Requirements Analysis: Concept Extraction and Translation of Textual Specifications to Executable Models

Leonid Kof

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

Abstract. Requirements engineering, the first phase of any software development project, is the Achilles' heel of the whole development process, as requirements documents are often inconsistent and incomplete. In industrial requirements documents, natural language is the main presentation means. This results in the fact that the requirements documents are imprecise, incomplete, and inconsistent. A viable way to detect inconsistencies and omissions in documents is to extract system models from them.
In our previous work we developed approaches translating textual scenarios to message sequence charts (MSCs) and textual descriptions of automata to automata themselves. It turned out that these approaches are highly sensitive to proper definition of terms (communicating objects for MSCs, states for automata). The goal of the presented paper is a systematic comparison of different term extraction heuristics, as a preliminary stage of MSC or automata extraction. The extracted terms were declared to communicating objects (in the case of MSCs) or to states (in the case of automata). The heuristics were compared on the basis of correctness of resulting MSCs and automata. We came to the conclusion that named entity recognition is the best performing technique for term extraction from requirements documents.

1 Requirements Documents are Inconsistent and Incomplete

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 [1]. This results in the fact that the requirements documents are imprecise, incomplete, and inconsistent, because precision, completeness and consistency are extremely difficult to achieve using mere natural language as the main presentation means.

According to Boehm [2], 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 defects of requirements documents. A practical way to detect errors in requirements documents is to convert informal specifications to executable models. In this case, errors in documents would lead to inconsistencies or omissions in models, and inconsistencies and omissions are easier to detect in models than in textual documents.

In our previous work [3–6] we developed approaches extracting behavior specifications (message sequence charts (MSCs) and automata) from requirements documents even in the presence of certain defects. It turned out that the approach to scenario-to-MSC translation is highly sensitive to the proper definition of communicating objects, and the approach producing automata is very sensitive to the proper definition of possible states:

– In the case of text-to-MSC translation, the algorithm tries to identify two communicating objects in every sentence: one before the first verb, and one after the last verb. Communicating objects are just elements of the previously constructed set of objects. Thus, in the case that the set of objects contains wrong terms, wrong communicating objects may be identified in some sentences, which leads to wrong MSCs. Details of the text-to-MSC translation can be found in [3, 4].
– In the case of text-to-automaton translation, the algorithm tries to identify a system state after the main verb of every sentence. This allows to translate sentences like "if ⟨`some condition`⟩, the system goes to ⟨`some state`⟩" to state transitions. The states are identified as elements of previously constructed set of states. Thus, in the case that the set of states contains wrong terms, wrong states may be identified in some sentences, which leads to wrong state transitions. Details of the text-to-automaton translation can be found in [6].

The goal of the presented work was to compare different term extraction heuristics. The heuristics were compared on the basis of correctness of resulting MSCs and automata: the extracted terms were declared to communicating objects (in the case of MSCs) or to states (in the case of automata). To evaluate the correctness of the automata, we used a manually constructed reference automaton. To evaluate the correctness of the MSCs, we used the evaluation rules developed in our previous work [5]. Surprisingly, it turned out that named entity recognition provided best performance in both cases, despite completely different writing styles.

Terminology: For the remainder of the paper we use the following terminology for MSCs: A *scenario* is a sequence of natural language sentences. An *MSC* consists of a set of *communicating objects*, or *actors*, a sequence of *messages* sent and received by these actors, and a sequence of *assertions* interleaved with the message sequence. Figure 1 illustrates the introduced MSC terminology. For automata, we use the standard definitions of *state* and *state transition* [7].

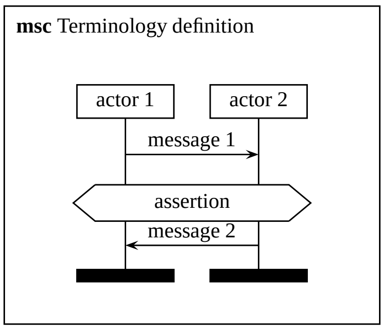

Fig. 1. MSCs: terminology definition

Outline: The remainder of the paper is organized as follows: Section 2 introduces the case studies used to evaluate the term extraction heuristics. Section 3 is the technical core of the paper, it presents the term extraction heuristics and their evaluation. Sections 4 and 5 present the summary of the paper and an overview of related work, respectively.

