SCENTOR: Scenario-Based Testing of E-Business Applications

Jeremiah Wittevrongel, Frank Maurer
The University of Calgary
jeremiah@cpsc.ucalgary.ca, maurer@cpsc.ucalgary.ca

Abstract

E-business software is often developed on a tight schedule, and testing needs to keep pace. Advice from proponents of approaches like Extreme Programming is that by testing continuously, it is actually possible to compress development cycles. In this paper we discuss a testing approach that supports developers with their task of creating automated functional test drivers for e-business applications. The main goal for the approach is to reduce the time and effort required to automate scenario tests for e-business applications. After motivating the approach, we give an abstract view of a tool designed and implemented to support the approach. Next, we give an example of its use, and finally proceed to a discussion of the architecture of the tool itself.

1. Introduction

Before sending e-business software live on the Web, we want to ensure it functions as expected. But e-business software development is different than traditional software development. The Internet has accelerated the required development pace, while the risks involved in using new and sometimes immature web technologies pose additional threats to completing the project on a tight schedule.

Approaches like Extreme Programming drastically shorten development cycles and may be particularly well-suited to mid-sized e-business projects. There is evidence that adopting only some of the practices in Extreme Programming will shorten project schedules and result in more successful software development processes [1].

One of the cornerstones of Extreme Programming is the use of continual testing throughout the development process. Unit tests created by the developers while they work on the system are combined with scenario tests based directly on customer requirements. Together, these tests form a continually evolving and always-available automated test suite.

Tools like JUnit [7] provide a lightweight testing framework for quickly implementing functional unit tests. The support is not specific, however, to e-business applications, meaning that developers may have additional work if they wish to adopt an approach involving continual automated testing.

SCENTOR [12] is an approach that aims to provide e-business-specific support for the generation of scenario-based tests using JUnit as a basis. In doing so, it is hoped that creating scenario tests for e-business applications takes less time, leaving more time for running the tests and developing the system. In a test-centric approach to development, this will result in a compressed schedule.

In Section 2, we motivate our work, and give some background. Section 3 illustrates how to derive test drivers from usage scenarios of a system. Section 4 gives an example of our approach, while Section 5 describes the system architecture. In Section 6, we discuss related work, and finally, Section 7 summarizes our results.

2. Motivation and background

E-business projects often have huge time-to-market pressure; there is not always a lot of time for testing. Priority One under extreme time pressure should be ensuring typical use scenarios can be completed.

This suggests that testing of e-business applications should focus on user-visible functionality. In Extreme Programming, this user-visible functionality is described in user stories. SCENTOR adds Unified Modeling Language (UML) sequence diagrams on top of this approach, which serves to make the scenarios slightly more formal.

This slightly more formal representation of the scenarios is enough to allow useful support and partial automation for the tedious task of writing automated test drivers. The developer would only need to add concrete parameters and expected results for each step in the scenario, and this forms a straightforward automated test driver.

Tests based on typical-use scenarios ensure that the focus of testing is on the most-used parts of the system.

The SCENTOR approach is also meant to support incremental testing. It should be possible to add test cases (individual tests with a single set of concrete parameter values and expected results) at any time to the existing test suites (collections of test drivers).

In addition to handling this basic functionality, a tool that supports scenario-based testing of e-business applications should do more. Many of the technologies used in the development of dynamic e-business applications will require that test drivers perform some special setup first, before executing the actual functional tests. SCENTOR integrates support for generating global setup code (test setup code performed only once immediately before an entire test suite is run). By doing so, it allows multiple test cases to share this (sometimes expensive) setup process.

The test execution sequence proceeds as follows: first, the global (common) setup runs once. Then a group of test cases is run, each possibly having its own setup code that will run immediately before it, and its own teardown code that will run immediately following it. Finally, any global teardown code is executed.

In this fashion, SCENTOR can not only save time required for test implementation, but can also reduce the time required to run a group of tests, while maintaining a modular test design.

3. From scenarios to test drivers

As a proof of concept, SCENTOR was designed and implemented to support scenario-based testing of e-business applications using Enterprise Java Beans (EJB) [9]. SCENTOR is also targeted towards lightweight development processes that include only a partial set of UML models while maintaining the Extreme Programming focus on the production of source code. Figure 1 shows the progression from scenarios to test drivers in SCENTOR.

