Life-cycle E-Commerce Testing with OO-TTCN-3 *

Robert L. Probert[1], Pulei Xiong[1], and Bernard Stepien[2]

[1] School of Information and Technology Engineering, University of Ottawa 800 King Edward Avenue, Ottawa, Ontario K1N 6N5 Canada
bob, xiong@site.uottawa.ca
[2] Bernard Stepien International Inc.
bernard.stepien@sympatico.ca

Abstract. E-Commerce systems have become ubiquitous. However, it is a challenge to create high quality e-commerce systems respecting time and budgetary constraints. In this paper, we present a life-cycle testing process for e-commerce systems by adapting OO-TTCN-3, an object-oriented extension of a formal test language TTCN-3, to enable the efficient specification of tests in object-oriented, e-commerce development environments. This extension is meant to ease life-cycle testing, facilitate test case reuse between different test phases, and provide a unified Abstract Test Suite (ATS) interface to test tools. A case study shows how to apply the approach to a typical e-commerce system based on a high-yield, risk-directed strategy.

1 Introduction

Web techniques have grown very fast in recent years. *Electronic Commerce* (E-Commerce) systems, which facilitate the conduct of business over the Internet with the assistance of Web techniques, have been adopted by more and more companies as the architecture of their enterprise information systems. It is a challenge to build a quality e-commerce system under the constraints of time and budget. Software testing is an essential and effective means of creating quality e-commerce systems.
Software testing is a *life-cycle* process which parallels the software development process [1, 2]. Generally, a complete software testing life cycle includes the following phases: test planning, verification and validation (V&V) of system analysis and design (AD) models, unit testing, integration testing, functional testing, and non-functional system testing. In these testing phases, we need to build and test various AD models, as well as the implementation, by applying multiple testing methods and utilizing multiple testing tools.

* This work was partially supported by Communications and Information Technology Ontario (CITO) in a collaborative project with IBM, and by Testing Technologies IST GmbH. The authors are grateful for the comments of the anonymous referees,which improved the paper considerably.

Life-cycle testing, however, does not guarantee testing software thoroughly. In the real world of software projects, software should be tested in a cost-effective way: executing a limited number of test cases, but still providing enough confidence. A *high-yield, risk-directed* test strategy introduced in [3] is a cost-effective approach of test case design: test cases that are high-yield (that have the highest probability of detecting the most errors) and high-risk-directed (that have the most serious consequences if they fail) are developed and executed with high priorities.
In this paper, we propose a life-cycle e-commerce testing approach based on a high-yield, risk-directed strategy by specifying test cases at an abstract level in *OO-TTCN-3*. The paper is organized as follows. In section 2 we discuss briefly the nature and architecture of web applications, and discuss related work on web application modeling and testing. In section 3 we introduce a life-cycle testing approach, and propose a useful object-oriented extension to TTCN-3. In section 4 we present a case study in which we apply the proposed testing approach. In section 5 we present a summary and discuss possible future work.
Web application, a term similar to *e-commerce system*, is also used in this paper. There is no essential difference between these two terms. Informally, we consider that *e-commerce system* is a term with broader scope than *web application*: an e-commerce system may consist of several relatively independent web applications.

2 Testing E-Commerce Systems: Background and Related Work

2.1 The Nature and Architecture of Web Applications

Generally, web applications are a kind of thin-client Client/Server (C/S) system. However, compared to traditional C/S software systems, web applications have natures which make them more complex to design and test: (1) server-based application architecture such that no special software or configuration are required on client side (2) navigation-based interaction structure (3) n-tiered system architecture such that the components of each tier may be distributed and run on heterogeneous hardware and software environments (4) independent of types, brands, and versions of web servers and browsers (5) the web server may concurrently process tens of thousands of requests from client applications (6) code contains a blend of object-oriented and procedure-oriented programming. Today, most e-commerce systems are running on the 4-tiered architecture, that is, client tier, web tier, business tier, and Enterprise Information System (EIS) tier. Each tier is built on component-based techniques.

