Towards Ontology-driven Requirements Engineering

Katja Siegemund[1], Edward J. Thomas[2], Yuting Zhao[2], Jeff Pan[2], and Uwe Assmann[1]

[1] Technische Universität Dresden, Germany
[2] University of Aberdeen, UK

Abstract. Requirements Engineering has a huge impact on success or failure of a software project. However, the acquisition, specification and evolution of goals and requirements from different stakeholders or sources often leads to incomplete, ambiguous, and faulty requirements. Therefore, the ability to detect and repair inconsistent and incomplete requirements is crucial to the successful modelling of requirements specifications. Ontologies provide the expressivity to sufficiently capture requirements and reasoning tasks can be performed to check the consistency and completeness of the requirements model. This paper presents a Meta Model for ontology-driven goal-oriented Requirements Engineering. Ontology consistency checking and rule driven completeness tests are combined to measure the validity and coverage of the evolving requirements model. Experiences from a first evaluation are described.

Key words: Requirements Engineering, Ontologies, Reasoning

1 Introduction

Usually, Requirements Engineering (RE)—i.e., the identification, specification and documentation of requirements—is the first task of a software development process. Results are documented in a Requirement Specification often treated as a contract between customer and developer. The importance of RE was already identified in several studies (e.g. [1]). Thus, an improved RE contributes to safer and better-quality software, safes time and money and also decreases the risk of overran budgets and project failures.

An important challenge for requirements engineering is to cope with inconsistencies or incompleteness in requirements specifications. Such inconsistencies result from the acquisition, specification, and evolution of goals and requirements from multiple stakeholders and sources [2]. It is frequently the case that changes of requirements have a particularly significant impact on the consistency of specifications. In order to regain consistency, requirements are removed from the specification which often leads to incompleteness. Zowghi et. al. ([3]) describes this vicious circle as a causal relationship between consistency, completeness and correctness. From a formal point of view, correctness is usually meant to be the combination of consistency and completeness. Therefore, the

ability to detect and repair inconsistent and incomplete requirements is crucial to the successful development of requirements specifications and will lead to a more correct RSB.

The paper is structured as follows: after some background information on RE in Section 2, we briefly sketch our solution approach in Section 3. In Section 4 we introduce Goal-Oriented Requirements Engineering and exemplify a way towards ontology-driven goal-oriented RE. Section 5 describes the implementation of the Requirements Ontology and the realisation of the completeness and consistency checks. First evaluation results are presented in Section 6. Finally, the paper concludes with an outlook and future work in Section 7.

2 Technical Background

The following sections give a brief overview about the main problems of RE addressed in this approach.

Completeness. Good requirements are meant to be consistent, necessary, unambiguous, and verifiable [4,5]. Additionally, they should be complete [6]. Missing or incomplete requirements propagate through the entire system development and lead to incomplete or incorrect designs, architectures, implementations and tests. Today, it is reasonably well known that missing or incomplete requirements lead to faulty software designs, implementations and tests resulting in software of improper quality or safety risks. Project cost and time overruns due to missing requirements and underestimated budgets may even cause a project to be aborted. Davis in [7] states completeness to be the most difficult of these (above mentioned) specification attributes to define and incompleteness the most difficult violation to detect. Complete metadata for requirements, that is data about that requirement rather than data listed in the requirement [6]), ensure completeness. Although some approaches exist that aim to ensure complete requirements (e.g. [8]), up to now there is no absolute way to determine the completeness of requirements in advance. It is reasonably well known, that requirement [specifications] will never be totally complete, finished or finalized as long as the system is in service and must evolve [6]. This may be a reason for today tools not to address this field. However, it is at least possible to improve and validate the completeness of information about these requirements (metadata). Though current RE tools provide means for capturing requirements, they fail in providing sufficient support for metadata about requirements and leave it to the requirements engineer to define them. Another shortcoming of RE tools is the lack of tests for completeness, that is, checking whether all important metadata are available. This way, the requirement engineer would detect missing but relevant information easily.

Consistency. The vicious circle of completeness, consistency and correctness shows the importance of a consistent Requirement Specification in order to improve its correctness. Nuseibeh and Russo in [9] argue that detecting and re-

pairing of inconsistent requirements is crucial to a successful development of requirement specifications. Inconsistent specifications restrain deriving useful information. Following [9] and [3] the reasons for inconsistent requirements are changes of requirement during software development. Various approaches have been proposed to handle inconsistent goals and requirements during Requirements Engineering. Most of them use formal logics to detect and handle such inconsistencies. Yang et. al. separate consistency checking in ontology-based requirements elicitation methods into two steps: (1) verifying the domain knowledge for consistency and (2) checking whether the requirements model keeps consistent under the restriction of a consistent domain model [10].

