns itself off after 30 seconds.
The driver leaves the car.	The driver leaves the car.
Use Case: show RPM (revolutions per minute)	*Use Case: show RPM (revolutions per minute)*
The driver switches on the car by turning the ignition key to the switched on position.	The driver switches on the car by turning the ignition key to the switched on position.
The car is switched on and the pointer of the rev meter display goes from the technical initial position to the initial position of the scale (0 min-1), damped as described below.	The pointer of the rev meter display goes from the technical initial position to the initial position of the scale (0 min-1), damped as described below.
The input signals from the motor are sent regularly.	**The motor regularly sends input signals.**
The system determines the engine speed and displays it.	**The system determines the engine speed.**
	The system displays the engine speed.
The driver switches off the car by turning the ignition key to the switched off position.	The driver switches off the car by turning the ignition key to the switched off position.
The car is switched off and the pointer of the rev meter display falls back to its technical initial position, damped as described below.	**The pointer of the rev meter display falls back to its technical initial position, damped as described below.**
The driver leaves the car.	The driver leaves the car.

Table 2. Original scenarios (left) and corrected scenarios (right)

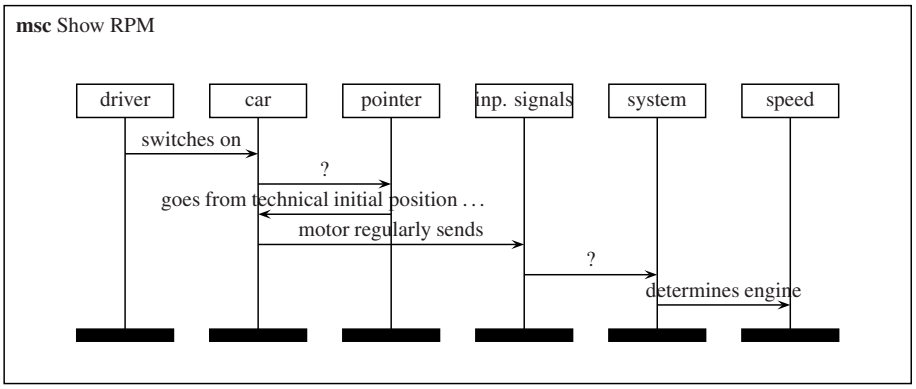

Figure 4. MSC (excerpt) for the scenario "Show RPM", extracted with the original ontology from [12]

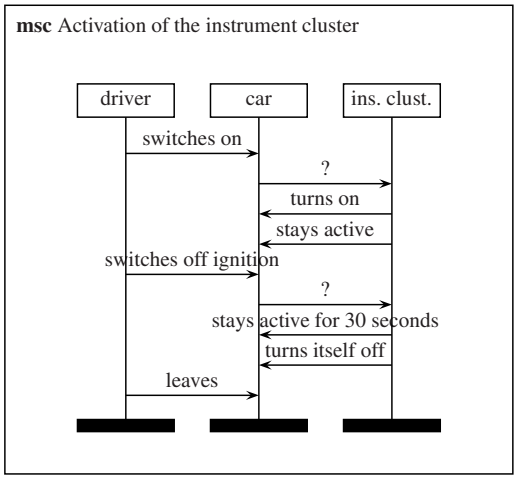

Figure 5. MSC for the scenario "Activation of the instrument cluster" from Table 2, extracted with the corrected ontology

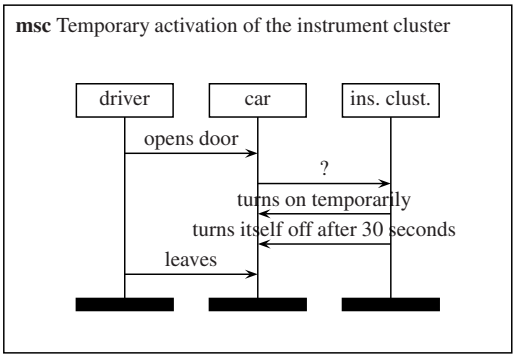

Figure 6. MSC for the scenario "Temporary activation of the instrument cluster" from Table 2, extracted with the corrected ontology

5.2. Ontology Validation via Construction of MSCs

In the presented approach the translation of sentences to MSC messages is strongly influenced by the available ontology: A word sequence is identified as a communicating object if and only if the ontology contains it. If the ontology does not contain all the necessary concepts, some communicating objects would not be identified. On the other hand, if the ontology contains dispensable concepts, these could be erroneously identified as communicating objects.

