DODT: Increasing Requirements Formalism using Domain Ontologies for Improved Embedded Systems Development

Stefan Farfeleder
Institute of Computer Languages
Vienna University of Technology
stefan.farfeleder@tuwien.ac.at

Thomas Moser
CDL Flex
Vienna University of Technology
thomas.moser@tuwien.ac.at

Andreas Krall
Institute of Computer Languages
Vienna University of Technology
andi@complang.tuwien.ac.at

Tor Stålhane
Department of Computer and Information Science
Norwegian University of Science and Technology
stalhane@idi.ntnu.no

Herbert Zojer
Infineon Technologies Austria AG
herbert.zojer@infineon.com

Christian Panis
Catena Radio Design bv
cpanis@catena.nl

***Abstract*—In times of ever-growing system complexity and thus increasing possibilities for errors, high-quality requirements are crucial to prevent design errors in later project phases and to facilitate design verification and validation. To ensure and improve the consistency, completeness and correctness of requirements, formal languages have been introduced as an alternative to using natural language (NL) requirement descriptions. However, in many cases existing NL requirements must be taken into account. The formalization of those requirements by now is a primarily manual task, which therefore is both cumbersome and error-prone. We introduce the tool DODT that semi-automatically transforms NL requirements into semi-formal boilerplate requirements. The transformation builds upon a domain ontology (DO) containing knowledge of the problem domain and upon natural language processing techniques. The tool strongly reduced the required manual effort for the transformation. In addition the quality of the requirements was improved.**

I. Introduction

The potential for errors in the development of embedded systems is coupled to system complexity which is increasing year after year. Verification and validation (V&V) is used both to prevent errors and to find and fix them. To be able to test whether a system is behaving as intended, a requirements specification to test against is needed. The quality of the requirements specification decides how meaningful and efficient V&V can be performed. From the viewpoint of V&V a requirement should be precise, i.e., not open to interpretation, and it should support automated verification to minimize costs. Requirements specifications using NL have weaknesses in both aspects as there is always the need for human interpretation. Thus requirements languages based on a formalism are used more often in embedded systems instead, especially in safety-critical environments.

The research leading to these results has received funding from the ARTEMIS Joint Undertaking under grant agreement N° 100016 and from specific national programs and/or funding authorities. This work has been supported by the Christian Doppler Forschungsgesellschaft and the BMWFJ, Austria.

However, in many cases the requirements engineer (RE) must take into account existing NL requirements, e.g. needs expressed by system stakeholders, or requirements adapted from standard documents or re-used from previous projects. To take advantage of requirements formalism these NL requirements need to be converted into formal requirements. By now the formalization of requirements is a primarily manual task, which therefore is both cumbersome and error-prone.

In this work we present a semi-automated process to convert NL requirements into a semi-formal representation called boilerplates requirements. The process is implemented by our tool DODT. Most of the work is done automatically by the tool but at some points the RE needs to intervene. We include an evaluation of the process on a set of requirements. The results show the advantages of our process: for the majority (60%) of the requirements the transformation can be done in an almost fully automated way.

The remainder of this paper is structured as follows: Section II discusses related work; section III motivates our research; section IV describes our process; section V contains the evaluation; and finally section VI concludes.

II. Related Work

This section summarizes related work on Requirements Engineering, Boilerplates and Natural Language Processing.

A. *Requirements Engineering*

Requirements Engineering is a discipline that deals with understanding, documenting, communicating and implementing customers' needs. There is a gap between informal NL requirements and requirements specified in a formal language, e.g., first-order logic. Stakeholders prefer informal requirements, because they can be understood and defined by everyone without training required. The downside is the inherent imprecision and ambiguity of NL statements. Formal languages provide vastly better possibilities for requirements analysis and verification against system models, e.g., model checking, but are difficult to use for non-experts.

