Ontology-Driven Guidance for Requirements Elicitation

Stefan Farfeleder[1], Thomas Moser[2], Andreas Krall[1], Tor Stålhane[3], Inah Omoronyia[4], and Herbert Zojer[5]

[1] Institute of Computer Languages, Vienna University of Technology
{stefanf,andi}@complang.tuwien.ac.at
[2] Christian Doppler Laboratory "Software Engineering Integration for Flexible Automation Systems", Vienna University of Technology
thomas.moser@tuwien.ac.at
[3] Department of Computer and Information Science, Norwegian University of Science and Technology
stalhane@idi.ntnu.no
[4] Irish Software Engineering Research Centre, University of Limerick
inah.omoronyia@lero.ie
[5] Infineon Technologies Austria AG
herbert.zojer@infineon.com

Abstract. Requirements managers aim at keeping their sets of requirements well-defined, consistent and up to date throughout a project's life cycle. Semantic web technologies have found many valuable applications in the field of requirements engineering, with most of them focusing on requirements analysis. However the usability of results originating from such requirements analyses strongly depends on the quality of the original requirements, which often are defined using natural language expressions without meaningful structures. In this work we present the prototypic implementation of a semantic guidance system used to assist requirements engineers with capturing requirements using a semi-formal representation. The semantic guidance system uses concepts, relations and axioms of a domain ontology to provide a list of suggestions the requirements engineer can build on to define requirements. The semantic guidance system is evaluated based on a domain ontology and a set of requirements from the aerospace domain. The evaluation results show that the semantic guidance system effectively supports requirements engineers in defining well-structured requirements.

Keywords: requirements elicitation, domain ontology, elicitation guidance, requirements engineering.

1 Introduction

A major goal of requirements engineering is to achieve a common understanding between all project stakeholders regarding the set of requirements. Modern IT projects are complex due to the high number and complexity of requirements, as well as due to geographically distributed project stakeholders with different

G. Antoniou et al. (Eds.): ESWC 2011, Part II, LNCS 6644, pp. 212–226, 2011.

backgrounds and terminologies. Therefore, adequate requirements management tools are a major contribution to address these challenges. Current requirements management tools typically work with a common requirements database, which can be accessed by all stakeholders to retrieve information on requirements content, state, and interdependencies.

Requirements management tools help project managers and requirements engineers to keep the overview on large amounts of requirements by supporting: (a) Requirements categorization by clustering requirements into user-defined subsets to help users find relevant requirements more quickly, e.g., by sorting and filtering attribute values; (b) Requirements conflict analysis (or consistency checking) by analyzing requirements from different stakeholders for symptoms of inconsistency, e.g., contradicting requirements; and (c) Requirements tracing by identifying dependencies between requirements and artifacts to support analyses for change impact and requirements coverage. Unfortunately, requirements management suffers from the following challenges and limitations:

- Incompleteness [5] of requirements categorization and conflict identification, in particular, when performed manually.
- High human effort for requirements categorization, conflict analysis and tracing, especially with a large number of requirements [5].
- Tracing on syntactic rather than on concept level: requirements are often traced on the syntactic level by explicitly linking requirements to each other. However, requirements engineers actually want to trace concepts, i.e., link requirements based on their meaning, which can be achieved only partially by information retrieval approaches like "keyword matching" [11][12].

The use of semantic technologies seems promising for addressing these challenges: Ontologies provide the means for describing the concepts of a domain and the relationships between these concepts in a way that allows automated reasoning [18]. Automated reasoning can support tasks such as requirements categorization, requirements conflict analysis, and requirements tracing. While these are very valuable efforts, we think what is missing here is additionally having a proactive and interactive guidance system that tries to improve requirements quality while actually eliciting requirements.

In this work we present the prototypic implementation of a semantic guidance system used to assist requirements engineers with capturing requirements using a semi-formal representation. Compared to the usual flow - write requirements, analyze requirements using the domain ontology, improve requirements - our approach directly uses the domain ontology knowledge in a single step. The semantic guidance system uses concepts, relations and axioms of domain ontologies to provide a list of suggestions the requirements engineer can build on to define requirements.

We evaluate the proposed semantic guidance system based on a domain ontology and a set of requirements from the aerospace domain. The evaluation results show that the semantic guidance system supports the requirements engineer in defining well-structured requirements. The tool managed to provide useful suggestions for filling missing parts of requirements in the majority of the cases (>85%).

