APPLYING ADVANCED UML BASED TESTING METHODOLOGY TO E-LEARNING

Jinghua Gao, Eda Marchetti, Andrea Polini
ISTI CNR
Via Moruzzi 1, Pisa Italy
{Jing.Hua, Eda.Marchetti, Andrea.Polini}@isti.cnr.it

ABSTRACT

We present an ongoing experience in the application of UML based testing methodology in an e-Learning environment. In particular, we focus on the interaction of learning objects and Learning Management Systems (LMS). This paper reports the application of the proposed technology for test case generation of the SCORM system.

KEYWORDS

e-Learning technologies, UML based testing, automatic test generation, SCORM.

1. INTRODUCTION

The e-Learning technologies have been studied for over 40 years, but recently they have seen a growing interest mainly due to the widespread diffusion of internet, mobile devices, open systems. However the development of e-Learning applications is still an expensive task, in which the "reuse" of already developed materials and interoperability between different applications are crucial saving considerable amount of money. Focusing precisely on solving the interoperability problem different organizations, such as IMS [16], ADL [1], deliver specifications and standards, which regulate the data interchange. Unfortunately a general agreement on what "interoperability" exactly means is not yet achieved, and different suppliers still develop their e-Learning tools and resources, including additional features with respect to the basic characteristics provided by standard specifications. Obviously, this prevents the interoperability and data exchange of e-Learning system, and makes their combination the most critical part of the e-Learning program. In this situation a possible solution is to adopt an accurate testing process, which verifies precisely the behaviour and features provided by an existing e-Learning system (LMS in the following).

For this, adopting the definition of interoperability presented in [20] i.e.: "e-Learning systems are defined to be interoperable when they can exchange the necessary data, using a common system infrastructure resulting in the expected end-system behaviour", in this paper we describe a possible testing process based on the AGEDIS [2] methodology. We use the documentation provided by the ongoing TELCERT project [23] (Technology Enhanced Learning Conformance - European Requirements and Testing), and in particular the UML [26] description of LMSs contained into the Application Profiles [20]. Derived from a base specification these last in fact contain the useful information for: tailoring the conceptual data schema to the specific needs of a community; mapping the localized conceptual schemas to a generic binding; defining for a general LMS the set of abstract APIs[1] that can be invoked by contents and the corresponding behaviour [20]. Thus by using this abstract information we first develop a UML model of the system, and then automatically derive once for each Application Profile a set of test cases useful for verifying that the contents deployed on LMSs are compliant with the requirements expressed in the Application Profile. The combined use of a model and an automatic approach for test case derivation has various advantages such as:

[1] These have the role of abstracting from the real implementation of each LMS and defining a unique set of elements on which the test generation can be focused. They can be implemented in the LMSs, using various technologies and even different names for the methods.

- a reduction of the time and cost necessary for test cases generation and implementation when a real LMS is considered . It will be only necessary to map the already derived test cases based on the abstract APIs on the real one);
- an easier modification and reuse of the test suite. Each change in the system model will be automatically reported to the corresponding set of test cases and test cases derived for the unchanged components in the model can be reused.

Model based testing is a widespread adopted techniques and in literature there are different proposal which can mainly divided main into classes: state machine-based and scenario-based approaches. Among the proposals for test case derivation we only mention the work of Offutt and Abdurazik [17], who first used the UML Statechart; Liuying and Zhichang [15], who use a formal semantic; Chevalley and Thevenod-Fosse [8] who proposed a probabilistic method based on transition coverage; Antoniol et al. [3], who used UML statecharts for covering selected paths in a FSM; Bochmann et al. [6] using FSMs, Tretmans [24] and Fernandez et al. [9], who used the Input/Output Transition Systems (IOTSs).

To our knowledge model-based testing has not yet received proper consideration inside the e-Learning community and in the literature there are no experiences reported describing the application of one of the mentioned approaches. The available documentation is focused on a testing process conducted on the basis of the experience and the skill of test developers without adopting a specific model. This paper intend to be the first attempt at technology transfer, with the aim of bringing into the e-Leaning environment all the advantages of a structured and formal testing approach.

The paper is organized as follows: In Section 2 we describe in details the AGEDIS methodology. In Section 3, focusing on a specific case study, we present the modelling process adopted for defining in UML the abstract API and LMS services and the automatic test case generation.. The conclusion are instead in and Section 4

2. AGEDIS AND AML LANGUAGE

In our work we reused the results of the AGEDIS (Automated Generation and Execution of Test Suites for DIstributed Component-based Software) [2] project, that begun in October 2000 and is just finished and has developed a methodology and a set tools for the derivation of test cases from models.

