Lessons Learned from an Application of Ontologies in Software Testing

He TAN [a], Vladimir TARASOV [a] and Anders ADLEMO [a]
[a] *School of Engineering, Jönköping University, Jönköping, Sweden*

Abstract. Testing of a software system is a resource-consuming activity that requires high-level expert knowledge. In previous work we proposed an ontology-based approach to alleviate this problem. In this paper we discuss the lessons learned from the implementation and application of the approach in a use case from the avionic industry. The lessons are related to the areas of ontology development, ontology evaluation, the OWL language and rule-based reasoning.

Keywords. application of ontologies, OWL, Prolog, lesson learned, test case generation, automated testing

1. Introduction

Manual software testing is made up of labor-intensive processes. Automated testing can significantly reduce the cost of software development and maintenance [1]. Despite successful achievements in automation of script execution and white-box testing, there is still a lack of automation of black-box testing of functional requirements. Because such tests are mostly created manually, which requires high-level human expertise, modern methods from the area of knowledge engineering are up to the challenge.

In our previous work [2,3,4] we have proposed and implemented an approach to automate software testing by modelling the testing body of knowledge with formal ontologies and reasoning with inference rules to generate test cases[1]. The experiment has demonstrated that the use of ontologies allows for automation of the full process of software testing, from the capture of domain knowledge in software requirements specifications (SRS) to the generation of software test cases. In this paper we discuss the lessons learned from the implementation and application of the approach. The lessons relate to the ontology development and evaluation, the use of OWL, and rule-based reasoning.

The rest of this paper is structured as follows. The automated testing process framework is presented in Section 2. Section 3 presents an implementation of the framework for a testing task in the avionics industry. We discuss the lessons learned from the project in Section 4. Section 5 describes related work. The conclusions of the study are given in Section 6.

[1]The study presented in this paper is part of the project Ontology-based Software Test Case Generation (OSTAG) that was financed by the Knowledge Foundation in Sweden, grant KKS-20140170.

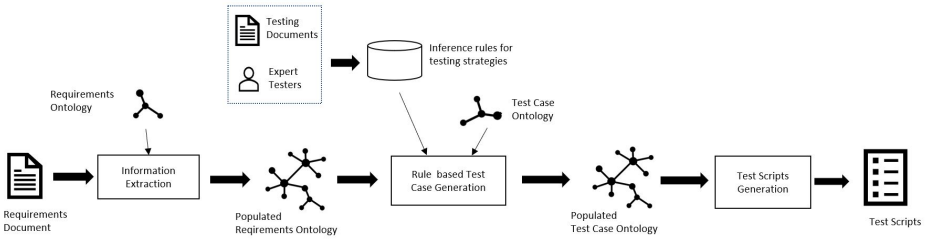

Figure 1. A framework for automation of testing process using ontologies

2. A Framework for Testing Process Automation Using Ontologies

A typical process of black-box testing of functional software requirements comprises two activities. During the first activity, software testers design test cases based on SRS and their own expertise from previous work on testing software systems. The second activity is to generate test scripts. Finally, the tests are carried out, either manually or using a test execution tool, based on the automated execution of test scripts. Fig. 1 presents a framework to automate such a testing process using ontologies. The framework was developed in our project, OSTAG.

Requirements are often described in well-structured or semi-structured textual documents. First, a requirements ontology is built, to represent the structure of the software requirements. With the help of the ontology, the requirements information is extracted from the text documents and then used to populate the ontology. The populated ontology serves as an input for the test case generator.

In situations where the testers' expertise is less structured, the information is acquired through interviews with experienced testers and examination of existing software test description (STD) documents. The acquired testing strategies are represented with inference rules that utilize the populated requirements ontology for checking conditions and querying data to generate test cases.

Furthermore, a test case ontology is provided to specify what should be contained in test cases and how each test case should be structured. The test case ontology is used in the test case generation step. The ontology is populated when test cases are generated. Finally, the populated test case ontology is employed to generate test scripts.