This work is organized in the following way: Section 2 presents related work. Section 3 motivates our research; section 4 introduces our approach to ontology-based guidance. Section 5 presents an evaluation of the tool and section 6 concludes and gives ideas about future work.

2 Related Work

This section summarizes related work, going from the broad field of requirements engineering to the more specific areas of elicitation guidance and finally pattern-based requirements.

2.1 Requirements Engineering

Requirements Engineering is a discipline that deals with understanding, documenting, communicating and implementing customers' needs. Thus, insufficient understanding and management of requirements can be seen as the biggest cause of project failure. In order to improve this situation a systematic process to handle requirements is needed [8]. The main activities of a requirements engineering process can be defined as follows [15]:

- **Requirements Elicitation**. Requirements elicitation involves technical staff working with customers to find out about the application domain, the services the system should provide and the system's operational constraints. The goal is to gather raw requirements.
- **Requirements Analysis and Negotiation**. Requirements analysis and negotiation is an activity which aims to discover problems and conflicts with the requirements and reach agreement on changes to satisfy all system stakeholders (people that are affected by the proposed system). The final goal is to reach a common understanding of the requirements between all project participants.
- **Requirements Documentation and Validation**. The defined requirements, written down in a software requirements specification, are validated against criteria like correctness, completeness, consistency, verifiability, unambiguity, traceability, etc.
- **Requirements Management**. Requirements management consists of managing changes of requirements (keeping them consistent), e.g., by ensuring requirements traceability (identification of interdependencies between requirements, other requirements, and artifacts).

These four steps can be summarized as requirements development. In addition, requirements management is a supporting discipline to control all requirements and their changes during the development life cycle and to identify and resolve inconsistencies between the requirements and the project plan and work products. One important method of requirements management is requirements tracing. Traceability can be defined as the *degree to which a relationship between*

two or more products of the development process can be established [1]. Gotel [7] and Watkins [21] describe why requirements tracing can help project managers in verification, cost reduction, accountability, change management, identification of conflicting requirements and consistency checking of models.

2.2 Elicitation Guidance

There are several approaches to guide users to specify requirements. PROPEL [3] is a tool that provides guidance to define property specifications which are expressed as finite-state automata. For the definition of a property the user is guided by a *question tree*, a hierarchical sequence of questions. There are separate scope trees for a property's behavior and its scope. Based on the answers, the tool chooses an appropriate property template. A team of medical personnel and computer scientists used the tool to formulate properties of medical guidelines.

Kitamura et al. [14] present a requirements elicitation tool that improves requirements quality by analysis. The tool analyzes natural language requirements and finds domain ontology entities occurring in the statements. According to these occurrences and their relations in the ontology, requirements are analyzed in terms of completeness, correctness, consistency and unambiguity. Suggestions are provided to the user in order to improve these metrics, e.g., if the tool finds that a domain concept is not defined in the requirements set, the suggestion would be to add a requirement about the missing concept.

REAS [6] is a tool that interactively detects imprecisions in natural language requirements. It consists of a spelling checker, a rule imposer, a lexical analyzer and a parser. Some of the rules enforced by the tool are writing short requirements, using active voice instead of passive and avoiding possessive pronouns (e.g., "it" and "its"). If the tool detects a violation of a rule, the requirements engineer is asked to improve the offending expression.

Compared to our own approach, PROPEL only deals with a rather specific kind of requirements. The other two approaches try to identify weaknesses after the requirements have been defined and do not actively propose wording while writing them.

2.3 Pattern-Based Requirements

Hull, Jackson and Dick [9] first used the term *boilerplate* to refer to a textual requirement template. A boilerplate consists of a sequence of attributes and fixed syntax elements. As an example, a common boilerplate is "⟨*system*⟩ `shall` ⟨*action*⟩". In this boilerplate ⟨*system*⟩ and ⟨*action*⟩ are attributes and `shall` is a fixed syntax element. It is possible to combine several boilerplates by means of simple string 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; a boilerplate requirement is thus defined by its boilerplates and its attribute values. The

authors did not propose a fixed list of boilerplates[1] but instead envisioned a flexible language that can be adapted or enriched when necessary.

Stålhane, Omoronyia and Reichenbach [20] extended boilerplates with a domain ontology by linking attribute values to ontology concepts. They adapted the requirements analyses introduced by Kaiya [13] to boilerplate requirements and added a new analysis called opacity. The requirement language used in this work is based on their combination of boilerplates and the domain ontology.