These considerations are exemplified in Figure 4. It shows the MSC for the use case "Show RPM", as presented in the bottom right part of Table 2, extracted using the original ontology from [12]. This MSC has obvious deficiencies: "input signals", "system", and "speed" are identified as communicating objects, "motor" is not identified as a communicating object, but, instead, just as a part of the message from "car" to "input signals".

Basing on the results of the validation of this MSC and also other extracted MSCs, the original ontology from [12] was corrected in the following way:

- The ontology branches containing non-hardware concepts (system messages, names of scale positions, sensor values (speed, temperature, ...)) were deleted.
- The concepts "motor", "sensor", "wheel speed sensor", "IC display", "digital speedometer display", "analog speedometer displays" were added to the branch "hardware" of the ontology. These concepts were used in the scenarios but missing in the ontology.

It is important to emphasize that all these changes, although performed manually in an ontology editor, were done in approximately two hours. This means that in a real software project, such an ontology adjustment would not be expensive either.

The necessity of ontology adjustment gives also another important hint for ontology construction: When constructing an ontology, it is vital to know what the ontology will be used for. If it is intended to serve as the project glossary, as in [12], it should contain all domain-specific concepts. If it is to be used for the analysis of scenarios, as in the presented paper, it should contain potential communicating objects only.

5.3. Extracted MSCs, Validation

When the ontology contains all the potential communicating objects and every sentence of the scenario satisfies our minimal grammatical requirements, every scenario can be translated to an MSC. Figures 5–7 show the MSCs constructed from scenarios shown in Table 2, right hand side. A simple comparison of Table 2 and Figures 5–7 shows that even for the sentences where the message sender or receiver are not explicitly mentioned, they are correctly identified. A comparison of the extraction results with other scenarios from [5] showed that for all the 37 scenarios that we rephrased to satisfy the grammatical requirements, the MSCs coincide with the scenarios, modulo identified missing messages, timers like "turns off **in 30 seconds**", and implicit message repetitions like "motor **regularly** sends ..."

Figures 5–7 show also that even in seemingly precise scenarios some information can be missing. To explicitly present these missing parts to the requirements analyst is also an important feature of the presented approach.

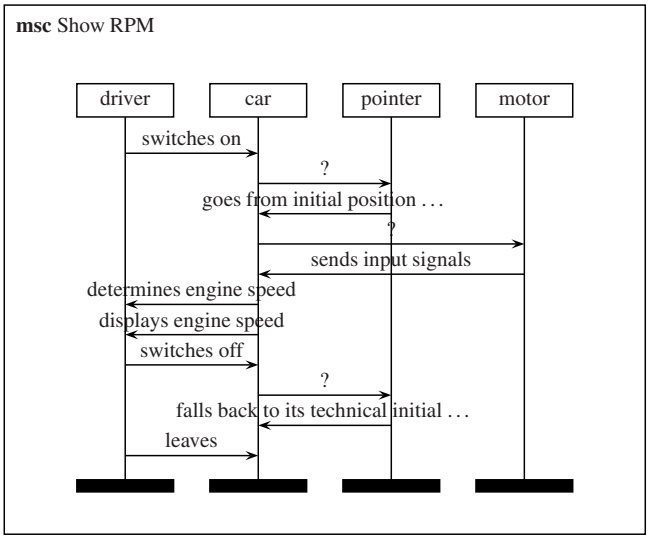

Figure 7. MSCs for the scenario "Show RPM" from Table 2, extracted with the corrected ontology

6. Related Work

The idea to use computational linguistic to analyze requirements documents is surely not new. Although Ryan claimed that natural language processing is not mature enough to be used in requirements engineering [16], there was a lot of work in this area in recent years.