3. An Implementation of the Framework

In this section we demonstrate an implementation of the framework. The implementation is to support testing of the components of an embedded sub-system within an avionic system. The case data were provided by the fighter aircraft developer, Saab Avionics. In the avionics industry, many of the systems are required to be highly safety-critical. For these systems, the software development process must comply with several industry standards, like DO178B. The requirements of the entire system, or units making up the system, must be analyzed, specified, and validated before initiating the design and implementation phases. The software test cases have to be manually inspected as well. The requirements and test cases that were used in the framework implementation were provided in text documents and the results were manually validated by avionic industry domain experts.

3.1. Requirements Ontology and Test Case Ontology

The *requirements ontology* (see Fig. 2) was built by ontology experts based on the structure of the textual requirements specification documents provided by Saab Avionics. Each requirement has a unique ID and consists of at least:

1. Requirement parameters, which are inputs of a requirement,
2. Requirement conditions,
3. Results, which are usually outputs of a requirement and exception messages.

Some requirements require the system to take actions. More details about the ontology can be found in [2].

The *test case ontology* (see Fig. 3) was also built by ontology experts based on the structure of test case descriptions created by the industrial partner. Each test case has a unique ID, addresses one requirement, and has prerequisite conditions. There is a list of ordered steps needed to be followed in each test case. Each step consists of input, procedure and expected result.

The ontologies were built using the ontology language OWL, and developed in the tool Protégé. The classes Test_Input, Test_Procedure and Test_Results are defined as the subclasses of OWLList [5], which is a representation of a sequence in OWL-DL.

