An Ontology Based Approach for Test Scenario Management

P.G. Sapna and Hrushikesha Mohanty

University of Hyderabad,
Hyderabad, India
sapna_pg@yahoo.com, mohanty.hcu@gmail.com

Abstract. Management of test scenarios involves ordering and selecting a set of test scenarios for testing with the objective of fulfilling criteria like maximizing coverage or discovering defects as early as possible. To do this, knowledge about the main activities in the domain and the interactions between them need to be maintained. This knowledge helps in categorizing scenarios as per the approaches used for testing. Current approaches to test management involve generating scenarios related to use cases. However, the use case based approach does not capture knowledge about the dependencies that exist among activities in the domain and therefore cannot provide complete information for scenario management. Ontologies provide a mechanism to represent this knowledge. They also provide a means to share and reason on knowledge that is captured. The objective of this work is to use ontologies to aid test management.

Keywords: UML, use case, scenario, ontology, test scenario management.

1 Introduction

One of the challenges faced in software engineering is building software with quality at reduced cost, time and effort [2]. This requires a clear understanding of the elements used to build a software and the relations among them. In this direction ontologies can aid in performing inference on knowledge gathered from requirements captured using UML diagrams. Ontologies also enable communication among different people involved in building a software and hence can be used as a common mechanism for requirements analysis.

Test management is a way of organizing artifacts of the system like requirements, scenarios and test cases such that it enables accessibility and usability with the objective of delivering quality software within constraints of cost and time. Requirements and scenarios are specified by UML models. Given the large number of use cases and scenarios for even a medium sized system, there is a need for effective scenario management. For this purpose, not only domain but also the process of testing should be specified comprehensively as well as understandably so that the stakeholders in testing can participate effectively. With the background of the wide use of ontology in software engineering, it is proposed to use ontologies for test management in this work. The use of analyzing the requirements of a domain through an ontology of software testing is to allow reasoning about the information represented [1]. For instance :

– Consistency. Suppose it is declared a and b are instances of the concept U. Also, it is defined that a and b are different individuals and the relation 'includes' is irreflexive and asymmetric. Then, if it is defined that a includes b and b includes a, then there is inconsistency.. This error can be detected on reasoning of test ontology.
– Inferred relationships. If concept X is equivalent to concept Y, and concept Y is equivalent to concept Z, then X is equivalent to Z.
– Membership. If a is an instance of concept U and U is a subconcept of Y, then it can be inferred that a is an instance of Y.
– Classification. Suppose it is declared that certain property-value pairs are a sufficient condition for membership in a concept A. Then, if an individual x satisfies the condition, it can be said that x is an instance of A.

This kind of inference and reasoning based on specific relationships can be better done on ontology repository than a use case based approach. In this work, an ontology is introduced to facilitate querying requirements for test management. The contribution of this work includes :

– Development of an ontology for software testing
– Querying the ontology

The ontology is implemented in OWL using Protege as the editor. Built ontology shows the relations among the testing concepts outlined in SWEBOK[1]. Section 1.1 provides a motivating example for using ontology. Section 1.2 discusses the use of an ontology as a repository for test management. The use of ontologies in different areas and particularly in testing is discussed in Section 2. Application of the steps to build an ontology for testing based on the testing concepts extracted from SWEBOK and the primitives of relevant UML diagrams is discussed in Section 3. The section includes design, implementation and transformation steps involved in building an ontology. Reasoning on the ontology is discussed in Section 4. Section 5 summarizes the work.

1.1 Motivating Example

This section discusses an example that motivates work on test management using an ontology. Fig. 1 shows the relationship between use case, scenario and classes. A requirement shown in a use case diagram and detailed by scenarios is implemented by classes. For querying on test process with respect to a requirement, there has to be traceability for the relations existing among the artifacts. e.g. 'project management' has scenario 'wage calculation' implemented by classes, say, 'pay-calculate', 'award-calculate', 'utility-award', etc. The relations among these artifacts can be shown using an ontology. Using an E-R diagram for the

[1] Software Engineering Book of Knowledge provides a consistent view of software engineering, its boundary and contents. Available at `www.swebok.org/`

same example does not show the impact on testing. That is, we need to show that for every use case that is to be tested, corresponding scenarios must be tested. Also, unit testing of all classes that make up the scenario must be tested. This requires a transitive relationship to be defined such that for each use case, all corresponding scenarios as well as related classes have to be tested. Usually class repositories are stored in an RDBMS, but still has limitations. Fig. 1. Entity Relationship diagram for the relation 'Use case has scenarios, and each scenario is made of classes'.