Ibrahim et al. [10] use boilerplate requirements in their work about requirements change management. They define a mapping from boilerplate attributes to software design artifacts (e.g., classes, attributes, operations) and add traceability links between requirements and artifacts accordingly. There are several other pattern based languages similar to boilerplates, e.g., requirements based on EBNF grammars [19]. Denger et al. [4] propose natural language patterns to specify requirements in the embedded systems domain. They include a metamodel for requirement statements and one for events and reactions which they use to check the completeness of the pattern language. Compared to boilerplate requirements, their patterns seem to be a bit less generic, e.g., some of the nonfunctional requirements used in our evaluation would be impossible to express.

Matsuo, Ogasawara and Ohnishi [16] use controlled natural language for requirements, basically restraining the way in which simple sentences can be composed to more complex ones. They use a frame model to store information about the domain. There are three kind of frames. The noun frame classifies a noun into one of several predefined categories. The case frame classifies verbs into operations and contains the noun types which are required for the operation. Finally the function frame represents a composition of several simple operations. The authors use these frames to parse requirements specifications, to organize them according to different viewpoints and to check requirements completeness. In contrast to domain ontologies, the frame-based approach seems to be harder to understand and to adapt by non-experts.

3 Research Issues

There have been presented several approaches to use ontologies to analyze requirements. These approaches try to measure quality aspects like completeness, correctness and consistency on a set of requirements. In [20] there is an analysis called *opacity* that basically checks if, for two concepts occurring in a requirement, there is a relation between them in the domain ontology. A conclusion of our review of this analysis was that, rather than first writing an incorrect requirement, then analyzing and improving it, a better approach would be to actually suggest the very same domain information which is used for the opacity analysis to the requirements engineer in the first place. There are two points to this idea:

[1] J. Dick maintains a list of suggestions at `http://freespace.virgin.net/gbjedi/books/re/boilerplates.htm` though.

- We want a system that automatically proposes at least parts of the requirements by using information originating from a domain ontology.
- We want a system that exploits relations and axioms of the domain ontology, i.e., a system that is more powerful than just a simple dictionary. The relations and axioms of the domain ontology represent agreed upon knowledge of stakeholders; by using them we hope to improve the precision of requirements.

We believe that a tool supporting these two points will increase efficiency by speeding up the process to get a high-quality requirements specification.

A semantic guidance system implementing these two points was added to our boilerplates elicitation tool (DODT). To test our assumption, we used the tool on requirements from the domain of the *Doors Management System* (DMS). The DMS is a use case developed by EADS for experimenting with and evaluating requirements engineering and modeling techniques within the CESAR project[2]. The DMS controls the doors of an airplane; its main objective is to lock the doors of an airplane while it is moving (in the air or on the ground) and to unlock them when the airplane is parked on the ground. The system consists of sensors that measure the state of the doors, actuators to lock and unlock the doors and computing elements that control the entire system.

4 Guidance for Boilerplate Requirements

This section presents our approach for guiding the requirement engineer using information of a domain ontology.

4.1 Boilerplate Requirements Elicitation

Figure 1 shows how requirements elicitation works using the boilerplates method. To specify a requirement, one or more boilerplates are chosen by the requirements engineer. The attribute values refer to entities in the domain ontology. During instantiation the attributes of the chosen boilerplates are set and a final boilerplate-based requirement is produced. The semantic guidance system affects the domain ontology, the attribute values and the instantiation. Its purpose is to suggest potential and suitable values for the attributes to the requirements engineer.

Table 1 lists the boilerplate attributes implemented by the tool. We reduced the number of attributes compared to the original suggestions in [9] and [20]. This was done in accordance with user wishes who were uncomfortable with the subtile differences between similar attributes and had problems deciding on which to use.

