Model Driven Testing of Web Applications Using Domain Specific Language

Viet-Cuong Nguyen
Department of Computer Science and Engineering
Faculty of Electrical Engineering
Czech Technical University in Prague
Prague, Czech Republic

***Abstract*—As more and more systems move to the cloud, the importance of web applications has increased recently. Web applications need more strict requirements in order to sup-port higher availability. The techniques in quality assurance of these applications hence become essential, the role of testing for web application becomes more significant. Model-driven testing is a promising paradigm for the automation of software testing. In the web domain, the challenge however remains in the creation of models and the complexity of configuring, launching, and testing big number of valid configuration and testing cases. This paper proposes a solution towards this challenge with an approach using Domain Specific Language (DSL) for model driven testing of web application. Our techniques are based on building abstractions of web pages using domain specific language. We introduce WTML - a domain specific language for modeling web pages and provide automatic code generation with a web-testing framework. This methodology and techniques aim at helping software developers as well as testers to become more productive and reduce the time-to-market, while maintaining high standards of web application quality.**

***Keywords*—Domain specific language (DSL); model-driven development; model-driven testing; WTML**

I. Introduction

Advances in web-based technologies today has led to the rapid growth in the number of web applications used in business. As the demand for mobility and internet-of-things requires more complexity in web applications, the existing testing frameworks used to test software system struggle to get up to speed. Methods from model driven can support the rapid evolution of such system by building an abstract model of a web application and use the created models instead of specific code to generate tests. In general, the model of the web application does not need to include all the details of the implementation, but should be precise enough to guarantee that the test cases represent actual use scenarios of the web application [1].

In this paper, we present an attempt to build an approach using Domain Specific Language for model driven testing of web application.

Our techniques are based on building abstractions of web pages and modeling state-machine-based test behavior using domain specific language. This is used to form a more generic testing framework that can apply to many web-based systems to save time and cost.

This paper is structured as follows: In the next section, we review some knowledge of Model-driven Development (MDD), model-based testing and domain specific language as background information. The subsequent section discusses the current challenge to build a testing platform that can automate the process from development to execution. In the next section, we introduce WTML (our designed DSL) for test modeling and test development of web-based applications. In the last section, we present some conclusions on the methodology of using a domain specific language in model-driven testing of web applications.

II. Background

Automated model driven testing has received much attention in recent years, both in academia and in industry. This interest has been stimulated by the success of model-driven development in general, by the improved understanding of testing and formal verification as complementary activities, and by the availability of efficient tool support [2]. Model driven engineering approach as a methodology could be described as follow:

A. Model driven engineering

Model-driven engineering (MDE) is a software development methodology, which focuses on creating and exploiting domain models. Models can be perceived as abstract representations of the knowledge and activities that govern a particular application domain. Models are developed though-out various phases of the development life cycle with extensive communication among product managers, designers, developers and users of the application domain. MDE aims to increase productivity by maximizing compatibility between systems, simplifying the process of design and promoting communication between individuals and teams working on the system [3].

The Object Management Group's (OMG) initiatives on MDE contain the Model-driven Architecture (MDA) specification. MDA allows definition of machine-readable applications and data models that enable long-term flexibility with regards to implementation, integration, maintenance, testing and simulation [4] [5]. There are two main modeling classes in MDA:

- Platform Independent Models (PIMs): these are models of the structure or functionality, which are independent of the specific technological platform used to

implement it.

- Platform Specific Models (PSMs): these are models of a software or business system, which are bound to a specific technological platform.

In the MDA, models are first-class artifacts, which are later integrated into the development process through the chain of transformations from PIMs through PSMs to coded application. The mapping and transformation between PIMs and PSMs are based on meta-model concepts. These concepts can be described by technologies such as Unified Modeling Language (UML), Meta Object Facility (MOF) or Common Warehouse Meta-model [3], [6], [12]. These languages are considered as general-purpose modeling languages. Currently, there are many challenges in implementing model driven testing due to the lack of standardization and tools. There are specific desired aspects for each application within its domain and this makes it difficult to design a tool that can be applied to all situations.

B. Model based testing

Model-based testing is application of model-based design for designing and optionally also executing artifacts to perform software testing or system testing. Models can be used to represent the desired behavior of a System Under Test (SUT), or to represent testing strategies and a test environment.

A model describing a SUT is usually an abstract, partial presentation of the SUT's desired behavior. Test cases derived from such a model are functional tests on the same level of abstraction as the model [10].

