Translation of Textual Specifications to Automata by Means of Discourse Context Modeling

Leonid Kof

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

Abstract. [**Context and motivation**] Natural language is the main presentation means in industrial requirements documents. In such documents, system behavior is specified either in the form of scenarios or in the form of automata described in natural language. The behavior descriptions 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.
[**Question/problem**] Formalization of textual behavior description can reveal deficiencies in requirements documents. Formalization can take two major forms: it can be based either on interaction sequences or on automata, cf. survey [1]. Translation of textual scenarios to interaction sequences (Message Sequence Charts, or MSCs) was presented in our previous work [2–4]. To close the gap and to provide translation techniques for both formalism types, an algorithm translating textual descriptions of automata to automata themselves is necessary.
[**Principal ideas/results**] It was shown in our previous work that discourse context modeling allows to complete information missing from scenarios written in natural language and to translate scenarios to MSCs. The goal of the approach presented in this paper is to translate textual descriptions of automata to automata themselves, by adapting discourse context modeling to texts describing automata.
[**Contribution**] The presented paper shows how the previously developed context modeling approach can be adapted in order to become applicable to texts describing automata. The proposed approach to translation of text to automata was evaluated on a case study, which proved applicability of the approach.

Key words: requirements analysis, behavior extraction, behavior modeling, natural language processing

1 Requirements Documents Suffer from Missing Information

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 [5]. 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. From the linguistic point of view, document authors may introduce three defect types, without perceiving them as defects, cf. Rupp [6]:[1]

[1] The following definitions are translations of the definitions from [6], in German

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 not always aware of these document defects. Even documents that are precise from the human point of view can omit some facts relevant for behavior specification. The goal of the presented paper is to translate texts to automata despite such defects.

According to Boehm [7], 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. Our previous work [2–4] focused on defects in scenarios, specially on the "deletion" defects. The goal of the previous work was to identify missing parts of scenarios written in natural language and to produce Message Sequence Charts (MSCs) containing the reconstructed information. The key idea was to model the discourse context and to infer the missing parts of scenarios from the context. In the case of MSCs, the discourse context model included the set of messages that are sent but not yet answered.

According to our survey of modeling techniques [1], all modeling techniques are either interaction-based (MSC-like) or automata-based. Similarly, texts describing system behavior fall in the same two categories: They either specify scenarios (interaction between system components) or give textual description of automata. The goal of the approach presented in this paper is to translate automata-based textual descriptions to automata. Together with our previous work, this provides extraction of both model types from textual documents. It turns out that, in the case of automata, the discourse context model is simpler than for MSCs and contains only a default initial state for incompletely specified state transitions (see Section 3 for details).

Contribution: The presented paper shows how the idea of discourse context modeling can be transferred to texts describing automata. It shows that a different approach to context modeling, even simpler than the approach developed to translate textual scenarios to MSCs, is sufficient to translate textual descriptions of automata to automata themselves.

Outline: 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 to translate texts to automata. 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 Steam Boiler

Authors of requirements documents tend either to forget facts that seem obvious to them or they are reluctant to precisely specify the context in which their statements apply. This is quite natural, and is just a part of the human process of focusing attention onto facts that seem most important at the moment of writing. This results in the problem that even precise specifications, as for example the Steam Boiler Specification [8], used in the presented work, cannot be analyzed on the sentence level.

The Steam Boiler Specification was chosen for the case study, as it was the standard benchmark for several case studies aiming to compare different formalization methods [9]. 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. It is easy to see that it makes no sense to analyze every sentence of the specification separately: Some sentences, as for example Sentence 1, Sentence 3, and Sentence 7, do not contain any explicit behavior specification. Others contain behavior information, but cannot be directly translated to state transitions, as they specify the state after the transition only. The initial state, *normal mode*, is common for all transitions and remains unspecified in the sentences describing transitions. Every such omission is a “deletion” defect in the sense of the definitions given in Section 1. The goal of the presented paper is to translate texts to automata despite such defects.