Nuseibeh and Russo provide a formal technique for detecting and repairing inconsistencies, based on a specific type of reasoning, called abduction. If a particular consistency rule is violated, their proposed abductive reasoning technique identifies (evolutionary) changes to perform on the specification, such that a particular consistency rule is no longer violated [9].

Other approaches treat the detection of conflicts between requirements often only regarding refinement relationships or conceptual overlapping (e.g. [11]). Moreover, most techniques consider binary conflicts only, that is, conflicts among two requirements. Although these approaches and solutions exist, up to now hardly any RE tool makes use of them.
Another problem are the numerous and often crosscutting[3] interrelationships among requirements that are not considered at all. Thus, there is no systematic support for detecting and resolving this kind of inconsistencies. One notable exception is [2]. To address these problems, our approach aims to detect such inconsistencies between various RE objects and provide solutions for repairing them.

Deficiencies of RE Methods. Analysis of available RE Methods and approaches exposed a number of problems and shortcomings described above. Additionally, some have also been reported in other approaches. The deficiencies of current RE methods and tools can be summarized as follows:

- Requirement knowledge is not sufficiently covered. Intentions, risks, obstacles and decisions are not documented during RE and thus, are not available at later stages during software development.
- Relationships among requirements are inadequately captured and are often limited to binary relations between requirements instead of defining which kind of relation is meant (e.g. excluding, alternative, generalization).
- Requirement problems (e.g. conflicts, unstated information) are detected too late or not all.
- Completeness and consistency are not verified.
- Models for RE need richer and higher-level abstractions [13].

[3] Requirements are scattered through the whole system. The may appear in Use-Cases or Test-Cases and be a part of metrics. Additionally, crosscutting requirements provide a description of the *overlap* between requirements - the first step to managing any inconsistency that arises at such overlap [12].

3 Solution Approach

Ontologies provide a formal representation of knowledge and the relationships between concepts. Thus, they are well suited to tackle the above mentioned problems and shortcomings of Requirements Engineering. The Web Ontology Language in particular, which is standardised by W3C, supports the open world assumption, allowing incomplete knowledge to be reasoned over. This characteristic of the language makes it suitable for use in Requirements Engineering. The separation of the concepts of consistency and completeness mean that an evolving requirements model can be checked for consistency, without the incompleteness of the model causing a problem. In contrast, closed world systems such as relational databases make no distinction between incomplete and inconsistent knowledge; any fact not known is assumed to be false.

The idea presented in this paper is to use an ontology for structuring the concepts, requirements and relationships captured during requirements elicitation. Here, goals play a crucial role for reaching a high level of completeness (see Section 4.1). This domain independent ontology can be instantiated with the requirement artefacts (goals, requirements, obstacles, etc.) for a particular project. This forms the Requirement Specification Base. TBox Reasoning techniques are applied for consistency checking. To validate the consistency, and subsequently the completeness of requirements we propose query answering techniques to detect incompleteness based on predefined completeness rules. The novel of this approach is twofold: (1) the Ontology-based Requirements Meta Model with a huge set of relevant meta data and requirement relationships and (2) the consistency and completeness rules for validating the Requirement Specification Base and providing concrete suggestions on how to solve causes for inconsistency and incompleteness. This approach allows for requirements reasoning based on formal semantics and thus, aims to resolve many of the shortcomings observed in other approaches.

4 Related Work

This section describes the main concepts of goal-oriented RE approaches, related work and sketches the way to our solution towards ontology-driven goal-oriented RE.

Conventional Requirements Engineering in its early times concentrated on what the system should do and how it should do it. This lead to fairly low-level requirements on data, operations, etc. [14]. *Goals* have become more and more popular for the early phases of Requirements Engieering (e.g. [15], [13], [16]). Yue showed in [17] that goals in requirements models provide a criterion for requirements completeness.

Goal-oriented Requirements Engineering must not be understood as a different or new form of Requirements Engineering. It is rather a supplement of the RE process, in the meanwhile self widespread. Goal-driven concepts have been adopted in many Requirements Engineering frameworks in numerous domain

and stages of Requirements Engineering. This indicates that goals are a core concept for RE in general. Thus, we decided to adapt to this methodology and support not only a general RE method, but also a goal-oriented RE. The next section explains the main advantages and concepts of goal-oriented Requirements Engineering. Subsequently, we describe our approach to enable ontology-driven goal-oriented requirements engineering.