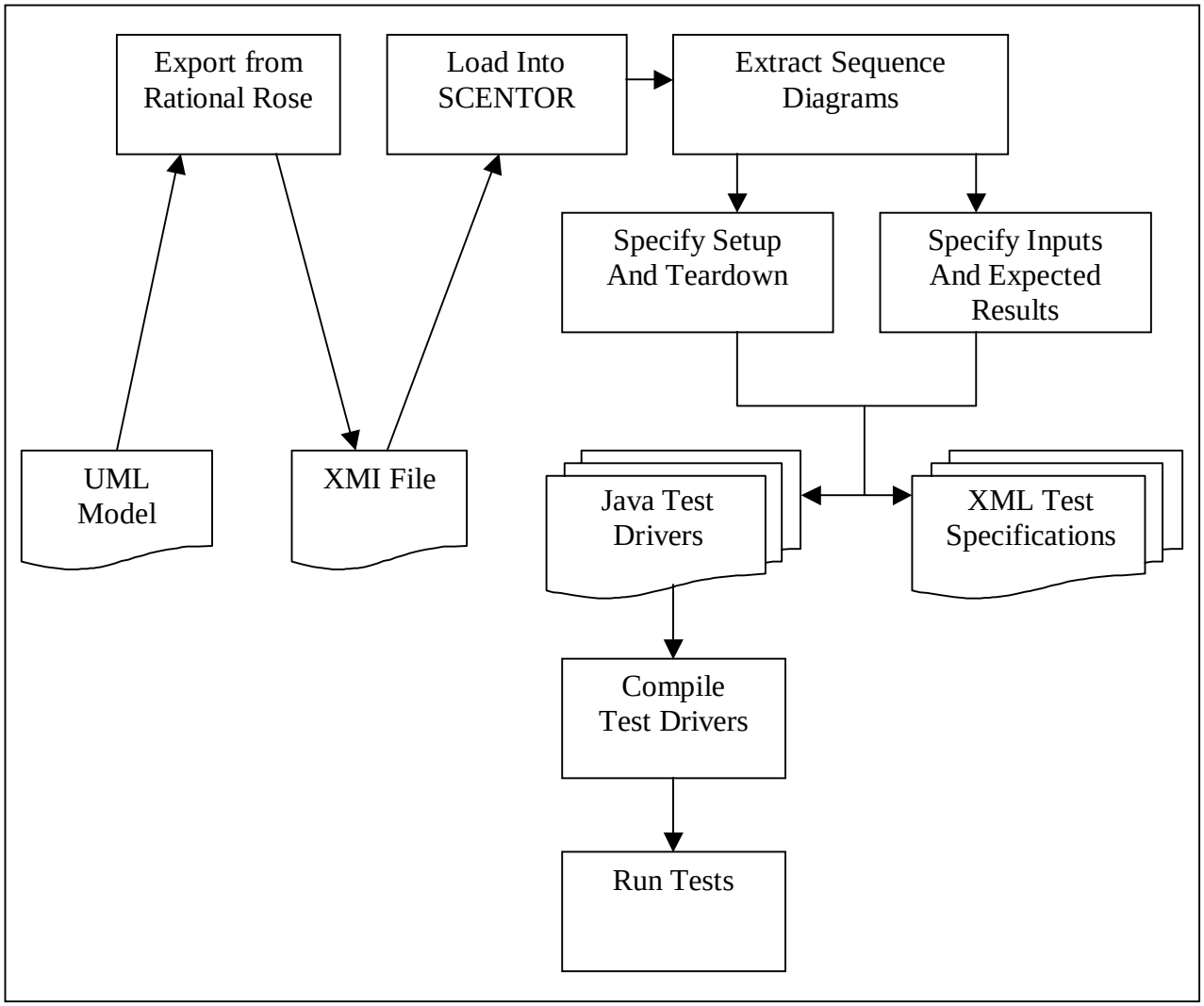

Figure 1: Moving from scenarios to test drivers

Scenarios are first modeled as UML sequence diagrams, using a CASE tool such as Rational Rose. This includes classes and methods participating in the scenario. The UML model is then exported from the CASE Tool in the vendor-independent XMI (XML Metadata Interchange) format.