[2] `http://cesarproject.eu/`

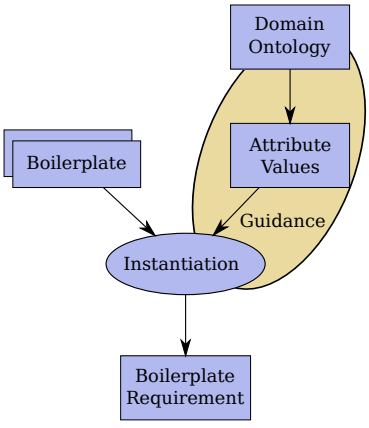

Fig. 1. Boilerplate Requirements Elicitation Flow

Table 1. Boilerplate Attributes and Values

Attribute	Description	Example Value
⟨*action*⟩	A behavior that is expected to be fulfilled by the system, or a capability	open the door
⟨*entity*⟩	A separate entity in the domain; not fitting into ⟨*user*⟩ or ⟨*system*⟩	door status
⟨*number*⟩	A numeric value denoting a quantity	100
⟨*operational condition*⟩	A condition or event that occurs during system operation	the user tries to open the door
⟨*system*⟩	Any part of the system; sub-class of entity	door
⟨*unit*⟩	Unit of measurement	millisecond
⟨*user*⟩	A person somehow interacting with the system, e.g., operating it; sub-class of entity	pilot

4.2 Domain Ontology

The domain ontology contains facts about the domain that are relevant to requirements engineering, i.e., facts that can be used to formulate and to analyze requirements. The domain ontology should be usable to specify requirements for several projects in the same domain. Thus adding concepts which are only relevant to a single product should be avoided. See section 6 for a discussion about combining several ontologies. This approach could be used to split the ontology into a common domain part and a product-specific one.

There are three kinds of facts about the domain stored in the domain ontology. The following list describes them. The tool uses an OWL[2] representation to store ontologies. A detailed mapping to OWL can be found in Table 2.

Table 2. Mapping of domain facts to OWL

Fact	OWL Expressions
Concept(name, definition)	Declaration(Class(*concept-iri*)) AnnotationAssertion(rdfs:label *concept-iri name*) AnnotationAssertion(rdfs:comment *concept-iri definition*)
Relation(subj, label, obj)	Declaration(ObjectProperty(*label-iri*)) AnnotationAssertion(rdfs:label *label-iri label*) SubClassOf(*subj-iri* ObjectAllValuesFrom(*label-iri obj-iri*))
SubClass(sub, super)	SubClassOf(*sub-iri super-iri*)
Equivalence($concept_1$, $concept_2$)	EquivalentClasses(*$concept_1$-iri $concept_2$-iri*)
Deprecated(concept)	AnnotationAssertion(*deprecated-iri concept-iri* `1`)

Concept: A concept represents an entity in the problem domain. The entity can be material (e.g., a physical component of the system) or immaterial (e.g., a temporal state). OWL classes are used to represent concepts. The reason for using classes instead of individuals is the built-in support for sub-classing.

A concept has two attributes, its name and a textual definition. The definition is intended to provide the possibility to check whether the correct term is used.

Relation: A relation is a labeled directed connection between two concepts. A relation contains a label which is expected to be a verb. The label, the relation's source and destination concepts form a subject-verb-object triple. Relations are used for guidance (section 4.3). Relations map to OWL object properties and object property restrictions.

Axiom: There are two types of axioms that are relevant to guidance: sub-class and equivalence axioms. The first one specifies that one concept is a sub-class of another concept, e.g., *cargo door* is a sub-class of *door*. This information is used to "inherit" suggestions to sub-class concepts, e.g., the guidance system infers the suggestion *the user opens the cargo door* from the base class' *the user opens the door*.

The equivalence axiom is used to express that two concepts having different names refer to the same entity in the domain. An example from DMS is the equivalence of *aircraft* and *airplane*. Ideally each real-world phenomenon has exactly one name. However, due to requirements coming from different stakeholders or due to legacy reasons, at times several names are required. It is possible to mark a concept as being *deprecated*; the tool will warn about occurrences of such concepts and will suggest using an equivalent non-deprecated concept instead.

In this work we assume the pre-existence of a suitable domain ontology. See [17] for ways of constructing new domain ontologies. The tool contains an ontology

editor that is tailored to the information described here. We found this editor to be more user-friendly than generic OWL editors like Protégé[3].

4.3 Guidance

When filling the attributes of a boilerplate, the tool provides a list of suggestions to the requirements engineer. The provided guidance depends on the attribute the requirements engineer is currently filling, e.g., the suggestions for ⟨*system*⟩ will be completely different than for ⟨*action*⟩. The idea is to apply an attribute-based pre-filter to avoid overwhelming the user with the complete list of ontology entities. Typing characters further filters this list of suggestions to only those entries matching the typed string.