2 Case Studies

2.1 Automaton: The Steam Boiler

The Steam Boiler Specification was chosen for the automaton-based case study, as it was the standard benchmark for several case studies aiming to compare different formalization methods [8]. This specification describes the steam boiler itself and states the requirements to the control program for the steam boiler. The steam boiler system consists of four pumps to provide the steam boiler with water, one controller for every pump, a device to measure the water level in the steam boiler, and a device to measure the quantity of steam coming out of the steam boiler. The goal of the control program is to maintain the water level between predefined marks, in order to prevent damage of the steam boiler. This water level should be maintained even in case of certain equipment failures. In the case of equipment failures, water levels between certain emergency marks are allowed. Water levels above/below emergency marks cause steam boiler damage.

The control program for the steam boiler should support a number of modes: initialization mode, normal mode, degraded mode, rescue mode, and emergency stop mode. For every mode, the specification describes the required program reactions to different operation situations. An example set of rules, applicable in the normal mode, is shown in Table 1. Table 2 shows the required behavior of the control program (tabular representation of the automaton), manually constructed on the basis of the specification. This manually constructed automaton will be used as the reference for the evaluation of the automatically extracted automata in Section 3.2.

Table 1. The steam boiler, specification excerpt (copied from [9])

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.
4. As soon as the program recognizes a failure of the water level measuring unit it goes into rescue mode.
5. Failure of any other physical unit puts the program into degraded mode.
6. If the water level is risking to reach one of the limit values M1 or M2 the program enters the mode emergency stop.
7. This risk is evaluated on the basis of a maximal behaviour of the physical units.
8. A transmission failure puts the program into emergency stop mode.

Table 2. Automaton for steam boiler control, manually constructed

Initial mode	Target mode	Transition condition
initialization	initialization	message steam-boiler-waiting not yet received
initialization	emergency stop	unit for detection of the level of steam is defective
initialization	emergency stop	failure of the water level detection unit
initialization	normal	all the physical units operate correctly
initialization	degraded	any physical unit is defective
initialization	emergency stop	transmission failure
normal	rescue	failure of the water level measuring unit
normal	degraded	failure of any other physical unit
normal	emergency stop	the water level is risking to reach one of the limit values
normal	emergency stop	transmission failure
degraded	normal	defective unit repaired
degraded	rescue	failure of the water level measuring unit
degraded	emergency stop	the water level is risking to reach one of the limit values
degraded	emergency stop	transmission failure
rescue	normal	water level measurement unit repaired
rescue	degraded	water level measurement unit repaired
rescue	emergency stop	the unit which measures the outcome of steam has a failure
rescue	emergency stop	the units which control the pumps have a failure
rescue	emergency stop	the water level risks to reach one of the two limit values
rescue	emergency stop	transmission failure

2.2 MSCs: The Instrument Cluster

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 the below example, taken from [10]:

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.

For this scenario, several translations to an MSC are possible: For example, the second sentence can be translated both to an assertion and to a message from the instrument cluster to the driver. The same is true for the fourth sentence. Due to the fact that, even for a single scenario, several translations to an MSC are possible, it makes no sense to use a single set of MSCs for evaluation. Instead, we use the following correctness rules for MSCs (cf. [5]):

– General statements that are actually irrelevant for the MSC (e.g., “There is no difference between rising and falling temperature values”) should be translated to assertions.
– General statements about the system state (e.g., “The instrument cluster is activated”) can be translated both to messages and to assertions.
– For a statement sequence like “X activates Y”, “Y is activated”, the first statement should be translated to a message, the second one to an assertion.

– If a statement does not have to be translated to an assertion due to one of the above rules, it should be translated to a message.
– If, for any particular actor, it is known that this actor cannot receive messages, as for example some sensors used in automobiles, no messages should be sent to this object.

These rules, applied manually, will be used to evaluate the influence of different term extraction heuristics on the correctness of the extracted MSCs in Section 3.3.

3 Term Extraction Heuristics and their Influence on Behavior Models

This section is the technical core of the paper. First, in Section 3.1 it presents the term extraction heuristics. Then, it presents the evaluation of the heuristics on the Steam Boiler Specification (Section 3.2) and on the Instrument Cluster Specification (Section 3.3).

3.1 Term Extraction Heuristics

Existing term extraction approaches are based either on named entity recognition or on the analysis of sentence structure, cf. [11, 12]. Basic ideas of both approaches are presented below.

