First Steps Towards Conceptual Schema Testing

Albert Tort and Antoni Olivé

Universitat Politècnica de Catalunya
{atort,olive}@lsi.upc.edu

Abstract. Like any software artifact, conceptual schemas of information systems can be tested. Testing conceptual schemas has some similarities with testing programs, but there are important differences. We present a list of six kinds of tests that can be applied to conceptual schemas. Some of them require complete schemas comprising both the structural and the behavioral parts, but we show that it is useful to test incomplete schema fragments. We introduce CSTL, a language for writing automated tests of executable schemas written in UML/OCL. We sketch the implementation of a test processor to automatically execute CSTL tests as many times as needed, following the style of the modern xUnit testing frameworks.

Keywords: Conceptual modeling, Testing, UML/OCL.

1 Introduction

The need and the importance of software testing in information systems engineering is undisputed. We adopt here the precise and concise definition of testing proposed by Meyer: "To test a program is to try to make it fail", from which the goal of testing becomes "to uncover faults by triggering failures" [9]. Many other verification techniques are used or are in research and development, but, in professional practice, testing continues to be the dominant technique.

Currently, most work in conceptual modeling assumes that conceptual schemas are executable, and therefore they are software [6, 8, 10]. Then, a question naturally arises: how can we test conceptual schemas?

Most of the work in software testing assumes that the System Under Test (SUT) consists of programs (objects, components) that provide only a set of operations and testing a SUT means calling those operations with appropriate context and input parameters, and checking that they return the expected outputs. For example, the recent UML Testing Profile (UTP) is based on this assumption [1] and the same happens in popular testing frameworks like JUnit [3].

If a conceptual schema were like an ordinary program, then its testing would not be very different from testing code. However, a conceptual schema is knowledge or, more precisely, it is the general knowledge that an information system needs to know about the domain and about the functions it has to perform [11].

In this work we advocate the use of testing during the elicitation of the conceptual schema as an early error detection practice to help increasing software quality.

In our approach, the conceptual schemas under test consist of a structural (sub)schema and a behavioral (sub)schema. The structural schema consists of a taxonomy of entity types, a set of relationship types (either attributes or associations), the cardinality constraints of the relationship types, and a set of other static constraints formally defined in OCL. Entity and relationship types may be base or derived. The behavioral schema consists of a set of event types. We adopt the view that events are similar to ordinary entities and, therefore, that events can be modeled as a special kind of entities, which we call event entities [12]. Each event type has an operation called *effect*() that gives the effect of an event occurrence. The effect is declaratively defined by the postcondition of the operation.

Testing conceptual schemas is as important as testing programs in projects that follow OMG's Model Driven Development (MDD) approach [8], when the transformation from Platform Independent Models (PIM) to Platform Specific Models (PSM) is fully automatic. This requires complete conceptual schemas, that is, conceptual schemas that include all structural and behavioral aspects.

However, we have found that it makes sense to test also incomplete conceptual schemas, as a means to increase their quality [7]. Even small fragments consisting of a few entity and relationship types, integrity constraints and derivation rules can be tested to uncover their faults.

2 Testing conceptual schemas

In this paper, we adopt UTP's terminology and consider that a test case is a "specification of one case to test the system including what to test with, which input, result, and under which conditions....A test case always returns a verdict." The verdict may be Pass, Fail, Inconclusive, and Error [1]. In general, we consider that the verdict is Error when the conceptual schema or the test case is ill-formed (is not a valid instance of the corresponding metaschema). The verdict is Fail if the knowledge represented in the conceptual schema produces unexpected results according to the specified assertions. Otherwise, the verdict is Pass.

We analyzed that when we test a conceptual schema, a test case includes one or more of the following test kinds:

- **Check that a given IB state is consistent**. This kind of test can be used to check that (1) the whole set of constraints and derivation rules behave as expected; and (2) the set of constraints defined in the schema is strongly satisfiable (because there is at least one consistent and non-empty IB state).

- **Check that a given IB state is inconsistent**. A conceptual modeler writes this kind of test to check that (1) the OCL constraints behave as expected; or (2) the whole set of constraints and derivation rules behave as expected.

- **Check the contents of a given IB state**. This kind of test can be used to check that (1) the structural schema can be instantiated to represent a particular domain state; (2) one or more derivation rules derive the expected results; (3) an

OCL navigational expression yields the expected results; or (4) the effect of one or more domain events implies an expected result on the IB.

- **Check that a domain event may not occur in a given IB state.** Domain event types and queries may have constraints. The meaning is that the instances of those types or queries may only occur in the domain if the constraints are satisfied. A conceptual modeler writes this kind of test to check that (1) the OCL event constraints behave as expected; and (2) the whole set of constraints defined in the event or query does not allow its occurrence as expected.