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 for the analysis of document quality were introduced, for example, by Rupp [15], Fabbrini et al. [7], Kamsties et al. [11], and Chantree et al. [6]. All these approaches have in common that they define guidelines for document writing and measure document quality by measuring the degree to which the document satisfies the guidelines. These approaches are barely comparable to the approach presented in this paper, as they do not perform any behavior analysis.

Other class of approaches, like for example those by Goldin and Berry [9] and Abbott [1], analyze the requirements documents, extract application specific concepts, and provide an initial model of the application domain. They do not perform any behavior analysis, either.

The approaches analyzing system behavior, as for example those by Gervasi and Zowghi [8] and Tawbi et el. [17] translate the text to executable models by analyzing linguistic patterns. In this sense they are very similar to the approach presented in this paper. Gervasi and Zowghi translate textual requirements to first order logic. They analyze only the information directly available in the document, without reconstructing missing objects and actions. Tawbi et al. state that the behavior can consist of pairs of service request and provision pairs, which is comparable to the stack idea building the core of the presented approach. However, they do not explicitly introduce stack discipline and do not provide automation.

To summarize, to the best of our knowledge, there is no approach to requirements documents analysis, able to analyze scenarios and to identify missing pieces of behavior, yet.

7. Conclusion

The presented approach solves three important problems of the early requirements analysis phase:

- It detects missing information in scenarios.
- It validates the previously extracted application specific ontology or glossary.
- It translates textual scenarios to MSCs.

The constructed MSCs can be used in further software development, when validated. Thus, the approach presented in this paper makes a contribution to behavior modeling and also to formalization of functional requirements.

8. Future Work

The work presented in this paper can be further developed in three directions:

Taking different sentence constructions into account: Some sentence forms, such as passive sentences,

sentences containing several action verbs, were treated as error cases in this paper. One direction for future work is the improvement of the linguistic part of the analysis, in order that such sentences can be treated as well. Furthermore, Tawbi et al. [17] list several linguistic forms that the same action can have. In the ideal case, all these forms should be taken into account and lead to the same message in the MSC.

Refinement of ontology extraction: An ontology that is to be used for analysis of scenarios should contain all the potential communicating objects, and only them. The ontology extraction procedure should be refined to be able to distinguish communicating objects from other nouns. The ideal ontology extraction procedure would also classify the objects as senders or receivers. Such a classification, although handcrafted, is used, for example, in CICO [8].

Tool-based validation of the produced MSCs: Broy et al. [4] proposed MSCs as the means of formalization of partially specified system behavior. To achieve such formalization, the presented approach can be combined with the tool Smyle [2]. Smyle takes a set of MSCs and generates other MSCs, corresponding to not yet specified parts of system behavior. The analyst has to decide which of the generated MSCs represent allowed system behavior. When Smyle has gathered enough information, it generates an automaton for each communicating object.

A solution to these problems would further improve the presented approach and make it industrially applicable.

References

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

[2] 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://smyle.in.tum.de/2006-12.pdf, accessed 01.02.2007.

[3] T. D. Breaux, M. W. Vail, and A. I. Anton. Towards regulatory compliance: Extracting rights and obligations to align requirements with regulations. In *RE '06: Proceedings of the 14th IEEE International Requirements Engineering Conference (RE'06)*, pages 46–55, Washington, DC, USA, 2006. IEEE Computer Society.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

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

[9] 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.

[10] 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.

[11] 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.

[12] L. Kof. *Text Analysis for Requirements Engineering*. PhD thesis, Technische Universitaet Muenchen, 2005.

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

[14] A. Ratnaparkhi. A maximum entropy model for part-of-speech tagging. In E. Brill and K. Church, editors, *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 133–142. Association for Computational Linguistics, Somerset, New Jersey, 1996.

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

[16] K. Ryan. The role of natural language in requirements engineering. In *Proceedings of IEEE International Symposium on Requirements Engineering*, pages 240–242. IEEE Computer Society Press, 1992.

[17] M. Tawbi, F. Velez, C. Souveyet, and C. B. Achour. Evaluating the CREWS-L'Ecritoire Requirements Elicitation Process, 1999. http://sunsite.informatik.rwth-aachen.de/CREWS/reports99.htm, accessed 01.05.2007.