In spite of the fact that separate analysis of every sentence is insufficient even for the relatively well-written Steam Boiler Specification, the existing approaches translating textual specifications to models analyze every specification sentence separately (cf. Section 4). It is the goal of the presented work, to capture context information in order to complete information not explicitly mentioned in sentences specifying state transitions.

Table 2 shows the required behavior of the control program, manually constructed on the basis of the specification. This manually constructed automaton will be used to evaluate the proposed text-to-automaton translation procedure in Section 3.

3 Translation of Texts to Automata

The process of text-to-automaton translation is motivated by the already tested and validated algorithm for text-to-MSC translation presented in [2–4]. The process of text-to-MSC translation consisted of three steps:

- identification of communicating objects,
- splitting of every sentence into segments,

Table 1. The steam boiler, specification excerpt (copied from [8])

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.

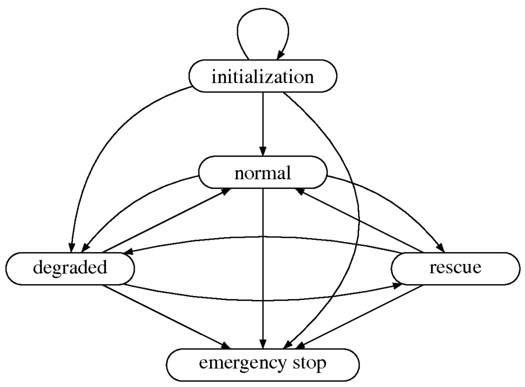


Fig. 1. Automaton for steam boiler control, manually constructed

– for every segment, translation of the segment to an MSC element. An MSC element can be either a message between two communicating objects or an assertion about system state.

The process of text-to-automaton translation follows similar steps: First, the set of potential states is determined. Then, every sentence is split into segments. Finally, segments are translated either to state transitions or to transition conditions. These steps are presented in Sections 3.1-3.3. Section 3.4 presents the results of the evaluation of the presented text-to-automaton translation on the Steam Boiler Specification.

To stay robust, the presented approach uses solely a part-of-speech (POS) tagger [10] on linguistic side and does not use more sophisticated techniques like Discourse Representation Theory (DRT) [11], as the use of techniques like DRT would render the approach highly fragile.

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

3.1 Identification of States

In our previous work [2–4] it was shown that the algorithm for text-to-MSC translation that inspired the presented work is highly sensitive to the proper definition of the set of communicating objects. Thus, it was to expect that the presented algorithm for text-to-automaton translation is sensitive to the proper definition of the set of states.

Identification of states is necessary for later decision whether to translate a particular sentence segment to a state transition. As a first approximation, it is possible to manually extract the names of the states explicitly listed in the text. However, this set of states can be incomplete. In our case study, this incompleteness resulted in missing transitions in the extracted automaton, cf. Section 3.4.

The name of a state can consist of several words, like “emergency stop mode”. Furthermore, the same mode can be called, for example, both “emergency stop mode” and “mode emergency stop” in the specification text. To automatically extract the different forms of the mode names, the following procedure was applied:

– The whole text was tagged by a part-of-speech (POS) tagger. The applied tagger [10] has a precision of about 97%, which makes it unlikely to be an error source.
– Following tags were considered: (1) tag “`VBD`”, identifying verbs in the past participle form (“been”, “done”), (2) any tag starting with “`NN`”, identifying different noun forms, and (3) tag “`JJ`”, identifying adjectives
– Following patterns were extracted from the tagged text:
  • Word “`mode`”, followed by any number of substantives (like in “`mode|NN rescue|NN`”), adjectives (like in “`mode|NN normal|JJ`”), or verbs in the past participle form (like in “`mode|NN degraded|VBD`”).

- Any number of substantives, adjectives, or verbs in the past participle form, followed by the word "`mode`".

Technically, the extraction of the above patterns from the tagged text was performed by the application of the UNIX tool `grep` with the following regular expressions:

- `mode|NN ([^|]*|(NN|VBD|JJ))+`
- `([^ |]*|(NN|VBD|JJ))+ mode|NN`

Here it is important to emphasize that the signal word "mode" used to identify state names, is specific to the Steam Boiler Specification. For other specification texts, it is necessary to provide other signal words or to use other extraction techniques: For example, in [4] the names of modeling elements were identified as subjects of sentences having particular grammatical features.