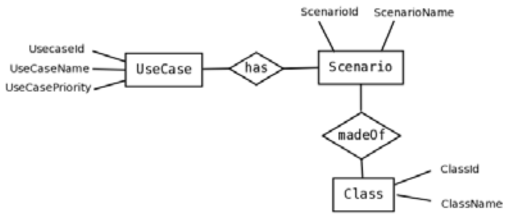


Fig. 1. Entity Relationship diagram for the relation 'Use case has scenarios, and each scenario is made of classes'

1.2 Ontology as a Repository

The common practice is to use a database to store the requirements of a system and query the same. However, it is not possible to capture all constraints related to a system using the representation provided by relational databases. Examples showing situations where a relational database is insufficient to represent the conceptual relations are discussed in Section 1.1.

For the following disabilities [7] found with RDBMS, ontology is considered as useful repository.

– Lack of hierarchy. Relational models have no notion of a concept/query hierarchy.
– Representing m:n relationships. Relation between entities are represented using 1:n relationship in RDBMS. m:n relationships are represented as a set of m, 1:n relationship.
– Access to data. There is need to have access to the database to query the database.
– Knowledge of a query language. User must have knowledge of a query language supported by the specific database.

Representing requirements using ontologies is found more useful than RDBMS for being expressive and flexible to manage. Querying on requirements stored in a relational repository requires prior knowledge on entity relations(i.e. domain structure) whereas in case of ontology, repository queries can be exploratory for traversing links among concepts that make the ontology. The use of ontologies in different areas and particularly in testing is discussed in Section 2.

2 Survey

A survey of current work on ontologies for software reveals that ontologies have been built with varied objectives in mind. One of the objectives of building ontologies is to facilitate collaboration of remote teams in multi-site distributed software development. SWEBOK has been used as the point of reference to build ontologies of software engineering [11, 12]. The ability of ontologies to query necessary and relevant information regarding the domain concerned is exploited. Dillon et al [3] have focused on the same issue by developing a software engineering ontology that defines common sharable software engineering knowledge as well as information about particular projects. The software engineering ontology consists of five sub-ontologies for software requirements, design, construction, testing and tools and methods. The software testing ontology in particular, consists of subontologies, namely: test issues sub-ontology, test targets sub-ontology,test objectives sub-ontology, test techniques sub-ontology and test activities sub-ontology. Both [11, 3] provide the advantage of development of a consistent understanding of the meaning of issues(terminology and agreements) related to a project by different people distributed geographically across locations.

A second objective for building ontologies is to access ontology mediated information [3]. Rules can be written about relationships between concepts in an ontology and the same can be used for query processing. The advantage over databases is that new facts can be inferred or reasoned with asserted facts. An example of this objective is the Protein Ontology built to integrate protein knowledge and provide a structured and unified vocabulary to represent concepts related to protein synthesis.

Thirdly, ontologies are used in the area of semantic web services [3]. Several issues need to be addressed in the area of web services that include, selection of architecture, discovery of service, selection of service and composition and coordination of services to meet requirements. Web services are semantically annotated to assist in the process of discovering and selection for which a combination of ontologies and Web 2.0 is used.

A fourth area in which ontologies are used is multi-agent systems [3, 9]. Multi-agent systems involve multiple autonomous agents that collaborate with one another to fulfil goals of the system. Agents have a knowledge base that provides some intelligence. Also, the system is distributed and decentralized where agents are geographically distributed and communicate mainly through message passing. To maintain coherence and consistency of knowledge among agents, an ontology is used as a common knowledge base that is shared by all agents. This facilitates communication and coordination among agents.