Named Entitiy Recognition. Named entity recognition (NER) aims at identification of standard classes of proper names, i.e. people, places, and organizations. For example, the phrase "`President Bush visits troops in Iraq`" contains two named entities: "`President Bush`" (person) and "`Iraq`" (place). In its most simple form, NER procedure just looks up every word in the predefined list of people, places, and organizations and decides whether the given word is a named entity. This procedure would work fine for "`Iraq`" and other country names, as the number of country names is finite. It makes no sense to apply the lookup procedure to compound names like "`President Bush`": By means of a lookup table we can identify a finite number of former presidents, but we would be unable to identify future ones.

To identify compound names, the following idea can be applied: We define a set of keywords, each keyword indicating either a person name or a place or an organization. For example, for people, the following keywords make sense: "president", "CEO", "professor", etc. In order to use the keyword approach to named entity recognition, it is sufficient to apply a part-of-speech (POS) tagger to the analyzed text. Then, we can just extract the sequence of nouns (words having tags starting with "`NN`") following the keyword. For example, for the sentence "President George W. Bush visits troops in Iraq" we would get the tagging "`President|NNP George|NNP W.|NNP Bush|NNP visits|VBZ troops|NNS in|IN Iraq|NNP`". If we extract the sequence of nouns following the word "president", we obtain the complete named entity, "`President George W. Bush`".

There already exist tools performing named entity recognition, as for example the C&C tool suite (`http://svn.ask.it.usyd.edu.au/trac/candc`). However, existing tools are trained on newspapers texts and, thus, are limited to recognition of standard classes of entities. For requirements analysis, we have to recognize other classes of entities, like system components or states, which makes a customized NER procedure necessary. We use the following heuristics: a named entity consists of

- the keyword, followed by any number of substantives (tags beginning with `NN`), adjectives (tag `JJ`), or verbs in the past participle form (tag `VBD`), or
- any number of substantives, adjectives, or verbs in the past participle form, followed by the keyword.

The decision to consider not only substantives, but also adjectives and verbs in the past participle form was motivated by the concrete form of named entities occurring in our case studies. It turned out in the case studies, that these additional terms do not result in wrong communicating objects or wrong states of the automaton.

Extraction Based on Sentence Structure. If we go beyond POS-tagging and use parsing [13], it becomes possible to extract terms with particular grammatical roles. For example, if we extract the grammatical subject from the sentence “The program enters a state in which it waits for the message steam-boiler-waiting to come from the physical units”, we get “the program”, and if we extract the prepositional object with the preposition “for”, we get “the message steam-boiler-waiting”.

In the case studies we used the following term extraction heuristics, already proven useful for the Instrument Cluster Specification (cf. [5]):

- subjects of active sentences containing a direct object, also occurring in passive sentences,
- subjects of active sentences containing *no* direct object, also occurring in passive sentences,
- subjects of active sentences containing a direct object, *not* occurring in passive sentences,
- subjects of active sentences containing *no* direct object, *not* occurring in passive sentences,
- subjects of passive sentences,
- direct objects.

This technique should be applied with caution, as the existing parsers are definitely less precise than POS-taggers, so the parser itself could become an error source.

3.2 Evaluation: Term Extraction from the Steam Boiler Specification

The Steam Boiler Specification describes different states and state transitions of the control program. An automaton is the most suitable representation of such a specification. To extract an automaton, it is necessary to know the potential states [6]. In our previous work, we investigated NER as the means of term (=state name) extraction. It turned out that NER works well for the Steam Boiler Specification. In the presented paper, we go further and compare previously applied approaches with term extraction based on sentence structure.

Table 3. Steam Boiler Specification, extracted terms

explicitly listed modes	initialization mode, normal mode, degraded mode, rescue mode, emergency stop mode
NER, keyword “mode”	mode emergency stop, mode normal, mode rescue, mode degraded, initialization mode, emergency stop mode, normal mode, standard operating mode, rescue mode, degraded mode
subjects of active sentences containing a direct object, also occurring in passive sentences	level, exactly n liters
subjects of active sentences containing *no* direct object, also occurring in passive sentences	water level
subjects of *active* sentences containing a direct object, *not* occurring in passive sentences	program, transmission failure, failure, limit, water level, water level measuring unit, calculation, unit, water level risks, units
subjects of *active* sentences containing *no* direct object, *not* occurring in passive sentences	initialization mode, message, steam, quantity, unit, physical units, physical unit, mode, program, normal mode, . . . (list pruned due to space limitations)
subjects of passive sentences	message, level, signal, corresponding decision, risk, units, water level, exactly n liters, calculation, water measuring unit, mode
direct objects	physical units, message, state, steam-boiler, steam, level, detection, emergency stop mode, water, valve, order, pump, . . . (list pruned due to space limitations)

