Chapter 5

Ontology-based Scenario Management

Management of test scenarios involves storing and retrieving scenarios fulfilling a criteria like selecting certain chunk of scenarios pertaining to a given use case. In order to achieve this, knowledge on domain entities and their relations are to be maintained. This knowledge helps in categorizing scenarios as per the approaches used for testing. Ontologies provide a mechanism to share and reason on knowledge that is captured. The objective of the work described in this chapter is to use ontologies to aid test management.

5.1 Introduction

One of the challenges faced in software engineering is building software with quality at reduced cost, time and effort [BCO09]. This requires a clear understanding of the elements that build a software and the relations among them. In this direction ontologies can aid in reusing and performing inference on knowledge gathered from requirements captured using UML diagrams. Ontologies also enable communication between different people involved in building a software and hence can be used as a common mechanism for requirements analysis.

Based on the work in previous chapters, use cases and scenarios have been
obtained. Now, during testing, test cases related to use cases and scenarios are to be picked as per a testing criteria like required percentage of test coverage. For the purpose, an ontology based approach for storage and retrieval of use cases and their corresponding scenarios is proposed. Ontology helps to retain domain specific relations among use cases and scenarios; it enables to reason over ontology facilitating retrieval of scenarios from repository for a given criteria. A metamodel for use case and scenario management is proposed so that a reasoner can be applied while querying. For the purpose, Protege, an open source ontology editor has been used. Further, the ontology metamodel is extended for test process management.

The work reported in this chapter is organized as follows: Section 5.1.1 provides motivating examples for using ontology. Section 5.1.2 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 5.2. Section 5.3 enumerates the steps involved in building an ontology. Application of the steps to build an ontology for testing based on the testing concepts is also discussed. Section 5.3.1 describes the process of implementing using Protege, an open source ontology editor. Extension of ontology using concepts extracted from SWEBOK\textsuperscript{1} and using the primitives of relevant UML diagrams is discussed in Section 5.5.

5.1.1 Motivating Examples

This section discusses examples that motivate work on test management using an ontology.

Example 1:

Fig. 5.1 shows the relationship between use case, scenario and classes. A requirement shown in a use case diagram and detailed by scenarios is implemented by

\textsuperscript{1}Software Engineering Book of Knowledge provides a consistent view of software engineering, its boundary and contents. Available at www.swebok.org/
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 same example does not show the impact on testing. That is, we can say 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 done. 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.

Figure 5.1: Entity Relationship diagram for the relation 'Use case has scenarios, and each scenario is made of classes'

Example 2:
The timetable scheduling problem in a college environment is another case where representing all relationships and constraints cannot be done explicitly in an RDBMS. The time table scheduling problem involves, allotting hours for a particular class belonging to a course. Entities of the system include instructor, class, course, classroom, classlength and subject. One of the challenges in the construction of the timetable is to ensure that constraints on faculty, courses, classrooms, time-slots, or subjects are not violated. Constraints include:
a. A class is taught by one instructor only.
b. An instructor cannot teach more than one class at a point of time.
c. A class may use more than one classroom. However, a class may use only one classroom at a time.
d. No classes share a classroom at the same time.
e. A class cannot have more that one course at a time.
f. A class has a scheduled length of one/two hours.

Though the basic relationships of subject, course, teacher, classroom, period can be represented, the other constraints and preferred options cannot be explicitly represented in an RDBMS. As shown in the above examples, all constraints on the system cannot be captured using RDBMS. However, ontologies help capture constraints related to concepts in the system.

5.1.2 Ontology for Repository and Reasoning

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 5.1.1.

For the following disabilities [MW] 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. Different areas where ontologies have been used is discussed in the next section.

The use of analyzing the requirements of a domain through an ontology of software testing is to allow reasoning about the information represented [AvH04]. For instance:

a. 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 $a$, there is inconsistency.. This error can be detected on reasoning of test ontology.

b. Inferred relationships. If concept X is equivalent to concept Y, and concept Y is equivalent to concept Z, then X is equivalent to Z.

c. 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 $Y$. For example, consider two concepts, 'Student' and 'MCA Student' where 'MCA Student' is a subconcept
of 'Student'. Also consider that 'y' is an instance of 'MCA Student'. Then, it can be inferred that 'y' is a 'Student'.

d. 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 $A$.