There have been numerous approaches to transform NL requirements into semi-formal and formal languages: into ACTL (action based temporal logic) formulae [1]; into Use Case Models [2]; into CSP process algebra [3]; into Circal process algebra [4]; into executable LSCs (live sequence charts) [5]; into scenarios [6]; and into temporal logic formulae [7].

Our transformation approach aims at covering a broader range of requirement statements. This includes requirements which cannot be expressed formally at all. It achieves this by being less ambitious in terms of formality.

B. Boilerplates

Hull, Jackson and Dick [8] first used the term *boilerplate* to refer to a textual requirement template. A boilerplate consists of a sequence of attributes and fixed syntax elements (FSEs). As an example, a common boilerplate is "⟨*system*⟩ `shall` ⟨*action*⟩". In this boilerplate ⟨*system*⟩ and ⟨*action*⟩ are attributes and `shall` is an FSE. It is possible to combine several boilerplates by means of simple concatenation. This allows keeping the number of required boilerplates low while at the same time having a high flexibility. During instantiation textual values are assigned to the attributes of the boilerplates.

Stålhane, Omoronyia and Reichenbach [9] extended boilerplates with a DO by linking attribute values to ontology concepts. The requirement language used in this work is based on their combination of boilerplates and the DO.

C. Natural Language Processing (NLP)

NLP refers to *systems that analyze, attempt to understand, or produce one or more human languages, such as English, Japanese, Italian, or Russian* [10]. Three important NLP applications related to this work are (a) *Part-Of-Speech (POS) Tagging* which categorizes the tokens of a sentence into different types, (b) *Stemming* which finds the root (stem) of a word for inflected forms, e.g., the singular for a plural word, and (c) *Syntax Parsing* which analyzes the syntax of a sentence and builds a syntax tree. Our tool uses the GATE[1] framework [11] for NLP. It provides a uniform interface to several NLP algorithms. Among others the *ANNIE POS tagger*, the *GATE Morphological analyzer*, and the *OpenNLP Parser*[2] are used.

III. Motivation

Our goal is to improve system development by increasing requirements formalism. We use boilerplates because they support automated requirement analysis and are usable for most requirements while at the same time being as readable as NL requirements. DODT supports the creation of new boilerplate requirements. The RE selects boilerplates and the tool provides suggestions gained from the DO.

[1]http://gate.ac.uk/

[2]http://incubator.apache.org/opennlp/

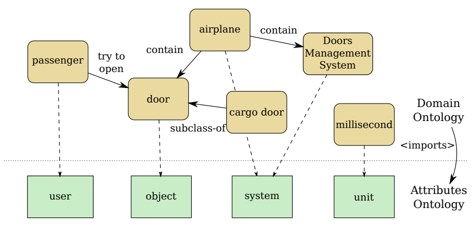

Fig. 1. DMS Domain Ontology

However, if the RE wants to transform existing NL requirements, a lot of manual work is still required which could often be performed semi-automatically. For the requirement "The passenger shall be able to open the door." and the boilerplate "⟨*user*⟩ `shall be able to` ⟨*action*⟩" it is clear to a human that the original requirement can be trivially transformed into a boilerplate requirement by assigning "The passenger" to ⟨*user*⟩ and "open the door" to ⟨*action*⟩. With the current tool support the RE first needs to select the boilerplate and then fill in the attributes, either by copy-and-paste or by selecting from the list of suggestions.

We therefore concentrated our research on tool support for better supporting this transformation. We expect the tool to help with (a) selecting boilerplates and (b) filling in the attributes. These two points are not entirely independent. If more than one boilerplate fits, the "quality" of the entire requirement depends on both the choice of boilerplates and the resulting attribute values. We thus soon realized that a promising approach is to try several boilerplates and resulting attribute values, and to compare and order the results. We also realized that not everything can be done automatically. Our expectation is that the tool is able to reduce the amount of manual work required from the RE who should be able to concentrate on the "difficult" cases, requiring major restructuring of a sentence. Also, the RE should not be overwhelmed by the choices he has to make. In order to increase the efficiency of the transformation, we want to use simple yes-or-no questions or lists with few entries to choose from.

