A TESTING METHODOLOGY FOR AN OPEN SOFTWARE E-LEARNING PLATFORM

Ana Cavalli, Stéphane Maag, Sofia Papagiannaki and Georgios Verigakis
GET / Institut National des Télécommunications
Laboratoire SAMOVAR / UMR CNRS 5157
9 rue Charles Fourier
F-91011 Evry Cedex, France
e-mails : {Ana.Cavalli, Stephane.Maag, Sofia.Papagiannaki, Georgios.Verigakis}@int-evry.fr

Fatiha Zaïdi
University of Paris Sud 11,
Laboratoire de Recherche en Informatique,
UMR CNRS 8623
Bat. 490 University of Paris Sud
91405 Orsay Cedex, France
e-mail : Fatiha.Zaidi@lri.fr

Abstract This paper presents an outline of a methodology for the test of an open software e-learning platform. The methodology includes a test generation method from UML descriptions, in particular Sequence, Activity, Class diagrams and Navigation maps. It also includes data modelling proposing two approaches: one based on OCL and the other on UML profiles. The studied open software e-learning platform is dotLRN, an open source enterprise-class suite of web applications and portal framework for supporting course management, online communities and collaborations. The work presented in this paper is being developed in the framework of the E-LANE[1] project, an European and Latin American collaboration for the creation of an open software integrated platform for e-learning.

Keywords: UML, dotLRN, XMI, OCL, testing methodology, test generation methods

[1] http://www.e-lane.org/

1. Introduction

In the last few years major progress has been achieved in the design of e-learning methodologies and platforms, particularly developing tools and methods to facilitate to users the access to these technologies. More recently, this evolution has strengthened the idea to develop open software platforms by integrating existing applications already used for e-learning. In this context, educational organizations can organize the teaching of a collection of courses using e-learning technologies, overcoming the geographical barriers, addressing to a larger audience and last but not least, by reducing the costs. To reduce the costs, organizations only need to deploy a powerful web server with the goal that the users may access these services using an open software platform connected to internet. Moreover, these platforms are designed to stimulate users to use and integrate new services.

This paper presents the work being developed in the framework of the European-Latin American New Education (E-LANE) project, an European and Latin American collaboration for the creation of an open software integrated platform for e-learning in which GET/INT is a partner. This project is part of the European Commission program @LIS Demonstration Projects. The base platform for the development of E-LANE is dotLRN, which is an open software platform composed by a suite of web applications and a portal framework of an e-learning system.

These open software e-learning platforms integrating existing applications developed by different teams need to be tested to insure that these applications are integrated together smoothly and correctly. The integration of new services by users also needs to be checked to guarantee the systems functionalities are preserved. In particular, these systems are exposed to many threats: a web server has to respond to requests from each node in the network. For instance, testing should guarantee that a user cannot gain permissions over the system he is not supposed to have. In addition to this, modern web applications are becoming increasingly complex and mission critical. This situation becomes even worse because of the lack of standardization in the web clients (browsers). Testing has to assert the system usability. For instance, an user is unable to complete a process because the content of the web page does not appear correctly or, due to a slow network connection, he may be unable to login because of an unrealistic timeout value. Our approach takes into account how these functional aspects have been implemented, checking that they conform to the specification.

In this paper, we propose a testing methodology for the dotLRN platform based on a test generation method and present some ideas of how

to test new services integrated by the users. We also give a short presentation of a new approach for testing e-learning services, which is complementary of the test generation method.

The methodology presented in this paper is inspired by the work presented in [1, 4], which describes how UML diagrams developed in the Analysis phase are analyzed in an automated way to produce test cases. In relation to this work the paper presents several new contributions. First at all, none of these work is well adapted to the test of a web application tool. In our work, we need to check the graphical user interfaces but also the content of the generated pages. There is an extension of UML for Web Applications but it is more focused on the Design View. One of the contributions of this paper is to provide an answer to these issues. Therefore, in order to produce diagrams which convey the information needed to feed our tests we considered the work presented in [8] and [7].