It is not mandatory to choose from the list of suggestions; the tool will not stop the requirements engineer from entering something completely different. In case information is missing from the domain ontology, an update of the ontology should be performed to improve the guidance for similar requirements. All changes to the domain ontology should be validated by a domain expert to ensure data correctness.

There are three types of suggestions for an attribute; Table 3 provides an overview over the suggestion types.

Concept: The tool suggests to use the name of a concept for an attribute. The tool generates two variants, just the plain name and once prefixed with the article "the". The idea is that most of the times using "the" will be appropriate but sometimes other determiners like "all" or "each" are more suitable and are typed in manually.

Verb-Object: The tool uses a relation from the domain ontology to suggest a verb phrase to the requirements engineer. The suggestion is the concatenation of the verb's infinitive form[4], the word "the" and the relation's destination object. This construction is chosen in order to be grammatically correct following a modal verb like "shall". An example from Figure 2 is the suggestion *check the door status*.

Subject-Verb-Object: For this kind of suggestion the entire relation including subject and object is taken into account. The suggestion text is "the", the subject, the verb conjugated into third person singular form, "the" and the object. An example from Figure 2 is the suggestion *the person tries to open the door*.

It is possible to combine several suggestions simply by selecting the first one, manually typing in "and" and selecting another suggestion.

For the classification of concepts into different attributes a separate ontology, the *attributes ontology*, is used. The attributes ontology contains an OWL class

[3] `http://protege.stanford.edu/`

[4] For building verb infinitives the *morphological analyzer* of the GATE project (`http://gate.ac.uk/`) is used.

Table 3. Suggestion Types

Type	Suggestion
Concept	*concept*
	`the` *concept*
Verb-Object	*verb (inf.)* `the` *object*
Subject-Verb-Object	`the` *subject verb (3rd sing.)* `the` *object*

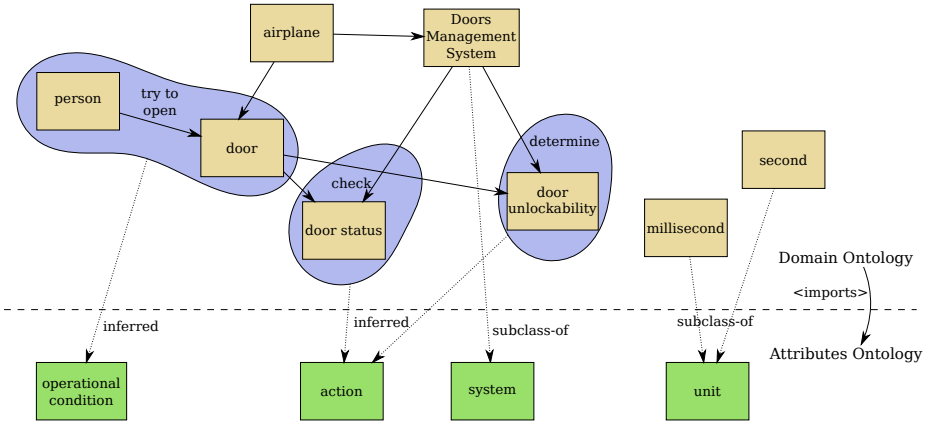

Fig. 2. Domain Ontology and Attributes Ontology

per attribute and the sub-class axioms mentioned in Table 1. The domain ontology imports the attributes ontology to use its classes. Domain concepts are linked to attributes by means of sub-class axioms which are stored in the domain ontology.

An example for the semantic guidance system is given in Figure 2. The domain ontology is shown in the upper part of the figure, the attributes ontology is below. The concept *Doors Management System* is a sub-class of class *system*, which in turn allows the tool to suggest using the *Doors Management System* for a boilerplate containing the attribute ⟨*system*⟩. The blue regions represent verb-object and subject-verb-object suggestions in the domain ontology. Their mapping to the attributes ⟨*action*⟩ and ⟨*operational condition*⟩ is inferred automatically by the tool.