4.1 Goal-Oriented Requirements Engineering (GORE)

As already manifested in its name, GORE puts much emphasize on *goals* which will be used to identify, describe and correlate requirements. Lamsweerde defines goals as "declarative statements of intent to be achieved by the system under consideration" [18]. Goals are formulated in terms of prescriptive assertions (as opposed to descriptive ones) [19]; they may refer to functional or non-functional properties and range from high-level concerns to lower-level ones [20]. Goals explore *why* certain requirements are necessary for the system to be. Thus, they capture stable information and provide means to separate stable from unstable information which enables a better reuse. There are several reasons to extend RE with the identification and formulation of goals. Lamsweerde describes in ([20]) the importance of goals for RE. Some of the main benefits of goal-orientation are:

- Goals provide a meaningful criterion for sufficient completeness of a requirement specification. "A requirement specification can be denoted as complete with respect to a set of goals if all the goals can be proved to be achieved from the specification and the properties known about the domain considered." [17]
- Specification of pertinent requirements, that are requirements that serve at least one of the identified goals [17].
- Goals may be satisfied by different alternative requirements. Thus, relationships between goals and requirements can help to choose the best one.
- Concrete requirements may change over time whereas goals pertain stable. Thus, stable information can be separated from volatile one.
- Goals drive the identification of requirements.

4.2 Enabling Ontology-Driven Requirements Engineering

Knowledge-driven techniques seem promising to overcome some of the shortcomings of current RE-practices. Kossmann et. al. defines Knowledge-driven Requirements Engineering when Requirements Engineering is guided not only by a process but as well by knowledge about the process and the problem domain [21].

In order to use knowledge-driven techniques, it is necessary to apply knowledge repositories that can be easily updated and utilised. Furthermore, inferencing and decision support should be possible to apply on such a repository.

Ontologies are useful for representing and interrelating various knowledge. Since RE involves knowledge capturing and analysis, there is a clear synergy between the ontological modelling of a domain and the modelling that a Requirements Engineer will perform during the requirements process [22]. Due to this overlap, numerous works dating back have addressed the use of ontologies in RE, e.g. [23], [24]. All formalisms for RE need a particular conceptualisation, and almost all of them are reducible to first order logic [22]. Thus, they have much in common with ontologies that are constructed by using a formal language.
Since the semantic web emerged, there has been a renewed interest in ontologies. There is an increasing amount of research devoted to utilising semantic web technologies in RE (e.g. [25], [21]) and software engineering in general.

However, most of them concentrate only on specific requirement artefacts, e.g. goals, requirements or use-cases and do not support reasoning over relations between all concepts. Sancho et. al. ([26]) describe a method for ontology-aided performance engineering. They propose the structure of an ontological database to relate Performance Engineering issues among themselves and with other non-functional requirements. The NFR/i* Framework first proposed in [27] focuses on qualitative reasoning for goal contribution and evaluation of alternative goal refinements. The KAOS Framework ([28]) concentrates mainly on goal satisfaction. Lamsweerde describes in [29] an interesting method for reasoning about alternative requirements and inspired our work regarding some of the requirement relationships and ontology properties. Lamsweerde introduces influence relations among requirements, stating whether a requirement has a positive or negative influence on another. By applying formal reasoning techniques it becomes possible to identify alternative requirements for those with a negative impact.
Other approaches aim to address a broader field of Requirements Engineering. The OntoRem approach ([21]) describes the development of a meta model for general RE concepts. Lee et. al. present an Ontology-based active Requirements Engineering Framework (Onto-ActRE) based on GORE [30]. While relationships of requirements with other concepts, such as goals and risks, are supported, they do not consider the variety of relations between requirements.

5 Ontology-Driven Requirements Engineering

This section describes the main architecture decisions, concepts and relationships of the Requirements Ontology (the Requirements Meta Model), the rules for verifying the consistency and completeness of requirements metadata.