The above procedure resulted in the extraction of the word sequences shown in Table 3. This table contains not only the states explicitly defined in the document, but also noise, "`standard operating mode`". However, as the case study has shown, this noise can be compensated for when constructing the automaton (cf. Section 3.4).

Table 3. Automatically extracted states

"**`mode`**", followed by other words:	mode emergency stop, mode normal, mode rescue, mode degraded
"**`mode`**", preceded by other words:	initialization mode, emergency stop mode, normal mode, standard operating mode, rescue mode, degraded mode

The procedure to extract the potential states of the automaton by extracting the named entities with the signal word "`mode`" was sufficient for the steam boiler case study. In general, it is easy to extend the procedure by adding further signal words. Furthermore, it is possible to integrate the above procedure with grammar-based methods from [4]. Applicability of every particular method depends on the writing style of the concrete document.

3.2 Categories of Sentences

One of the prerequisites for the text-to-automaton translation is the assignment of every (sub)sentence to one of the four categories: "state transition", "transition condition", "context setting", or "irrelevant", cf. Section 3.3. The assignment of sentence segments to categories takes place in the following steps:

1. Splitting of every sentence to segments
2. Assignment of segments to categories on the basis of grammatical information only
3. Re-assignment of segments to categories, by using context information

Each of these steps is described below.
Sentence splitting: To split sentences, just the following assumption is made: punctuation marks are correctly placed to separate subsentences. The splitting process itself is rather simple. Punctuation symbols and the words "if" and "when" are used as splitting

marks. Additionally, the conjunctions "and" and "or" are used as splitting marks, unless they directly follow an adjective or a number. This heuristics prevents splitting of expressions like "if the water level lies between N1 and N2, ...". A splitting example is shown in Table 4.

Table 4. Splitting example

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

Assignment of segments to categories on the basis of grammatical information: On total, we differentiate four classes of sentence segments:

- Segments translated to transitions, like "the program enters either the mode normal" in the example in Table 4. Such segments are called "state transition" in the remainder of the paper.
- Segments translated to transition conditions, like "as soon as this signal has been received" in the example in Table 4. Such segments are called "transition condition" in the remainder of the paper.
- Segments that are not translated to any element of the automaton, but setting the context for the subsequent segments, like the first sentence in Table 1. Such segments are called "context setting" in the remainder of the paper.
- Segments that are irrelevant for the text-to-automaton translation, like the third sentence in Table 1. Such segments are called "irrelevant" in the remainder of the paper.

Identification of the four segment classes is possible on the basis of the POS tags and the previously extracted set of states. The identification consists of two phases. In the first phase, every sentence segment is marked on its own. In the second phase, the decision of the first phase is revised by taking the neighbors of the analyzed segment into account. In the first phase, the assignment of the sentence segment to one of the four classes is fairly simple:

- If the sentence segment does not contain any reference to a state (element of the extracted set of states), it is marked as "irrelevant". This holds, for example, for the first segment in Table 4.
- If the sentence segment contains a reference to a state, but first occurrence of the state is not preceded by a verb, this segment is marked as "context setting". A word is considered as a verb if the POS tagger assigns a tag starting with "`VB`" to this word. For example, in Table 1, the header ("normal mode") and the first sentence set the context for the translation of the following sentences.

– Otherwise, the sentence segment is marked as “state transition”.

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

Re-assignment of segments to categories, by using context information: To take context into account, it is necessary to revise the “context setting”-marks first. For example, the fourth segment in Table 4 is marked as “context setting” in the first phase, although it actually specifies a state transition. Here, the following heuristics is applied: If, for a given sentence, any of its segments is marked as “state transition”, then all segments marked as “context setting” are relabeled to “state transition”. This compensates for potentially missing verbs in some sentence segments. In the case of the example shown in Table 4, it marks the fourth segment as “state transition” and leaves the other marks unchanged.