Other contributions given by this paper are the following: to present the techniques and tools which are used for the conformance testing of the platform. The methodology we propose includes the following steps: description of the web services in UML, in particular use cases using Activity diagrams, Sequence diagram, Class diagram and Navigation map; export of diagrams into a XMI format; parsing of XMI formats in order to generate test scripts; generation of tests scripts (using the different languages for scripts). This paper also presents data modelling proposing two approaches: one using Object Constraint Language (OCL) [10] and the other using a UML profile.

Finally, this paper presents the perspectives of this work, in particular we shortly introduces a new approach for testing that is complementary to the previous one presented in this paper. In this new approach, we propose to describe in OCL the expected properties of the web service and to check these properties on the service execution traces.

In the following sections the article first describes the dotLRN platform (section 2). Section 3 presents how to specify dotLRN using UML models for testing. In section 4, we describe the tools to automate the test generation from these models. Section 5 introduces the two approaches proposed for data modelling and section 6 presents the perspectives for this work. Finally, section 7 gives the conclusion.

2. The dotLRN e-learning platform

dotLRN [2] is a web based e-learning platform and portal framework, designed to support course management, online communities and collaboration. It is open source and built over the OpenACS [3], a toolkit for building scalable, community-oriented web applications.

In dotLRN there are three main portal types: *user*, *class* and *community* portals. The user portal is the private space that each user owns, while the class and community portals contain all the pages related to a specific class or community. Each of these three portal types is divided into four sections: the main space, the calendar, the files and the control panel. The pages in dotLRN are composed of *portlets*. Portlets are small portals that have some specific functionalities, like the forums, the news or the calendar.

A user portal is created automatically whenever a new user is registered in dotLRN, but the class and community portals are created by the *site-wide administrator* according to the needs of the users. When a new class or community portal is created, the site-wide administrator assigns one ore more users as administrators of this portal. For example, for computer science class portal, the administrators can be the professor and his teaching assistants, while for a photography group community portal, the administrator will be one or more students. The responsibilities of a portal administrator are to add content, customize the layout and decide the policy of the portal.

When a class or community has an *open* policy, any user can join, while when the policy is *closed* only the administrator can add users. A third policy exists, the *wait* policy, where any user can ask to join and then the administrator will decide to accept or deny this user.

3. UML models for testing

In order to derive our test cases, we need to grasp and describe the functionality of the system in a formal way. A model-based approach development of Web applications can be performed using UML techniques and UML notation. We use a design methodology which is based on a UML extension for hypermedia [9]. It consists of three main steps that constitute the conceptual model, the navigation map, and the presentation model.

The conceptual model is built taking into account the functional requirements captured with use cases. We discover the use cases of the

[2] http://www.dotlrn.org
[3] http://openacs.org/

system under consideration and document them in a *Requirements Document* which contains the scope of the system, the main actors, the use case diagram and textual description of each use case [6]. The output of this step is not directly used as an input to our test suite but it is important in order to design the UML diagrams which actually constitute our input data.

From this conceptual model, the navigation space model, also represented as a static class model, is constructed. It defines a view on the conceptual model showing which classes of this latter may be visited through navigation in the Web application. Finally, a dynamic presentation model is represented by UML sequence diagrams describing the collaborations and behaviors of the navigational objects and access primitives. In order to specify our model, the following diagrams are involved:

- Class Diagram to introduce the main classes of the system.
- Activity Diagram for each actor to display dependencies among the use cases.
- Navigation Map to provide information about the dynamic content of the web pages.
- Sequence Diagram for each use case describing the main and the alternative scenarios of the use case to represent the dynamic presentation model.