Figure 3 shows the boilerplates for two requirements and some of the suggestions provided by the guidance system. The information that *Doors Management System* is a *system* and that *second* and *millisecond* are values for attribute *unit* is stored in the domain ontology itself. The suggestions *check the door status* and *determine the door unlockability* are inferred from the domain ontology relations. The knowledge to suggest verb-object pairs for the attribute *action* is a built-in feature of the tool. The attribute *operational condition* turns out to be the most difficult one in terms of providing useful suggestions. The reason for this is that

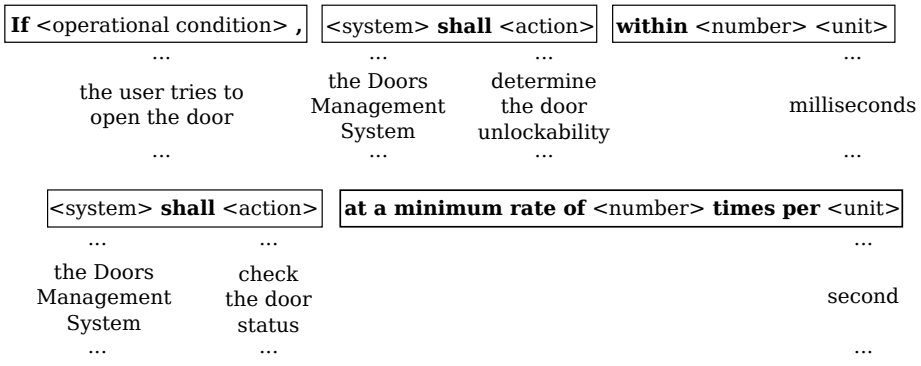

Fig. 3. Boilerplates and Suggestions

there are many grammatical ways to describe conditions, a subject-verb-object triple being only one of them. Therefore the tool does not only suggest those triples for conditions; instead all concepts, verb-object pairs and subject-verb-object triples are provided in order to use those phrases in conditions.

5 Evaluation

As mentioned before we evaluated the semantic guidance system with a domain ontology and a set of requirements from the Doors Management System.

5.1 Setting

The use case contains a set of 43 requirements specified using natural language text. Various types of requirements are included: functional, safety, performance, reliability, availability and cost. Each requirement was reformulated into a boilerplate requirement using DODT. The semantic guidance system was used to assist in filling the boilerplate attributes.

The domain ontology for DMS was specifically developed for usage with the boilerplates tool. The data for the DMS ontology was initially provided by EADS and was then completed by the authors. Table 4 lists the number of concepts, relations and axioms of the DMS ontology.

Figure 4 shows the graphical user interface of the tool. At the top of the interface boilerplates can be selected. The center shows the currently selected boilerplates and text boxes for the attribute values of the requirements. The list of phrases below the text boxes are the suggestions provided by the semantic guidance system. Typing in the text boxes filters the list of suggestions. The tool shows the textual definitions of the concepts as tooltips. Selecting a list entry will add the text to the corresponding text box. The bottom of the interface lists all requirements. Expressions that refer to entities from the domain ontology are

underlined with green lines; fixed syntax elements of boilerplates with black. If nouns missing from the domain ontology were to be seen, they would be highlighted with the color red.

Table 5 present statistics about the suggestions produced by the guidance system for the DMS ontology.

Table 4. Ontology Measurements

Entity	Count
Concepts	107
Relations	70
Axioms	123
SubClass	108
to Concepts	15
to Attributes	93
Equivalence	15

Table 5. Guidance Suggestions

Type	Count
Concept	212
Verb-Object	69
Subject-Verb-Object	101
Total	382

Table 6. Evaluation Results

Item	Count
Requirements	43
Boilerplates	21
Attributes	120
Complete suggestions	36
Partial suggestions	39