5.2 State of the Art

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 [WC05, WCD05]. The ability of ontologies to query necessary and relevant information regarding the domain concerned is exploited. Dillon et al [DCW08] 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 [WC05, DCW08] 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 [DCW08]. 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 [DCW08]. 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 [DCW08, NPT08]. 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 [DCW08]. A software test ontology(SwTO), that deals with the software testing domain has been built by [BCO09]. The ontology is used along with a test sequence generator to generate test sequences to test the operating system domain, Linux. Zhu et al [ZH05] 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.

5.3 The Proposed Approach

One of the well-known methods to build an ontology is the Methontology strat-
egy [MF97]. 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, conceptu-
alization, formalization, integration, implementation, evaluation, documentation
and maintenance. Noy et al [NM01] follow an iterative design process to build an
ontology. A set of steps to build an ontology is proposed. This work adapts ideas
from both works.

5.3.1 Building an Ontology for testing

In this section, the steps discussed in Section 5.3 are applied to build an ontology
for testing.

5.3.2 Design of Ontology

5.3.2.1 Defining concepts

Steps described in building an ontology are applied in the context of software
testing and are explained below:
1. **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. 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'.

2. **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, priority*; different types of priority include *customer assigned priority* and *structure based priority*.

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

4. **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 concept it describes must be determined. The properties are attached to concepts.

5. **Creating instances**: Instances are individuals belonging to a concept.

5.3.2.2 **Defining relations between concepts**

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**

i. *UseCase* is a sub concept of *System*.

ii. Individuals of *UseCase* relate itself to individuals of the *Actor* concept through the *hasActor* property.

iii. A *UseCase* must be related to at least one actor in the *Actor* concept through the *hasActor* property.

**Use case: Properties**

i. *hasActor* is an inverse object property of *isActorOf*, whose domain is *UseCase* and range is *Actor*.

ii. *hasUseCasePrecondition* is a functional object property, whose domain is *UseCase* and range is *UCPrecondition*.

iii. *hasUseCasePostcondition* is a functional object property, whose domain is *UseCase* and range is *UCPostcondition*.
iv. *hasUseCasePriority* is a functional object property, whose domain is *UseCase* and range is *UCPriority*.

v. *hasUseCaseName* is an object property.

**Actor : Condition**

i. *Actor* is a sub concept of *System*.

ii. Individuals of *Actor* relate itself to individuals of *UseCase* concept through *hasUseCase* property.

iii. An *Actor* must be related to at least one *UseCase* in the *UseCase* concept through the *hasUseCase* property.

**Actor: Properties**

i. *associate* is a functional object property, whose domain is *Actor* and range is *Usecase*.

ii. *hasPriority* is a functional object property, whose domain is *Actor* and range is *ActorPriority*.

**Scenario : Condition**

i. *Scenario* is a sub concept of *System*.

ii. Individuals of *Scenario* relate itself to individuals of *UseCase* concept through *hasScenario* property.

iii. A *Scenario* must be related to at least one *UseCase* in the *UseCase* concept through the *hasScenario* property.

**Scenario: Properties**

i. *hasScenario* is an inverse object property of *isScenarioOf*, whose domain is *UseCase* and range is *Scenario*.

ii. *hasPrecondition* is a functional object property, whose domain is *Scenario* and range is *Precondition*.

iii. *hasPostcondition* is a functional object property, whose domain is *Scenario* and range is *Postcondition*.

iv. *hasPriority* is a functional object property, whose domain is *Scenario* and range is *Priority*.

**Activity : Condition**

i. *Activity* is a sub class of *System*.
ii. Individuals of Activity relate itself to individuals of Scenario concept through hasActivity property.

iii. An Activity must be related to at least one Scenario under a UseCase concept through the hasActivity property.

**Activity: Properties**

i. hasActivity is an inverse object property of isActivityOf, whose domain is Scenario and range is Activity.

ii. hasActivityPrecondition is a functional object property, whose domain is Activity and range is Precondition.

iii. hasPostcondition is a functional object property, whose domain is Scenario and range is Postcondition.

iv. hasPriority is a functional object property, whose domain is Scenario and range is Priority.