The developer would then load the XMI file into SCENTOR, and could optionally load a previously created test specification (in an XML format [11]) as well (not shown in Figure 1).

Tests are specified based on the UML sequence diagrams. A small suite of tests is typically based on the set of messages sent by a single object in a UML sequence diagram. This single object could, for example, represent the whole user interface of the system. The developer needs only to add concrete parameter values to

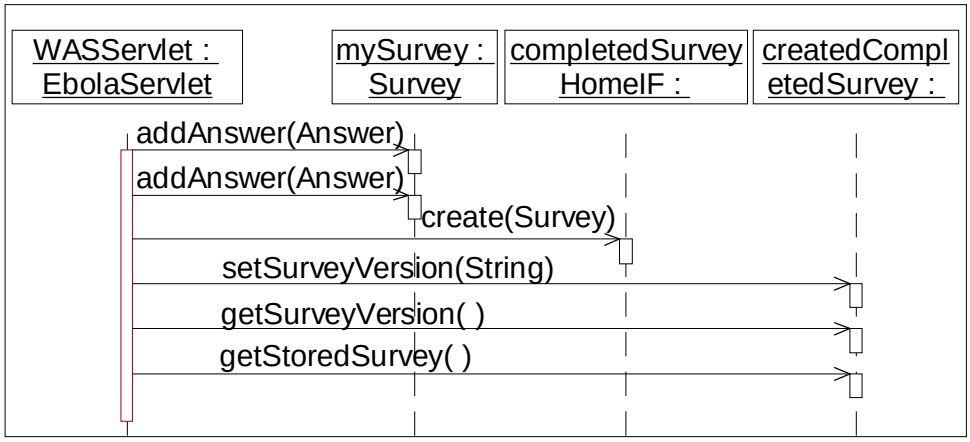

Figure 2: A scenario expressed as a UML sequence diagram

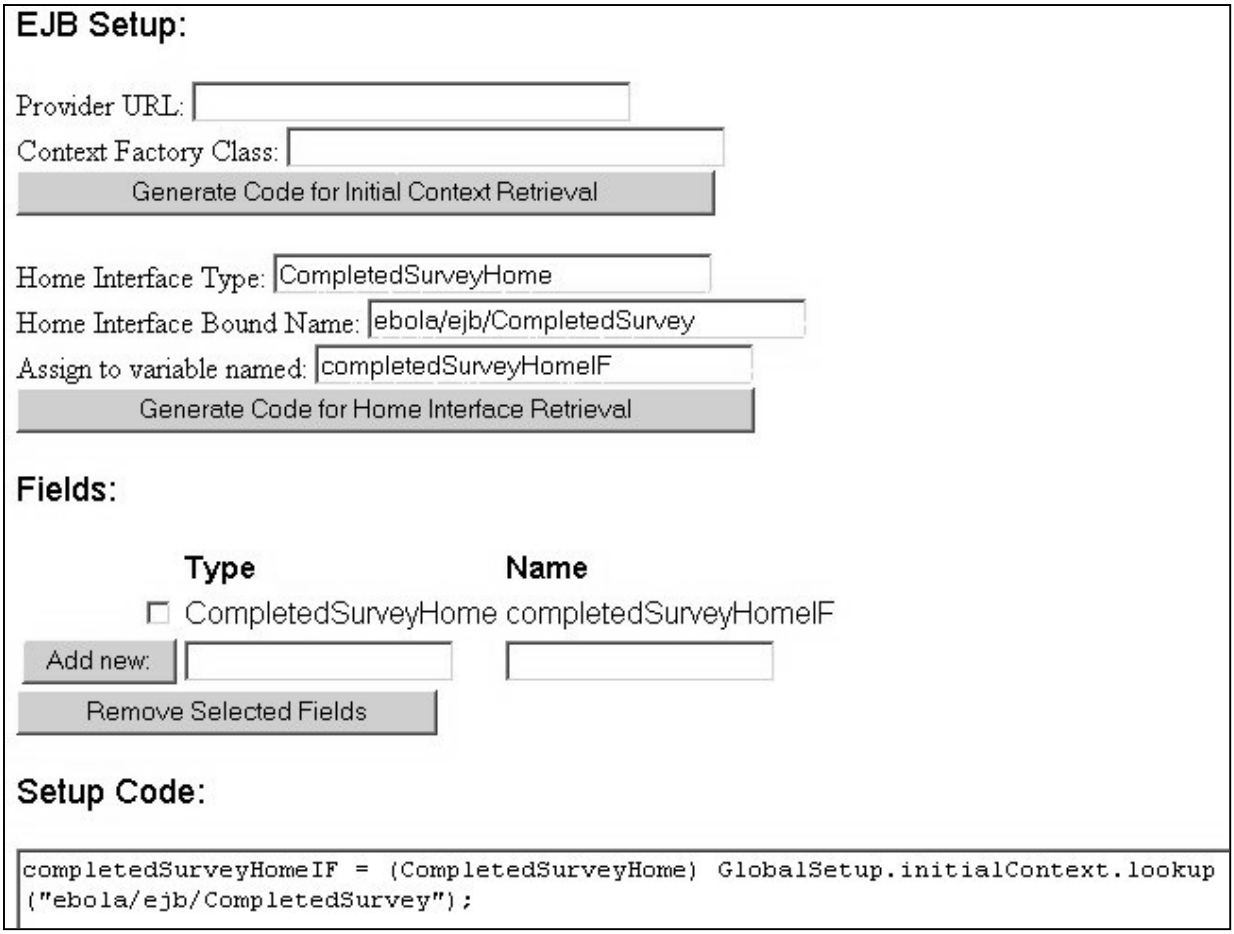

Figure 3: Screenshot showing EJB setup code generation

the method call and specify the expected results of the method call. To preserve the focus on source code and avoid the need for developers to learn another language, we decided to let the developer enter these results in the underlying programming language (Java) instead of using other formal languages (e.g. UML's object constraint language, OCL).

SCENTOR concretely helps in the development of test drivers for Enterprise Java Beans by supporting generation of common EJB-specific setup code. This setup code would normally be shared among a group of test cases in such a way that it is only executed once, regardless of the number of tests running. Creating the initial context for Bean lookups and retrieving references to the EJB Home Interfaces are two tasks that are well suited for this type of setup code; SCENTOR supports both.

The main benefit of sharing the setup code among a group of tests comes with reduced execution time for a large group of tests. Looking up and retrieving a reference to an EJB Home Interface takes quite some time, and repeating that lookup for each test means test drivers run much more slowly than if the call is only done once for the entire suite.

When the developer adds more test cases to a suite that has shared setup code of this nature, the setup code is automatically inherited by the new test cases.

Developers can compile and execute the generated test drivers from within SCENTOR if they wish, or may employ another compilation and execution environment.

The Test specifications can be saved as an XML files following the SCENTOR Test Specification DTD [11]; these files can later be loaded into SCENTOR for modifications or additions.

4. An example

Consider the EBOLA Project [13], which includes an online survey implemented using Enterprise Java Beans [9]. In this example, SCENTOR is used to create automated test drivers for part of EBOLA's online survey code.

First, the scenarios to test are modeled in Rational Rose as UML sequence diagrams. Figure 2 shows one of the sequence diagrams for our set of scenarios. This UML model is exported to XMI format, and loaded into SCENTOR.

Two suites of tests are created, one for each of the two enterprise beans that comprise this part of EBOLA. These suites are each based on a sequence diagram that includes an EJB instance as a participant.