Currently, testing usually comprises between 30% and 70% of all software development projects. Hence, a good testing methodology and toolset will enable software developers and testers to become more productive and reduce the time-to-market, while maintaining high standards of software quality.

The purpose of the model-driven testing in the web domain is to provide a framework that helps developers perform the following tasks:

- Create models of web applications or pages: This enables developers to create the abstraction of the components. Developers can later use the model created as a skeleton for the test project. In this way, the test plan can be reviewed and simulated to discover problems in the implementation or model before the actual code is ready for test.
- Model behaviors: The behaviors and interactions of the web application are modeled using the modeling language to later support test case generation. These behavior models simulate the features of the web application.
- Generate test cases for the web components. The tools generate tests using data from the component (page) models and the behavior models. It is often a good practice to have the test cases that cover all required test specifications.

Test execution: The generated tests can be later executed either manually or automatically by some triggers. This test execution automatically compares the observed results with the results predicted by the model. Thus, developers can walk through a unit test case to examine each test interaction and identify where the test failed.

C. Domain specific language

In software development and domain engineering, a domain specific language is a programming or specification language dedicated to a particular problem domain, a particular problem representation technique, and/or a particular solution technique. The concept is not new. Special-purpose programming languages and all kinds of modeling or specification languages have always existed, but the term has become more popular due to the rise of domain specific modeling [7].

Adoption of domain specific language can be a solution to several problems encountered in various software development aspects. A DSL can reduce the costs related to maintaining software [8]. In comparison to other techniques, DSL is considered as one of the main solutions to software reuse [9]. On the other hand, using DSL also promotes program readability and makes its understanding easier. This enables users without experience in programming to create the models or programs as long as they possess knowledge of the targeted domain.

Another advantage of a DSL for modeling is the ability to generate more verification on the syntax and semantics than a general modeling language. This can reduce errors (and burden) on the debugging process.

III. CHALLENGE

The process of web application development starts with concepts, mock-ups and requirements. After that, following a lot of iterations, more and more mature prototypes are gradually created towards a working solution. Testing needs to be performed within every iteration in this process. This nature makes testing web applications a routine task from designing the tests to tests execution and report. When maintaining such systems, any change to the system also requires the execution of a complete regression test. Therefore, there is a need to build a testing platform that can automate this testing process from development to execution.

There exist many model-based testing approaches and tools that vary significantly in their specific designs, testing target, tool support, and evaluation strategies. In the web domain, there is a noticeable increase in the number of model-driven testing techniques in recent years. Firstly, the challenge in this area is to have a good design of a modeling language that used to represent the system. Secondly, there is the challenge for effectively defining the process of test case generation and evaluation. There are several aspects of a model-driven testing technique that need to be considered:

- Effective Modeling Language: The modeling language used to model system, can be a generic UML approach or a domain specific language, should bring up good solution on the web domain while being easy to read

and to understand. This language needs to be effective and designed with agility support to ensure that models can adapt to changes seamlessly.

- Automation: This is an important aspect in model driven development, it is the ability to generate final artifacts from high-level specifications. Automation also enables test case generation and execution mechanism to perform easily without manual refinements.
- Good Tool Support: The tool chain and platform support is essential for any approach. This allows the integration with other parts of the development process. This means that the platform should provide tools for editing, debugging, compiling and transformation. The tools should also be able to be integrated together with other languages and platforms without a lot of effort.

Although there exist many techniques that tend to vary significantly in their design, they usually don't provide adequate results in every applicable domain [11]. There are also challenges in other aspects of the modeling process. On one hand, the model has to be written in a notation powerful enough to describe any elements of the web page. At the same time, it has to be abstract enough to ease the process of model creation and promote software reuse.

IV. Our Approach: DSL for Web Page Modeling

Our approach is based on the principle of raising the level of abstraction by modeling web pages and describes their behaviors using the theory of State Machines. In order to check the conformance between the application and the model, the automated process for generating test cases from the model is used. Our approach uses DSL to develop the testing model together with the functional web page model development. We aim at introducing a DSL and the tool set that fit for this purpose.

In this approach, designing a new DSL with the support for modeling at a good abstraction level is crucial. This DSL can later be used for automatic generation of the model artifacts and code that implement the services. In theory, a general modeling language could also be used for this purpose but an appropriately designed DSL can perform the same job much more effectively.