Other ontologies that have been built include the disease ontology, manufacturing ontology and different financial system ontologies [3]. A software test ontology(SwTO), that deals with the software testing domain has been built by [2]. The ontology is used along with a test sequence generator to generate test sequences to test the operating system domain, Linux. Zhu et al [13] in their work present an ontology of software testing. They discuss use of an ontology in a multi-agent software environment context to support the evolutionary development and maintenance of web-based applications. Software agents use the ontology as the content language to register into a system and for test engineers and agents to make test requests and report results. The paper describes how the concepts of the ontology and the relations between them are defined in UML.

Thus, ontologies have found use in different areas like software engineering, semantic web services and multi-agent systems. The next section details the steps involved in building an ontology with focus on testing.

3 Building an Ontology for Testing

One of the well-known methods to build an ontology is the Methontology strategy [6]. A generally applicable method to construct domain knowledge model and validate the same is proposed. The ontology development process is composed of the following steps: planning, understanding, knowledge elicitation, conceptualization, formalization, integration, implementation, evaluation, documentation and maintenance. Noy et al [8] follow an iterative design process to build an ontology. Steps to build an ontology is proposed. This work adapts ideas from both works. The set of steps to build an ontology is given in the subsection below.

3.1 Design of Ontology

Steps described in building an ontology are applied in the context of software testing and are explained below:

a. Determine domain and scope of ontology : The objective of this work is to provide a framework for testing, specifically specification based testing wherein specification is captured using UML. Representation and management of use cases and scenarios are in focus while building an ontology. The objective is to use this ontology for enumerating test scenarios to form a test suite for testing.

In the ontology, concepts related to requirements and scenarios like, use cases,related users(actors), scenarios related to each use case are included. At the same time concepts like type of testing and resource are referred from SWEBOK. In this work, the objective of building an ontology is to provide a means to manage scenarios. Some of the questions that the ontology should answer includes:

- Which requirements involve use of activity ‘x’?
- What are the scenarios required to test use case ‘UC’?
- What are the use cases that include use case ‘UC’ ?
- What are the use cases that extend use case ‘UC’ ?
- List all scenarios of the use case ‘UC’ that include use case ‘UC1’ having priority ‘p’.

b. Defining concepts in the ontology : To define an ontology for testing, a list of terms and the related properties are listed. For example, important terms related to this work include: use case, actor, scenario, activity diagram, class, priority, testtype, testsuite, etc; different types of priority include customer assigned priority and technical priority. Fig. 3 shows the concepts in the ontology.

c. Create a class hierarchy : First, terms that independently describe objects are considered. The terms form classes in the ontology. For example, use case, actor, scenario, priority form classes in the ontology. The next step involves organizing classes into hierarchical taxonomy. For example, customer assigned priority and technical priority are subclasses of the class priority. i.e.customer assigned priority is a type of priority. Therefore, customer priority is a subclass of the priority class.

d. Defining properties and constraints : Concepts themselves are incapable of answering questions such as those enumerated in Step 1. There is need to describe internal structure of concepts. For example, consider the concept use case. Properties related to the concept includes haspriority, hasscenario, hasactor. For each property, the class it describes must be determined. The properties are attached to concepts. With reference to Fig. 1, the properties include has between concepts UseCase and Scenario and property madeOf between concepts Scenario and Class.

e. Creating instances : Instances are individuals belonging to a concept. Instances of concepts in Fig. 1 are shown in Fig. 2.

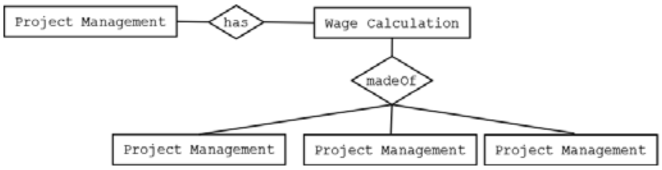


Fig. 2. Instances of concepts in Figure 1

3.2 Defining Relations between Concepts

Classes and relations between the concepts are shown in Fig. 3. For each concept, conditions are defined as to how the concepts interact for realizing an objective. An example of sufficient and necessary condition for some of the concepts is elaborated below:

Use case - Condition. The concept UseCase is a sub concept of System. The hasActor property is used to relate individuals of the UseCase concept with the individuals of the Actor concept. Also, a UseCase must be related to atleast one actor of the Actor concept through the hasActor property.