AML (AGEDIS Modeling Language) [2] is the specific language defined and used within the AGEDIS project, and it is supported by the Objecteering UML Editor tool [18]. The syntax of AML is fully compliant with that used by UML 1.x, but it only uses a subset of the available UML diagrams, such as state machines, class diagrams, and object diagrams. In addition to UML, AML allows the definition of test directives used for annotating the design with special labels that add information to be reused for the test cases generation step. Test directives are a set of restrictions, constraints, and goals defined by the user to drive the test generator establishing useful test cases. Test directives include:

- *Coverage criteria*, which are used to generate a set of test cases with respect to a given coverage constraint. Coverage criteria are indicated as a set of expressions over object variables and expressed in a specific AML diagram called "*test purpose diagram*".
- *Test constraints*, which are used by the test generator during selection of the relevant execution sequences.
- *Test purposes*, which look like state machines highlighting a specific aspect of the behaviour that must be tested.

The AGEDIS methodology generates test cases applying an iterative process with the following six steps [25]:

- Build a behavioural model of the SUT (System Under Testing). AGEDIS uses the tool Objecteering together with the AML profile to create the system model.
- Annotate the model with testing information. It is possible to either describe the interfaces between the model and SUT, or to use test directives to annotate the model.
- Automatically generate a test suite. This is the core function of AGEDIS; a test case generator is able to automatically generate the test case from the developed model.
- Review the model test information and test suite with developers and customers;
- Execute the test suite automatically and log the results;
- Review of the results repeating steps 2 to 5 until the coverage and quality goals of the test are achieved.

In the next section we show the application of AGEDIS to a case study. However, with respect to the six steps above, we only report the results obtained running the first four and stopping the process to the generation of test cases since we do not have the required information for executing the tests.

3. CASE STUDY

In this section we present an application of the AGEIS methodology for testing the correct implementation of the APIs developed by a specific LMS. We consider in particular the Shareable Courseware Object Reference Model (SCORM) platform [21], which is one of the most accepted standard in the e-Learning community. In the following sections, by using the SCORM documentation we focus on the derivation of a proper AML model documentation, and the application of the AGEDIS tools for deriving the test cases.
It must be noted that on the SCORM web site [21] a test suite for verifying the conformity of the learning objects with the behaviour of specific LMS is already available, though it is unclear the test strategy adopted for deriving it: our overall impression is that skilled test developers, using their personal knowledge and previous experiences, developed the test cases. Even if (from a practical point of view) the ad hoc test case derivation can be very profitable, without a specific reference model the testing generation can be an unrepeatable process, which depends greatly on the talent of the involved testers.

We focused therefore on the interactions between the LMS, and the learning objects (in SCORM referred as Sharable Content Object (SCO)). Learning objects are considered the minimum unit that can be reused and assembled to structure a course. To enable communication between the LMS and the SCO, SCORM defines a set of APIs. These can be invoked by the SCOs for interacting with LMS by using get/set/commit methods. A common behaviour in this context is the learner's interaction with content objects when this is displayed by the web browser (SCORM targets, the Web as a primary medium for delivering instruction).

Derivation of the AML-Model

The first steps applying the AGEDIS methodology is the creation of an abstract model which represents the behaviour of the system that we intend to test. In our case adopting AML, using the available SCORM documentation and the information enclosed in [5], we developed the relative model.
We begin with the definition of a class diagram representing the components of the systems. In particular in the class diagram we enclosed four different components:

1. Content, which is the learning object that we intend to test;
2. JSWrapper, which is the API locator. It is responsible for finding the proper API on the APIs interface when the getAPI method is invoked by the Content component;
3. `API`, which represents the API interface implementation. It contains the methods that can be invoked by the learning objects (content). We consider here a simplified version of the API in which the error management is ignored. The methods in the API are: `lmsInitialise`, `lmsFinish`, `lmsCommit`, `lmsGetValue`, `lmsSetValue`.
4. `LMS`, which is the learning management system.