2.2 Web Application Modeling and Testing

There has been much research into web application modeling and testing. UML (Unified Modeling Language) is suited to modeling web applications, but needs some extensions to describe web-specific elements. Some

extensions to UML were proposed in [4] to represent web-specific components, such as client pages, server pages, forms, hyperlinks, and Java Applets. The authors of [5] discussed how to apply extended UML to build a business model, navigation model and implementation model.
Several approaches have been proposed to support web application testing. In [6], an object-oriented architecture for web application testing was proposed which contains several analysis tools that facilitate the testing of web-specific components. In [7], the authors presented a testing methodology for web applications based on an object-oriented web test model, which captures test artifacts from three different aspects: the object aspect, the behavior aspect, and the structure aspect. In [8], the authors proposed a definition of unit and integration testing in the context of web applications. The testing approaches discussed above focus on testing web applications after they have been built. Alternatively, in [9], the authors presented a formal method which integrates testing into the development process as early as the system analysis and design phases. The paper demonstrated how to use the Specification and Description Language (SDL), Message Sequence Charts (MSCs), the Tree and Tabular Combined Notation (TTCN), and industrial-strength system design tools such as Telelogic TAU to develop and test a CORBA-based e-commerce system.
These testing approaches contributed to the improvement of web application quality. However, they only applied to part of development life-cycle: either in the analysis and design phases or the implementation phase. In addition, the test cases developed in a testing approach are written in a specific script language and depend on proprietary or specific industrial tools. These test scripts can not be reused by other testing approaches. In this paper, we intend to propose a life-cycle testing approach which leverages multiple testing methods in a life-cycle testing process. Furthermore, we integrate all the testing phases by specifying test cases on the abstract level with Object-Oriented TTCN-3. These abstract test cases can be easily reused in the whole testing process, and are independent of specific testing tools.

3 A Life-cycle Testing Approach with OO-TTCN-3

3.1 A Life-cycle Testing Process

Life-cycle e-commerce testing is a process of applying multiple testing methods to AD models and implementations in different testing phases, with assistance of various test tools, as shown in Figure 1. In this process, we specify all test cases in OO-TTCN-3. The dashed lines in Figure 1 indicate the possible ATS reuse. The ATS expressed in OO-TTCN-3 provides a unified interface. This makes test scripts independent from specific test tools. This also facilitates ATS reuse between different test phases: e.g., test scripts for unit testing can possibly be reused by integration testing without any modification, and different test tools may be used for the unit testing and integration testing.

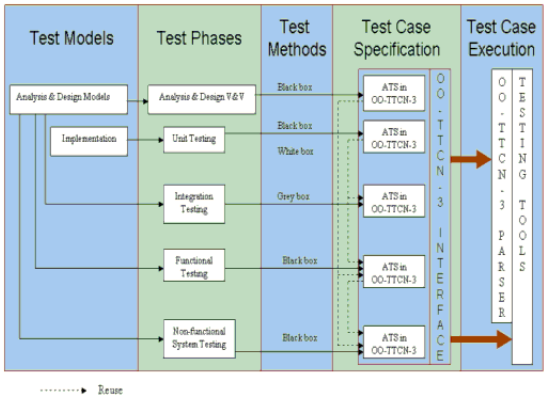

Fig. 1. Life-cycle Testing Process Model with OO-TTCN-3 for E-Commerce Systems

3.2 Introduction to TTCN-3

TTCN-3 has been developed and standardized by ITU and ETSI (European Telecommunication Standards Institute) for general testing purposes. An ATS specified in TTCN-3 is independent of languages, platforms, and testing tools. TTCN-3 is built from a textual core notation on which several optional presentation formats are defined, such as the tree and tabular format (TFT) and the Graphical Presentation Format (GFT) [10, 11]. Complex distributed test behavior can be specified at an abstract level in TTCN-3 flexibly and easily in terms of sequences, alternatives, loops and parallel stimuli and responses. Practical applications of TTCN-3 were introduced in [12–15]. In addition, an extension for TTCN-3 were proposed in [13] to handle specific issues in testing real-time systems.