Several test cases are created in each of the two test suites. Now, the EJB specific setup code is added. The

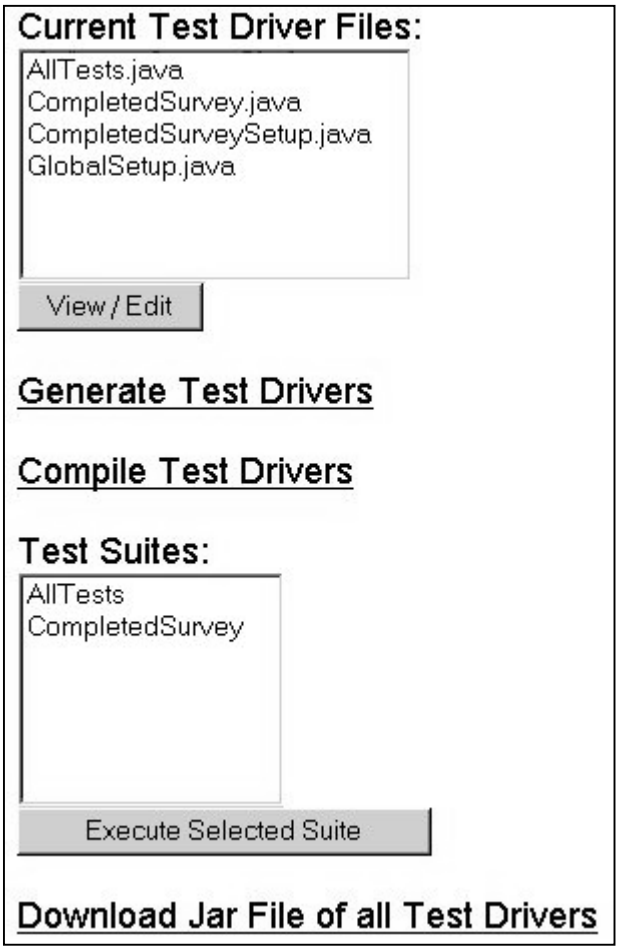

Figure 4: Compilation and execution

global test suite's setup code creates an initial context, which can then be accessed by any test it contains (directly or indirectly). Since each of the smaller test suites deals with a single EJB, the home interface will be retrieved in its setup code. Figure 3 shows the generation of code for the lookup of an EJB Home Interface.

There are two important results of this shared setup process. First, the initial context is only created once regardless of the number of tests run. The Home Interfaces are also only looked up once. Second, the initial context is available to all the tests we have specified (they share it), and the Home Interfaces are then accessible to any test in the suite whose setup retrieves them. This minimizes the time required to run the test drivers; the expensive setup operations that all the tests require are only executed once.

Finally, the test drivers can be generated, compiled, and run from within SCENTOR. The test drivers will connect to an EJB server on the network, execute, and display the results to the user. The user interface for this is shown in Figure 4.