We then define an Object Diagram, which corresponds to a specific implementation of the environment described in the class diagram. Finally, following the sequence of calls admitted for a typical SCORM section, we define the relative State Machines, which simulate the systems behaviour. For the sake of clarity we decided to divide the state machine into three separate components (Figure 1), that will be integrated at run time by the AGEDIS tools into a unique diagram. Specifically, the State Machine (A) focuses on the overall system behaviour which always foresees the invocation of LMSInitialise and LMSFinish methods, at the beginning and end of every learning object section of interaction, respectively. Between these method calls, the invocations of the set, get, commit methods can be performed. These include a cycle of set or get or commit calls (State machine (B)) or any sequence or combination of these three methods (State Machine (C)).

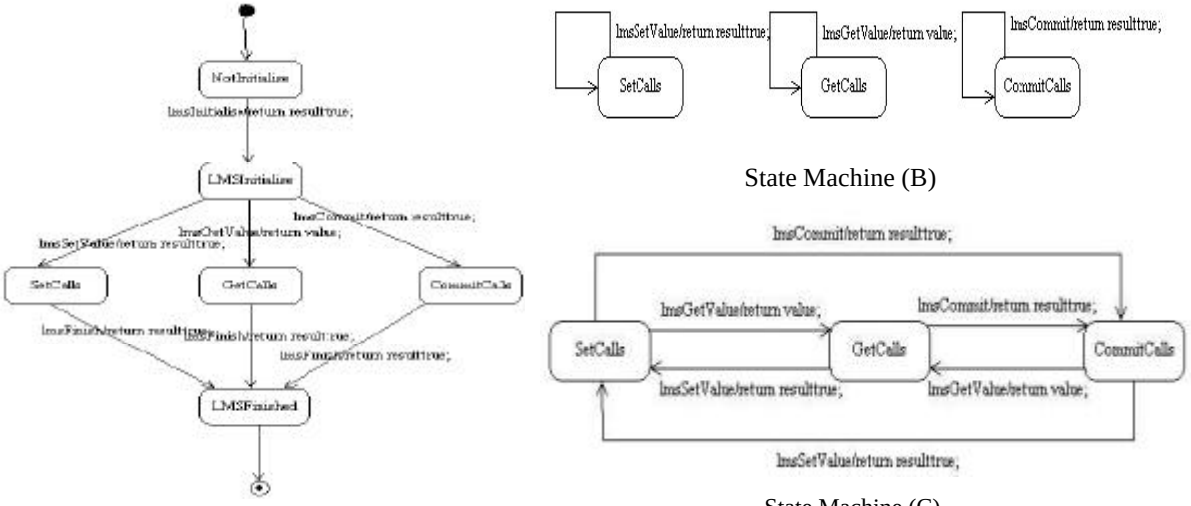

Figure 1 Different State Machines

Test Purpose Definition

In this section we describe the derivation of the AGEDIS test purpose, which focuses on the behaviour of the system. Graphically, it looks like a state machine, but the states considered are represented without actions or events. In particular, states from different classes can be used in one diagram and triggers do not represent processing of events by the local machine, but they match the transitions of the model. From a technical point of view the test purpose must be labelled with the stereotype <<test purpose>> to distinguish it from the state machine. The peculiarity of this diagram is that it is possible to create different test purposes, each one focused on a specific system function.

In our case study, due to the simplicity of the developed state machines, we decided to define a unique test purpose that merged all system functions (see Figure 2). The states must be opportunely labelled with specific marks (tagged values) that will be used by the test generator for deriving the test suite. In our case study these are:

- *Init:* indicating the initial state of the test purpose.
- *Accept*: indicating a successful termination of the test.

For the sake of simplicity in our case study, we ignored the error management and we used only the marks {init} and {accept}. Specifically because we are interested in testing all the functions of the system, we consider as initial and final state the state `tpnotinitialise` and `tpfinish` respectively. In the test purpose between these two states all the paths involving the `tpgetvalue, tpsetvalue` and `tpcommit` are admitted.

Considering instead the transitions, they can be labelled using AML language, by triggers and guards. The former describe the emission/reception of events at the specification level and are represented using regular form expressions such as:

- `sourceObject?targetObject.sigName()/opName()` indicating that targetObject must receive a signal or an operation call from sourceObject.
- `sourceObject!targetObejct.sigName()/opName()` the sourceObject must send a signal or an operation call to targetObject.

When the definition of the SourceObject and the TargetObject is irrelevant, it is possible to substitute their name with a "*" in the trigger expression. For instance, in the test purpose of the figure above we write `*!*.lmsInitialize.return(resulttrue)` in the first transition for indicating that we do not care which are the sender and the receiver of the event.