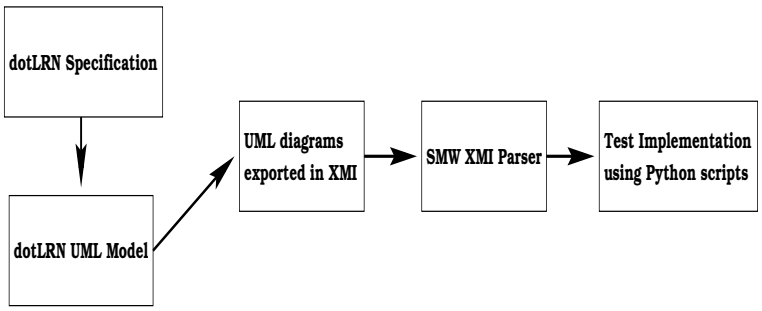

Figure 1. Outline of our methodology

From these three previous steps, we progress in our testing process. These diagrams are exported in an XMI format. This activity is supported by all the modern CASE tools like ArgoUML and Rational Rose. Afterwards the XMI is parsed and we produce a program which connects

to the web server and makes requests according to the given scenario in the Sequence Diagram. Finally, the web page of the response is examined to verify if it conforms with the specification. Figure 1 presents briefly the different steps of our study.

3.1 From Conceptual Model to Navigation Map

In order to specify the dotLRN platform we first need to build a conceptual model of the application domain taking into account the functional requirements captured with use cases. Techniques such as finding classes and associations, and defining inheritance structures are used.

Then, the Navigation Map of a web application introduced in [8] is used because it provides information about the dynamic content of each web page which is part of the system as well as the links between the different web pages. This information is essential during the parsing of the HTML pages (section 4.3).

The Navigation Map is a Class diagram where each web page is a class and a link between two pages is an association between the two respective classes. This extension of UML for web applications introduces a number of *stereotypes* (such as indexes, guided tours, queries and menus), *tagged values* and *constraints*.

The Table 1 summarizes the mapping we use between web entities and metamodels in a class diagram.

3.2 Modelling Use Case Dependencies

The use cases of a system are not independent. Apart from the *include* and *extend* relationships among them, there are also sequential dependencies. In order one use case to be executed, another should have taken place before. For instance, in dotLRN the user should login before being able to do anything else. Since the automation of the testing procedure is also of concern, we have to describe somehow these dependencies. We achieve this by introducing an activity diagram where the vertices represent use cases and edges are sequential dependencies between the use cases.

An edge in such a diagram denotes that the use case in the tail has to be executed before the use case in the head. *fork* and *join* are used when some use cases should be executed independently in order another one to take place. For instance, in dotLRN “Add subject” and “Add term” are two independent use cases which should be synchronized so as the “Add class” can be tested.

Web Entity	UML Metamodel	Tagged Values
web page	≪web page≫ class	**TitleTag:** the title of the page **RelativeURL:** the relative URL of the page **BodyTag:** the set of attributes for the ¡body¿ tag
page scripts	≪web page≫ class operations	
page variables	≪web page≫ class attributes	
link	≪link≫ association	
form	≪form≫ class	**Method:** GET or POST
form types	≪form≫ class attributes stereotyped by ≪form input≫	
portal page	≪portal page≫ class	**TitleTag:** the title of the page **RelativeURL:** the relative URL of the page **BodyTag:** the set of attributes for the ¡body¿ tag
portal element	≪portal element≫ class	**BelongTo:** the ≪portal page≫ it belongs

Table 1. Mapping Web Applications to UML Class diagrams. Stereotypes are presented inside ≪ ≫.

Furthermore, in this diagram we have included also the parameters of the use cases. The reason is that sometimes it is easier to realize the dependencies between the parameters of the use cases. For instance, in the above example, in order to add a new class, the administrator should provide information about the term(Term.name) and the subject of the class(Subject.name). As a consequence, there is a dependency between the "Add class" use case and the "Add Term" and "Add Subject" use cases.