Use case - Properties. hasActor is an inverse object property of isActorOf, whose domain is UseCase and range is Actor. hasUseCasePrecondition is a functional object property, whose domain is UseCase and range is UCPrecondition. hasUseCasePostcondition is a functional object property, whose domain is

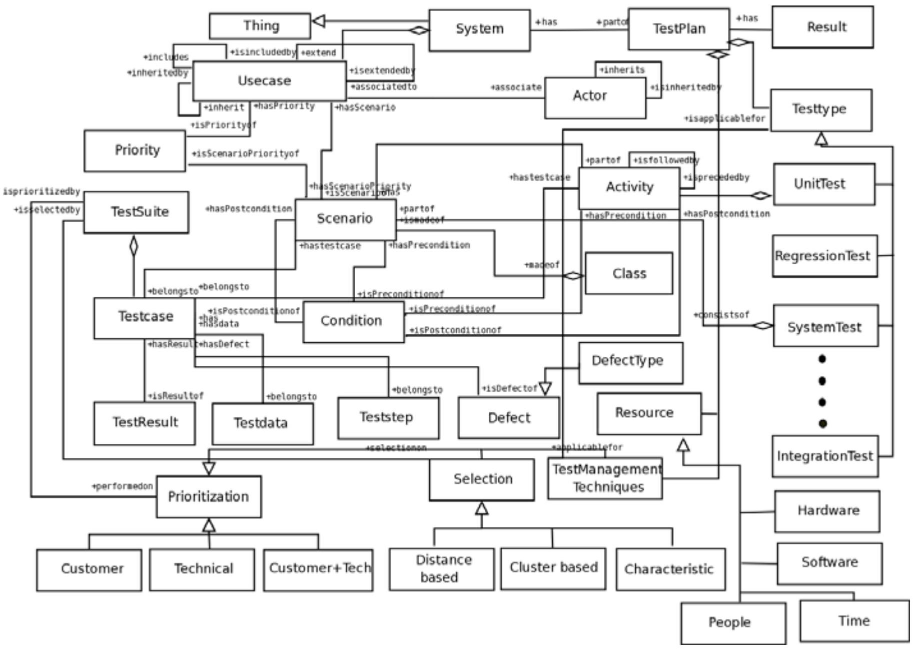


Fig. 3. Class diagram showing relation among concepts of the ontology

UseCase and range is UCPostcondition. hasUseCasePriority is a functional object property, whose domain is UseCase and range is UCPriority. hasUseCaseName is an object property.

3.3 Implementation Using OWL

OWL has been chosen to implement the ontology due to its knowledge representation capabilities(concepts, individuals, properties, relationships and axioms) and the possibility to reason about the concepts and individuals[5]. Other ontology languages that can be used include RDF, DAML and DAML+OIL. Given use case and activity diagrams, an XSLT(eXtensible Stylesheet Language Transformation) document is written that defines the transformation rules to covert an XML file to OWL file[4]. An XSLT processor aids in performing the transformation based on the rules. Following are the transformation steps from XML to OWL using XSLT as shown in Fig. 4.

– The use case and activity diagrams serve as input to the transformation process. In this work, scenarios have been generated from activity diagrams.
– Run the XSLT processor. The XSLT file provides the template for transformation. The input file is read and if it finds the predefined pattern, it replaces the pattern with another according to the rules.