The extracted terms are listed in Table 3. Additionally to the terms extracted by their grammatical roles and by NER, Table 3 lists also the terms explicitly mentioned in the specification as mode names. To evaluate the influence of the term extraction heuristics on the behavior extraction, each set of the extracted terms was used as a set of potential states for the automata extraction algorithm [6]. The resulting automata were evaluated according to following criteria:

1. The manually constructed automaton, presented in Table 2, was used as reference.
2. If none of the modes listed in the first line of Table 3 was present in the set of potential states, the corresponding automaton was not evaluated: In this case the automaton can contain correct transitions solely by coincidence.
3. Transition conditions were considered as equivalent, if they had at least two common words.
4. For every transition, it was evaluated whether the transition has the same initial state (called “source” in Table 4) and the same target state as in Table 2.

Evaluation results are presented in Table 4. Each column presents the number of transitions (total, missing, etc.) obtained with the corresponding set of potential states. For sets of potential states containing the word “mode”, evaluation was performed twice: for the original set of potential states and for the set of potential states with the word “mode” manually removed from the set. The reason is that “mode” is a constituent of the explicitly listed state names, so the presence of “mode” in the set of potential states can lead to the state called just “mode”, and ignoring of the real mode names.

Different heuristics combinations were not considered, as it was known from our previous work [5] that unnecessarily extracted terms reduce the quality of behavior

Table 4. Steam Boiler Specification: evaluation results

	Transitions						
	total	missing	correct condition only	correct condition and target	correct condition and source	correct condition, target, and source	wrong transitions
explicitly listed modes	12	9	—	—	—	**11**	1
NER, keyword "mode"	20	1	—	—	—	**19**	1
subjects of active sentences containing a direct object, also occurring in passive sentences	modes completely missing						
subjects of active sentences containing *no* direct object, also occurring in passive sentences	modes completely missing						
subjects of *active* sentences containing a direct object, *not* occurring in passive sentences	modes completely missing						
subjects of *active* sentences containing *no* direct object, *not* occurring in passive sentences	32	5	15	—	—	—	17
subjects of *active* sentences containing *no* direct object, *not* occurring in passive sentences, without "mode"	12	16	4	—	—	—	8
subjects of passive sentences	modes completely missing						
direct objects	29	12	5	3	—	—	21
direct objects, without "mode"	29	11	5	3	1	—	20

models, and NER (second line of Table 4) resulted in an almost perfect automaton: Out of 20 state transitions in Table 2, 19 were correctly identified, which implies the recall of 95%. 19 out of 20 identified transitions were correct, which implies the precision of 95% too. To summarize, for the Steam Boiler Specification, NER was proven the best heuristics to extract potential states of the automaton.

3.3 Evaluation: Term Extraction from the Instrument Cluster Specification

In our previous work we already evaluated the influence of term extraction heuristics based on sentence structure on the quality of the produced MSCs [5]. It turned out that the best heuristics was to declare subjects of active sentences, containing a direct object, *not* occurring in passive sentences, to communicating objects for MSCs. Additionally, this set of communicating objects was augmented with the objects initializing the scenarios: "driver" and "car". Due to good performance that NER has shown on the Steam Boiler Specification, we wanted to investigate how NER can be applied to the Instrument Cluster Specification.

Manual analysis of the Instrument Cluster Specification has shown that following keywords are sensible for NER application to the Instrument Cluster Specification: display, flasher, indicator, indication, light, lights, position, sensor, sensors, warning. Table 5 shows the named entities extracted with these keywords. Some terms present in

Table 5 contain not only nouns but also verbs (e.g., “blinks”). This error was introduced by the POS tagger.

Table 5. NER for the Instrument Cluster Specification

Keyword	**Extracted terms**
display	digital display, dot matrix display, display RPM, warning display, error display, rev meter display, display tolerance, analog display, ... (list pruned due to space limitations)
flasher	warning signal flasher, hazard warning signal flasher, signal flasher
indicator	indicator lights, engine speed indicator, analog indicator, diagram indicator, indicator lights blinks
indication	audible indication, authentic indication, indication (optical display), indication scale, maximum respective minimum possible corresponding indication
light	engine control light, light control unit
lights	instrument cluster lights, warning lights, warning display lights, error display lights, warning symbol lights, ... (list pruned due to space limitations)
position	technical initial position, initial position, technical final position, technical position, degree technical final position, degree technical initial position, pointer position
sensor	outside temperature sensor
sensors	wheel speed sensors, defect wheel speed sensors
warning	hazard warning, engine warning, ice warning, corresponding warning, section warning, warning tone, warning characteristics, ... (list pruned due to space limitations)