3.3 Object-Oriented TTCN-3

In this paper, we do not intend to make TTCN-3 fully object-oriented. Instead, our extension only focuses on specifying inheritance hierarchies and aggregation relationships between test modules.

Inheritance When we test an object-oriented system, if an inheritance relationship in the system is introduced during design in accordance with the Substitution Principle, an inheritance relation also will hold for the test cases for this inheritance hierarchy [16]. As shown in Figure 2, class B is derived from class A in accordance with the Substitution Principle, and TM_A and TM_B are test modules for class A and B, respectively. There exists an inheritance relationship between TM_B and TM_A. The allowed ways of introducing inheritance as required by the Substitution Principle, and corresponding test case design considerations are as follows [16].

- A new operation, say b_op1, is added to the interface of B and possibly a method is created to implement the operation. In this way, specification-based test cases will now be required for the operation

in TM_B. If the operation has an implementation, implementation-based test cases need to be added to comply with coverage criteria.
- If an operation in B, say a_op1 which is inherited from A, has not changed in any way, either in specification or in implementation, the test cases for this operation in TM_A still apply to TM_B, which means the test cases do not need to be rerun in TM_B if they have passed in the execution of TM_A.
- The specification of an operation in B, say a_op2 which is inherited from A, has changed. In this case, new specification-based test cases are required for the operation in TM_B, which will satisfy any weakened preconditions and check outputs for the new expected results from any strengthened postconditions. The test cases for this operation in TM_A must be re-run. If the expected results need to be revised according to the strengthened postcondition, the test cases in TM_B need to be overridden.
- An operation in B, say a_op3 which is inherited from A, has been overridden. In this case, all the specification-based test cases for the operation in TM_A still apply in TM_B. The implementation-based test cases need to be reviewed. Some test cases need to be overridden, and new test cases need to be added to meet the test criteria for coverage.

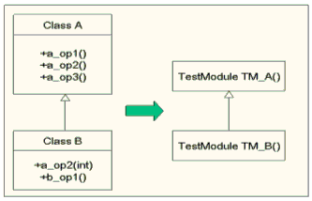

Fig. 2. Class Inheritance Hierarchy and Corresponding Test modules

In short, test cases in TM_A can be reused in TM_B. TTCN-3 provides a way to reuse definitions in different modules by using the import statement. However, as a procedure-oriented language, TTCN-3 is incapable of specifying the inheritance relationship between TM_A and TM_B. Therefore, we extend TTCN-3 with a fundamental object-oriented mechanism, namely inheritance (extend and override), to make it capable of specifying derived test modules. The extension helps to specify the inheritance hierarchies in test modules clearly, and it eases the reuse of test case definitions in unit testing at the class level. For example, for a simple inheritance hierarchy shown in Figure 3, we can develop test cases based on the specification and implementation (if any) of an abstract class Account, and specify them in OO-TTCN-3 in test module AccountTest, even before the creation of two subclasses: EXAccount and BankAccount. After the two subclasses have been designed and implemented, the test cases in AccountTest are ready to be reused for test module EXAccountTest and BankAccountTest which are inherited from AccountTest. In addition, if any subclass is derived from the class Ac-

count in the next development iteration, the test module for the new subclass can also benefit from the existing testing module hierarchy. In section 4.3.3, we show how to use OO-TTCN-3 to specify a derived test module.