There are three essential requirements to the DSL design that we aim to achieve during the creation of a DSL to ensure the quality of the language. Firstly, the language needs to be effective, while being easy to read and to understand. Secondly, as the modeling language can raise the level of abstraction away from programming code by directly using domain concepts, automation needs to be achieved to generate final artifacts from these high-level specifications. This automatic transformation at the same time has to fit the requirements of the specific domain. Finally, the DSL has to be able to provide support via tools and platforms. The DSL needs to be able to integrate with other parts of the development process. This means that the language is used for editing, debugging, compiling and transformation. It should also be able to be integrated together with other languages and platforms without a lot of effort.

The starting point for a DSL for web page modeling is an abstraction of a web page. This abstraction model comprises the effective elements that are involved in the testing process and, optionally, the behavior of the transitions to be simulated and validated during the test execution. Following diagram depicts the simplified syntax rules of a page model:

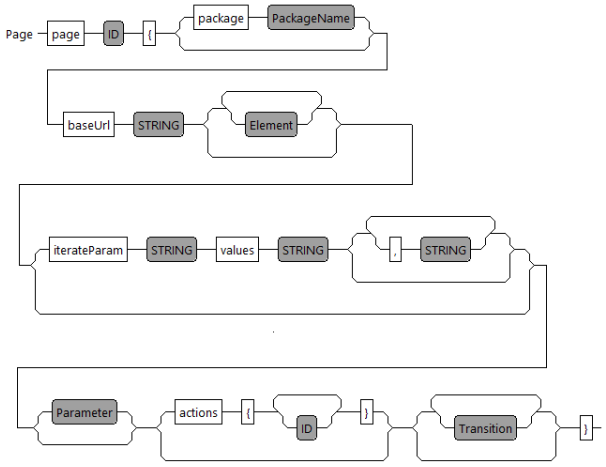

Fig. 1. Simplified syntax of a Page in WTML

The semantics of the language expressions starts with the page definition identified by its name (ID). In order to have package information for code generation, a package name can be optionally declared. Base URL is then assigned to each page. This gives us the possibility for customization of the parameters for the URL. Main information for a page is the elements. A page can have arbitrary number of elements. In order to query elements in a web page, we identified it with the XPath expression. The syntax for an element can be seen as in Fig. 2.

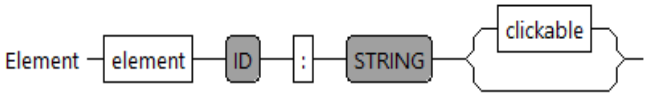

Fig. 2. Syntax of an Element model in a page

Each element starts with the keyword element followed by its name (ID). We then use a string literal to store its XPath expression. An element can optionally be clickable, this can be declared by the keyword *clickable*.

We then define the parameters of the page. A page can have any number of parameters. Each parameter starts with the keyword *param* as in Fig. 3.

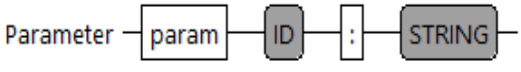

Fig. 3. Syntax of parameters in a page

Another type of parameter can be seen in the next block in a page as in Fig. 1 is the set of parameters to later be used in the code generation process to repeatedly test against. This is

defined by the keyword *iterateParam*. The parameters for iteration are comma-separated.

This information is enough if we just simply want to model a page for testing. The last components are the actions and transitions. These are the optional components to define the actions and transition between pages. This can be used later when we want to use state machine to model the test cycle of the whole web platform.

To demonstrate the simplicity of the model creation process in this approach, we can see how simple it is to write a textual model of a web component from a web application in a case study.

```
page RatingPage{
   baseUrl "http://www.webtest.org/login"
   element content "//*[@id='content']"
   element user "//*[@id='user']"
   element submit "//*[@id='submit']" clickable
   iterateParam itemID "12,13,14,15,16"
   param action "add"
}
```

This eight-line-of-code model at this abstraction level allows us to be very flexible on building the elements and logic needed for the test. At this level code reused is heavily promoted. This can be reused on many pages yet enables us to generate large amount of codes for test automation. Our benchmark pointed out that 90 lines of Java code were generated from this. This means we saved a significant amount of time that was otherwise supposed to be spent on test development. Overall, even if we take into account the time spent on developing and learning a new DSL such as WTML, this could still potentially provide a good productivity gain in test development.

V. Automation of Test Generation with WTML

According to IEEE standards, a test case is "a documentation specifying inputs, predicted results, and a set of execution conditions for a test item". As the definition states, there are three required elements in a test case; test inputs, predicted results and conditions of execution. IEEE's definition also clarifies the difference between test data and test case.