5.1 Requirements Ontology

In Figure 5.1 we present the architecture of ODRE. A so-called Requirement Meta Model is generated from the knowledge of the requirements analysis and builds the TBox of the Requirements Ontology. It formalizes the RE concepts, as well as relationships between requirement artefacts. This domain independent

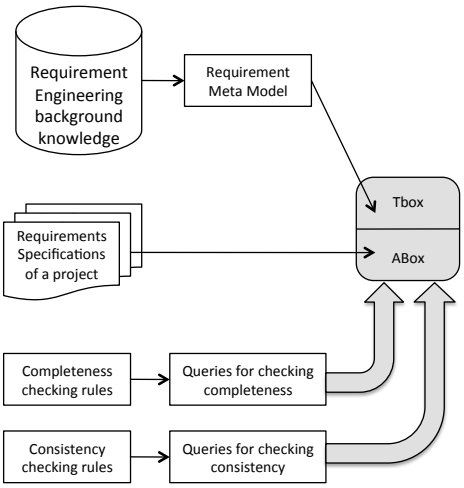

Fig. 1. The architecture of ODRE (Ontology-Driven Requirement Engineering)

ontology TBox can be instantiated with the requirement artefacts for a particular project. This way, the ABox is generated from the results of requirement analysis of an application project. It builds the requirement Specification Base. Once the Requirement Elicitation has been completed and the Requirements Ontology is filled with all information, we execute consistency and completeness queries, generated from consistency checking rules (cf. Section 5.2) and completeness checking rules (cf. Section 5.3), respectively. These rules are used to enable the validation of the Requirements Specification Base of the project. The following sections present the rules for completeness and consistency checking and exemplarily explain how they have been implemented so far.

The Requirements Meta Model was mainly inspired by a UML Meta Model for GORE presented in [31]. Especially the requirement relationships have been adapted, comprehensively extended and transferred to ontological concepts. Furthermore, we adapted the categorization of non-functional requirements published by ISO/IEC 9126 standard to comply to this well-established and acknowledged categorization.
In our approach, a huge number of GORE concepts are modelled as OWL Classes, while relations, relationships and most of the meta data are modelled as object properties. The Ontology consists of 102 classes, 39 object properties and 6 data properties, implemented with Protege 4.0. The ontology is available at http://purl.org/ro/ont.

The class Artifact is the most fundamental concept of this ontology. It subsumes the concepts Problem and RequirementArtifact. RequirementArtifact encapsulates the concepts Challenge, Goal, Story and Requirement.

As already described in section 5 requirement artifacts have various relationships among each other. These relationships are mainly modelled as object prop-

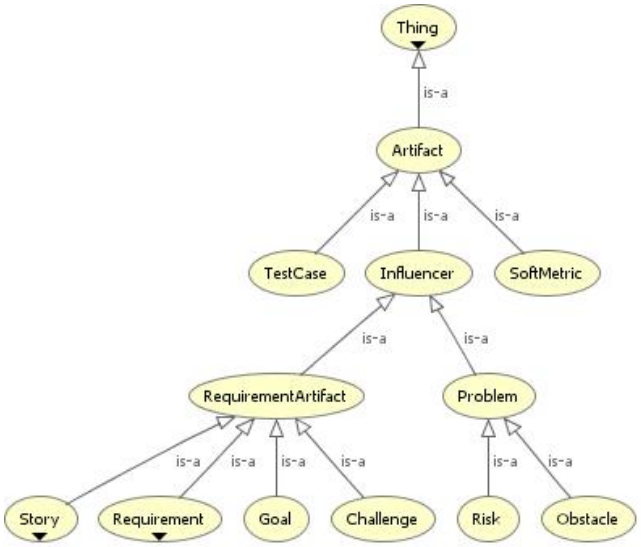

Fig. 2. Extract of the main concepts in the Requirements Ontology

erties. For example a requirement may negatively contribute to a goal, others positively. To capture this valuable information, we provide the object properties isNegativeContribution to and isPositiveContributionTo between goals and requirements. This data is used for later queries and supports the user in recognizing the quality of his decisions regarding goal satisfaction. Functional and non-functional requirements are identified by means of Scenarios and Use Cases. Use Cases and Scenarios can be related to requirements via the object property isRelatedTo. The class Attribute contains different attributes all requirements can be associated with, e.g. Priority and Cost. Additionally, this class has the subclass State. The requirement status will be automatically assigned after verification. Thus, the state can become Checked, Invalid, etc. We put much emphasis on the modelling of relationships between requirements. Therefore, we provide eight object properties to capture alternative, coexistent, excluding and conflicting requirements and to denote generalizations, specialisations and refinements of requirements. Decisions in the elicitation process may possibly derive in conflicts. These conflicts will have an impact on further decisions regarding requirements, Use Cases and the final design of the software. Thus, it is important to document such conflicts. This is realised by the OWL property "isInConflictWith". Additionally, conflicts help in identifying inconsistencies between Project Work and Requirements. Finally, the class Refinement and RefinementReason as well as three associated object properties are used to track refinement changes (source and target) of goals and requirements. Furthermore, enabling the user to describe the reason for such a refinement eases further decisions and enables the traceability of them.