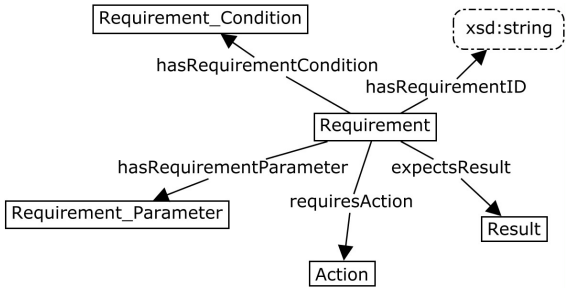

Figure 2. The key entities in the *requirements* ontology

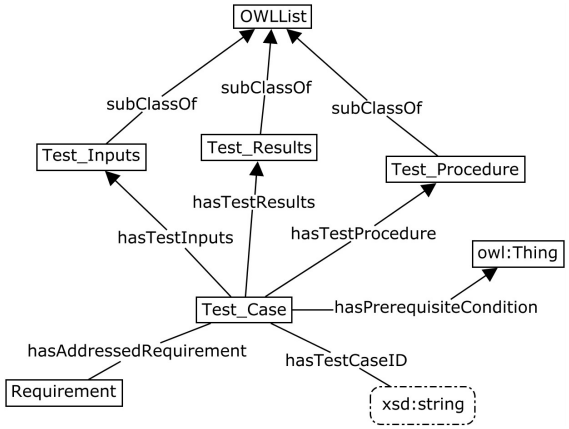

Figure 3. The key elements in the *test case* ontology

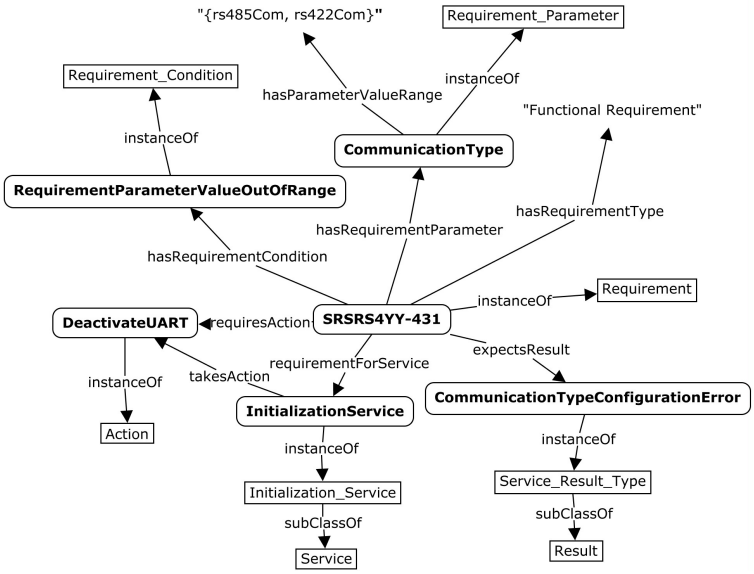

Figure 4. Ontology fragment of the SRSRS4YY-431 requirement specification

3.2. Population of the Requirements Ontology

The populated ontology contains 147 individuals in total in the experiment. Fig. 4 shows a fragment of the populated ontology for one particular functional requirement, SRSRS4YY-431. The requirement states that if the communication type is out of its valid range, the initialization service shall deactivate the UART (Universal Asynchronous Receiver/Transmitter), and return the result "comTypeCfgError". In Fig. 4, the rectangles represent the concepts of the ontology; the rounded rectangles represent the individuals (instances); and the dashed rectangles provide the data values of datatype property for individuals.

3.3. Approach to Test Case Generation based on Inference Rules

Test cases are generated through deriving information from the populated requirements ontology with the help of inference rules. A necessary task to solve to derive test cases from the requirements ontology is to represent testers' expertise on how they use requirements to create test cases. Such expertise embodies inherent strategies for test case creation, knowledge that can be expressed in the form of heuristics represented as if-then rules. This kind of knowledge is acquired from two sources. First, literature on software testing contains some general guidelines, e.g. boundary value testing. These general test strategies apply to all domains. Second, expert testers were interviewed to capture their expertise that is specific to particular types of software systems and/or particular domains. Such testing knowledge needs to be acquired for each domain type. Additionally, existing test cases and their corresponding requirements were examined and analysed. The details of the test case generation with inference rules are provided in [4].

The condition (if-part) of a heuristic rule is formulated using the ontology instances (individuals) representing the requirement and connected hardware parts, input/output

parameters and the like. The instructions for generating a test case part are expressed in the action part (then-part) of the rule. The Prolog programming language was chosen for coding the acquired inference rules. The requirements ontology was first translated into the Prolog syntax, to prepare the ontology for the inference rules. The translated ontology can be loaded as part of the Prolog program, and the ontology entities can be directly accessed by the Prolog code. The inference engine that is built-in into Prolog was used to execute the coded rules to generate test cases. An example of the inference rule written in Prolog that implements the acquired test case generation heuristic rule for the requirement SRSRS4YY-431 is given below:

```
% construct TC procedure
1   tc_procedure(Requirement, [Service, WriteService, ReadService,
                               recovery(Service)]) :-
    % check condition for calls #2-4
2   action(Requirement, deactivateUART),
    % get service individual for calls #1,4
3   service(Requirement, Service),
    % get individuals of the required services
4   type(WriteService, transmission_service),
5   type(ReadService, reception_service).
    % check the required action
6   action(Requirement, Action) :-
    objectPropertyAssertion(requiresAction, Requirement, Action).
    % retrieve the service of a requirement
7   service(Requirement, Service) :-
      objectPropertyAssertion(requirementForService, Requirement, Service).
    % check the type of an instance
8   type(Individual, Class) :-
      classAssertion(Class, Individual).
```

Line 1 in the example is the head of the rule consisting of the name, "input" argument and "output" argument, which is the constructed procedure as a Prolog list. The list is constructed from the retrieved ontology entities and special term functors. Line 2 encodes the condition of the heuristic. Lines 3-5 are the queries to retrieve the relevant entities from the ontology. The predicates 6-8 are auxiliary and help perform the actual retrieval of the required entities from the ontology.

Each test case is generated sequentially, from the prerequisites part to the results part. The generated parts are collected into one structure (Prolog term).

3.4. Population of the Test Case Ontology

During the generation phase, all created test cases are stored in the Prolog working memory as a list. When the test case list is complete, the next phase of the ontology population starts. The test cases in the list are processed consecutively. For each test case, an instance is created with object properties relating it to the addressed requirement and test case parts. After that, instances with object properties are created for the four test case parts: prerequisites, test inputs, test procedure and expected test results. If the parts contain several elements, OWL lists are used for the representation.