In order to generate test cases, test data must first be generated. Test data can be generated using different sources that describe the system, system's behavior or how the system will be used, such as source code, system specifications, system requirements, business scenarios and use cases. Our approach utilizes specification-based method for test case generation.

In this approach, we focus on the verification of the web system against the design specification that was available on the test models. This comprises of abstract information on the available operations and its parameters. Information from the specifications allows generation of test cases for boundary-value analysis, equivalence class testing or random testing [13].

In WTML platform, in order to generate the tests, we first need to generate the model implementation of the page to be tested against. A sample on how the page in Java was generated is as bellow:

```
@Page
@ComponentScan(basePackages=
                {"net.webmodeling.testing"})
public class FirstPage {
    private final static String
                baseUrl = "http://www.testpage.org/";
    private final static String
                iterateParamName = "value";
    @Autowired
    private AutoBrowser browser;

 @Value("#{'AAA,BBB,CCC'.split(',')}")
 private List<String> iterateParams;
 private static final By
  rating = By.xpath("//*[@id='viewcomments_click']");

    public String getRating() {
        return browser.getTextValue(rating);
    }
    public void clickOnRating() {
        browser.clickOn(rating);
    }
...
```

From the web page model syntax as seen on previous section, *iterateParam* is used when we want to iterate over a set of input parameters when testing a page. This becomes handy especially on the development of regression tests.

Another important aspect is *@Page* annotation, we introduced this annotation to inject special configuration to a page. This allows us to use Spring framework for processing pre- and post- Java bean creation. Testing data is injected directly into the page from the test models by using Spring *@Value* annotation. All setters and getters are also automatically generated from elements in the models.

This approach also provides a solution for automating regression testing. These tests are the reuse of the existing test cases from the previous system tests. Regression testing is performed when additions or modifications are made to an existing system. Since this could be run and generated automatically, regression testing could be performed anytime using WTML platform when there is a requirement.

VI. Integration with Other Platform

One of the essential features of the modeling tool is the ability to integrate with other platforms. Selenium is a suite of tools to automate web browsers across many environments. WTML can utilize Selenium to provide automatic simulation with browser. WTML raises the level of abstraction by modeling the elements and actions on the web page. This model will then be used as input to generate code for modeling page accordingly. We use Java as the target language. Using Spring framework dependency injection we then can integrate layered architecture in the code generated. Configurations are injected into JUnit tests via Spring annotation.

To support WTML platform, we created our defined annotations in Java, this Page annotation consists of Configuration that can be later injected and directives to load

the application context. We also defined our browser implementation in order to integrate with web driver from Selenium and provide automatic processing. In general this browser is defined in the following way:

```
...
@Component
public class AutoBrowser {
    private static final int TIME_OUT_SEC = 10;
    private static final Logger LOGGER =
          LoggerFactory.getLogger(AutoBrowser.class);
    @Autowired
    private WebDriver webDriver;

    public void clickOn(By location) {
        webDriver.findElement(location).click();

    }

    public WebElement findElement(By location) {
        return webDriver.findElement(location);
    }
    public void goToPage(String url) {
        webDriver.get(url);
    }
    public void goToUrlWithParam(String baseUrl,
                        Map<String,String> params) {
        final StringBuilder pageUrl =
                        new StringBuilder();
        pageUrl.append(baseUrl + "?");
        for (Map.Entry<String, String>
                  entry : params.entrySet()) {
            pageUrl.append(entry.getKey());
            pageUrl.append("=");
            pageUrl.append(entry.getValue());
            pageUrl.append("&");
        }
        goToPage(pageUrl.toString());
    }
    public void goToUrlWithSingleParam
              (String baseUrl, String paramName,
                                 String paramValue) {
        final StringBuilder pageUrl =
                           new StringBuilder();
        pageUrl.append(baseUrl + "?");
        pageUrl.append(paramName);
        pageUrl.append("=");
        pageUrl.append(paramValue);
        goToPage(pageUrl.toString());
    }

    @PreDestroy
    private void destroy() {
        webDriver.quit();
    }
    public int getNumberOfElements(By location) {
        return webDriver.findElements(location)
                    .size();
    }
    public String getTextValue(By location) {
        return webDriver.findElement(location)
                    .getText();
    }
    public String getAttributeValue(By location,
                          String attributeName) {
        return webDriver.findElement(location)
                    .getAttribute(attributeName);
    }
    public String getCssValue(By location,
                          String propertyName) {
        return webDriver.findElement(location)
                    .getCssValue(propertyName);
    }
...
```

After the configuration of Selenium web driver is defined and loaded, we inject web driver into our *AutoBrowser*, this way we keep the Selenium code separated from our browser logic. This allows us to only focus on the requirements and logics of code generation and automation test runners. After that we define all necessary methods for our automated browser such as *getNumberOfElements* from a given XPath address inside any page.