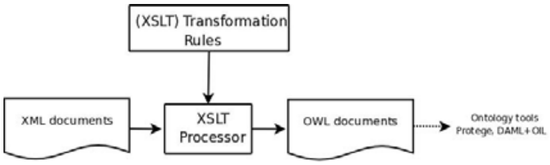


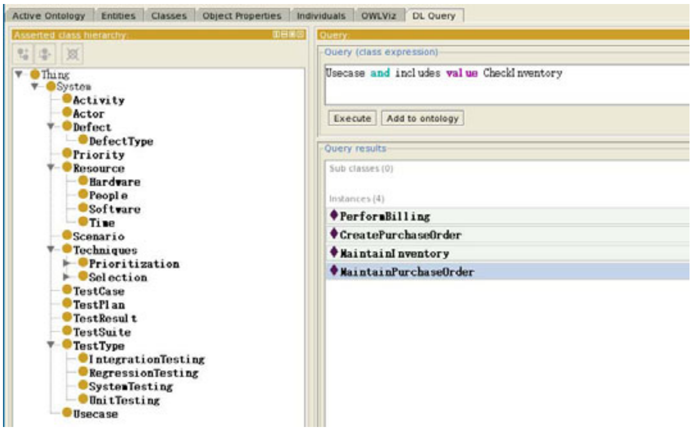
Fig. 4. The transformation process

– Output of the process is OWL specification. Ontology tools like Protege[2] can be used to verify syntax and validate semantics captured. It is to be noted here that test scenario generation algorithm in this work generates scenarios for system testing.

4 Results - Querying the Ontology

As shown in Fig. 5 the Protege OWL editor has support to edit and execute rules through the Query window using Pellet/FACT++ as reasoner. A user expresses requirements as a query and submits it to the Protege tool to obtain results. Queries include determining the number of tests that passed based on some criteria, tests based on a criteria that failed as well as displaying results. A sample of the queries that can be given to the ontology include:

1. What are the use cases the include use case 'x' ?
2. List all scenarios have test result = fail.
3. List all actors related to use cases and scenarios having priority 'y'.
4. List all scenarios for use case 'uc'. (This lists all scenarios for use case 'uc' as well as for use cases that are included by use case 'uc').

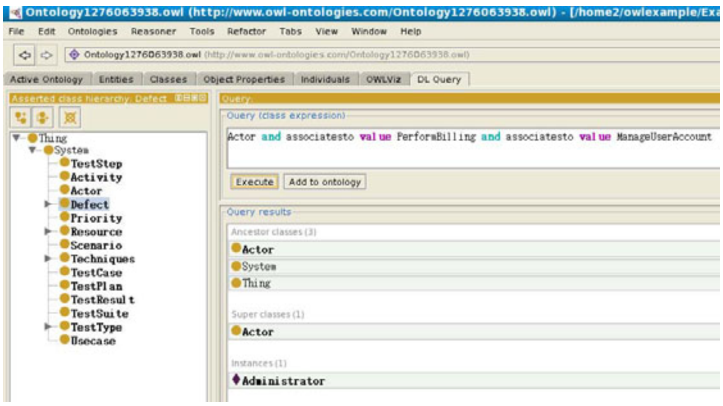


```
Usecase and includes value CheckInventory
```

```
PerformBilling
CreatePurchaseOrder
MaintainInventory
MaintainPurchaseOrder
```

Fig. 5. Querying the ontology - list all use cases that include use case 'Check Inventory'

[2] Protege is an open source ontology editor and knowledge-base framework. Available at `http://protege.stanford.edu`


Ontology1276063938.owl (http://www.owl-ontologies.com/Ontology1276063938.owl) - [/home2/owlexample/Exa

- Thing
  - System
    - TestStep
    - Activity
    - Actor
    - Defect
    - Priority
    - Resource
    - Scenario
    - Techniques
    - TestCase
    - TestPlan
    - TestResult
    - TestSuite
    - TestType
    - Usecase

Query (class expression)

```
Actor and associatesto value PerformBilling and associatesto value ManageUserAccount
```

Query results

Ancestor classes (3)
- Actor
- System
- Thing

Super classes (1)
- Actor

Instances (1)
- Administrator

Fig. 6. Querying the ontology - list all actors associated with both use cases 'Perform Billing' and 'Manage User Account'