There are two Prolog predicates from the ontology population layer that perform ontology population: `ontology_comment` and `ontology_assertion`. The former is used to insert auxiliary comments in the ontology. The latter asserts OWL axioms representing test case elements in the test case ontology. The four additional

Table 1. Test case from the STD (left column) and the corresponding generated test case by applying inference rules to the populated requirements ontology (right column)

...	...
Test Inputs 1. According to table below. 2. <uartId> := <uartId> from the rs4yy_init call 3. <uartId> := <uartId> from the rs4yy_init call 4. <parity> := rs4yy_noneParity	**Test Inputs:** 1. <parity> := min_value - 1, <parity> := max_value + 1, <parity> := 681881 2. <uartID> := <uartID> from the initializationService call 3. <uartID> := <uartID> from the initializationService call 4. <parity> := noneParity
Test Procedure 1. Call rs4yy_init 2. Call rs4yy_write 3. Call rs4yy_read 4. Recovery: Call rs4yy_init	**Test Procedure:** 1. Call initializationService 2. Call writeService 3. Call readService 4. Recovery: Call initializationService
Expected Test Results 1. <result> == rs4yy_parityCfgError 2. <result> == rs4yy_notInitialised 3. <result> == rs4yy_notInitialised, <length> == 0 4. <result> == rs4yy_ok	**Expected Test Results:** 1. <result> == parityConfigurationError 2. <result> == rs4yyNotInitialised 3. <result> == rs4yyNotInitialised, <length> == 0 4. <result> == rs4yyOk
...	...

predicates, `populate_w_prereq`, `populate_w_inputs`, `populate_w_procedure` and `populate_w_results`, create OWL statements for the test case parts by processing lists associated with each part. The the name prefix and initial number of each of these predicates are passed to construct instances of an OWL list representing this test case part.

Finally, the newly asserted axioms are serialized in the ontology source file by the ontology serialization layer. It takes care of translating the OWL assertions from the Prolog syntax into the OWL functional-style syntax with the help of a definite clause grammar.

3.5. Test Scripts Generation

In order to carry out testing, test cases need to be transformed into executable procedures. Such procedures are usually programs written in a language like Python or C. However, our project partner, Saab Avionics, follows a strict quality assurance process. According to their process, all test cases have to be thoroughly inspected as plain text by the quality assurance team before they are signed of for actual execution. For this reason, the test cases were translated into plain English in the OSTAG project. The translation was implemented by the verbalization process on the test case ontology. The verbalization starts with the test case instance and then iterates through the OWL lists representing the four test case parts. In a similar way, the test cases from the test case ontology can be transformed into actual executable procedures in a programming language.

In the implementation, 37 test cases were translated into plain text descriptions according to the STD provided by Saab Avionics. An example of a test case description is given in Table 1.

3.6. Evaluation of the Generated Test Cases

As we observed, there is an almost one-to-one correspondence between the texts in the generated test scripts and the STD provided by Saab Avionics (see for example the test case in Table 1). The evaluation result validated that it is possible to automate the testing process using the framework. Even more so, on some occasions the generated test scripts texts indicated a discrepancy to the corresponding test scripts texts in the STD document. These discrepancies were presented to and evaluated by personnel from Saab Avionics and on occasions, the observed discrepancies indicated a detected error in the STD document.

4. Lesson Learned

In this section we discuss our choices for the implementation of the framework and lessons learned from the effort to use ontologies and inference rules to automate testing.

4.1. Building Ontologies of Software Requirements and Test Cases

The knowledge acquisition bottleneck is a common problem that we faced when building an ontology. It is difficult to bring domain experts into the ontology development, and it is also challenging for them to express their knowledge in a way appropriate for ontology construction. During the OSTAG project, we had the impression that the software engineers from Saab Avionics, from the beginning to the end of the project, struggled with understanding the ontology engineering technology.

In the project, instead of acquiring knowledge from domain experts, the ontology experts relied on the textual documents, i.e. SRS and STD, as the sources to acquire the knowledge of requirements and test cases. The documents provided by Saab Avionics were written in a well-structured manner and had been manually inspected to achieve compliance with the industry standards for developing highly safety-critical systems. In this way, it was easy to engineer the ontologies of requirements and test cases from the provided documents. However, natural language is inherently ambiguous. For example, several requirements of the embedded system component could be interpreted differently. Therefore, the evaluation and validation of the ontologies became necessary.