5.2 Rules for Consistency Checking

We base our notion of consistency and completeness on the violation of consistency and completeness rules. Therefore, we predefined two set of rules in natural language and realised the appropriate checks by querying techniques. We will have a brief look on the concrete realisation of these rules separately for consistency and completeness.

The consistency rules check for valid relations between the requirement artifacts, e.g. whether a goal is indeed connected with at least one requirement. Moreover, requirement relationships are verified in order to prevent concurring requirement, conflicts, and so on. We predefined six consistency rules containing the rule that has to be met, the output if it is violated and possible solutions.

To realise these rules in the requirements ontology, we have encoded them as a set of DL axioms which guarantee that if these rules are fulfilled, the ontology will be logically consistent. Any deviation from the rules will result in an inconsistent ontology. One such rule is modelled in as follows:

$ChosenRequirement \sqsubseteq \neg\exists isExclusionOf.ChosenRequirement$
This means that no requirement that was specified to exclude any other requirement can be included in one and the same requirement subset.

Other consistency rules are specified in a similar way. The mapping between the axioms in the ontology and the rules is given in the ontology using annotation properties on the particular axioms. If one of these axioms is violated, we can use an explanation service to find which axioms caused this violation. The reasoner can determine the axioms in the inconsistency justification that represent a consistency rule. The additional axioms are used to find which individuals had caused the error. This allows us to explicitly mention the invalid requirements in the error message that is displayed to the user. For inconsistencies we distinguish errors and warnings. Errors must be resolved by the requirements engineer and warnings should be resolved. A template is used to define generic error messages for each consistency rule. For inconsistencies we distinguish errors and warnings. Errors must be resolved by the requirements engineer and warnings should be resolved. A template is used to define generic error messages for each consistency rule and suggestions to resolve found inconsistencies.

Example We assume the following individuals and relationships as an extract of the ontology resulting from the Requirement Analysis:

$isExclusionOf(Functional\_Requirement5, Functional\_Requirement7)$
$ChosenRequirement(Functional\_Requirement5)$
$ChosenRequirement(Functional\_Requirement7)$

This set of axioms results in an inconsistency with regard to the previously defined consistency rule. If one requirement excludes another, they cannot both be included in the set of requirements chosen for the final model.

We use the approach described above to derive errors and warnings displayed to the user. To support the requirements engineer in error elimination, the system suggests a list of possible solutions. The concrete error message and the suggestion to resolve our inconsistency is shown below:

> **Excluding requirements must not be included in one chosen requirement subset.**
>
> Error.
> "The following requirements exclude others: Functional Requirement5."
> "Please choose one of the following options:"
>
> Suggestion.
> Exclude the following requirements from the chosen requirement set: Functional Requirement5
> Find alternatives for: Functional_Requirement5 or
> Revise the requirement relationships of (Functional_Requirement5, Functional_Requirement7).

After the Requirements Engineer has done the appropriate actions to remove errors and warnings, the requirement subset has to be checked for consistency once more. In case the set is now consistent, all requirements will be automatically asserted "Checked" and "Valid" in the Ontology. If not, another revision and check becomes necessary until the the Requirement Specification Base is consistent or the analyst refuses to do any further action. In the latter case, all the requirements will be asserted "Checked" as well, but all invalid requirements will be asserted "Invalid". This way it is possible to display all invalid requirements and change them manually or check for options without being forced to run too many consistency checks. However, it is always possible to have another consistency check.

5.3 Rules for Completeness Checking

We defined 43 Completeness Rules to check the completeness of the requirement artifacts, e.g. whether goals, requirements, use-case descriptions and so on have been specified and if all required attributes and relations have been specified. According to Firesmith ([6]) the most important metadata to capture is:

- Project-Unique Identifier (PUID)
- Prioritization
- Rationale
- Source
- Status (may include more than one kind of status)
- Verification Method

These metadata were integrated in our completeness rule set and extended by further completeness criteria regarding the existence of requirement artifacts and their relations among each other. An extract of these rules is shown below:

Every Functional Requirement (FR) must define whether it is mandatory or optional.
IF FR is not mandatory AND not optional
THEN print error: "You did not specify whether the following FRs are mandatory or optional"
"Please specify whether the following FRs are mandatory or optional: [FR_n]."