The guard is a condition used for limiting the situation in which a transition can occur. The value of the guard in a test purpose must match the guard value in the state machine. Otherwise there will be some errors when you compile the model. The transition in test purpose can be fired only when the trigger and the guard expression are satisfied.

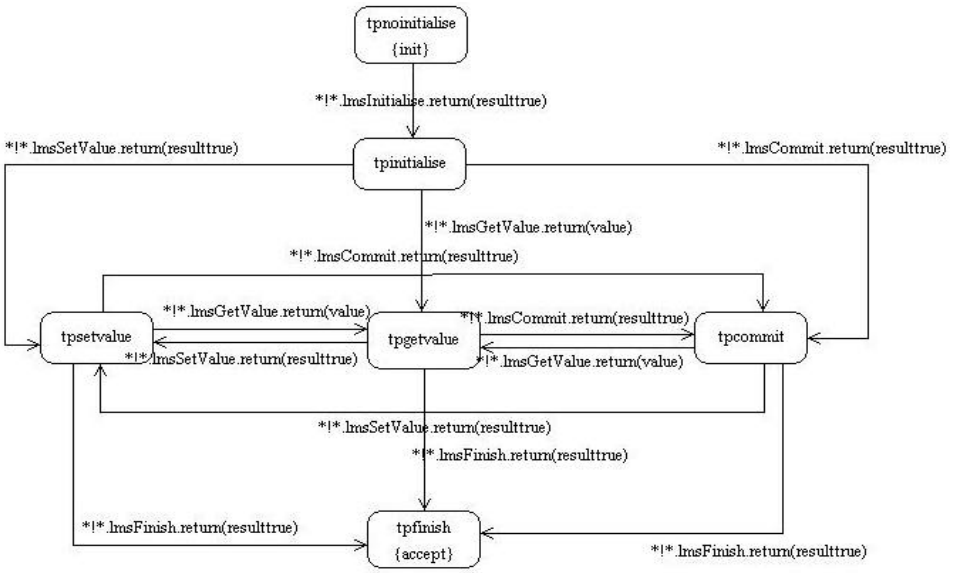

Figure 2 The test purpose diagram

Test Cases Derivation

Once the test purpose has been defined, it is possible to derive the test cases. To do this is necessary to compile the model using one of the available AGEDIS tool for turning it into Intermediate Format (IF), which is the internal language of AGEDIS. During this process, the information of the state machine and test directives are encoded and a test directive file (TE) is automatically produced for the corresponding test purpose. Using the IF and TE files the test generator is able to use the information of the model and the test purpose, and generating as many test cases as specified by the user. Referring to the case study presented in Section 3 and the test purpose of Figure 2 the generation of test cases is carried out considering all the different paths between the states labelled {init} and {accept}. In particular, each path is then transformed into a different test case. In Figure 3 we report the description of two of the test cases derived.

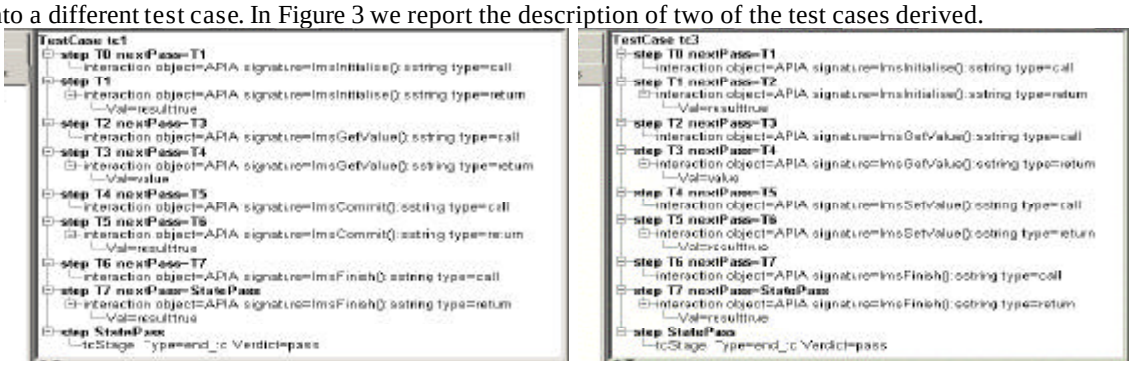

Figure 3 Test case A & B

4. CONCLUSION

We present our experience in applying model-based testing for testing the interaction between learning objects and a specific LMS. In particular, we focus on UML-based testing, adopting the AGEDIS methodology for test case derivation. To this end we describe in detail the necessary steps for modelling the system behaviour and consequently deriving the test cases. However, this is an ongoing experience, we are still investigating which could be the best approach for model –based testing in e-Leaning environment. Moreover considering in particular the SCORM application, we are planning to compare the performance of the automatically generated test suites with those provided by the SCORM documentation.