The population of the requirements ontology was done manually in the current project. To increment the level of automation of the testing process, the ontology population step also could be automated using ontology learning methods [6,7]. However, no well-established tools for automatic ontology population can support domain-specific applications. As a step forward, we developed a method based on BNF grammar [8] as a lightweight solution for automatic ontology population. The evaluation results showed that a fully automated ontology population with high quality is a challenging task.

4.2. Evaluation and Validation of Ontology

Although ontology evaluation is relevant and important in ontology-based applications, little attention has been paid to this topic. A number of different ontology evaluation criteria, or features as we call them, have been discussed in literature (e.g. [9,10,11]), but mainly from a theoretical point of view. To actually do ontology evaluation, the first challenge is to determine the relevant quality features that are to be evaluated for a specific application. We identified the following three quality features as the most relevant for this application:

- *Ontology usability* is defined as a set of attributes that describes the effort needed by a human to make use of an ontology. The feature is especially important when the ontology is going to be used by application domain experts who are normally not ontology experts.
- *Ontology applicability* is defined as the quality of the ontology being correct or appropriate for a particular application domain or purpose. In this case, the feature is about whether the ontologies support well the step of rule-based test case generation within the framework of automated testing, as presented in Fig 1.
- *Ontology correctness* is defined as the degree to which the information asserted in the ontology conforms to the information that should be represented in the ontology. For example, the requirements ontology should accurately represent the information in SRS.

The second, but also more challenging issue, is how to facilitate the evaluation of these features. However, not many practical methods and tools exist for ontology evaluation. The methods based on metrics [12,13] do not help us much in the evaluation of the three features. In the article [3] we reported on the methods and tools we developed to support the evaluation of the ontologies in the application. Our experience is that evaluation methods and/or tools should provide user-friendly functionality to assist non-ontology experts when they are involved in the evaluation process. The domain experts from Saab Avionics were involved in the validation of the populated requirements ontology. The level of difficulty to learn to master an ontology tool, like Protégé , is too high and not intuitive to them as first time users. To help the domain experts in their evaluation and validation work, a technique known as verbalization was used and worked very well.

4.3. Use of OWL as Ontology Language

OWL, a standard ontology language recommended by the Semantic Web community, was chosen as the ontology language. The expressiveness of OWL allowed for the capturing of SRS and STD details as well as for representing knowledge on both software requirements and the underlying hardware components. Being the standard language, OWL alleviates potential interoperability issues. There are different serialisations: both XML-based, e.g. OWL/XML, and text-based, e.g. Turtle and functional-style syntax. We chose the latter option because it allowed for interoperability with the test case generator and it was easier to work with for the knowledge engineers.

OWL is based on description logics. It supports automatic checking of the consistency and completeness of a model. OWL is an expressive language that allows for representation of intrinsic software requirements peculiarities. The OWL definitions are split

into the TBox and ABox. The distinction between TBox and ABox, however, is not always clear and obvious. Should DeactivateURT be represented as a subclass of Action or an instance of it? It actually depends on the purpose of knowledge modelling. Since the test case generation inference rules are formulated in terms of ontology individuals, classes and relations, our experience is that TBox should be simple and easy to understand and instance data should be handled in a simple fashion. Hence, the level of OWL expressiveness should be adjusted according to the domain of discourse and the purpose of knowledge representation. Sophisticated OWL restrictions are extremely difficult to verify by domain experts and are a complication factor for domain-level reasoning with the ontology. Furthermore, the TBox should be reusable as much as possible to model requirement specifications in different applications across the domain.