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

– If a sentence segment is marked as “irrelevant” and directly precedes a segment marked as “state transition”, then the former segment is relabeled to “transition condition”. This allows to mark the first segment of the example in Table 4, “as soon as this signal has been received”, as “transition condition”.
– After the above step, if a sentence segment is marked as “irrelevant” and directly precedes a segment marked as “transition condition”, the former segment is relabeled to “transition condition”. This allows to treat compound conditions, like “if message A or message B is received,. . . ”.
– If a sentence segment is marked as “irrelevant” and directly follows a segment marked as “state transition”, then the former segment is relabeled to “transition condition”. This allows to treat conditions like “⟨`some transition`⟩ if ⟨`some condition`⟩”.
– After the above step, if a sentence segment is marked as “irrelevant” and directly follows a segment marked as “transition condition”, the former segment is relabeled to “transition condition”. This allows to treat compound conditions, like “⟨`some transition`⟩ if ⟨`some condition`⟩ or ⟨`some other condition`⟩”.

When this relabeling process is finished, we have enough information to translate the text to an automaton.

The process of sentence splitting is purely syntactic, which is its major advantage: this makes sentence splitting independent of writing style of a particular document author. Furthermore, this allows to treat grammatically different types of conditions, like “if something happens” and “as soon as something happens”, in a uniform way.

3.3 Context Modeling and Generation of Transitions

When every sentence segment is assigned to one of the four classes (“state transition”, “transition condition”, “context setting”, or “irrelevant”), we can use this information to translate the text to an automaton. The actual text-to-automaton translation exploits the fact that sentence segments marked as “context setting” or “state transition” always refer to a state. The translation algorithm sequentially goes through the marked sentence segments. Depending on the sentence segment class, it performs the following actions:

– Segments marked as "irrelevant" are ignored.
– If the translation algorithm comes across a sentence segment marked as "context setting", the state contained in this segment becomes the default initial state for the transitions generated afterwards.
– If the translation algorithm comes across a sentence segment marked as "state transition", then several transitions are generated. The initial state of the transitions is always the current default initial state (context), the target state is the state taken from the "state transition" segment under analysis. The transitions conditions depend on the neighbors of the segment under analysis:
  - If the "state transition" segment under analysis is followed by a contiguous block of "transition condition" segments, then a state transition is generated for every segment from the "transition condition" block. The textual representation of every "transition condition" segment becomes a transition condition in the generated automaton.
  - If the "state transition" segment under analysis is preceded by a contiguous block of "transition condition" segments, and the "state transition" segment under analysis is the first "state transition" segment of its sentence, then a state transition is generated for every segment of the "transition condition" block, in the same way as above.
  - If no state transition can be generated due to the above two rules, the translation algorithm re-analyzes the current "state transition" segment and extracts the word sequence preceding its main verb. The word sequence preceding the main verb becomes the transition condition. This allows to handle constructions like "a transmission failure puts the program into the mode emergency stop". In this case, "a transmission failure puts" becomes the transition condition, cf. Table 5.
  - If all the above rules fail, a transition with an empty transition condition is generated.

By inferring the initial states of transitions, the presented algorithm visualizes presuppositions of the document author. This can be used for validation, in particular to proof whether the document author and the document reader interpret the specification in the same way.

The generated automaton is flat: it contains neither parallel nor nested states. Generation of such constructions would require deep semantic analysis, going far beyond capabilities of the existing linguistic tools.

The above rules were implemented in a Java program. This program generates automata represented as table, like Table 2 or Table 5. At the moment, the generated transition conditions are represented in natural language, and are not automatically analyzable. In the long run, the presented approach should be integrated with the approach by Gervasi and Zowghi [12]. Gervasi and Zowghi can translate conditions written in a restricted natural language to logical formulae. This translation would allow to perform further analysis of the automata, like for example completeness of input coverage.

3.4 Evaluation

Three case studies were performed to evaluate the presented approach. The case studies were performed on the same text, namely on Section 4 of the Steam Boiler Specification [8]. This section describes the required behavior of the steam boiler control program. This section was manually cut out of the document and submitted to the text-to-automaton translation.