The ontology built following the steps above is shown in Figure 5.2 below:

![Class diagram showing relation among concepts of the ontology](image)

Figure 5.2: Class diagram showing relation among concepts of the ontology
5.3.3 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 [MAQ]. Other ontology languages that can be used include RDF, DAML and DAML+OIL.

Given the 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 [DuC]. 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 Figure 5.3.

- Input. The use case and activity diagrams serve as input to the transformation process. In this work, scenarios have been generated from activity diagrams\(^2\).

- 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.

- Output of the process is OWL specification. Ontology tools like Protege\(^3\)

\(^2\)It is to be noted here that scenario generation algorithm in this work generates scenarios for system testing

\(^3\)Protege is a free, open source ontology editor and knowledge-base framework. Available at http://protege.stanford.edu
Steps described in building an ontology (Section 5.3.1) applied in the context of software testing are explained below:

1. **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, priority*; different types of priority include *customer assigned priority* and *structure based priority*. Figure 5.4 shows some of the classes defined.

![Figure 5.4: Concepts in the ontology](image)

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

3. **Defining relations between concepts** Concepts and relations between the concepts are shown in Figure 5.6. For each concept, conditions are
defined as to how the concepts interact for realizing an objective.

4. **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 concept it describes must be determined. The properties are attached to concepts.

5. **Creating instances**: Instances are individuals belonging to a concept.
Figure 5.7: Properties of concepts

Figure 5.8: Instances belong to concepts
5.4 Querying the Ontology

As shown in Figure 5.10 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:

a. What are the use cases the include use case ’x’?

b. List all actors related to use cases and scenarios having priority ’y’.

c. 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’).

d. List all scenarios used by an actor.

Three examples querying the ontology is shown. In the first example(Figure 5.9 & Figure 5.10), use cases, 'PerformBilling', 'CreatePurchaseOrder' and 'MaintainInventory' include use case 'CheckInventory'.

![Figure 5.9: Querying the ontology - an example](image)

The constraint applied on relation 'include' is that it is transitive. Hence, the query 'UseCase and includes value CheckInventory' gives the above three use
Figure 5.10: Querying an ontology - List all use cases that include a particular use case 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. 5.11 show the results of the query.

A third example requires that all actors associated to 'PerformBilling' be listed. Fig. 5.12 show the results of the query.

5.5 Extension of Ontology

For test management, not only domain, but also the process of testing should be specified comprehensively so that stakeholders in testing can participate effectively. Besides the concepts obtained as shown in Figure 5.2, testing concepts outlined in
Figure 5.11: Querying an ontology - List all actors related to use cases 'Perform-Billing' and 'Manage User Account'

Figure 5.12: Querying an ontology - List all actors associated to a particular use case
the SWEBOK are considered. Concepts obtained from SWEBOK include TestPlan, Result, TestType, UnitTest, RegressionTest, SystemTest, IntegrationTest, Resource, Hardware, Software, People, Time, DefectType, TestSuite, TestResult and TestData. The concepts applied to build an ontology for testing is shown in Figure 5.13. The advantage of the ontology is in managing the process of testing by querying, reasoning and inferring on the concepts.

A few examples of queries possible on the ontology are:

- What are the resources required in a particular TestPlan?
- List all classes related to testing a scenarios.
- What are the scenarios involved in RegressionTest?
- Give the scenarios selected using a particular testing technique.
- What is the people resource required for a particular testplan?

5.6 Summary

Scenario management involves selecting a set of scenarios for testing in order to meet project management criteria. Knowledge about entities in the domain and interactions between them provide significant input for scenario selection. This chapter presented an approach to query knowledge captured related to a domain for the purpose of selecting scenarios for testing. To effectively carry out the querying an ontology of software testing using related UML artifacts are built. Actors, related functionalities(use cases), scenarios, activities, priority of use cases and scenario have been considered. Protege, an OWL editor is used for querying and reasoning as shown in Section 5.4. Ontologies also help different users of a system like architects, developers and test engineers have a common reference point. For this, concepts from SWEBOK have been adopted and added to the
Figure 5.13: Class diagram showing relation among concepts of the ontology
ontology. Thus, an ontology aids in better understanding of requirements and goals of the system.