Finally, in the diagram the use cases are organized in group according to the object are associated with. These objects are instances of the classes in the Class diagram. Figure 2 shows the respective activity diagram for the Administrator. According to this latter, "Add department" should precede "Add subject". Also, "Add term" and "Add subject" should occur before "Add class", and "Add user" and "Add class" should take place before the execution of "Assign user to class". Finally, "Manage User" depends on "Add User" since first the user should be added to the system and then the administrator can edit his profile and modify his permissions.

In the testing phase, before simulating the scenarios in the Sequence diagrams these activity diagrams should be scanned to obtain the sequence by which the use cases will be tested.

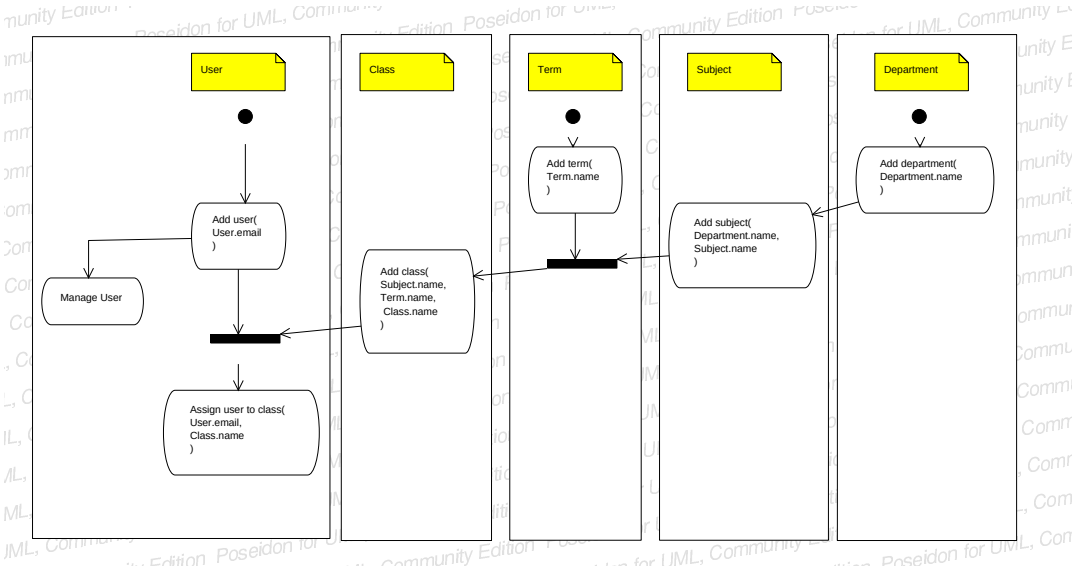

Figure 2. Activity diagram dependencies for "Assign user to class" use case

3.3 Sequence Diagram

In UML, a Sequence diagram realizes the interaction of objects via the interchange of messages in time. Similarly, as in activity diagrams the objects are instances of a class described in the Class diagram. Usually the sequence diagrams describe a single scenario.

We enumerate the messages as described in [4] so we can illustrate a number of alternative scenarios in the same diagram. According to this convention, capital letters denote alternatives (error messages). By adopting this tactic we can derive easily the different *Message_Sequences* [1] related to the same use case. Figure 3 shows the respective sequence diagram for the "Login" use case.

Our Sequence diagrams are also parameterized since input parameters can influence the execution and constitute separate *Choices* [1]. Such a parameter can be the email of a User. Whether this email belongs to a registered user (exists in the database) or belongs to a new user (does not exist in the database) determines what is going to happen later. In the former case the dotLRN page is displayed otherwise a warning appears in the Log In page.
During the testing procedure, if there are such branches and parameters