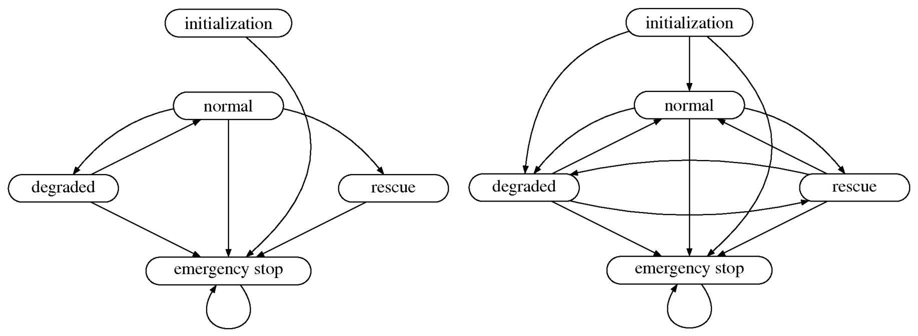
The case studies differed in the definition of the states:

1. In the first case study, the algorithm for text-to-automaton translation was provided with the set of states explicitly listed in the following specification sentence:
   The program operates in different modes, namely: initialization, normal, degraded, rescue, emergency stop.

   Thus, the translation algorithm was provided with the following set of states: "initialization mode", "normal mode", "degraded mode", "rescue mode", "emergency stop mode".
2. In the second case study, the algorithm for text-to-automaton translation was provided with the automatically extracted set of states shown in Table 3. The state name "`standard operating mode`" was manually removed from the set, as it does not represent a real state of the control program.
3. In the third case study, the algorithm for text-to-automaton translation was again provided with the automatically extracted set of states shown in Table 3. In contrast to the second case study, however, "`standard operating mode`" was not removed from the set.

The first case study produced the automaton shown in Figure 2(a). When compared with the manually constructed automaton, shown in Figure 1, the automaton in Figure 2(a) definitely lacks several state transitions.

The second and the third case studies produced the same automaton, shown in Figure 2(b) and Table 5. Interestingly, the state "`standard operating mode`" did not result in any additional state transitions in the third case study. As the output algorithm ignores such standalone states, this state is not presented in Figure 2(b).



(a) Translation with explicitly mentioned states (b) Translation with automatically extracted states, cf. Section 3.1

Fig. 2. Automaton for steam boiler control, automatically extracted

To evaluate the text-to-automaton translation, we compare the manually constructed automaton with the generated one. If we compare the graphical representations, i.e. Fig-

Table 5. Automaton for steam boiler control, automatically extracted

Initial mode	Target mode	Transition condition
initialization	emergency stop	the unit for detection of the level of steam is defective – that is, when v is not equal to zero – the program enters
initialization	emergency stop	the program realizes a failure of the water level detection unit it enters
initialization	normal	all the physical units operate correctly
initialization	degraded	any physical unit is defective.
initialization	emergency stop	a transmission failure puts
normal	rescue	as soon as the program recognizes a failure of the water level measuring unit it goes
normal	degraded	failure of any other physical unit puts
normal	emergency stop	the water level is risking to reach one of the limit values m1 or m2 the program enters
normal	emergency stop	a transmission failure puts
degraded	normal	once all the units which were defective have been repaired, the program comes
degraded	rescue	as soon as the program sees that the water level measuring unit has a failure, the program goes
degraded	emergency stop	the water level is risking to reach one of the limit values m1 or m2 the program enters
degraded	emergency stop	a transmission failure puts
rescue	degraded	as soon as the water measuring unit is repaired
rescue	normal	
rescue	emergency stop	it realizes that one of the following cases holds: 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	a transmission failure puts
emergency stop	emergency stop	the program stops.

ure 1 with Figure 2(b), we see that the automata coincide, except for the loop in the "`initialization`" mode in Figure 1 and the loop in the "`emergency stop`" mode in Figure 2(b). Manual analysis of the steam boiler specification shows that the behavior in the emergency stop mode is underspecified. It can be interpreted both as a loop and as its absence: "once the program has reached the emergency stop mode, the physical environment is then responsible to take approrpiate actions, and the program stops". As for the loop in the "`initialization`" mode, its extraction requires semantic analysis, going beyond the capabilities of the available linguistic tools. This loop stems from the sentence "the program enters a state in which it waits for the message steam-boiler-waiting to come from the physical units". It is not yet possible for linguistic tools to interpret the word "wait" as a state loop. Hard-coding generation of loops for words like "wait" would make the approach highly dependent on the writing style and would make generalization extremely difficult.