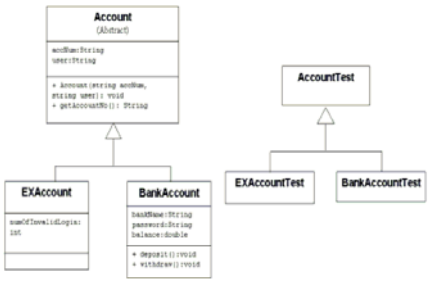

Fig. 3. Partial Class Diagram for Account, EXAccount and BankAccount

Extending TTCN-3 with an Inheritance Mechanism To extend TTCN-3 for specifying inheritance relationships between test modules, we investigate which elements in a test module are likely to be reused, and which elements can be extended or overridden in its derived test module. The result is shown in Table 1. From the table, we see that almost all of the elements can be reused directly, and some elements can be reused by means of an extend or override mechanism.

Table 1. Element reuse in a test module in TTCN-3

Elements			Reuse	Extend	Override
Definition	**Type Definition**	Built-in	N	N	N
		User-defined	Y	N	N
		RP Signature	Y	N	Y
	Test Data	Constant	Y	N	N
		Data Template	Y	N	N
		Signature Template	Y	N	Y
	Test Configuration	Communication Port	Y	Y	N
		Component	Y	Y	N
	Behavior	Function	Y	N	Y
		Named Alternatives	Y	N	Y
		Test Case	Y	N	Y
Control			N	N	N

We extend TTCN-3 by adding two key words: *private* and *public*, to indicate if an element is ready to be reused in a derived test module, and we assume that if an element is not specified explicitly with the key word private, the element is defined to be public by default. We also

propose to add a key word *extends*, which is used to specify that a test module is inherited from another test module. The modified TTCN-3 syntax in BNF form is as follows (the sequence numbers correspond to those defined in Annex A in [10]):

```
1. TTCN3Module::=TTCN3ModuleKeyword TTCN3ModuleId[extends TTCN3ModuleId]
 BeginChar
 [ModuleDefinitionsPart] [ModuleControlPart]
 EndChar
 [WithStatement] [SemiColon]
52. PortDefBody::=PortTypeIdentifier[extends PortTypeIdentifier] PortDefAttribs
73. ComponentDef::=ComponentKeyword ComponentTypeIdentifier
 [extends ComponentTypeIdentifier]
 BeginChar [ComponentDefList] EndChar
```

Aggregation There may also exist an aggregation relationship between test modules, e.g. between test modules for functional testing and those for unit testing. In Figure 4, a functional test scenario derived from the User Login use case consists of four steps. Each step corresponds to one test case defined in the test module for unit testing. The functional test scenario can be realized by executing these test cases. This relationship can be expressed as an aggregation relationship in UML. In section 4.5, we show how to specify a test module for a functional test scenario by reusing test cases defined in unit testing.

Functional Test Scenario	Unit Test Modules & Test Cases	
Use Case: User Login	Test Module	Test Case
Step 1: open index page; index page displays	hyperlinkCheck	Hyperlink_tc
Step 2: click "login" hyperlink; login page displays		Hyperlink_tc
Step 3: input "user ID" and "password", click "login" button; logonSuccess page displays	LoginServletTest	Login_tc
Step 4: click "logout" hyperlink; logout page displays	hyperlinkCheck	Hyperlink_tc

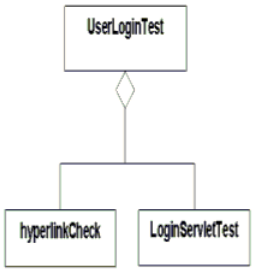

Fig. 4. Aggregation Relationship between Test Modules

4 Case Study

In this section we present part of a case study in which we apply our life-cycle testing approach with OO-TTCN-3 to a typical e-commerce system based on a high-yield, risk-directed strategy. The complete case study can be found in [17].

4.1 Overview of EX System

The EX system is an on-line currency exchange system which acts as a broker to provide customers the best exchange rate between

Canadian Dollars (CND) and U.S. Dollars (USD) among its linked banks at remote sites. The entire EX system can be divided into three subsystems: EX web application subsystem (EAS), bank subsystem (BS), and post office subsystem (PS). EAS obtains quotes from banks and presents the best one to web clients (WC).