5. List of all unit tests for the Software under Test.
6. List all scenarios used by an actor.

Two examples querying the ontology is shown. In the first example(Fig. 5), use cases, 'PerformBilling', 'CreatePurchaseOrder' and 'MaintainInventory' include use case 'CheckInventory'. The constraint applied on relation 'include' is that it is transitive. Hence, the query 'Usecase and includes value CheckInventory' gives the above three use cases as well as the use case 'MaintainPurchaseOrder' which includes use case 'CreatePurchaseOrder'. This inference is useful in testing the use case 'Maintain Purchase Order' where the scenarios belonging to 'Check Inventory' also have to be tested. A second example requires that all actors related to both use cases 'PerformBilling' and 'Manage User Account' be given. Fig. 6 show the results of the query.

5 Conclusion

In this work, an approach for test scenario management is proposed. The increasing size of software systems makes it necessary to manage the test scenarios in an efficient way. In this regard, ontologies help to analyze the relationship between requirements captured using UML diagrams. Scenarios generated from activity diagrams(i.e. each use case has atleast one activity diagram) along with activities, are linked to use cases. Further, software testing concepts adopted from SWEBOK, are used to build an ontology for test management. Protege, an OWL editor is used for querying and reasoning. Thus, ontologies have a common reference point for architects, developers and test engineers engaged in software development. In particular, the use of ontologies for test scenario management to achieve desired quality in terms of test coverage is shown.

References

1. Antoniou, G., van Harmelen, F.: A Semantic Web Primer. The MIT Press, Cambridge (2004)
2. Bezerra, D., Costa, A., Okada, K.: swtoI (Software Test Ontology Integrated) and its Application in Linux Test. In: Proceedings of Ontose 2009, pp. 25–36 (2009)
3. Dillon, T.S., Chang, E., Wongthongtham, P.: Ontology-based software engineering-software engineering 2.0. In: Proceedings of the 19th Australian Conference on Software Engineering, pp. 13–23. IEEE, Los Alamitos (2008)
4. DuCharme, B.: XSLT Quickly. Manning Publications
5. Mendes, O., Abran, A., Montral Qubec, H.K.: Software engineering ontology: A development methodology. In: Metrics News
6. Juristo, N., Ferndandez, M., Gomez-Perez, A.: METHONTOLOGY: From Ontological Art Towards Ontological Engineering. AAAI Technical Report SS-97-06, 15(2) (1997)
7. Mabotuwana, T., Warren, J.: An ontology based approach to enhance querying capabilities of general practice medicine for better management of hypertension. Artificial Intelligence in Medicine (2009)
8. Noy, N.F., McGuinness, D.L.: Ontology Development 101: A Guide to Creating Your First Ontology. Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical Informatics Technical Report SMI-2001- 0880, 15(2) (2001)
9. Nguyen, C.D., Perini, A., Tonella, P.: Ontology-based Test Generation for Multi-agent Systems. In: AAMAS 2008: Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems. International Foundation for Autonomous Agents and Multiagent Systems, pp. 1315–1320 (2008)
10. Andrea Rodriguez, M., Egenhofer, M.J.: Determining semantic similarity among entity classes from different ontologies. IEEE Transactions on Knowledge and Data Engineering 15(2), 442–456 (2003)
11. Chang, E., Wongthongtham, P., Cheah, C.: Software Engineering Sub Ontology for Specific Software Development. In: Proceedings of 29th Annual EEE/NASA Software Engineering Workshop (SEW 2005), pp. 27–33. IEEE, Los Alamitos (2005)
12. Wongthongtham, P., Chang, E., Dillon, T.: Towards Ontology-based Software Engineering for Multi-site Software Development. In: 3rd IEEE International Conference on Industrial Informatics (INDIN), pp. 362–365. IEEE, Los Alamitos (2005)
13. Zhu, H., Huo, Q.: Developing software testing ontology in UML for a software growth environment of web-based applications. Software Evolution with UML and XML, 263–295 (2005)