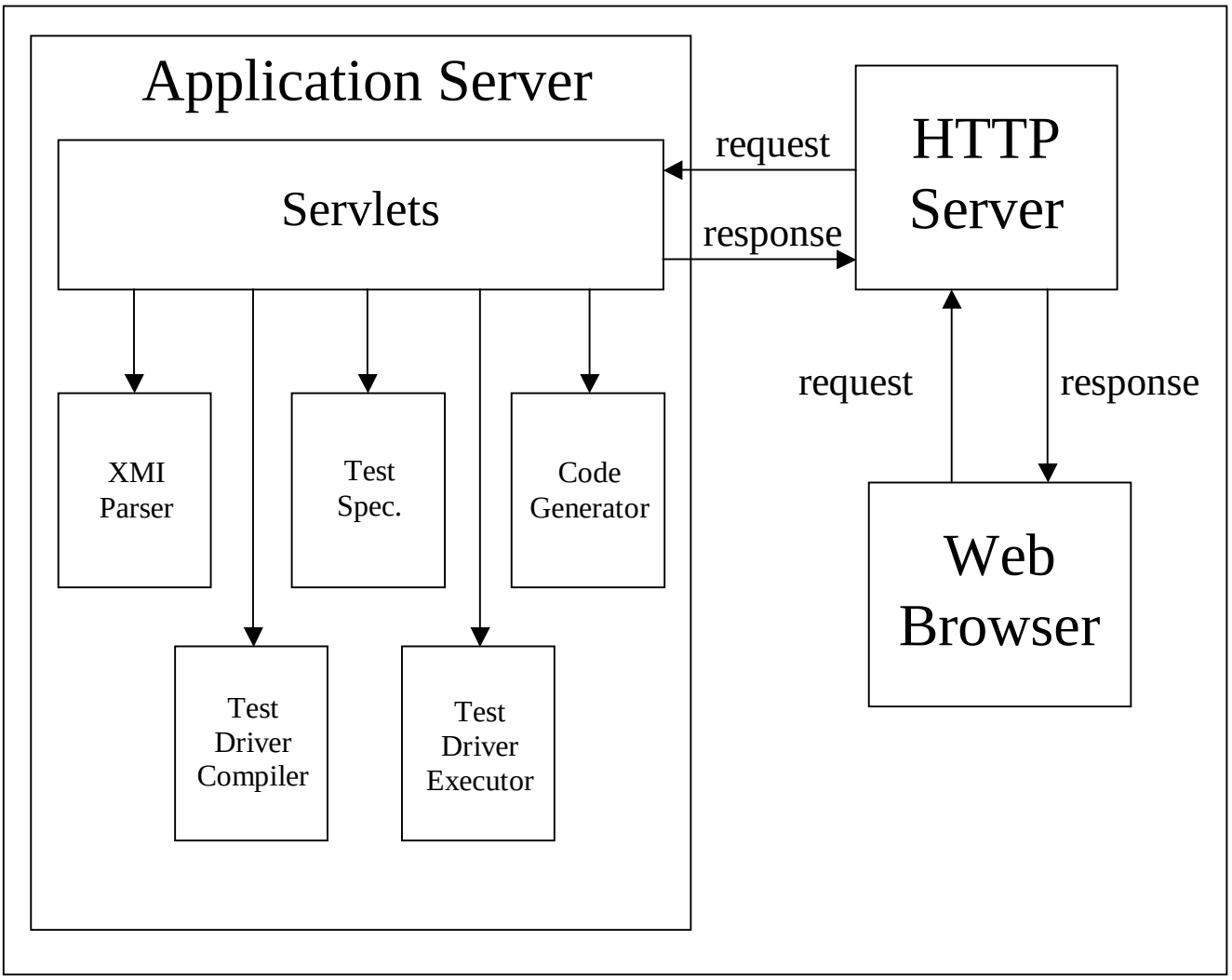

Figure 5: Tool Architecture for SCENTOR

5. System architecture & implementation

SCENTOR is made available as a service on the Internet. Users can connect to the SCENTOR web site [12], define tests and execute them on our machines[1]. An overview of the architecture can be found in Figure 5.

Java servlets present a web interface to the user, and also handle all other communication with the user (such as transferring files). These servlets connect to the major components of SCENTOR, which include the XMI parser, the test specification, the code generator, the test driver compiler, and the test driver executor.

The XMI parser reads in vendor-independent XMI files created with a CASE tool, and extracts the sequence diagrams from them. In addition, the type information for objects participating in the sequence diagrams is extracted, if present.

The test specification maintains the test cases, test suites, and test setup components created by the developer. It is also responsible for generating an XML representation that can be saved by the user, and loaded into SCENTOR in the future for further modifications and additions.

Once a test specification is created, the code generator translates the specification into Java source code. This source code depends on the JUnit testing framework for compilation and Execution.

The test driver compiler and executor are merely front-ends to a java compiler and a java virtual machine, respectively. The test drivers can be compiled and run directly from SCENTOR, using these components.

As mentioned earlier, SCENTOR is available on the Internet as a web application. As such, two more components are part of the SCENTOR architecture: a web browser and a web server. The web browser presents the

[1] SCENTOR is open-source software and can also be downloaded from the Scentor web site.

user interface, and the web server forwards requests and responses between the web browser and the servlets.

6. Related Work

SCENTOR takes some ideas concerning lightweight development processes from Extreme Programming discussed in [1], [2], and [10]. Not all the principles of Extreme Programming need be adopted to realize benefits [1]. One of the most prevalent principles, however, is the use of continual testing which includes both unit tests and functional scenario tests. SCENTOR aims to support continual scenario testing. We assume an environment where UML is used to describe the scenarios to be able to automatically generated test drivers – avoiding repetitive tasks seen in JUnit-based testing. In a complete implementation of Extreme Programming, the scenarios (user stories) would be written in English on index cards.

JUnit [7], and other related tools, provide a simple framework for unit testing of software. Scenario tests can be specified using JUnit, but no specific support for scenario tests is included. Further, JUnit contains no support specific to the testing of e-business applications.