- **Check that a domain event may occur in a given IB state.** This kind of test checks that the effect of a domain event occurrence is as expected. A conceptual modeler writes this kind of test to check that (1) the OCL event constraints behave as expected; (2) the whole set of constraints defined in the event behave as expected; and (3) the method and the derivation rules of the derived constant attributes and associations produce the expected results (satisfaction of postconditions and static constraints).

- **Check that a predefined query produces the expected results.** This kind of test checks that the effect of a query gives the expected answer. A conceptual modeler writes this kind of test to check that: (1) the query constraints behave as expected; (2) the effect of one or more previously occurred domain events has

3 The Testing Approach

Figure 1 shows the relationship between the definition and execution of a conceptual schema and the definition and processing of its tests. Our approach comprises the following main elements: *a language* to formally write tests, a *test processor* to interpret, manage and automatically run the collections of tests and an *information processor* for executing the conceptual schemas under test.

3.1 The CSTL language

The *Conceptual Schema Testing Language* (CSTL) is a textual procedural language for writing automated tests of executable conceptual schemas written in UML/OCL. CSTL is inspired in, and is an evolution of ASSL [4] but oriented to the testing activity. Moreover, we are designing CSTL in the style of the modern xUnit testing frameworks, by formalizing the test assertions [3]. By this way, tests expressed in CSTL may be automatically executed as many times as needed.

CSTL provide constructs for specifying the test kinds enumerated in Section 2. Note that all test kinds involve an IB state that must be specified by the conceptual modeler.

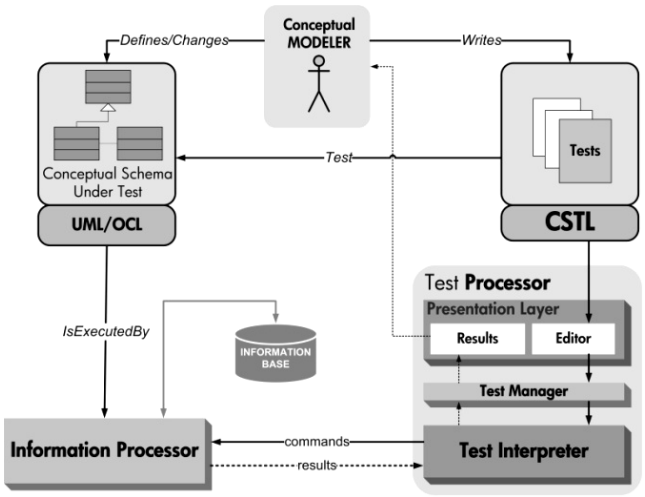

Fig. 1. Test processing and conceptual schema execution

A CSTL program consists of a fixture (may be empty), a set (may be empty) of fixture components, and a set of one or more test cases. The fixture is a set of statements that create a state of the IB and define the values of the common program variables. It is assumed that the execution of each test case starts with an IB state and the contents of the variables as defined by the fixture. With this assumption, the test cases of a program are independent each other, and the order of their execution is irrelevant.

In CSTL, there are three kinds of test cases: concrete, abstract and abstract invocation. A concrete test case is a set of statements that builds a state of the IB, defines values of its variables, and executes one or more tests of one of the six test kinds described in the previous section.

An abstract test case is a parameterized test case intended to be invoked one or more times in the same program. The parameters of an abstract test case may include fixture components and variables. A fixture component is a named set of statements that create a fragment of the state of the IB and define the values of a set of variables.

An abstract test case invocation is the invocation of an abstract test case with the desired values of the parameters.

The execution of a concrete test case or of an abstract test case invocation always returns a verdict. The verdict is obtained from the verdicts of the test kinds executed by the test case.

The result of the invocation of a CSTL program always returns also a verdict, which is obtained from the verdicts of its test cases.

3.2 The Test Processor

We are developing a *test interpreter* that reads a CSTL program and executes its statements. The test interpreter coordinates the execution of the tests (setting up fixtures, computing verdicts, and so on) and invokes the services of the information

processor to create, remove and change entities, attributes and associations of the IB, and also to evaluate OCL expressions over the IB. Moreover, it shows the results of the test execution. The test manager stores the CSTL programs and requests their execution to the test interpreter. The test manager also keeps track of the test results, and maintains test statistics.

Figure 2 shows the result of the execution of an example CSTL program. There are two test cases that have failed, and therefore the global verdict is Fail. Note that the test processor indicates the number of the lines where the tests have failed, and an explanation of the failure in natural language.

