Using Application Domain Ontology to Construct an Initial System Model

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

Abstract

This paper describes a case study on application of natural language processing in very early stages of the requirements engineering. In our previous work [7] we have shown how natural language processing can be applied to build an application domain model from the requirements document. In this paper we want to go one step further. We want to build an initial system model. To do so, we match the extracted application domain model and the meta model of the specification formalism. It turns out that such a matching is possible and it yields a sensible model, extensible to a full–fledged specification.

1 Introduction

Precise specification is a key success factor for a software project. Formal specification is ideal for the software developer, but it is not reasonable to require the author of the requirements document, who is seldom familiar with formal methods or even with the concept of "specification", to provide a formal description. State of the art are informal requirements documents written in natural language.

Using techniques described in section 2, we can extract a domain ontology from the text. The ontology itself is a valuable basis for communication between the domain expert and the requirements engineer. Though, to build an application, we have to go further and to map the extracted ontology to implementable concepts, like components, messages etc.

The first approach trying to extract domain knowledge and to map domain concepts to data types was that by Abbott [2]. It declares common nouns to data types, verbs to operators etc. This assumption renders the analysis results highly dependent on the writing style. Our aim is also to eliminate the drawbacks of Abbott's approach. We use other techniques, that seem more promising, to classify terms used in the text.

The primary aim of this paper is to show how the results of text analysis can help in building a system model. As in [7], our aim is also to discover weaknesses of the requirements document and to guide the writer to better requirements specification.

The paper is organized in the following way: section 2 introduces the methods of the ontology extraction and the tools implementing these methods, section 3 describes the case study and section 4 sums up the results.

2 Ontology Extraction

This section is a short summary of [7]. It describes just the techniques used for ontology extraction. The extraction results are presented in section 3.1.

2.1 Subcategorisation Frame Extraction

A verb subcategorization frame is a predicate with its arguments (subject and objects). Subcategorization frames are used by the tool ASIUM [6] for term classification and clustering. We used the parser by Michael Collins [5] to produce the parse tree. We cut subtrees according to certain heuristics in order to extract application domain concepts. A description of the heuristics can be found in [7].

2.2 Taxonomy Building

The tool ASIUM [6] is based on the assumption that concepts used in the same grammatical context must be related. It builds clusters of nouns occurring in the same context[1] and looks for common words in different clusters. There are different measures for cluster overlapping. If overlapping of two clusters exceeds the previously set threshold, the tools asks the user if the clusters should be joined.

The user can join such clusters to larger ones. When joining clusters, the user can introduce a generic term describing both joined clusters. In this way the user builds a tree of "is-a" relations, which is the domain taxonomy.

[1]context = verb subcategorization frame

2.3 Association Mining

The tool KAON [8] borrows its main idea from data mining: It considers the text as a database transaction and counts how often certain concepts occur in the same transactions. It is also possible to force it to use finer grade transactions, just by splitting the text into smaller chunks, each chunk representing a new transaction. The tool offers an own concept extraction facility, based on Part–of–Speech tagging and extraction of tag patterns.

The user can also set the minimal support and confidence values that make associations interesting. For every found association the user can decide if it should be included in the ontology. Given a taxonomy, KAON generalizes the association rules as described in [9].

3 Case Study

The steam boiler specification [4] was chosen for the case study, because this specification was also used as a benchmark for different formal specification techniques. The specification describes a system consisting of the boiler itself, a valve, four pumps and a couple of measuring devices. The goal of the control program is to maintain certain level of water in the boiler in order to avoid the damage of the boiler. The control program should be fault tolerant and maintain the proper water level despite failures of some hardware units.

The case study was conducted in the following way: we started with the application domain ontology extracted in [7] and with the AutoFocus meta model shown in figure 1. We tried to map the extracted concepts and associations to the concepts offered by meta model. This mapping is done manually, but is guided by the extracted ontology. If some parts of the extracted ontology were not clear enough to be mapped, we looked for the corresponding chunks of the specification text for clarification.