With the integration of Selenium, we are able to perform automatic browser actions. This enables us to write automated tests for a web application directly in WTML, which allows for better integration in existing unit test frameworks.

VII. Related Work

In the UML world, there has been effort on proposing techniques to automatically generate and execute test cases starting from a UML model of the Web Application by Filippo Ricca and Paolo Tonella [14]. This approach requires a manual work in several phases. There is manual work on the creation of models for testing and in the test refinement phase. Our approach has an advantage of fully automation in test case generation using the abstract web model and its action.

Alessandra Cavarra, Charles Crichton, Jim Davies, Alan Hartman and Laurent Mounier [15] presented the approach on test case generation utilizing UML. The authors' approach is based on extending UML using UML profiling capabilities. In these approaches, two profiles are created for different purposes. The first one is used to model the system under test by extending class diagrams, object diagrams, and state diagrams to support testing properties. The other profile is used to capture the test directives which are composed of the object diagrams and state diagrams. A transformation is then used to verify and produce scripts that can later be used to generate test cases.

A model-based testing approach is presented by Bouquet et al. in [16]. Their approach is based on a combination of class, object, and state diagrams, which can be found in UML and OCL expressions to automatically generate test cases from these models. Test cases are generated using a test generator that takes these diagrams and constraints as input. The authors discuss the need to alter the semantics of OCL to allow OCL expressions to have a side effect on the system state. In an overview of model driven testing techniques from the work of Mussa M., Ouchani S., Al Sammane W. and Hamou-Lhadj A. [11], the authors pointed out the shortcomings of this approach that it violates OCL semantics, which may hinder the acceptance of the approach by the UML community. One possible solution is to use an action language to express expressions that change the state of the system.

There has been also a direct attempt to use UML activity diagrams to generate test cases for Java programs in the work of Chen, Qiu and Li [17]. The approach is based on the

generation of test cases then compares the running traces with the activity diagram to reduce the test case set. The disadvantage of this approach is the limitation to the UML activity diagram that makes it impossible to obtain concurrency or loops for the tests.

Deutsch, Sui and Vianu in [18] introduced an approach that models data-driven web applications. This approach used Abstract State Machine to model the transitions between pages, determined by the input provided to the application. The structure and contents of web pages, as well as the actions to be taken, are determined dynamically by querying the underlying database as well as the state and inputs. The properties to be verified concern the sequences of events (inputs, states, and actions) resulting from the interaction, and are expressed in linear or branching-time temporal logic. This approach has an advantage of wide-range error detection. However, this leads to complex models that can make the integration with development methodologies not feasible.

Q. Yuan, J. Wu, C. Liu and L. Zhang [19] present an automatic approach to generate test cases of a given business process of a web service. The modeling of business process uses notations from Business Process Execution Language and UML activity diagrams. This approach is an example of applying MDA and the conformed transformation techniques. This approach aimed to build concise test models using given notations and generate test cases from this models. The advantage of this approach is the ability to apply in many types from unit testing to integration testing.

VIII. Conclusions

With the strict requirements of web-based systems, techniques to assure the quality of these systems play a very important role in the development process. Model-driven testing tools reduce overall testing time by supporting the reuse of many common testing functions. They also enhance test quality and complexity by offering a systematic approach to test suite generation. In this paper, we outlined the theoretical ideas and analysis from lessons learned during the real industry implementation of the framework. The approach introduced in our research provides a methodology for using a domain specific language in model-driven testing of web applications. Adopting WTML in combination with the MDA initiative allows early testing of model-driven systems and eases the sharing of models between the system developers and the system testers.

WTML was designed at the appropriate abstraction level to have better model readability and more support for integration. This is aimed at reducing test maintenance costs, since changes happen at the model level and are captured by the test models. When there are changes, we only have to regenerate the tests from the test models and all test cases are updated to the new specifications. This framework also enhances team communication because the model, test cases provide a clear, unambiguous, and unified view of both the system under test and the test. This technique decouples the testing logic from the actual test implementation. This makes the test architecture more robust and scalable. The shortcomings of this approach include a learning curve needed to adopt a new modeling language and the limitation of test behaviors based only on the possible elements modeled in a page abstraction.