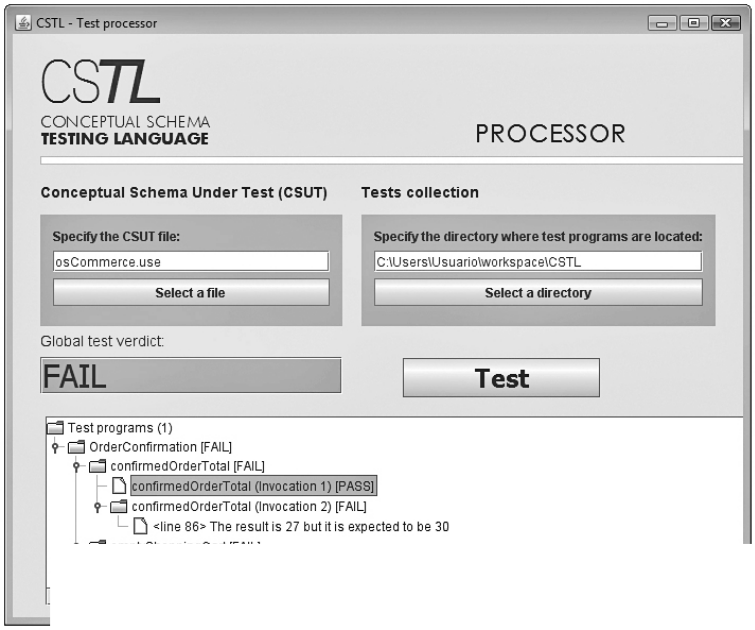

Fig. 2. Test processor screenshot

3.3 The Information Processor

A conceptual schema can be executable if there is a general-purpose *information processor* (virtual machine) able to behave according to the structural and behavioral rules defined in the conceptual schema [5].

We are implementing the information processor reusing USE [7] as much as possible. CSTL will be able to deal with richer conceptual schemas because: (1) it allows derived entity and relationship types; (2) in particular, it allows derived constant relationship types [11]; (3) events and predefined queries are conceptualized as entities and not as operation invocations [12]; and (4) it deals with conceptual schemas that allow multiple classification of entities.

4 Conclusions

We have seen that, like any software artifact, conceptual schemas of information systems can be tested with the goal of "uncover faults by triggering failures" [9]. We have shown that testing conceptual schemas has some similarities with testing programs, but there are important differences. We have presented a list of six kinds of tests that can be applied to conceptual schemas. Some of these test kinds require conceptual schemas that include all structural and behavioral aspects, but we have seen that it makes sense to test also incomplete conceptual schemas. Small fragments consisting of a few entity and relationship types, integrity constraints and derivation rules can be tested to uncover their faults and, therefore, to increase their quality [7].

We have introduced CSTL, a textual procedural language for writing automated tests of executable conceptual schemas written in UML/OCL. As far as we know, this is the first proposal of a language for testing conceptual schemas designed in the style of the modern xUnit testing frameworks. We are implementing a Test Processor that manages and executes CSTL programs.

We believe that our work opens new directions for research and development in conceptual modeling. First, it is necessary to develop a methodology for testing conceptual schemas. In particular, it seems interesting to develop a test-driven conceptual modeling methodology, similar to the popular Test-Driven Development [2]. Second, it is necessary to develop coverage criteria that measure the degree to which a conceptual schema has been tested. Finally, conceptual schema testing should be integrated with other existing verification techniques and tools.

5 References

1. Baker, P.; Ru, Z.; Graabowski, J.; Haugen, O; Williams C. Model-Driven Testing. Using the UML Testing Profile. Springer, 2008.
2. Beck, K.: Test-driven development: By example. Addison-Wesley Prof. (2003)
3. Gamma, E.; Beck, K. JUnit: A cook's tour. Java Report, pp. 27-38, 1999.
4. Gogolla, M.; Bohling, J.; Richters, M. Validating UML and OCL models in USE by automatic snapshot generation. Software & System Modeling, 4(4), 2005, pp. 386-398.
5. Griethuysen JJ van (ed) Concepts and terminology for the conceptual schema and the information base. ISO TC97/SC5/WG3, 1982.
6. Insfrán, E., Pelechano, V., Pastor, O. Conceptual Modeling in the eXtreme. Information and Software Technology, 44 (2002) 659-669.
7. Lindland, O.I.; Sindre, G.; Solvberg, A. “Understanding Quality in Conceptual Modeling”. IEEE Software, March 1994, pp. 42-49.
8. Mellor, S.J.; Balcer,M.J. Executable UML. A Foundation for Model-Driven Architecture. Addison-Wesley, 2002
9. Meyer, B. Seven Principles of Software Testing. IEEE Computer, August 2008, pp. 99-101.
10. Olivé, A. Conceptual Schema-Centric Development: A Grand Challenge for Information Systems Research. CAiSE 2005, LNCS 3520, pp 1-15.
11. Olivé, A. Conceptual Modeling of Information Systems. Springer, 2007.
12. Olivé, A,; Raventós, R. "Modeling events as entities in object-oriented conceptual modeling languages". Data&Knowledge Engineering 58 (2006) pp. 243-262.