3.1 Extracted Ontology

This taxonomy was extracted using the techniques described in sections 2.1 and 2.2:

- Message sources
  - water-level-measuring-unit
  - steam-level-measuring-unit
  - pump-controller
- Message receivers
  - physical-unit
  - control unit
  - pump
- Potentially failing hardware
  - water-level-measuring-unit
  - steam-level-measuring-unit
  - pump
  - control-unit
  - pump-controller
- Operation modes
  - waiting-state
  - emergency-stop-mode
  - normal-mode
  - degraded-mode
  - initialization-mode
  - rescue-mode
- Messages
  - signal
  - message
  - message-stop
  - message-steam-boiler-waiting
  - message-open-pump
  - message-close-pump
  - message-pump-failure-detection
  - ...
- Actuators
  - valve
  - pump
- Failures
  - failure
  - pump-failure
  - transmission-failure
  - pump-controller-failure
  - water-level-measuring-unit-failure
  - steam-level-measuring-unit-failure

This taxonomy was also enriched by associations. Every association involves two concepts and has the form like

- transmission-failure CAUSES emergency-stop-mode
- pump-controller CONTROLS pump
- message-pump-control-repaired-acknowledgement IS-SENT-BY control-unit
- ...

For the sake of brevity we do not want to list all the associations here.

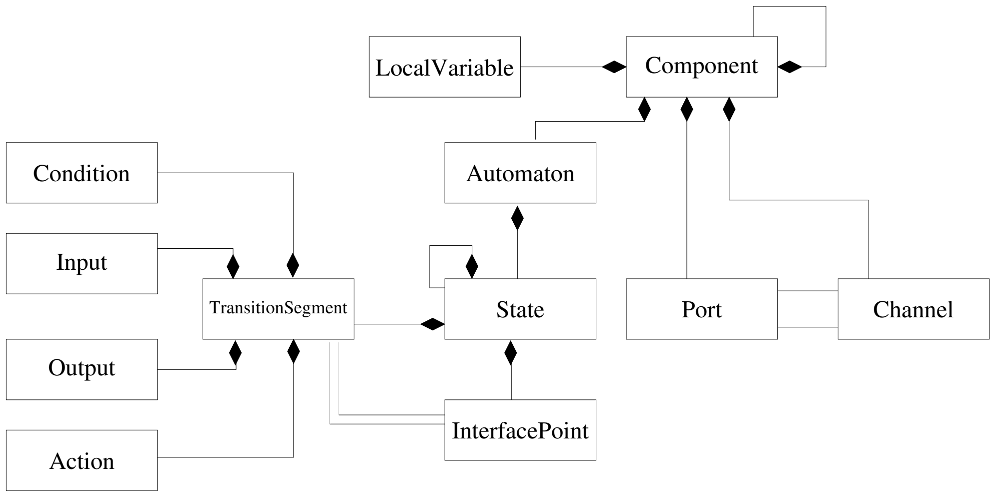


Figure 1. AutoFocus Meta Model

3.2 AutoFocus Meta Model

AutoFocus [1] is a CASE tool for modeling distributed reactive systems. It models a system as a set of components, connected by channels and communicating via message-exchange. Figure 1 shows the concepts used to build AutoFocus models and relations between them. In the diagram we omit the multiplicity specifications and role names, because we can not use them to map the results of text mining anyway.

We want to use this meta model in the following way: first of all, we determine the components building up the system (class "Component" in figure 1). Then, we determine communication ways (channels, class "Channel") between the components. To specify the component behavior, we attribute an automaton and a set of local variables (class "LocalVariable")

3.3 Matching the Ontology and the AutoFocus Meta Model

In this section we describe the process of matching the extracted ontology and the AutoFocus meta model. In the first step we try to map every extracted concept *class* to an AutoFocus concept. In the second step we map different associations to suitable concepts.

3.3.1 Messages

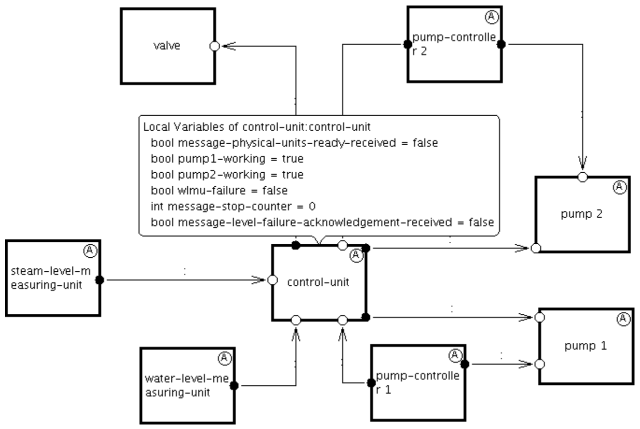
It is impossible to map the messages to AutoFocus concepts. "Message" or a similar concept does not exist in AutoFocus. Although AutoFocus uses message exchange as communication means, the channels are untyped and we can not define an alphabet for every channel. One can also see in figure 1 that the channel is only related to its ports (end points) and to a component, but there is no concept of channel alphabet. To the contrary, "message" is used in the specification text in the sense "element of the channel alphabet", so it can not be mapped to AutoFocus model.