On total, the Steam Boiler Specification contains 42 scenarios. 41 out of 42 scenarios were used for evaluation, due to technical difficulties of batch processing one of the scenarios. To evaluate the MSCs resulting from scenarios, the rules introduced in Section 2.2 were used. Additionally to these rules evaluating MSCs as a whole, wrong messages were counted: a message was considered as wrong, only if it was sent to a communicating object not able to process this message type. This implies that an MSC can be wrong without containing a single wrong message, for example if some sentence that should be translated to a message is translated to an assertion. The decision, whether a sentence is translated to a message or to an assertion is influenced by the set of communicating objects too (see [5] for details).

For the first attempt, all terms extracted by NER were joined, this set was declared to the set of communicating objects. However, two important terms, namely “driver” and “car” were missing from this set, which resulted in completely wrong MSCs (cf. first line of Table 6). For the second attempt, the set of NER-extracted terms was augmented with “driver” and “car”, and for the third attempt additionally with “instrument cluster”. (“Instrument cluster” is an important term in the Instrument Cluster Specification, but it is not identified as a named entity.) This augmentation resulted in many more correct MSCs, which can be seen in the second and third line of Table 6.

To investigate whether the results obtained with the usage of grammatical subjects can be further improved by NER, we augmented the set of communicating objects used before (subjects of active sentences, containing a direct object, *not* occurring in passive sentences + “driver” + “car”) with different NER-extracted terms. It turned out that in this case NER barely influences the results: in some cases the resulting MSCs were exactly the same as without NER, and in some cases the MSCs were different, but the

Table 6. Instrument Cluster Specification: evaluation results

		correct MSCs		wrong messages	
		total	percentage	total	percentage
all NER-extracted terms		0	0%	17	3,5%
all NER-extracted terms + “driver” + “car”		40	**97,5%**	0	0%
all NER-extracted terms + “driver” + “car” + “instrument cluster”		40	**97,5%**	0	0%
subjects of active sentences, containing a direct object, *not* occurring in passive sentences + “driver” + “car”	without NER	33	80%	20	4.2%
	+ NER, keyword “display”	33	80%	20	4.2%
	+ NER, keyword “flasher”	33	80%	20	4.2%
	+ NER, keyword “indicator”	33	80%	20	4.2%
	+ NER, keyword “light”	33	80%	20	4.2%
	+ NER, keyword “lights”	MSCs completely identical to the case without NER application			
	+ NER, keyword “position”				
	+ NER, keyword “indication”				
	+ NER, keyword “sensor”				
	+ NER, keyword “sensors”				
	+ NER, keyword “warning”				

percentage of correct MSCs was the same. This can be explained by the fact that the set of grammatical subjects contains unnecessary communicating objects that cause both wrong messages and wrong MSCs. To summarize, for the Instrument Cluster Specification, NER was proven the best heuristics to extract potential communicating objects.

4 Summary

Requirements engineering is an important project phase, influencing all later development phases. Requirements documents are mostly written in natural language, which implies incomplete and inconsistent requirements. Translation of natural language descriptions to executable models is a viable way to deal with incompleteness and inconsistency. The previously developed approaches to text-to-MSC/automaton translation turned out to be highly sensitive to proper term extraction, as a prerequisite for their application. In the presented paper, we systematically compared different term extraction heuristics and came to the conclusion that named entity recognition (NER) provides best performance on two requirements documents, despite of rather different writing styles. However, it is still an open question whether NER provides best results on any requirements document. An answer to this question, obtained in further case studies, would open the way to systematic concept extraction from industrial documents.

5 Related Work

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 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 our work: 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 [18], Abbott [19], or Sawyer et al. [20] analyzes the requirements documents, extracts application specific concepts, and provides 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 [21], Gervasi and Zowghi [22], and Avrunin et al. [23] 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. 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. 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 two approaches, as the language is not restricted.

If we consider a more general area of software engineering, it is possible to apply natural language processing to other types of documents too. Witte et al. [12] introduced an approach to software documentation analysis, allowing to link software documentation to code. They extract instances of previously defined concepts, like variables, classes, methods, etc., using named entity recognition with keywords motivated by the previously constructed ontology of programming concepts. In the presented paper, we have to deal with a more challenging situation, as there exists no ontology of modeling concepts, valid for all application domains.