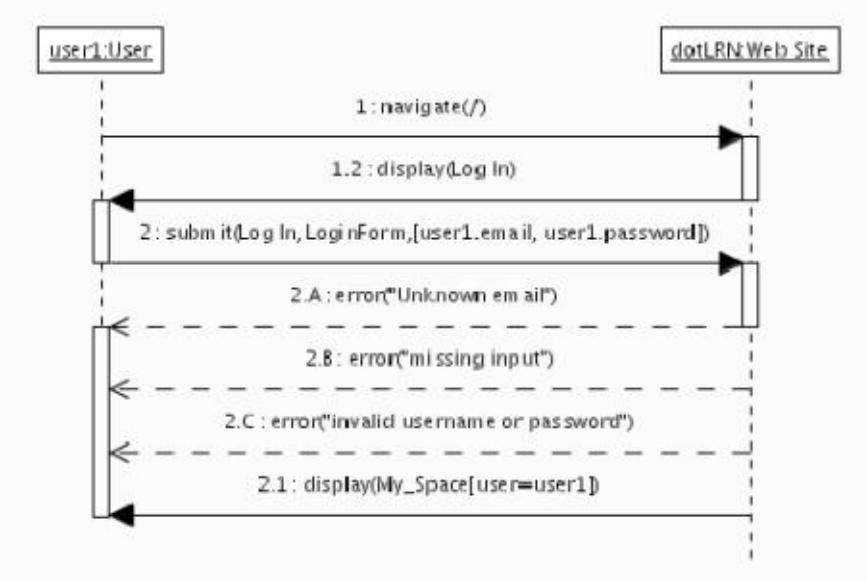

Figure 3. Sequence diagram for "Login" use case

then the produced program has to fork to test all the different possibilities.

Table 2 summarizes the actions we use in the Sequence diagram organized as HTTP requests of the user and possible HTTP responses returned to the user by the server, since the success or the failure of our tests depends upon these requests and the respective responses. Since the system under testing is a web application there are three possibilities for the user: either navigates to a URL or requests a web page through another one (clicks on a link to the wanted page) or submits information by filling an HTML form. The system answers either directly by returning the requested page (display) or by giving an error message.

4. Automating the test generation

To automate the test generation, our goal is first to parse the UML diagrams obtained from the previous steps. Therefore, based on these diagrams, we generate the necessary requests to the dotLRN server and then check if the server's replies are as expected by the previous models. Since dotLRN is a web application, the requests are HTTP requests that simulate a user navigating the site through a web browser. The possible actions are to fill and submit a form, to click on a link or to navigate

HTTP request	**HTTP possible responses**
navigate(url:String): User makes an HTTP request for a url	**display(page:WebPage)**: Web server returns the requested web page
link(target:String): User clicks in a HTTP link	**display()ege:WebPage**: Web server returns the target web page
submit(page:WebPage, form:Form, data:List): User submits an HTTP form	**display(page:WebPage)**: in case of legitimate input the web server responses with a new web page **error(msg:String)**: in case of wrong input the web server responses with the previous page displaying a warning message

Table 2. Actions of the Sequence Diagram

to a given URL. Similarly the server's replies are HTTP Responses that can either contain an HTML page, a redirection to another URL or an error message. Assuming the first case, the HTML page of the response has to be parsed to see if its contents are the expected ones.

Based on these requirements, we had to choose the components that were required to build our test suite.

4.1 The programming language

Since we are dealing with UML, it would be more efficient to choose an object-oriented language. We also wanted this language to provide easy string handling and a high level of abstraction for network communications. The programming language that we found as the most suitable was Python [5]. Python is a modern object-oriented language which combines remarkable power with very clear syntax. Its built-in modules provide numerous functions that facilitate string handling and networking.

4.2 Parsing and executing the UML

To parse the UML diagrams we could either use the API of a UML tool, or export the diagrams in an XMI format that would allow to parse them using an XML parser. XMI is a standard created by the Object Management Group (OMG) to represent UML diagrams in XML. Exporting to XMI was the solution we preferred since it does not tie us to a specific tool.

Although we could use any XML parser to parse the XMI, due to the high complexity of the standard we decided to use a specialized

XMI parser. The one we used was the parser included in the System Modelling Workbench tool [4]. It is free, open-source and also is written in Python, making it easier to integrate with our code. Being open-source it also enabled us to fix some incompatibility issues that appeared when used with XMI produced by the Poseidon tool.