3.3.2 Components and Channels

To model the components (class Component in figure 1), we can pick several categories of our taxonomy: Message-sources and receivers, actuators and "potentially failing hardware" are candidates for components. These categories are not disjoint and we get the following set of components: {*steam-level-measuring-unit, water-level-measuring-unit, valve, control-unit, pump-controller, pump*}
This set also shows a limitation of our text mining approach: According to the specification text, we need four pumps and four pump-controllers, but this does not follow from the results of our text analysis. To extract this information, we would need precise semantics evaluation and not just syntax– and word-frequency–based analysis.

In the next step we want to connect the components by channels. There is no category in our extracted taxonomy that could correspond to channels. There is also only one association between concepts that can be used to determine channels: "pump-controller" CONTROLS "pump". This implies a channel from the pump-controller to the controlled pump. For other channels we have to use the sentence

> The program communicates with the physical-units through messages which are transmitted over a number of dedicated lines connecting each physical-unit with the control-unit.



```
Local Variables of control-unit:control-unit
  bool message-physical-units-ready-received = false
  bool pump1-working = true
  bool pump2-working = true
  bool wlmu-failure = false
  int message-stop-counter = 0
  bool message-level-failure-acknowledgement-received = false
```

Figure 2. Components and channels

Such a sentence is not yet analyzable with the currently available text analysis tools. To decide about the direction of the channels, we can use the classification of the physical-units as message-senders and message-receivers (see section 3.1). This yields the component structure shown in figure 2. (We instantiate just two pumps and pump-controllers in order not to over–complicate the picture. The extension to four pumps is straightforward.) The box attached to control-unit in figure 2 lists the local variables of control-unit. We will explain the origin and meaning of local variables in section 3.3.4.

3.3.3 Components States

When mapping our extracted concepts to states (class State in figure 1), we have to decide which extracted state belongs to which component. First of all, there are two kinds of states: "operation modes" and "failures". Some states can be assigned by the means of extracted associations: For example, we have associations of the kind "pump-failure" IS-FAILURE-OF "pump". The other failures can be assigned by the means of their names. The only failure that can not be assigned in such a way is "transmission-failure", so we assign it to the control-unit. "Operation modes" are states of the control-unit. So, we get the following set of states for the control-unit:
{*initialization-mode, normal-mode, waiting-state, degraded-mode, rescue-mode, emergency-stop-mode, transmission-failure* }

All the components that can have a failure and that got "failure" as a state in the previous step also need a "working" state as a counterpart of "failure". For example, for the pump-controller we get the states and transitions shown in figure 3. In such a way all the components but valve get their internal states. According to the results of text mining,

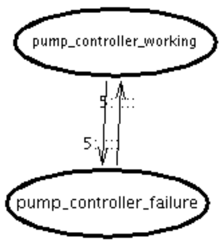


Figure 3. State transition diagram for pump-controller

valve has no failure and is not involved in any association, so it remains stateless.

3.3.4 State Transitions

Associations Involving Operation Modes. In the very first step of state transition modelling we model the transitions between the "failure" and the "working" states. As the results of text mining do not provide any further information about these transitions, we model them as ϵ–transitions (see also figure 3).

To model other state transitions, we use the associations between concepts, extracted from the text. These associations often involve an operation mode and a message. This yields, however, only a half transition: we lack either the start or the destination mode. For example, the association

> "normal-mode" IS-ENTERED-AFTER "message-physical-units-ready"

contains only information about the input message and the goal state, which is not enough to construct a state transition. To solve this problem, we look for the lines of our specification text where certain association occurs. Due to the structure of the specification text, we get the start state in such a way.