TOTEM [3] is a project that investigates ways in which UML diagrams can support derivation of test drivers for all levels of the system. In contrast with the lightweight UML modeling assumed by SCENTOR, however, TOTEM assumes a much greater use of UML in the development process.

Scenario-based testing of software in general is not a new idea. Basing test scenarios on requirements (as Extreme Programming does) is also not a new idea. Discussions of scenario-based testing in general and basing test scenarios on requirements can be found in many published works, including [5], [6], and [8]. SCENTOR's contribution in this area is a framework that applies scenario-based testing to e-business projects that utilize lightweight development processes.

SCENTOR assists developers in generating automated test drivers. The effectiveness of test automation in general, and the effectiveness of specific automation techniques has been the basis of much discussion, including [4], [8], and [14].

7. Summary and future work

SCENTOR assists developers by forming a bridge between user scenarios and functional test drivers. By removing some of the repetitive, mechanical work required when creating automated test drivers, SCENTOR also aims to reduce the time required to develop them. Further, by supporting factored setup code explicitly, it is possible to realize a speed increase when executing large sets of tests. Test execution speed is crucial in a framework where regression testing is a on-going activity (as in Extreme Programming).

Future plans include an empirical evaluation of SCENTOR's effectiveness in reducing the time required for automated test development. This is the next step for the SCENTOR project.

In addition, there are other plans for the future of SCENTOR. Currently, SCENTOR only generates functional test drivers. We plan to extend SCENTOR's code generation capabilities to also generate test drivers for load testing. Essentially, the test drivers generated for load testing would simulate a load of N users performing scenarios simultaneously. This can be used to measure EJB server performance under varying loads.

SCENTOR also currently has no facility for displaying the imported UML sequence diagrams graphically. Such an addition to the user interface would allow for much more intuitive selection of scenarios to use as the basis for test cases.

8. References

[1] K. Beck, "Embracing Change with Extreme Programming", *Computer*, IEEE CS Press, Los Alamitos, Calif., vol. 32, no. 10, Oct. 1999, pp. 70-77.

[2] K. Beck, *Extreme Programming Explained: Embrace Change*, Addison Wesley, Reading, Mass., 1999.

[3] L. C. Briand, *Testing of Object-oriented software sysTEms with the Unified Modeling Language (UML),* http://www.sce.carleton.ca/faculty/briand/totem/totem.htm (current Feb. 10, 2001).

[4] M. Fewster and D. Graham, *Software Test Automation: Effective use of Test Execution Tools*, Addison Wesley, Harlow, England, 1999.

[5] M. R. Lyu, ed., *Handbook of Software Reliability Engineering*, IEEE CS Press, Los Alamitos, Calif., 1996.

[6] B. Marick, *The Craft of Software Testing*, Prentice Hall, Englewood Cliffs, N.J., 1995.

[7] Object Mentor, Inc., *JUnit, Testing Resources for Extreme Programming*, http://www.junit.org/ (current 10 Feb. 2001).

[8] W. E. Perry, *Effective Methods for Software Testing*, Second ed., John Wiley & Sons, New York, 2000.

[9] Sun Microsystems, *Enterprise JavaBeansTechnology*, http://java.sun.com/products/ejb (current 23 Feb., 2001).

[10] J. D. Wells, *Extreme Programming: A Gentle Introduction*, http://www.extremeprogramming.org (current 10 Feb. 2001).

[11] J. Wittevrongel, *The SCENTOR Test Spec. DTD*, http://sern.ucalgary.ca/~milos/dtd/scentorTestSpec.dtd (current 23 Feb. 2001).

[12] J. Wittevrongel, *The SCENTOR Web Site*, http://sern.ucalgary.ca/~milos/projects/scentor_intro.htm (current 10 Feb. 2001).

[13] L. Wong, *The EBOLA Web Site*, http://sern.ucalgary.ca/~milos/projects/ebola/ (current 10 Feb., 2001).

[14] K. Zallar, "Practical Experience in Automated Testing," *Methods and Tools*, Martinig and Associates, vol. 8, no. 1, Spring 2000, pp. 2-9.