If we compare the table representations, Table 2 and Table 5, we see that they coincide except for the already discussed loops, if we ignore phrasings for the transition conditions. Furthermore, due to the applied sentence splitting algorithm, transition conditions in Table 5 are sometimes grammatically incomplete. A closer analysis shows that the transition conditions in Table 2 and Table 5 are semantically equivalent and differ in their lexical representation only. The only exception is the transition from the rescue mode to the normal mode that lacks a transition condition. This transition originates from the sentence "as soon as the water measuring unit is repaired, the program returns into mode degraded or into mode normal". The exact transition condition to "`mode normal`" is not specified. In Table 2 it was just guessed that the transition conditions to "`mode normal`" and "`mode degraded`" coincide.

Anyway, none of the automata, neither the manually constructed nor the automatically extracted can be directly used for further system development. Both automata rather serve to visualize the specification and thus to ease its validation. For this reason differences in lexical representations of transition conditions are unimportant.

To summarize, the presented approach to text-to-automaton translation is able to translate texts about automata to automata themselves and the translation result is precise enough to be used for behavior validation.

4 Related Work

Ryan [13] claimed that natural language processing is not mature enough to fully automate requirements engineering. In the same paper he admitted, however, that natural language processing can be useful to support human analysts. There was a lot of work aiming to support human analysts 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 to the assessment of document quality were introduced, for example, by Rupp [6], Fabbrini et al. [14], Kamsties et al. [15], and Chantree et al. [16]. 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 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 [17], Abbott [18], or Sawyer et al. [19] 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 [20], Gervasi and Zowghi [12], and Avrunin et al. [21] translate requirements documents to executable models by analyzing linguistic patterns. In this sense they are similar to the approach presented in this paper. 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. The presented paper goes further than the above two approaches, as the language is not restricted, and the assumptions about phrasing are minimal: It is solely assumed that punctuation marks are correctly placed.

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, model context information and use this context modeling to complete the information missing from the text when translating the text to an executable model.

5 Summary

The approach presented in this paper automates parts of the step from requirements documents to design. Despite minimal assumptions about the structure of the sentences to be translated, the approach is effective, which was shown in case studies. The translation of texts to design imitates the way how human analysts would model the discourse context. This context model is then applied to infer information not explicitly stated in the behavior specification.

The presented approach relies, in its pure form, on the writing style of the Steam Boiler Specification. Under following assumptions, it can be generalized and applied to other specifications too:

The set of system states is known: In the presented work, the set of states was extracted from the specification, but, in general, it is possible to provide the approach with a predefined set of states. It is important that the provided set of states be complete: if a state is missing, some sentences may be wrongfully identified as "irrelevant" instead of "context setting" or "state transition", which would definitely hurt the correctness of the generated automaton. Presence of noise states ("standard operating mode" in the presented case study), however, can be compensated for, as long as the noise states do not occur in sentences identified as "state transition".

Sentences describing state transitions contain a reference to the target state, as in "if . . . , the system goes into ⟨target state⟩". Given that the initial state of a state transition can be inferred from the context, this allows to extract a complete state transition.

Context setting is stated explicitly, either in paragraph titles or in describing sentences like "⟨context state⟩ is the state in which . . . "

Comma setting is correct: "if ⟨condition⟩, then ⟨action⟩" or "⟨action⟩ if ⟨condition⟩".

Before being used in the further development process, the generated behavior model has to be validated. Validation is necessary for at least two reasons:

- The original requirements document can contain inconsistencies or omissions.
- The applied linguistic tools do not offer 100% precision, and errors introduced by the linguistic tools may interfere with the presented heuristic for automata construction.