4.2 Verify and Validate the System Analysis and Design Models

During the system AD phases, the AD models, e.g., UML use cases, need to be verified and validated. For example, guided inspection can be used to verify the use case model for correctness, completeness and consistency [22]. Then, test scenarios can be developed from the verified use cases: one is the normal scenario which is considered low-yield and low-risk/medium-risk; one or more are for alternative and exceptional scenarios which are considered high-yield and high-risk/medium-risk. Figure 5 lists part of the test scenarios derived from the User Login use case. Some design models, e.g., activity diagrams, can be used to analyze and improve the coverage of the test scenarios [17].

Test Scenario	Yield	Risk	Priority
[TS001] **Pre-conditions:** 1. Login page is displayed 2. User account is not locked **Action:** Enter valid *User ID* and *Password*, click *Login* button **Post-conditions:** Web page with account information is displayed, and a session is created	Low	Medium	Low
[TS007] **Pre-conditions:** 1. Login page is displayed 2. User account is **locked** **Action:** Enter valid *User ID* and *Password*, click *Login* button **Post-conditions:** 1. Web page with account locked message is displayed 2. User account is locked	High	High	High

Fig. 5. Test scenarios for the User Login use case

The test scenarios above can be specified in GFT, as shown in Figure 6. These GFTs can be used to validate whether the design models conform to the requirements, e.g. comparing them with the MSCs created in the development process.

After the GUI design has been done, test cases can be developed from test scenarios and the GUI design. The development of high-yield test cases consists of three basic steps: (1) Partition input conditions into equivalence classes. (2) Create test data based on boundary-value analysis. (3) Design test cases to cover all test scenarios [17]. These test cases then are ready for functional testing.

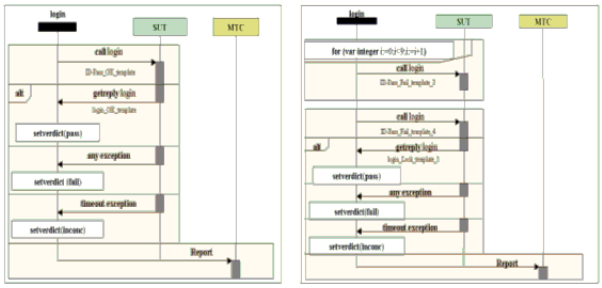

Fig. 6. Test Scenario 1 & 7 in GFT

4.3 Unit Testing

The unit testing level corresponds to the n-tier architecture of web applications: client tier testing, web tier testing, business tier testing, and EIS tier testing (which we will not discuss in this paper). This is because each tier is a relatively independent set of components. The techniques used with these components are similar in the same tier, but may be different in different tiers. Also, each tier has different responsibilities.