4.3 Parsing the HTML pages

Since HTML mixes presentation and content data, the HTML output of dotLRN does not allow us to extract the information we want without first looking the implementation details. To avoid this we need to change the page templates of dotLRN in order to provide the data in a more formal way. We achieve this by adding *id* attributes to the tags we want to query. For example, to the td tag that contains the user's name in the user pages will have an attribute *id="username"*. That way we can query any page independently of the implementation of the layout of the page.

4.4 Example implementation

In this section we give the skeleton of a possible solution. In the code that follows we have left out some code (mainly some functions) in order to reduce the size and increase the clarity.

```
1  from urllib import urlopen
2  from smw.io import XMIStreamer
3  from smw.metamodel import UML14
4
5  class TestSuite:
6      returnedPage = None
7
8      def validateLink(self, link):
9          operation = self.getOperation(link)
10
11         if operation.name == "navigate":
12             url = self.getOperationParameters(operation)
13             fd = urlopen(serverbase + url)
14             self.returnedPage = fd.read()
15             fd.close()
16
17         elif operation.name == "display":
18             params = self.getOperationParameters(operation)
19             pageTemplate = generatePageTemplate(params)
20
21             parser = dotHTMLParser(pageTemplate)
22             parser.feed(returnedPage)
23
```

[4] http://www.abo.fi/~iporres/html/smw.html

```
24      def execute(self, source):
25          xmi = XMIStreamer(UML14)
26          fd = open(source, "r")
27          model = xmi.loadFromStream(fd)
28          fd.close()
29
30          sequenceDiagram = self.getSequenceDiagram(model)
31          for link in self.getLinks(sequenceDiagram):
32              self.validateLink(link)
```

Function *execute* (line 24) is the main function of the class and it reads the XMI code from the file defined in the variable *source* that is given as a parameter. It then isolates the Sequence Diagram (for this example we assume that only one exists) and then validate one by one all its messages (links). All the getX functions (like getSequenceDiagram and getLinks) are trivial to write and they consist of navigating through the structures generated by SMW to get a specific data. They are assumed to be defined inside the class.

The validation of each link depends on the operation. If the operation is *navigate* (lines 11-15) then we have to extract the destination URL from the parameters and then get the requested page. We assume that destination is a relative URL, so we use the *serverbase* variable (line 13) to make it absolute. The page is then kept in the *returnedPage* variable to be used by the following commands.

In the case of a *display* operation (lines 17-22), we create a template of the page based on the operations parameters and then use our HTML parser to compare the returnedPage with the template. The skeleton of the parser is as follows.

```
1   from HTMLParser import HTMLParser
2
3   class dotHTMLParser(HTMLParser):
4       pageTemplate = None
5
6       def __init__(self, pageTemplate):
7           self.pageTemplate = pageTemplate
8
9       def handle_starttag(self, tag, attrs):
10          for attr in attrs:
11              if "id" in attr:
12                  validateElement(tag, attr)
```

The *dotHTMLParser* class inherits the HTMLParser class and overrides the *handle_starttag* function to search for elements that have an *id* attribute. Every such element will be validated according to the pageTemplate that was given during the instantiation (the details of the validation are not shown).

Similarly to the two example operations we can write the code to handle the rest of the supported operations.

5. Inserting Test Data into the Model

In order to execute our tests and observe the behavior of the system it is necessary to substitute the formal parameters by the actual values. For instance in case we want to test whether a user with a valid email and password can log in successfully in the system, we have to provide to the test a valid pair of email and password. In analogous works the tester provides this input through the user interface of the test suite before the execution of each test. In our case since we are interested in automatically creating some scripts which will be immediately executed we would like to integrate somehow this information in our model.

Below we present two different approaches on how to deal with this problem.