In the initialization-mode we get following associations:

- "normal-mode" IS-ENTERED-AFTER "message-physical-units-ready"
- "message-program-ready" IS-SENT-IN "initialization-mode"
- "message-program-ready" IS-SENT-UNTIL "message-physical-units-ready"

The first association can be translated directly into a state transition from "initialization-mode" to "normal-mode" (figure 4). The others give rise to different problems. The second and the third association can only be interpreted as a transition without state change. To stop sending "message-program-ready" when "message-physical-units-ready" is received, we need an indicator variable showing that the message-has been received (see also figure 2).

Waiting-state is involved in just one association:

- "waiting-state" WAITS-FOR "message-steam-boiler-waiting"

This association does not indicate the goal state. The textual context of the original sentence gives no information either, so we introduce a new state, called "UNKNOWN", as the universal source and sink for such transitions.

When trying to determine in a similar way transitions from/to "emergency-stop-mode", we discover following associations:

- "transmission-failure" CAUSES "emergency-stop-mode"
- "water-level-measuring-unit-failure" CAUSES "emergency-stop-mode"

To interpret the second association as a transition, we introduce the variable "wlmu-failure"[2] as a local variable of control-unit.

Introduction of this variable gives rise to an important idea: we have to differ between "wlmu-failure" as a state of the water-level-measuring-unit and "wlmu-failure" as a failure detected by the control-unit. In general, we have to differ between real failures and the failures detected by the control-unit. To indicate the detected failures, we introduce indicator variables to the control-unit. Figure 2 shows three of them: two "pumpX-working" indicators and "wlmu-failure". We can easily add more indicators for other detected failures.

Associations Involving Failures. When considering associations involving failures, we find three associations not yet taken into account in our model:

- "transmission-failure" IS-DETECTED-BY-WRONG "message"
- "wlmu-failure" IS-DETECTED-BY-WRONG "message-level"
- "pump-controller-failure" IS-DETECTED-BY-MISSING-REACTION-TO "message-start"
- "message-level-failure-detection" DETECTS "wlmu-failure"

The first, second and third associations can not yet be translated into model elements: They require detailed modelling of system behavior, which we can not provide basing solely on the results of text mining. We have to know what a wrong message means (unexpected or something else), we have to keep record of the $should$–water level and we have to know the $should$–reaction to the "message-start". The only thing we can model is transmission-failure: as wrong message can be received in every operation mode, each mode gets a transition to "transmission-failure" (figure 4).

[2]In the following, we abbreviate "water-level-measuring-unit" as "wlmu" and "steam-level-measuring-unit" as "slmu"

The fourth association stems from the following text fragment:

> LEVEL-FAILURE-DETECTION: This message is sent (until receipt of the corresponding acknowledgement) to indicate to the physical-units that the program has detected a failure of the water-level-measuring-unit.

We model this sentence in the following way: If the variable "wlmu-failure" is set, the control-unit sends the "message-level-failure-detection", which we can model in our state transition diagram (figure 4). To model the "until"–part of the requirement, we introduce the variable indicating the receipt of the acknowledgement message (see also figure 2).

The control-unit sends the "message-level-failure-detection" to other physical-units. This implies that we have to introduce a state or a variable "wlmu-failure" to the units other than control-unit as well.

Associations Involving Messages. The only association involving messages that is not yet mapped to the model, is the following one:

- "message-stop", SENT 3 TIMES, CAUSES "emergency-stop-mode"

It stems from the sentence

> stop: when the message-stop has been received three times in a row by the program, the program must go into emergency-stop-mode.

To satisfy this requirement, we introduce counters/indicator variables, similar to the pump-failures (see figure 2).

Further Specification Steps. We are aware that our model, presented mainly in figure 4, is not complete.

Following steps has to be done yet:

- Byproducts of mapping text mining results:
  - Introduce all the necessary indicator variables and use them really as pre- and postconditions of transitions
  - Introduce extra states or variables for "…-acknowledgement to stop sending "…-detection" messages.