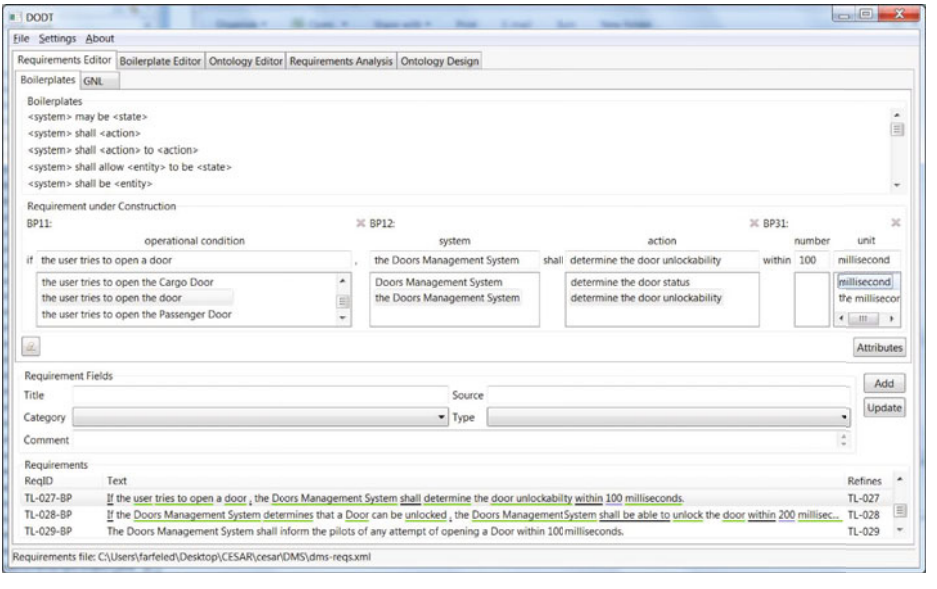

Fig. 4. DODT Screenshot

5.2 Results

Table 6 lists the major results of the evaluation. For 43 requirements, we used 21 different boilerplates. The boilerplate which was used most often (16 times) is ⟨*system*⟩ `shall` ⟨*action*⟩. The 43 boilerplate requirements have a total of 120 attributes. For 36 attributes out of 120 (30%) the semantic guidance system was able to suggest the entire attribute value without any need for a manual change. For another 59 attributes (57.5%) the guidance could suggest at least parts of the attribute value. This leaves 25 attribute values (12.5%) for that the guidance was no help. For partial matches, these are some of the reasons the attribute values had to be modified:

- A different determiner is used than the suggested "the", e.g., "a", "each" or "all".
- The plural is used instead of singular.
- A combination of two or more suggestions is used.
- A subordinate clause is added, e.g., "each door that could be a hazard if it unlatches".

Reasons for no guidance are these:

- Numbers for the ⟨*number*⟩ attribute cannot be suggested.
- Words are used that do not exist in the domain ontology.

Future work will include setting up an evaluation to compare the elicitation time with and without the semantic guidance system. However, due to the high percentage where the guidance was able to help (>85%) we are confident that efficiency improved, even though the presentation of explicit numbers has to be postponed to future work.

We also hope to improve the quality of requirements using the tool. We did a qualitative comparison of the original DMS requirements and the boilerplate requirements. These are our findings:

- Boilerplate requirements encourage using the active voice. In our evaluation the original requirement "Information concerning the door status shall be sent from the Doors Management System to ground station..." was turned into "The Doors Management System shall send information concerning the door status to ground station..." 8 requirements were improved in this way. In some cases missing subjects were added.
- Requirements like "There shall be..." and "It shall not be possible to..." were changed into "The *subject* shall have" and "The *subject* shall not allow...". Such changes make it obvious what part of the system is responsible to fulfill the requirement. To determine the right value for *subject* the original stakeholders should be asked for clarification. Due to timing constraints this was not possible and plausible values were inserted by the authors.
- During the requirements transformation we found that the original requirements used different expressions for seemingly identical things, e.g., "provision to prevent pressurization" and "pressure prevention means" or "airplane"

and "aircraft". Such synonyms are either stored as an equivalence axiom in the domain ontology or, preferably, stakeholders agree upon the usage of one term.
- Using boilerplates improved the overall consistency of the requirements. The original requirements set contains a mixture of "must" and "shall". While using one or the other is probably a matter of taste, one form should be picked and used consistently.
- The guidance system corrected a few typographic errors in the original requirements, e.g., "miliseconds".

We found the tool to be easy to use and user-friendly. This sentiment is shared by the partners in the CESAR project who are also currently evaluating the tool.

6 Conclusion and Future Work

Requirements should be made as consistent, correct and complete as possible to prevent detecting and correcting errors in later design phases. With respect to the three points raised in the introduction about requirements engineering, this work intends to be the foundation for further analyses, e.g., by facilitating requirements categorization with ontology knowledge.

We presented a tool for the elicitation of boilerplate requirements that includes a semantic guidance system which suggests concept names and phrases that were built from relations and axioms of the domain ontology. The tool managed to provide useful suggestions in the majority of the cases (>85%). We realize that the tool needs to be evaluated in a larger context, i.e., more requirements and a larger ontology. We will address this in our future research work.

The selection of suitable boilerplates for a requirement is not always trivial and requires a bit of experience with the boilerplates method. Thus a feature we want to explore and possibly add to the tool is the semi-automatic conversion of natural language requirements into boilerplate requirements.