5.1 Modelling Test Data with OCL

For that purpose in [11] OMG defines a Data Pool as a collection of values which is used by test components as a source of values for the execution of the test cases. Unfortunately they do not describe how these values can be integrated into the model but they just propose that they can be represented by utility parts or be logically described by constraints. In [3] the authors use OCL constraints expressed in a static model in order to identify the test domains from which test cases can be selected. However this approach requires extra effort in order to discover these test domains and also to extract random values from them. However, we think it is possible to use OCL expressions in order to directly assign values to attributes and to parameters and relate them to the model entities. In OCL, through *invariants* and *init* expressions one can assign values to an attribute but since these constraints have to be always true all the instances of an entity will hold these values. This is different from what we need in our case since we want to define different attribute values for different instances. For instance, we need two instances of a class user to have different values for the email and the password.

However, in a "let" expression we can define a variable which contains all the related values. Since we want these variable to be visible outside the specific constraint we define them inside a "definition" constraint. According to the specification, a "definition" constraint must be attached to a Classifier. All variables and operations defined in this constraint are known in the same context as where any property of the

Classifier can be used. Moreover, OCL 2.0 introduces Tuples in which several values can be composed. Combining all these, we could associate a classifier with a number of values of different type which represent an instance. For example, in case we have defined a class "User" with two attributes "email" and "age" and we want to represent two user instances "user1" and "user2" we could define the following OCL expression:

```
context User
def: let user1 = Tuple{email: String = 'Sofia', age: Integer = 12}
     let user2 = Tuple{email: String = 'Georgios', age: Integer = 5}
```

The advantages of using OCL is that OCL relies in types of UML diagrams so it is strongly related to our model and also it is easy to be checked for syntactic correctness and for consistency with the model using either the OCL tools integrated into the CASE tools or more sophisticated OCL parser. Moreover OCL expressions appear in the XMI representation of the model. However, Tuples are only introduced in OCL 2.0 and many CASE tools like Poseidon and ArgoUML do not support this version.

5.2 Modelling Test Data with UML Profiles

Another approach to the problem of representing data values in UML is the use of a UML Profile. UML Profiles is an extension mechanism provided by UML that enables us to add specific features. UML profiles are implemented using *stereotypes* and *tagged values*.

Different solutions could be proposed for the design of a suitable UML Profile, but our solution will focus on simplicity and ease of implementation. When we want to represent an instance **I** of a class **C**, we can define **I** as a new class having the same attributes and operations with **C**. Each attribute of **I** will have a tagged value containing the actual value of the respective attribute of **C**. If we want a tagged value to reference to an instance **J** of some other class then we use the notation /J. Additionally, to show that **I** is of type **C**, it will have a stereotype ≪C≫.

This method requires some extra development effort in order to check the correctness of the given values. However, contrary to the first method, it can be used with older versions of UML that are supported better from most of the currently available open source CASE tools.

6. Perspectives

The software tool presented in the previous sessions perform the parsing of the UML specification, the generation and translation of the tests

to the XMI form and their execution on the dotLRN platform. Another option that we have is in the way we execute the UML diagrams, that is it may gives us also another set of possibilities. The approach that we introduced automatically parses and executes the UML diagrams. However it could be equally possible instead to generate the code that will do the specified test. The generated code could be written in any language and will have the benefit that it can be used to run the tests from any computer, since it will run without the need of our software. Of course this will need further development effort, but since it is independent of the overall design it can be left for later stages of the project.

Concerning the test, in a first step, we started with the test of communication interfaces and user requirements. Next steps will be the test of authentication mechanisms and application contents (for instance, to check the content of the required web page). For that purpose, we need to treat data at the specification level. Indeed, to test if a user is permitted to access to the system, we need to check if it enters his proper login and password. To deal with this problem, two approaches were proposed: one based on OCL expressions and another one on UML profiles. However, both present advantages and drawbacks and we plan to investigate further both solutions.

Another issue that we still investigate is related to a new test method. The method presented in this paper is based on a UML description of the system to be tested. Experimentation results, with this approach, are very promising. The automation of the testing procedure reduces the time and costs to produce e-learning reliable software tools. In addition, the test of users requirements contributes to the design of tools with easy and convivial interfaces.