4.4. Use of Ontology-Based Reasoning

The use of OWL results in a machine-readable model that can be used for reasoning in different ways. While there are several OWL reasoners, such as Pellet or HermiT that allows for high-level reasoning at the level of descriptions logics, domain-specific reasoning requires either coding in a general-purpose language, e.g. Java, or formalisation of custom inference rules and utilisation of an inference engine to execute the rules. Test case generation requires the use of procedural knowledge—strategies that are used by software testers to create test cases. This type of knowledge is difficult to capture directly within an ontological model as soon as ontologies are designed to represent declarative knowledge. A rule language provides an explicit way to capture and store procedural knowledge. A rule-based formalisation of procedural knowledge could be more flexible and easier to maintain compared to program code-based formalisation. Moreover, domain-level reasoning are easier to validate by experts when the reasoning steps are represented by inference rules compared to being hard-wired into code.

The results of reasoning during the test case generation were saved in a new, test case ontology through population. This exemplifies the use of reasoning for domain specific transformation from one ontological model to another. However, we learned that the expressiveness level of OWL in the ontology should be limited to OWL Lite. Otherwise, the higher expressiveness could cause performance problems during reasoning with rules or it would be difficult to use in program code.

4.5. Use of Prolog as a Rule Language

SWRL (Semantic Web Rule Language) is a frequent choice for reasoning over OWL ontologies. It can support both forward-chaining and backward-chaining reasoning depending on the capabilities of an inference engine used to power SWRL rules. Although SWRL is widely used for OWL reasoning, it has rather limited facilities for expressions in rule antecedents. This limitation, together with the necessity to complement it with an external inference system, would complicate the full implementation of the procedural semantics of test case generation. Hence, we chose Prolog as both rule language and inference mechanism for the implementation of the test case generator.

Prolog supports backward-chaining reasoning, which fits to the problem-solving type in our case. It was natural to implement rules starting with the goal—a test case or test case part to generate. Prolog has a built-in inference engine with automatic conflict

resolution that reduced the effort required to manage rule firing. Apart from the inference mechanism necessary to execute rules, Prolog has standard programming facilities. Thus, no other programming language was needed to develop a complete test case generator. Our choice of rule language implied the translation to/from Prolog but it was not difficult to implement due to the availability of the serialisation of OWL in functional-style syntax. The advantage of the translation was that the ontology statements could be directly queried from the Prolog rules. One more aspect to consider is the assumption on the truth values of the assertions in an ontology. Prolog is based on close-world assumption, while OWL is based on open-world assumption. However, it did not create any difficulties as soon as the required reasoning was performed in a limited and closed domain.

The disadvantage of Prolog is that it lacks support for OWL specific semantics. We solved this by adding additional Prolog rules implementing, for example, subsumption reasoning. In this way, the support for most of the parts of the OWL semantics could be brought into a Prolog program. We also learned that the acquisition of inference rules could be a problem. A partial solution that we found was to map particular test cases to the corresponding requirements and then to discuss it with the expert testers. The last drawback that we learned was that the use of individuals (ABox) makes the rules fragile. As soon as individuals in an ontology could change frequently, the rules need to be updated with every change. To alleviate this problem, classes and relations (TBox) were used when possible instead of individuals' names.

5. Related Work

There have been presented a number of projects with a focus on using formal models, such as finite state machines, input/output transition systems, and UML, to support testing activities [14,15]. The use of ontologies as formal model in software testing has not yet received much attention, at least not as much as in other stages of the software life-cycle process [16]. In [17], Happel and Seedorf present possible ways of utilizing ontologies for the generation of test cases, and discuss the feasibility of reusing domain knowledge encoded in ontologies for testing. In practice, however, few tangible results have been presented. Most of the research have had a focus on the testing of web-based software and especially web services, e.g. [18,19].

The use of ontologies in requirements engineering date back to the 1990s, e.g. [20, 21,22]. There exists a clear synergy effect between the ontological modelling of domain knowledge and the modelling of requirements performed by requirement engineers [23]. Recently, a renewed interest in utilizing ontologies in requirements engineering has surged due to the appearance of semantic web technologies [17,23]. Most of the research deals with inconsistency and incompleteness problems in requirement specifications through reasoning over requirements ontologies.