Client Tier Typical components in the client tier are html files, scripts embedded in an html file, and Java Applets. They run on the client side, usually within the content of a browser. From a functional perspective, common testing activities in the client tier include validating that every hyperlink in an html file is valid and that scripts and applets act as expected. Testing models based on the analysis of data flow and control flow of source code can be used to derive test cases [7]. The following TTCN-3 script demonstrates how to check the validity of a hyperlink.

```
module hyperlinkCheck{
modulepar {charstring testtype};
type record url {charstring protocol, charstring host, charstring portNum,
 charstring path}
template url hyperlink_index := {protocol:=https://,host:=www.site.uottawa.ca,
 portNum:=:1180, path:=ex/index.html}
template charstring status_ok := 200;
template charstring status_timeout := 408;
type port hyperlinkPortType message {out url; in charstring}
type port mCPType message {in verdicttype}
type port pCPType message {out verdicttype}
type component hyperlinkComponent {port hyperlinkPortType hyperlinkPort;
 port pCPType CP}
type component mtcComponent {port mCPType CP;
 port hyperlinkPortType hyperlinkPort;
 var integer activePTCs := 0;}
type component TSI {port hyperlinkPortType hyperlinkTSI}
function hyperlink_check (in url hyperlink, mtcComponent theSystem)
 runs on hyperlinkComponent {
  map(self:hyperlinkPort, theSystem:hyperlinkPort);
  hyperlinkPort.send(hyperlink);
  alt {
```

```
    [] hyperlinkPort.receive(status_ok) {setverdict(pass)}
    [] hyperlinkPort.receive(status_timeout) {setverdict(inconc)}
    [] hyperlinkPort.receive() {setverdict(fail)} }
  CP.send(getverdict); }
testcase hyperlink_tc(in url hyperlink, integer loops, out integer passedTCs,
 integer failedTCs,integer inconcTCs)runs on mtcComponent system mtcComponent{
  var verdicttype theVerdict;
  var hyperlinkComponent theNewPTC[loops];
  for (i:=1;i<=loops;i:=i+1) {
   theNewPTC[i]:= hyperlinkComponent.create;
   activePTCs  := activePTCs + 1;
   connect(mtc:CP, theNewPTC[i]:CP);
   theNewPTC[i].start(hyperlink_check(hyperlink, system)); }
  while (activePTCs > 0) {
    CP.receive(verdicttype:?)-> value theVerdict;
    activePTC := activePTC - 1;
    setverdict(theVerdict);
    if (theVerdict == pass)   { passedTCs := passedTCs + 1; }
    else if (theVerdict == fail)   { failedTCs := failedTCs + 1; }
    else if (theVerdict == inconc) { inconcTCs := inconcTCs + 1; } }
  all component.done; }
function basicFunctionality() return verdicttype {
  var verdicttype localVerdict;
  var integer nrP := 0, nrF := 0, nrI := 0;
  localVerdict := execute(hyperlink_tc(hyperlink_index,1,nrP,nrF,nrI));
  return localVerdict; }
control {
  var verdicttype overallVerdict;
  if (testtype == basicFunctionality) {
    overallVerdict := basicFunctionality(); } } }
```

Web Tier Components running in the web tier are JSP files, Java Servlets, and CGI programs etc. Web components are also identified by URLs, in the same way as html files, but run on the server side. In addition, servlet and CGI programs may utilize parameters wrapped in an HTTP request. These components are usually referred to as server pages, while html files are referred as client pages. Test modules in TTCN-3 for server pages are similar to those for client pages, but with a significant difference: using procedure-based ports which are based on a synchronous communication mechanism to specify procedure calls in remote entities, instead of message-based ports which are based on asynchronous message exchange. The following code segment shows how to specify a test module for testing Login Servlet by using a procedure-based port:

```
signature loginServlet(in url url_loginServlet, charstring ID,
 charstring password) return boolean exception (reasonType);
template loginServlet validLogin := {url_loginServlet := url_login_template,
 ID := C001, password := C001}
type port loginPortType procedure {out loginServlet}
```

Business Tier The objects in the business tier are used to implement the business logic of web applications. The objects can be represented by class diagrams, possibly with constraints written in Object Constraint Language (OCL). Test cases can be derived from the class diagrams and constraints. Often, there exists an inheritance relationship between these test cases, as we have discussed

in section 3.3.1. Test modules for specifying these test cases can be specified in OO-TTCN-3, which can specify the inheritance relationship between test modules appropriately.
In the EX system, Account is an abstract class (see Figure 3). It contains two attributes: accNum and user. One of the methods defined and implemented in the class Account is getAccountNo(), which returns the account number of the current user. Two subclasses, BankAccount and EXAccount, are derived from Account. BankAccount is used to describe the attributes and behaviors of bank accounts. EXAccount is used to describe the accounts in the EX system. The signatures and implementations of getAccountNo() do not change in the two derived classes. Therefore, test cases developed for getAccountNo() can be reused by the test modules for BankAccount and ExAccount. In addition, we only need to run the test suites once, either in the test module for BankAccount or in the module for EXAccount, to validate the method in the three classes. This also avoids testing the abstract class Account, which cannot be instantiated and is difficult to test directly. The following is a code segment which shows how to specify the test modules in OO-TTCN-3:

```
module AccountTest {
signature Acc_constr(in charstring AccNum,charstring user)exception(charstring);
signature getAccountNo () return charstring exception (charstring);
testcase getAccountNo_tc() runs on mtcType system mtcType{...} }
module BankAccountTest() extends AccountTest {
control {execute(getAccountNo_tc());} }
```