ACKNOWLEDGEMENT

The authors would like to thank the Telcert Project for the materials and the financial support. We also gratefully acknowledge Dr. Hartman of the IBM Haifa Research Laboratory and former leader of the Agedis project for providing the tool and support.

REFERENCES

[1] ADL Advanced Distributed Learning. Available at http://www.adlnet.org.

[2] AGEDIS documentation available at http://www.agedis.de

[3] Antoniol, G. et al, September 30-October 4,2002, A Case Study Using the Round-Trip Strategy for State-Based Class Testing, *Proceedings of IEEE ISSRE2002*, Germany, pp.383-397.

[4] Basanieri, F, et al. September 30 - October 4, 2002The Cow Suite Approach to Planning and Deriving Test Suites in UML Projects, *Proc. UML 02, LNCS* 2460, Dresden, Germany, pp. 383-397.

[5] Bell J., et al. Application Profile for UK FE to appear on http://www.opengroup.org/telcert/.

[6] Bochmann, G.V. et al. 1994Testing: Review of Methods and Relevance for Software Testing", *Proc. Int. Symp. on Soft. Testing and Analysis (ISSTA)*, Seattle, pp. 109-124.

[7] Briand, L. et al, 2002, A UML-Based Approach to System Testing, *Journal of Software and Systems Modeling (SoSyM)* Vol. 1 No.1 pp. 10-42.

[8] Chevalley, P. et al, 8-12 October 2001, A UML-Based Approach to System Testing, *Journal of Software and Systems Modeling(SoSyM)*, Vol. 1 No.1 2002 pp. 10-42.

[9] Fernandez, J. et al, 1997, An Experiment in Automatic Generation of Test Suites for Protocols with Verification Technology, Special Issue of *Science of Computer Programming*, Vol.29, pp.123-146.

[10] Fraikin, F. et al, September 2002, SeDiTeC - Testing Based on Sequence Diagrams, *Proceedings of IEEE CASE 02*, Edingburgh.

[11] Graubmann, P. et al, 2000, HyperMSCs and Sequence Diagrams for use case modeling and testing, *Proceedings of UML 2000 LNCS* Vol.1939, Pages 32-46.

[12] Harel, D. et al, 2003, Specifying and Executing Behavioural Requirements: The Play In/Play-Out Approach, *Journal of Software and System Modelling(SoSyM)*

[13] HEFCE99/39 Use of TLTP materials in UK higher education, 1999 on line at http://www.hefce.ac.uk/pubs/hefce/1999/99_39.htm

[14] Horton, W. et al, 2003, E-Learning Tool and Technologies, John Wiley & Sons.

[15] Liuying, L. et al, 22-25 September 1999. Test Selection from UML Statecharts, *Proceedings of 31st Int. Conf. On Technology of Object-Oriented Language and System*. Nanjing, China.

[16] LMS Global Learning Consortium, Inc. Available at http://www.imsglobal.org (Last updated on 14/9/2004)

[17] Offutt, J. et al, October 2000. Generating Test from UML Specifications. *Proceedings of UML 99*, Fort Collins, CO.

[18] Objecteering on line at http://www.objecteering.com

[19] Ryser, J. et al, June 2000, Using Dependency Charts to Improve Scenario-Based Testing, *Proceedings of TCS2000* Washington D.C..

[20] Smythe, C. State of the Art Report on Technologies and Techniques for Testing, to appear on http://www.opengroup.org/telcert/.

[21] SCORM documentation available at http://www.adlnet.org/index.cfm?fuseaction=scormabt

[22] SCORM 2004: January 2004 The SCORM Sequencing and Navigation, http://www.adlnet.org, Version 1.4.

[23] TELCERT project http://www.opengroup.org/telcert

[24] Tretmans, J., 1996, Conformance Testing with Labelled Transition Systems: Implementation Relations and Test Generation, *Computer Networks and ISDN Systems*, Vol,29, pp. 49-79.

[25] Trost, J. et al, June 2002. *AGEDIS Modeling Language(AML) Tutorial*, available at http://www.agedis.de

[26] UML Documentation available at http://www.uml.org/#UML2.0.

[27] UMLAUT Project, Available at http://www.irisa.fr/UMLAUT/