Every FR must specify AT LEAST ONE property hasRequirementRelationship.
IF FR has NOT specified a property "hasRequirementRelationship"
THEN print error: "You did not specify any requirement relationship for the Functional Requirements [FR_n].
Please check the relationships for the following Functional Requirements: [FR_n]."

Example We assume an extract of the following individuals and relationships:

$isRelatedTo(Goal2, UseCase7)$
$NonFunctionalRequirement(NonFunctionalRequirement1)$
$isOptional(NonFunctionalRequirement1, true)$
$FunctionalRequirement(FunctionalRequirement1)$

This raises the following error messages and suggestions for our completeness rules:

Every Requirement must state whether it is mandatory or optional.
Error.
"You did not specify whether the following FR are mandatory or optional: FunctionalRequirement1."
"Please specify this attribute for the FR: FunctionalRequirement1."

Every FR must specify AT LEAST ONE requirement relationship.
Warning.
"You did not specify any requirement relationship for the following FR: FunctionalRequirement1."
"Please check whether there exists any relationship to another requirement for the FR FunctionalRequirement1."

The completeness rules have been realised as SPARQL queries. If the patterns encoded in the rule match the entailed ontology, then the particular completeness rule being tested failed, and the requirements model is deemed incomplete. The results of the query identify those parts of the ontology which are incomplete. Since we are looking for incomplete requirements, we had to allow the query to work in a closed world environment. This requires the use of negation as failure in the query, which is available in the draft SPARQL 1.1 specification using the "NOT EXISTS" keyword and functionality. Generally, OWL uses the open world assumption (OWA), which states that every fact not known is undefined,

therefore it is difficult to find incomplete requirements, as these requirements are treated as present but unknown by an OWL reasoner. The SPARQL 1.1 specification supports querying under OWL, and it supports negation in queries, which is treated as negation as failure by the SPARQL specification. In this case, we used the SPARQL 1.1 support in the TrOWL reasoner (REF) to support our implementation. The implementation of these database style constraints is similar to that described by Sirin and Tao in [32]. Therefore, we use a combination of open world and closed world assumption. OWA allows us to check the logical consistency of incomplete requirements without being forced to have a complete set of metadata. Closed world reasoning using negation as failure is employed when the Requirements Engineer decides to check completeness during RE or he decides to complete RE and finalises the requirements.

Verification of the Requirement Specification Base

We implemented the ontology using Protege 4.0, and created the completeness constraints as queries in SPARQL, using the SPARQL 1.1 OWL profile and support for negation as failure against OWL knowledge bases. The reasoner and query engine used is the TrOWL tractable reasoner[4]. This is particularly well suited to this application, as once a consistent and complete set of requirements is loaded, it is possible to make a SPARQL endpoint available permanently to allow traceability links to be established between the artifacts in the requirements ontology and the rest of the software engineering life cycle. The TrOWL reasoner also supports explanation of consistency checks for OWL-DL ontologies, and supports justification of query answers. The additional functionality for rewriting the reasoning explanations and justifications into english has been implemented into the RELib package.

6 Evaluation

ODRE has primarily been evaluated within the MOST Project[5]. The evaluation criterion for the discussed ontology are that (i) all requirement artefacts should be possible to capture, (ii) all inconsistencies and incomplete metadata are detected and (iii) all requirement metadata are finally complete and consistent. During development, we have simulated its usage by instantiating it against few case studies of the project, i.e. semantic modeling of network physical devices ([33]) and validating Component-Based Implementations of Business Processes ([34]).

After instantiation, we evaluated our implementation by performing the completeness and consistency checks over this ontology. The evaluation was performed on a 2GHz dual core MacBook Pro. The evaluation was an excellent test of the ability of our approach to deal with incomplete and inconsistent specifications. The ontology was then stored in the Quill OWL2-QL repository using

[4] Available at http://trowl.eu

[5] Marrying Ontologies with Software Technology, http://www.most-project.eu/

Semantic Approximation [35] and the completeness checks were performed. The Case Study ontology failed 14 of the 37 tests included in the RELib test suite, and a total of 132 different individual problems were identified. Thus, the evaluation proved the detection of incomplete and inconsistent requirement artefacts, which was one of our main goals. The total time required to test the completeness constraints and generate the English explanations from the justifications provided by the reasoner was 808ms. The performance of the system is therefore good enough to be used as part of an ongoing cycle of testing and revision of the ontology. However, in order to prove the applicability of our Ontology-driven Requirements Engineering method and the impact of reasoning over requirement data, another and exceeded evaluation will be conducted in the near future. In this evaluation we will not only test the quality of the developed requirement artefacts, but also the effort and time needed for specification until a certain predefined level of completeness and consistency is reached.