For evaluation we applied the tool to requirements of the *Doors Management System* (DMS) - a use case developed by EADS for experimenting with and evaluating requirements engineering techniques within the CESAR project[3]. DMS controls the doors of an airplane. Its main objective is to lock the doors of an airplane while it is moving and to unlock them when the airplane is parked on the ground.

IV. Semi-Automatic Conversion to Boilerplates

This section describes DO and Transformation Process.

A. Domain Ontology

Our approach uses a DO which stores information: concepts, relations and axioms. Please refer to [12] for more details about the DO. Fig. 1 shows a small subset of the DMS DO we used for the evaluation. The top part shows the

[3]http://cesarproject.eu/

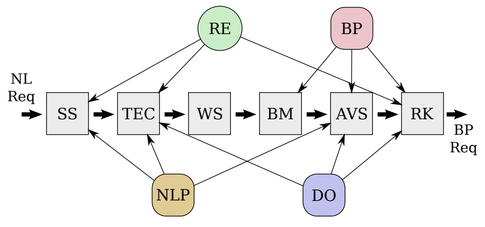

Fig. 2. Transformation Process

concepts as nodes, and relations and axioms as edges. The bottom part shows a special ontology, the attributes ontology, which contains the boilerplate attributes as concepts. The dotted arrows show the links between concepts and boilerplate attributes, e.g., "passenger" is a ⟨*user*⟩.

B. Transformation Process

Our process (Fig. 2) to transform NL requirements into boilerplate requirements is semi-automatic. Most of the work is done automatically but at some points the RE has to select from a list of suggestions and finally to validate and correct the results if necessary. The process inputs are manual work of the RE, a set of boilerplates (BP), NLP techniques and DO. The transformation process consists of six steps:

1) Sentence Selection (SS): A boilerplate requirement represents a single sentence. NL requirements which consist of several sentences are split into several requirements. The tool uses the GATE sentence splitter to detect sentence beginnings and ends. If more than one sentence is detected, the RE is asked to select the sentence to transform into the boilerplates representation. In the evaluation we found that sometimes only the first sentence is an actual requirement and that the following sentences are comments, explanations or rationales. Such information should be clearly separated from the requirement statement, e.g., using an requirement attribute.

2) Typographic Error Correction (TEC): To correct typographic errors in the NL requirement, the DO concepts are used as a dictionary. For any sequence of nouns (detected by a POS tagger) which is not contained in the DO, the tool computes the Damerau-Levenshtein distance [13] (the number of single letter edits to get from one expression to another) to each of the DO entities. If there are DO entities similar to the noun sequence, i.e., below a threshold for the distance, the RE is asked to select from a list of possible corrections.

3) Word Substitution (WS): The goal of this step is to transform a requirement in such a way that matching boilerplates can be found in the next step. User-defined substitution rules are applied to replace expressions of the original requirement with synonymous FSEs. The substitutions we used in our evaluation are to replace occurrences of "must", "should" and "will" with the word "shall" in order to match our set of boilerplates based upon "shall" and to replace "in case that" with "if". The actual transformations that should be applied here strongly depend on the original NL requirements and on the used boilerplates. Therefore the tool allows the RE to configure the substitution rules.

4) Boilerplate Matching (BM): For each boilerplate the tool searches all possible ways the boilerplate matches the requirement. For the matching only FSEs are considered. They match if identical words or punctuation exist in the requirement. The text between FSEs will be assigned to the attributes in the next step. A boilerplate can match at several starting positions in a requirement and also in different ways for the same starting position, e.g., for the boilerplate "`if` ⟨*operational condition*⟩ `,`" each comma following one occurrence of "if" will be considered a different match. After all matches have been found, conflicts between matches are computed. Two matches conflict if they span overlapping parts of the requirement. Then all combinations of match sets that are *maximal* - in the sense that no further non-conflicting matches can be added - are computed.