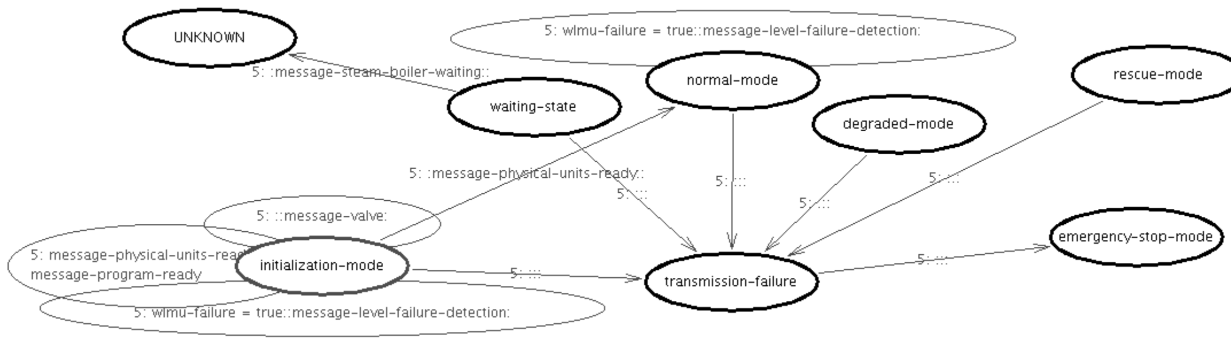

UNKNOWN
5: wlmu-failure = true::message-level-failure-detection:
5: :message-steam-boiler-waiting::
normal-mode
rescue-mode
waiting-state
degraded-mode
5: :message-physical-units-ready::
5: :::
5: :::
5: :::
5: :::
5: ::message-valve:
emergency-stop-mode
5: message-physical-units-ready
message-program-ready
initialization-mode
5: :::
5: :::
transmission-failure
5: wlmu-failure = true::message-level-failure-detection:

Figure 4. State transition diagram for control-unit

- Other steps, that are necessary, but that can not be done using solely the results of text mining
  - Control the channel alphabets and the directions of message-flow. Further channels (additional to those shown in figure 2) can become necessary.
  - Eliminate the state “UNKNOWN” and properly implement state changes
  - Implement the state changes on the basis of system dynamics (water level etc.) and not only on the basis of messages

We do not want to go further in the implementation, because here purely manual work begins. Text mining gives us no more support. The goal of our work was to show how far we can get using the results of text mining and what kind of problems occur when mapping the results of text mining to the concepts of an existing specification formalism.

4 Conclusion

In this paper we presented a case study showing how to get from a textual specification to an initial version of formal model. We started with the results of text analysis performed in [7] and with a meta model of the specification formalism. The results of the text mining were mapped to the concepts offered by the specification formalism.

Although the result of the modelling is neither complete nor executable, we do not want to go further: Our goal is not to build a full–fledged specification, but to show how far we can get using text mining.

Using the results of text mining, we could get surprisingly far: we modeled the system architecture (components and connections), internal states of the components and state transitions. We believe that together with approaches described in [7] we have a powerful method of requirements analysis.

Acknowledgements

Here we want to thank Alexander Pretschner, Tobias Hain, Jan Romberg and Manfred Broy for fruitful discussions and help with AutoFocus modelling.

References

[1] The AutoFocus Homepage. http://autofocus.in.tum.de/index-e.html, accessed 17.10.2003.

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

[3] J.-R. Abrial, E. Börger, and H. Langmaack. *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control*, volume 1165 of *LNCS*. Springer–Verlag, 1996.

[4] J.-R. Abrial, E. Börger, and H. Langmaack. The steam boiler case study: Competition of formal program specification and development methods, in [3], 1996. http://www.informatik.uni-kiel.de/˜procos/dag9523/dag9523.html.

[5] M. Collins. *Head-Driven Statistical Models for Natural Language Parsing*. PhD thesis, University of Pennsylvania, 1999.

[6] D. Faure and C. Nédellec. Asium: Learning subcategorization frames and restrictions of selection. In Y. Kodratoff, editor, *10th European Conference on Machine Learning (ECML 98) – Workshop on Text Mining*, Chemnitz Germany, April 1998 1998.

[7] L. Kof. An Application of Natural Language Processing to Requirements Engineering — A Steam Boiler Case Study. Contribution to ICSE 2004.

[8] A. Maedche and S. Staab. Discovering conceptual relations from text. In W.Horn, editor, *ECAI 2000. Proceedings of the 14th European Conference on Artificial Intelligence*, pages 321–325, Berlin, 2000. IOS Press, Amsterdam.

[9] R. Srikant and R. Agrawal. Mining generalized association rules. *Future Generation Computer Systems*, 13(2–3):161–180, 1997.