7 Outlook and Conclusion

In this paper we have presented an approach to capturing and validating a set of software project requirements using OWL ontologies and reasoning technology. The approach combines the open world reasoning capabilities of OWL to allow the consistency of incomplete requirements to be validated during the requirements engineering process, and extends OWL with closed world constraints to further check the completeness of the requirements model against several well-established metrics such as ISO/IEC 9126.

We have implemented our approach and have performed an in-depth evaluation using an extant set of project requirements. This evaluation has shown that the approach is capable of dealing with a reasonably complex set of requirements from a real-world problem, and can quickly identify where these requirements are inconsistent or incomplete. The performance is such that it can be integrated into the requirements engineering workflow without becoming a burden on the requirements engineer.

Further work in this area concentrates on guidance of the requirements engineering process and traceability through other stages of the software development cycle. We developed a guidance ontology for our approach which is connected to the Requirements Ontology. This enables user guidance through the prescribed process of requirements engineering. This guidance includes not only a task list for the user but also incorporates the described consistency and completeness checks at the appropriate time as well as real-time display of detected errors and suggestions for resolving them. All these features have been integrated into a Eclipse Plugin and can be accessed through a yet rudimentary java user interface.

The completed requirements ontology can be integrated with other ontological models which cover other aspects of the software development process, links between the artefacts in the requirement ontology and those artefacts generated at later stages can therefore be queried later for traceability purposes.

References

1. Tracy Hall, Sarah Beecham, and Austen Rainer. Requirements Problems in Twelve Software Companies: An Empirical Analysis. *IEE Proceedings - Software*, 149(5):153–160, 2002.
2. Axel van Lamsweerde, Robert Darimont, and Emmanuel Letier. Managing conflicts in goal-driven requirements engineering. *IEEE Transactions on Software Engineering*, 24:908–926, 1998.
3. Didar Zowghi and Vincenzo Gervasi. The Three Cs of Requirements: Consistency, Completeness, and Correctness. In *Proceedings of 8th International Workshop on Requirements Engineering: Foundation for Software Quality, (REFSQ'02)*, 2002.
4. Donald Firesmith. Specifying Good Requirements. *Journal of Object Technology*, 2:77–87, 2003.
5. Ian Sommerville and Pete Sawyer. *Requirements Engineering: A Good Practices Guide*. John Wiley & Sons, 1997.
6. Donald Firesmith. Are Your Requirements Complete? *Journal of Object Technology*, 4(1):27–44, 2005.
7. Alan M. Davis. *Software Requirements: Analysis and Specification*. Prentice Hall Press, Upper Saddle River, NJ, USA, 2nd edition edition, 1993.
8. Ronald S. Carson. Requirements Completeness: A Deterministic Approach, 1995.
9. Bashar Nuseibeh and Alessandra Russo. Using Abduction to Evolve Inconsistent Requirements Specifications. In *The Use of Logical Abduction in Software Engineering 25*, 1999.
10. Yang Ying-ying, Li Zong-yon, and Wang Zhi-xue. Domain knowledge consistency checking for ontology-based requirement engineering. In *CSSE '08: Proceedings of the 2008 International Conference on Computer Science and Software Engineering*, pages 302–305, Washington, DC, USA, 2008. IEEE Computer Society.
11. Xuefeng Zhu. Inconsistency Measurement of Software Requirements Specifications: An Ontology-Based Approach. In *ICECCS '05: Proceedings of the 10th IEEE International Conference on Engineering of Complex Computer Systems*, pages 402–410, Washington, DC, USA, 2005. IEEE Computer Society.
12. Bashar Nuseibeh. Crosscutting Requirements. In *AOSD '04: Proceedings of the 3rd international conference on Aspect-oriented software development*, pages 3–4, New York, NY, USA, 2004. ACM.
13. John Mylopoulos, Lawrence Chung, and Eric Yu. From Object-oriented to Goal-oriented Requirements Analysis. *Communications of the ACM*, 42(1):31 – 37, 1999.
14. Alexei Lapouchnian. Goal-oriented Requirements Engineering: An Overview of the Current Research, 2005.
15. Annie I. Antón. Goal-Based Requirements Analysis. In *ICRE '96: Proceedings of the 2nd International Conference on Requirements Engineering (ICRE '96)*, page 136, Washington, DC, USA, 1996. IEEE Computer Society.
16. L. Liu and E. Yu. From requirements to architectural design - using goals and scenarios, 2001.
17. K. Yue. What does it mean to say that a specification is complete? *Proc. IWSSD-4, Fourth International Workshop on Software Specification and Design*, 1987.
18. Axel van Lamsweerde. Requirements Engineering in the year 00: A Research Perspective. In *International Conference on Software Engineering*, pages 5–19, 2000.
19. Pamela Zave and Michael Jackson. Four dark corners of requirements engineering. *ACM Transactions on Software Engineering and Methodology*, 6(1):1–30, 1997.