Nevertheless, we decide to propose another step-wise approach to automate the test generation and execution. This approach has no interaction with the system under test as the test generation method presented in this paper. Moreover, we do not need to use sets of UML diagrams. This new approach is well fitted for service developers that do not intend to spend time at the analysis phase. The approach is based on the verification of service properties directly on the service trace execution. For that purpose, we install a *sniffer* between the client and the server to obtain dynamically all the *html* pages of each request on the service. At the specification level, we associate to each class and each operation, properties that have to be verified once the execution of the service is performed. These properties are written using the OCL language and represent the requirements of the web service that we want to test. This language is not usable straightforward. We need to transform it in order to check it on the traces. Thus, we translate the OCL described prop-

erties in regular expressions. Afterwards, these regular expressions are those that we will check on the real traces execution. At GET/INT, we have developed a tool to perform this verification [2].

This approach using OCL constraints allows to define formally functionalities that services are intended to supply. Furthermore, this solution can also be used for the test of new services. Using this approach, we are able to check if the new added service (i.e. new functionality) works properly according to the OCL defined constraint. Moreover, we are able to test that existing services are preserved, performing regression tests by reusing OCL constraints already checked.

7. Conclusion

We have presented in this paper a new approach to test the functionality of an e-learning, web based system, the dotLRN platform. This platform presents the advantage to be an open source toolkit for building scalable, community-oriented web applications.

The method and the software tool we propose has been applied to dotLRN platform but they are generic enough and can be applied to other e-learning and web based systems.

The test method is based on the test of objectives, which are selected taking into account the experts and designers advice. These tests are based on use cases and cover all relevant aspects of the system behavior. Even if we cannot guarantee a total error coverage, it can be guaranteed for the selected tests. We propose a UML models technique for testing, that relies on a design methodology divided in three main steps, i.e. the conceptual model, the navigation model and the presentation model.

References

[1] F. Basanieri, A. Bertolino, and E. Marchetti. The Cow_suit Approach to Planning and Deriving Test Suites in UML Projects. *Proc. Fifth International Conference on the Unified Modeling Language - the Language and its applications UML 2002, LNCS 2460, Dresden, Germany*, pages 383–397, September 2002.

[2] Emmanuel Bayse, Ana Cavalli, Manuel Nunez, and Fatiha Zaidi. A passive Testing Approach based on Invariants: Application to the WAP. *Computer Networks journal*, 2003.

[3] Mohammed Benattou, Jean-Michel Bruel, and Nabil Hameurlain. Generating Test Data from OCL Specification. In *Proceedings of the ECOOP'2002 Workshop on Integration and Transformation of UML models (WITUML'2002)*, 2002.

[4] L. C. Briand and Y. Labiche. A UML-Based Approach to System Testing. *Software and Systems Modeling*, 1(1):10–42, 2002.

[5] Jr. Drake Christopher A. Jones, Fred L. *Python & XML*. O'Reilly & Associates, 1st edition, 2001.

[6] A. Cockburn. *Writing Effective Use Cases*. Addison-Wesley, 1st edition, 2000.

[7] J. Conallen. Modelling Web Application Architectures with UML. *Communications of the ACM*, 42(10):63–70, 1999.

[8] J. Conallen. *Building Web Applications With UML*. Addison-Wesley, 2nd edition, 2002.

[9] N. Koch, H. Baumeister, R. Hennicker, and L. Mandel. Extending UML to Model Navigation and Presentation in Web applications. *Proc. of Modeling Web applications in the UML Workshop, UML2000*, 2000.

[10] UML Object Constraint Language (2.0 submission). http://www.omg.org/cgi-bin/apps/doc?ad/03-01-07.pdf.

[11] UML Testing Profile (Draft Adopted Specification). http://www.omg.org/docs/ptc/03-07-01.pdf.