Validation of the produced automaton can make apparent the ambiguities or omissions in the document, not perceived by a human analyst. Validation of the automaton becomes especially valuable if the automaton generated by the presented approach radically differs from the manually constructed automaton. This can mean that the requirements text has several interpretations and thus should be made more precise before used in the further development steps. When the generated automaton is validated, it can be used in the further development process. Thus, the presented approach makes a contribution both to document improvement and validation and to the transition from requirements to design.

6 Future Work

The approach presented in this paper is a proof-of-concept that discourse context modeling can be successfully applied to translate specification texts to behavior models. It can be further developed in different directions. First of all, the case study used to evaluate the approach was relatively small. A larger case study would allow more significant conclusion about the precision of the proposed approach. Secondly, the generated transition conditions, as for example those shown in Table 5, sometimes contain unnecessary words. This problem arises from the fact that it is not possible to determine the boundaries of the subordinate clauses without parsing the sentence. In the presented work, mere part-of-speech (POS) tagging was applied instead of parsing, as POS tagging is much more precise (97% precision for tagging [10] vs. approx. 80% for parsing [22]). To combine the advantages of both technologies, the presented approach can be augmented in such a way that POS tagging is used for the actual text-to-automaton translation and parsing is used to determine clause boundaries. In this way it is possible to generate better transition conditions.

To validate the generated automata, a technique similar to CREWS-SAVRE [23] can be applied: In the original version of CREWS-SAVRE, a sequence of events is taken as input, and, for this sequence, questions like “What happens if the specified event does not occur?” are generated. In a similar way, for every state transition of the generated automaton, we could generate questions like “What happens if the input signal necessary for the transition does not occur?”, “What happens if the input signal necessary for the transition occurs several times?”, etc.

Developments sketched above would further improve the presented approach and make it industrially applicable.

References

1. Kof, L., Schätz, B.: Combining aspects of reactive systems. In: Ershov Memorial Conference. Volume 2890 of LNCS., Springer (2003) 344–349
2. 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
3. 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
4. Kof, L.: From Textual Scenarios to Message Sequence Charts: Inclusion of Condition Generation and Actor Extraction. In: 16th IEEE International Requirements Engineering Conference, Barcelona, Spain, IEEE Computer Society Conference Publishing Services (2008) 331–332

5. Mich, L., Franch, M., Novi Inverardi, P.: Market research on requirements analysis using linguistic tools. Requirements Engineering **9** (2004) 40–56
6. Rupp, C.: Requirements-Engineering und -Management. Professionelle, iterative Anforderungsanalyse für die Praxis. Second edn. Hanser–Verlag (2002) ISBN 3-446-21960-9.
7. Boehm, B.W.: Software Engineering Economics. Prentice-Hall (1981)
8. 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)
9. 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)
10. Curran, J.R., Clark, S., Vadas, D.: Multi-tagging for lexicalized-grammar parsing. In: 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics, Sydney, Australia, 17-21 July. (2006)
11. Blackburn, P., Bos, J., Kohlhase, M., de Nivelle, H.: Inference and computational semantics. CLAUS-Report 106, Universität des Saarlandes, Saarbrücken (1998)
12. Gervasi, V., Zowghi, D.: Reasoning about inconsistencies in natural language requirements. ACM Trans. Softw. Eng. Methodol. **14** (2005) 277–330
13. Ryan, K.: The role of natural language in requirements engineering. In: Proceedings of IEEE International Symposium on Requirements Engineering, IEEE Computer Society Press (1992) 240–242
14. 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
15. 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
16. 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
17. 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
18. Abbott, R.J.: Program design by informal English descriptions. Communications of the ACM **26** (1983) 882–894
19. 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
20. Vadera, S., Meziane, F.: From English to formal specifications. The Computer Journal **37** (1994) 753–763
21. 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
22. Clark, S., Curran, J.R.: Wide-coverage efficient statistical parsing with ccg and log-linear models. Comput. Linguist. **33** (2007) 493–552
23. Maiden, N.A.M.: CREWS-SAVRE: Scenarios for Acquiring and Validating Requirements. Automated Software Engineering **5** (1998) 419–446