20. Axel van Lamsweerde. Goal-Oriented Requirements Engineering: A Guided Tour. In *RE '01: Proceedings of the Fifth IEEE International Symposium on Requirements Engineering*, page 249, Washington, DC, USA, 2001.
21. M. Kossmann, R. Wong, M. Odeh, and A. Gillies. Ontology-driven Requirements Engineering: Building the OntoREM Meta Model. In *Information and Communication Technologies: From Theory to Applications, 2008. ICTTA 2008. 3rd International Conference on*, pages 1 – 6, 2008.
22. Glen Dobson and Peter Sawyer. Revisiting ontology-based requirements engineering in the age of the semantic web. In *Dependable Requirements Engineering of Computerised Systems at NPPs*, 2006.
23. John Mylopoulos, Alex Borgida, Matthias Jarke, and Manolis Koubarakis. Telos: Representing Knowledge About Information Systems. *ACM Trans. Inf. Syst.*, 8:325–362, 1990.
24. Anne Dardenne, Axel van Lamsweerde, and Stephen Fickas. Goal-directed Requirements Acquisition. *Sci. Comput. Program.*, 20:3–50, 1993.
25. Haruhiko Kaiya and Motoshi Saeki. Ontology based requirements analysis: lightweight semantic processing approach. In *Proc. Fifth International Conference on Quality Software (QSIC 2005)*, 2005.
26. Pere P. Sancho, Carlos Juiz, Ramon Puigjaner, Lawrence Chung, and Nary Subramanian. An approach to ontology-aided performance engineering through NFR framework. In *WOSP '07: Proceedings of the 6th international workshop on Software and performance*, pages 125–128, New York, NY, USA, 2007. ACM Press.
27. J. Mylopoulos, L. Chung, and B. Nixon. Representing and using nonfunctional requirements: A process-oriented approach. *IEEE Trans. Softw. Eng*, 18(6):483–497, 1992.
28. R. Darimont, E. Delor, P. Massonet, and A. van Lamsweerde. Grail/kaos: an environment for goal-driven requirements engineering. In *Proceedings of the 19th international conference on Software engineering*, ICSE '97, pages 612–613, New York, NY, USA, 1997. ACM.
29. Axel van Lamsweerde. Reasoning about alternative requirements options. In Alexander Borgida, Vinay K. Chaudhri, Paolo Giorgini, and Eric S. K. Yu, editors, *Conceptual Modeling: Foundations and Applications*, volume 5600 of *Lecture Notes in Computer Science*, pages 380–397. Springer, 2009.
30. Seok Won Lee and Robin A. Gandhi. Ontology-based Active Requirements Engineering Framework. 2006.
31. Rodrigo Cerón, Juan C. Dueñas, Enrique Serrano, and Rafael Capilla. A metamodel for requirements engineering in system family context for software process improvement using cmmi. In *PROFES*, pages 173–188, 2005.
32. Evren Sirin and Jiao Tao. Towards integrity constraints in owl. In Rinke Hoekstra and Peter F. Patel-Schneider, editors, *OWLED*, volume 529 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.
33. Krzysztof Miksa, Marek Kasztelnik, Pawel Sabina, and Tobias Walter. Towards semantic modeling of network physical devices. In *MoDELS Workshops*, volume 6002 of *Lecture Notes in Computer Science*, pages 329–343, 2009.
34. Jens Lemcke, Andreas Friesen, and Tirdad Rahmani. Validating component-based implementations of business processes. In *Electronic Business Interoperability: Concepts, Opportunities and Challenges*, chapter 7, pages 124–151. IGI Global, 2011.
35. J. Z. Pan and E. Thomas. Approximating OWL-DL Ontologies. In *the Proc. of the 22nd National Conference on Artificial Intelligence (AAAI-07)*, pages 1434–1439, 2007.