While creating the DMS ontology, we found there are concepts for which the suggestions provided by the semantic guidance system really help but which are not specific to the problem domain. The most prominent example are measurement units like "second" or "kg". So what we want to do is to collect such entities into a separate ontology and to extend the tool to be able manage several ontologies. This could be handled by adding OWL import declarations to the "main" domain ontology. Such domain-independent ontologies can then be easily reused for guidance in other domains.

Acknowledgments

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.

References

1. IEEE Recommended Practice for Software Requirements Specifications. IEEE Std 830 (1998)
2. OWL 2 Web Ontology Language Direct Semantics. Tech. rep., W3C (2009), http://www.w3.org/TR/2009/REC-owl2-direct-semantics-20091027/
3. Cobleigh, R., Avrunin, G., Clarke, L.: User Guidance for Creating Precise and Accessible Property Specifications. In: 14th International Symposium on Foundations of Software Engineering, pp. 208–218. ACM, New York (2006)
4. Denger, C., Berry, D., Kamsties, E.: Higher Quality Requirements Specifications through Natural Language Patterns. In: 2003 IEEE International Conference on Software - Science, Technology and Engineering, pp. 80–90. IEEE, Los Alamitos (2003)
5. Egyed, A., Grunbacher, P.: Identifying Requirements Conflicts and Cooperation: How Quality Attributes and Automated Traceability Can Help. IEEE Software 21(6), 50–58 (2004)
6. Elazhary, H.H.: REAS: An Interactive Semi-Automated System for Software Requirements Elicitation Assistance. IJEST 2(5), 957–961 (2010)
7. Gotel, O., Finkelstein, C.: An Analysis of the Requirements Traceability Problem. In: 1st International Conference on Requirements Engineering, pp. 94–101 (1994)
8. Gottesdiener, E.: Requirements by Collaboration: Workshops for Defining Needs. Addison-Wesley, Reading (2002)
9. Hull, E., Jackson, K., Dick, J.: Requirements Engineering. Springer, Heidelberg (2005)
10. Ibrahim, N., Kadir, W., Deris, S.: Propagating Requirement Change into Software High Level Designs towards Resilient Software Evolution. In: 16th Asia-Pacific Software Engineering Conference, pp. 347–354. IEEE, Los Alamitos (2009)
11. Jackson, J.: A Keyphrase Based Traceability Scheme. IEEE Colloquium on Tools and Techniques for Maintaining Traceability During Design, 2/1–2/4 (1991)
12. Kaindl, H.: The Missing Link in Requirements Engineering. Software Engineering Notes 18, 30–39 (1993)
13. Kaiya, H., Saeki, M.: Ontology Based Requirements Analysis: Lightweight Semantic Processing Approach. In: 5th Int. Conf. on Quality Software, pp. 223–230 (2005)
14. Kitamura, M., Hasegawa, R., Kaiya, H., Saeki, M.: A Supporting Tool for Requirements Elicitation Using a Domain Ontology. Software and Data Technologies, 128–140 (2009)
15. Kotonya, G., Sommerville, I.: Requirements Engineering. John Wiley & Sons, Chichester (1998)
16. Matsuo, Y., Ogasawara, K., Ohnishi, A.: Automatic Transformation of Organization of Software Requirements Specifications. In: 4th International Conference on Research Challenges in Information Science, pp. 269–278. IEEE, Los Alamitos (2010)
17. Omoronyia, I., Sindre, G., Stålhane, T., Biffl, S., Moser, T., Sunindyo, W.: A Domain Ontology Building Process for Guiding Requirements Elicitation. In: 16th REFSQ, pp. 188–202 (2010)
18. Pedrinaci, C., Domingue, J., Alves de Medeiros, A.K.: A Core Ontology for Business Process Analysis. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) ESWC 2008. LNCS, vol. 5021, pp. 49–64. Springer, Heidelberg (2008)
19. Rupp, C.: Requirements-Engineering und -Management. Hanser (2002)
20. Ståhane, T., Omoronyia, I., Reichenbach, F.: Ontology-Guided Requirements and Safety Analysis. In: 6th International Conference on Safety of Industrial Automated Systems (2010)
21. Watkins, R., Neal, M.: Why and How of Requirements Tracing. IEEE Software 11(4), 104–106 (1994)