5) Attribute Value Splitting (AVS): In this step, text occurring between FSEs is assigned to boilerplate attributes. When there are two or more adjacent attributes, a decision has to be made about where the text should be split. The tool iterates over all possible ways (word boundaries) to split the text and computes ratings. The rating is composed of a syntactic part and a semantic part. The syntactic part checks if the splitting agrees with the syntax tree of the requirement by increasing the rating if the children of a syntax tree node are assigned to the same attribute. The semantic part checks if the expressions assigned to an attribute are semantically sound by using the DO links from concepts to boilerplate attributes. The splitting with the best rating is then used.

6) Ranking (RK): The BM step typically generates several solutions on how to map the original NL requirement to the boilerplate representation. It is the responsibility of the RE to choose one of the proposed solutions. However, in order to support the RE with this decision, the tool computes a ranking of the solutions. This ordering is computed from the percentage of the requirement text that could be transformed, the number of words that could be matched with FSEs and the quality of the matching between boilerplate attributes and attribute values. ⟨*user*⟩ would be considered a better match than ⟨*system*⟩ if the corresponding attribute value contains "passenger" (linked to ⟨*user*⟩ in the DO).

After the six steps described above the RE can adjust the requirement before saving it. Parts of the requirement which could not be transformed are displayed by the tool. Step BM creates a maximal matching which in some cases splits a requirement into too many boilerplates. This is a deliberate design choice as removing an extra boilerplate requires much less effort than adding a missing one. Refinement links are automatically created to connect the original NL requirement with the new one in order to provide requirements traceability.

V. Evaluation

The DMS use case contains 43 NL requirements: functional, safety, performance, reliability, availability and cost. For each of the 43 requirements we used the tool to suggest a transformation into boilerplates. We measured how much and what

TABLE I
ONTOLOGY MEASUREMENTS

Entity	Count
Concepts	107
Relations	70
Axioms	123

TABLE II
RESULTS DATA

Item	Result
# NL Reqs.	43
# BP Reqs.	47
# Typogr. errors corrected	8
avg. # Suggestions	2.74
Unmodified Reqs.	60%

kind of interaction of the RE was necessary. Finally we compared the resulting boilerplate requirements with another set of boilerplate requirements which was transformed manually, and with the original requirements. We re-used the boilerplates used for the manual transformation in this evaluation. The data for the DMS ontology was initially provided by EADS and was completed by the authors. Table I lists the number of concepts, relations and axioms.

Table II summarizes the evaluation results. Seven of the 43 requirements consisted of two sentences, all of them were correctly detected by the SS step. From those second sentences we regarded four as requirements, resulting in a total of 47 boilerplate requirements. The TEC was able to correct 8 errors (inconsistent spellings like "pressurisation" vs. "pressurization" and misspellings like "miliseconds"). The BM step produced 2.74 suggestions on average, with a maximum of 13 and a standard deviation of 1.82, in our opinion a reasonable number of suggestions to select from. The AVS heuristic seems to work well: All six occurrences of adjacent attributes were handled correctly. RK managed to put the boilerplate matching found to be best-fitting by us at the first position in 31 out of 39 cases (79.4%).

The 47 resulting requirements contain a total of 97 instantiated boilerplates. For 19 requirements we needed modifications after the transformation; 22 extra boilerplates were removed from the requirements and two were added. The remaining 60% needed no further modifications.

Comparing the resulting requirements with the ones we transformed manually, we made the following observations:

- Requirements formulated using passive voice have been converted to active voice in the manual transformation to fit better into the boilerplate representation. Tool support for this is something we want to look into.
- Statements like "it shall be possible" and "there shall be means to" have been converted to include a subject in the manual transformation, e.g., "the DMS shall be able to". Such statements need further manual work after our transformation process.
- Some of the requirements started with "*subject* shall be designed such that" and then continued with the actual requirement. In the manual transformation we simply removed this prefix.
- In the manual transformation we dropped some redundant expressions while doing the transformation. Our process does not try to do that.