Prolog has been used as a reasoner for OWL ontologies in a number of cases. For example, in [24] the authors describe an approach to reason over temporal ontologies that translates OWL statements to clauses in Prolog and then uses the built-in inference mechanism. In [25] an OWL ontology and OWLRuleML rules are translated into Prolog clauses, which are then used to infer new facts by the Prolog inference engine. Our approach for automatic test case generation has utilised a similar idea, but we have used OWL functional-style syntax for the OWL to Prolog translation which makes queries to the ontology as close as possible to OWL syntax.

6. Conclusions

In the OSTAG research project, we proposed an ontology-based approach to automate the test process. The approach was applied and implemented in a use case from the avionics industry. In this paper we described the lessons learned from the project, especially those relevant to the practical issues in areas of ontology development, ontology evaluation, the OWL language and rule languages.

The implementation of the application has shown that it is possible to capture the knowledeg in well-structured requirements specifications and test cases using ontologies and to represent them in the OWL language. The test case generation strategies is a type of procedural knowledge. They can be captured by rule languages. The lessons learned about the OWL language and rule languages can be summarized as: (1) The expressivity level of ontologies has to be limited on the level that is sufficient for its specific application, if the ontology has to be populated, and the ontology has to be evaluated by domain experts especially when no tool is available to support domain expert evaluation, and (2) Care has to be taken not to make the rules too dependent on ABox by formulating rules in terms of TBox that is much more stable.

The main challenges lied in the ontology development and evaluation area. The time and effort needed to manually develop and populate the ontologies was reasonably high. One solution to this problem is to automate the process, but our preliminary experiments with the ontology learning methods have shown that the ontology development cannot be fully automated. Ontology experts are needed to finish the task. The cost will depend on the quality of the implemented ontology learning methods. Another solution is to introduce boilerplates or semantic patterns in textual documents when they can be used as the main source of domain knowledge. This solution would require a change in the process where the documents are developed.

An ontology evaluation is necessary and has to be arranged around usability, applicability and correctness, especially when the ontology is applied as a backbone for a solution in a domain-specific application. It is challenging to involve domain experts in the task due to the difficulty of using ontology tools. As soon as it become difficult for domain engineers to work directly with ontology editors, tools like ontology verbalization can be used instead of directly using an ontology editor. When an editor has to be used, a preparatory training session and the assistance of ontology engineer's are beneficial.

References

[1] B. Beizer, *Software testing techniques*, Dreamtech Press, 2003.

[2] H. Tan, I. Muhammad, V. Tarasov, A. Adlemo and M. Johansson, Development and evaluation of a software requirements ontology, in: *7th International Workshop on Software Knowledge-SKY 2016 in conjunction with the 9th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management-IC3K 2016*, 2016.

[3] H. Tan, V. Tarasov, A. Adlemo and M. Johansson, Evaluation of an Application Ontology, in: *Proceedings of the Joint Ontology Workshops 2017 Episode 3: The Tyrolean Autumn of Ontology Bozen-Bolzano*, CEUR-WS, 2017.

[4] V. Tarasov, H. Tan, M. Ismail, A. Adlemo and M. Johansson, Application of Inference Rules to a Software Requirements Ontology to Generate Software Test Cases, in: *OWL: Experiences and Directions – Reasoner Evaluation: 13th International Workshop, OWLED 2016, and 5th International Workshop, ORE 2016, Bologna, Italy, November 20, 2016, Revised Selected Papers*, M. Dragoni, M. Poveda-

Villalón and E. Jimenez-Ruiz, eds, Lecture Notes in Computer Science, Vol. 10161, Springer International Publishing, Cham, 2017, pp. 82–94. ISBN ISBN 978-3-319-54627-8.

[5] N. Drummond, A.L. Rector, R. Stevens, G. Moulton, M. Horridge, H. Wang and J. Seidenberg, Putting OWL in Order: Patterns for Sequences in OWL., in: *OWLED*, Citeseer, 2006.

[6] A. Gangemi, V. Presutti, D. Reforgiato Recupero, A.G. Nuzzolese, F. Draicchio and M. Mongiovì, Semantic web machine reading with FRED, *Semantic Web* **8**(6) (2017), 873–893.