To summarize, to the best of our knowledge, there is no approach to requirements documents analysis, that is able to analyze documents written in non-restricted language, and to extract behavior information from them.

References

1. Mich, L., Franch, M., Novi Inverardi, P.: Market research on requirements analysis using linguistic tools. Requirements Engineering **9** (2004) 40–56
2. Boehm, B.W.: Software Engineering Economics. Prentice-Hall (1981)
3. Kof, L.: Scenarios: Identifying missing objects and actions by means of computational linguistics. In: 15th IEEE International Requirements Engineering Conference, New Delhi, India, IEEE Computer Society Conference Publishing Services (2007) 121–130
4. Kof, L.: Treatment of Passive Voice and Conjunctions in Use Case Documents. In Kedad, Z., Lammari, N., Méthais, E., Meziane, F., Rezgui, Y., eds.: Application of Natural Language to Information Systems. Volume 4592 of LNCS., Paris, France, Springer (2007) 181–192
5. Kof, L.: From Textual Scenarios to Message Sequence Charts: Inclusion of Condition Generation and Actor Extraction. In: 16th IEEE International Requirements Engineering Con-

ference, Barcelona, Spain, IEEE Computer Society Conference Publishing Services (2008) 331–332
6. Kof, L.: Translation of Textual Specifications to Automata by Means of Discourse Context Modeling. In Glinz, M., Heymans, P., eds.: Requirements Engineering: Foundation for Software Quality 15th International Working Conference, REFSQ 2009. Volume 5512 of LNCS., Springer (2009) 197–211
7. Broy, M.: Informatik. Eine grundlegende Einführung. Volume 2. Springer (1998)
8. Abrial, J.R., Börger, E., Langmaack, H.: Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control. Volume 1165 of LNCS. Springer (1996)
9. Abrial, J.R., Börger, E., Langmaack, H.: The steam boiler case study: Competition of formal program specification and development methods. In Abrial, J.R., Borger, E., Langmaack, H., eds.: Formal Methods for Industrial Applications. Volume 1165 of LNCS., Springer (1996)
10. 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 11.01.2007.
11. Kof, L.: Natural Language Processing: Mature Enough for Requirements Documents Analysis? In Montoyo, A., Mu¯noz, R., Methais, E., eds.: Application of Natural Language to Information Systems. Volume 3513 of LNCS., Alicante, Spain, Springer (2005) 91–102
12. Witte, R., Li, Q., Zhang, Y., Rilling, J.: Ontological Text Mining of Software Documents. In Kedad, Z., Lammari, N., Méthais, E., Meziane, F., Rezgui, Y., eds.: Application of Natural Language to Information Systems. Volume 4592 of LNCS., Paris, France, Springer (2007) 168–180
13. Clark, S., Curran, J.R.: Wide-coverage efficient statistical parsing with ccg and log-linear models. Comput. Linguist. **33** (2007) 493–552
14. Rupp, C.: Requirements-Engineering und -Management. Professionelle, iterative Anforderungsanalyse für die Praxis. Second edn. Hanser–Verlag (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, Greenbelt, Maryland, IEEE Computer Society (2001) 97–105
16. Kamsties, E., Berry, D.M., Paech, B.: Detecting ambiguities in requirements documents using inspections. In: Workshop on Inspections in Software Engineering, Paris, France (2001) 68 –80
17. Chantree, F., Nuseibeh, B., de Roeck, A., Willis, A.: Identifying nocuous ambiguities in natural language requirements. In: RE '06: Proceedings of the 14th IEEE International Requirements Engineering Conference (RE'06), Washington, DC, USA, IEEE Computer Society (2006) 56–65
18. Goldin, L., Berry, D.M.: AbstFinder, a prototype natural language text abstraction finder for use in requirements elicitation. Automated Software Eng. **4** (1997) 375–412
19. Abbott, R.J.: Program design by informal English descriptions. Communications of the ACM **26** (1983) 882–894
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** (2005) 969–981
21. Vadera, S., Meziane, F.: From English to formal specifications. The Computer Journal **37** (1994) 753–763
22. Gervasi, V., Zowghi, D.: Reasoning about inconsistencies in natural language requirements. ACM Trans. Softw. Eng. Methodol. **14** (2005) 277–330
23. Smith, R.L., Avrunin, G.S., Clarke, L.A., Osterweil, L.J.: Propel: an approach supporting property elucidation. In: ICSE'02: Proceedings of the 24th International Conference on Software Engineering, New York, NY, USA, ACM (2002) 11–21