Compared to the original NL requirements we have achieved the following improvements:

- We increased the requirements' formalism. The separation into boilerplate attributes and FSEs allows using more advanced analysis methods, e.g., categorization of requirements by boilerplates or by attributes. Measurement values and units are explicit, which allows for automatic processing of requirements, e.g. for verification.
- We increased the understandability by linking requirement terms to domain concepts with textual descriptions.
- We increased the internal uniformity of requirements, all requirements now use "shall".
- Our process managed to correct typographical errors.

VI. Conclusion and Future Work

In this paper, we presented a semi-automated process for the transformation of NL requirements into semi-formal boilerplate requirements. Our evaluation shows that for 60% of the requirements manual interaction is minimal. We also presented the quality improvements that were achieved. Using boilerplate requirements instead of NL requirements will lead to improved embedded systems development.

In the future we will research tool support for advanced sentence restructuring, e.g., translation of passive voice into active voice, and whether it is possible and useful to create new boilerplates from a set of NL requirements. The ranking algorithm is experimental at this stage and needs to be evaluated on more requirements.

References

[1] R. Nelken and N. Francez, "Automatic Translation of Natural Language System Specifications into Temporal Logic," in *Proc 8th Int. Conf. Computer Aided Verification*. Springer, 1996, pp. 360–371.

[2] S. Seresht and O. Ormandjieva, "Automated Assistance for Use Cases Elicitation from User Requirements Text," in *Proc. 11th Workshop on Requirements Engineering*, vol. 16, 2008, pp. 128–139.

[3] G. Cabral and A. Sampaio, "Formal Specification Generation from Requirement Documents," *Electronic Notes in Theoretical Computer Science*, vol. 195, pp. 171–188, 2008.

[4] R. Fernandes and A. Cowie, "Capturing Informal Requirements as Formal Models," in *Proc. 9th Australian Workshop on Requirements Engineering*, 2004, pp. 1–8.

[5] M. Gordon and D. Harel, "Generating Executable Scenarios from Natural Language," *Proc. 10th Int. Conf. Computational Linguistics and Intelligent Text Processing*, pp. 456–467, 2009.

[6] X. Liu, "Scenario Elicitation from Natural Language Requirements," in *2nd Int. Workshop on Education Technology and Computer Science*, vol. 2. IEEE, 2010, pp. 252–255.

[7] S. Konrad and B. Cheng, "Facilitating the Construction of Specification Pattern-based Properties," in *Proc. 13th Int. Conf. on Requirements Engineering*. IEEE, 2005, pp. 329–338.

[8] E. Hull, K. Jackson, and J. Dick, *Requirements Engineering*. Springer, 2005.

[9] T. Stålhane, I. Omoronyia, and F. Reichenbach, "Ontology-Guided Requirements and Safety Analysis," in *Proc. 6th Int. Conf. on Safety of Industrial Automated Systems*, 2010.

[10] J. Allen, "Natural Language Processing," in *Encyclopedia of Computer Science*, 4th ed., R. Anthony, E. Reilly, and D. Hemmendinger, Eds. John Wiley & Sons, 2003, pp. 1218–1222.

[11] H. Cunningham *et al.*, "GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications," in *Proc. 40th Anniversary Meeting Association for Computational Linguistics*, 2002, pp. 168–175.

[12] S. Farfeleder *et al.*, "Ontology-Driven Guidance for Requirements Elicitation," in *Proc. 8th Extended Semantic Web Conf. (in press)*, 2011.

[13] V. Levenshtein, "Binary Codes Capable of Correcting Deletions, Insertions, and Reversals," *Soviet Physics Doklady*, vol. 10, no. 8, pp. 707–710, 1966.