Domain specific language such as WTML can be applied to automation testing of web-based applications and pages. In practice, this approach has initially gain adoption in testing of web systems in the financial industry where the authors had the chance to work with. Our future work will continue on the improvement of the framework in terms of consistent methodology, wider code generation coverage and more efficient notations and syntax.

Acknowledgment

This work has been supported by the Department of Computer Science and Engineering, Faculty of Electrical Engineering and by the grant of Czech Technical University in Prague number SGS14/078/OHK3/1T/13.

References

[1] F. Bolis, A. Gargantini, M. Guarnieri, E. Magri, L. Musto, "Model-Driven Testing for Web Applications Using Abstract State Machines", in M. Grossniklaus, M. Wimmer, ed., *Current Trends in Web Engineering* vol. 7703, (Springer Berlin Heidelberg, 2012), pp. 71-78.

[2] J. Peleska, "Industrial-Strength Model-Based Testing - State of the Art and Current Challenges", *Electronic Proceedings in Theoretical Computer Science* 111 (2013), pp. 3-28.

[3] X. Qafmolla, V. Nguyen, Automation of Web Services Development Using Model-driven Techniques. In Institute of Electronics Engineers, The 2nd International Conference on Computer and Automation Engineering (ICCAE 2010), pp. 190-194, 2010.

[4] Object Management Group (OMG): Meta Object Facility (MOF) Core. Retrieved March 20, 2012, http://www.omg.org/spec/MOF/2.4.1/, 2012.

[5] Object Management Group (OMG): The Architecture of Choice for a Changing World. Retrieved April 20, 2013, http://www.omg.org/mda, 2013.

[6] V. Nguyen, X. Qafmolla, Agile Development of Platform Independent Model in Model Driven Architecture. In Proceedings of the 2010 Third International Conference on Information and Computing , Vol. 2. IEEE Computer Society, Washington, DC, USA, pp. 344-347, 2010.

[7] Wikipedia: Domain specific language. Retrieved January 15, 2013, from http://en.wikipedia.org/wiki/Domain specific_language, 2013.

[8] A. Deursen, P. Klint, Little languages: Little maintenance. Journal of Software Maintenance, pp. 75-93, 1998.

[9] C. W. Krueger, "Software reuse", *ACM Computing Surveys (CSUR)* 24, 2 (1992), pp. 131--183.

[10] Wikipedia, "Model Driven Testing", *Model Driven Testing* (2014).

[11] M. Mussa, S. Ouchani, W. Al Sammane, A. Hamou-Lhadj, "A Survey of Model-Driven Testing Techniques", in *Quality Software, 2009. QSIC '09. 9th International Conference on* (, 2009), pp. 167-172.

[12] X.Yu, Y. Zhang, T. Zhang, L. Wang, J. Hu, J. Zhao, X. Li, A model-driven development framework for enterprise Web services. Information Systems Frontiers, pp. 391-409, 2007.

[13] M. Bozkurt, M. Harman, Y. Hassoun, "Testing Web Services: A Survey", Department of Computer Science, King's College London (2010).

[14] F. Ricca, P. Tonella, "Analysis and Testing of Web Applications", in *Proceedings of the 23rd International Conference on Software Engineering* (Washington, DC, USA: IEEE Computer Society, 2001), pp. 25--34.

[15] A. Cavarra, C. Crichton, J. Davies, A. Hartman, L. Mounier, "Using UML for automatic test generation", in *In international symposium on testing and analysis ISSTA* (Springer-Verlag, 2002).

[16] F. Bouquet, C. Grandpierre, B. Legeard, F. Peureux, "A Test Generation Solution to Automate Software Testing", in *Proceedings of the 3rd International Workshop on Automation of Software Test* (New York, NY, USA: ACM, 2008), pp. 45-48.

[17] C. Mingsong, Q. Xiaokang, L. Xuandong, "Automatic Test Case Generation for UML Activity Diagrams", in *Proceedings of the 2006 International Workshop on Automation of Software Test* (New York, NY, USA: ACM, 2006), pp. 2-8.

[18] A. Deutsch, L. Sui, V. Vianu, "Specification and verification of data-driven Web applications", *Journal of Computer and System Sciences* 73, 3 (2007), pp. 442 - 474. Special Issue: Database Theory 2004.

[19] Q. Yuan, J. Wu, C. Liu, L. Zhang, "A model driven approach toward business process test case generation", in *Proc. of the 10th International Symposium on Web Site Evolution (WSE)* (2008), pp. 41-44.