[7] G. Petasis, V. Karkaletsis, G. Paliouras, A. Krithara and E. Zavitsanos, Ontology population and enrichment: State of the art, in: *Knowledge-driven multimedia information extraction and ontology evolution*, Springer-Verlag, 2011, pp. 134–166.

[8] M. Ismail, Ontology Learning from Software Requirements Specification (SRS), in: *European Knowledge Acquisition Workshop*, Springer, 2016, pp. 251–255.

[9] A. Gómez-Pérez, Ontology evaluation, in: *Handbook on ontologies*, Springer, 2004, pp. 251–273.

[10] A. Gangemi, C. Catenacci, M. Ciaramita and J. Lehmann, Modelling ontology evaluation and validation, in: *European Semantic Web Conference*, Springer, 2006, pp. 140–154.

[11] D. Vrandečić, Ontology evaluation, in: *Handbook on ontologies*, Springer, 2009, pp. 293–313.

[12] S. Tartir and I.B. Arpinar, Ontology evaluation and ranking using OntoQA, in: *International Conference on Semantic Computing (ICSC 2007)*, IEEE, 2007, pp. 185–192.

[13] H. Hlomani and D. Stacey, Approaches, methods, metrics, measures, and subjectivity in ontology evaluation: A survey, *Semantic Web Journal* **1**(5) (2014), 1–11.

[14] A. Petrenko, A. Simao and J.C. Maldonado, Model-based testing of software and systems: recent advances and challenges, Springer, 2012.

[15] M. Utting, B. Legeard, F. Bouquet, E. Fourneret, F. Peureux and A. Vernotte, Recent advances in model-based testing, in: *Advances in Computers*, Vol. 101, Elsevier, 2016, pp. 53–120.

[16] F.B. Ruy, R. de Almeida Falbo, M.P. Barcellos, S.D. Costa and G. Guizzardi, SEON: a Software Engineering Ontology Network., in: *Knowledge Engineering and Knowledge Management: 20th International Conference, EKAW 2016, Bologna, Italy, November 19-23, 2016, Proceedings 20*, Springer, 2016, pp. 527–542.

[17] H.J. Happel and S. Seedorf, Applications of ontologies in software engineering, in: *Proceedings of Workshop on Sematic Web Enabled Software Engineering (SWESE) on the ISWC*, 2006, pp. 5–9.

[18] Y. Wang, X. Bai, J. Li and R. Huang, Ontology-based test case generation for testing web services, in: *Autonomous Decentralized Systems, ISADS'07. Eighth International Symposium on*, IEEE, 2007, pp. 43–50.

[19] H.M. Sneed and C. Verhoef, Natural language requirement specification for Web service testing, in: *Web Systems Evolution (WSE), 2013 15th IEEE International Symposium on*, IEEE, 2013, pp. 5–14.

[20] J. Mylopoulos, A. Borgida, M. Jarke and M. Koubarakis, Telos: representing knowledge about information systems, *ACM Transactions on Information Systems (TOIS)* **8**(4) (1990), 325–362.

[21] S. Greenspan, J. Mylopoulos and A. Borgida, On formal requirements modeling languages: RML revisited, in: *Proceedings of 16th international conference on Software engineering*, IEEE Computer Society Press, 1994, pp. 135–147.

[22] M. Uschold and M. Gruninger, Ontologies: principles, methods and applications, *The Knowledge Engineering Review* **11**(02) (1996), 93–136.

[23] G. Dobson and P. Sawyer, Revisiting ontology-based requirements engineering in the age of the semantic Web, in: *Proceedings of International Seminar on Dependable Requirements Engineering of Computerised Systems at NPPs*, 2006, pp. 27–29.

[24] N. Papadakis, K. Stravoskoufos, E. Baratis, E.G. Petrakis and D. Plexousakis, PROTON: A prolog reasoner for temporal ontologies in OWL, *Expert Systems with Applications* **38**(12) (2011), 14660–14667.

[25] L. Laera, V. Tamma, T. Bench-Capon and G. Semeraro, Sweetprolog: A system to integrate ontologies and rules, in: *International Workshop on Rules and Rule Markup Languages for the Semantic Web*, Springer, 2004, pp. 188–193.