4.4 Integration Testing

The purpose of integration testing is to make sure all components of a software system work together properly. ATS defined for unit testing can be reused directly for integration testing. Figure 7 is the partial ORD (Object Relation Diagram) for the EX system. The test module LoginServletTest defined for web tier testing is ready for integration testing, which includes the components login server page, LoginServlet servlet, logonSuccess server page, and EXBean, and interactions between these components.

4.5 Functional Testing

The purpose of functional testing is to ensure the behaviors of a software system meet its functional requirements. Test scenarios in GFT developed at the system analysis and design phases can be used to generate test cases for functional testing. Actually, a bidirectional mapping between the core language and GFT is defined in [11], which makes it is possible to generate test scripts in core language from the scenarios in GFT automatically, or vice-versa,

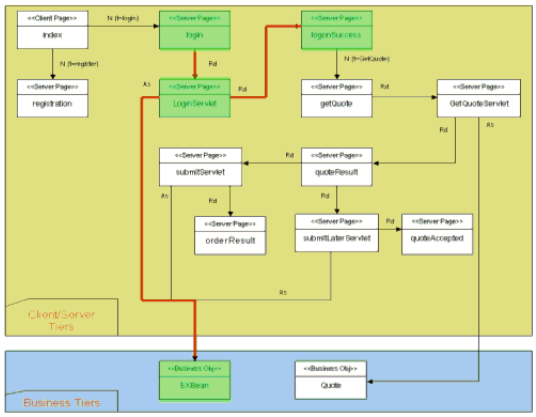

Fig. 7. Partial Object Relation Diagram for EX system

given specific tool support. On the other hand, test modules in TTCN-3 can be built manually. We can specify test cases developed in the system AD phases in TTCN-3. In addition, test scripts produced in unit testing can be reused in functional testing. The following is a code segment to test User Login functionality, which utilizes part of the definitions from test modules hyperlinkCheck and LoginServlettest.

```
function validLogin() return verdicttype {
  localVerdict:=execute(hyperlinkCheck.hyperlink_tc(hyperlink_index,1,0,0,0));
  if (localVerdict != pass) {return localVerdict;}
localVerdict := execute(hyperlinkCheck.hyperlink_tc(hyperlink_login,1,0,0,0));
  if (localVerdict != pass) {return localVerdict;}
  localVerdict := execute(LoginServletTest.validLogin_tc());
  if (localVerdict != pass) {return localVerdict;}
  localVerdict:=execute(hyperlinkCheck.hyperlink_tc(hyperlink_logout,1,0,0,0));}
```

4.6 Non-functional System Testing

Non-functional system testing is the process of testing an integrated software system to verify that the system meets its specified non-functional requirements. Test cases for non-functional system test cases, such as performance tests, can be specified in TTCN-3. Test scripts developed in the previous testing stages, such as functional testing, may be reused for non-functional testing. The following is an example of performance testing: adding a function in the hyperlinkCheck test module to simulate 1000 times of clicking on index.html, and then observing how many requests timeout or fail:

```
function performanceTesting() return verdicttype {
  localVerdict  := execute(hyperlink_tc(hyperlink_index,1000,nrP,nrF,nrI)); }
```

4.7 Concrete Requirements and Results

The above abstract test suite can be executed after transforming it into executable code in an implementation language such as Java.

This has been achieved by using one of the many commercially available TTCN-3 tools, like in our case, TTthree [18]. After that the actual execution of this code can be performed using a runtime environment like TTman [18] that allows a user to select a given test case to be executed. However, the abstract test suite can be executed only after organizing some adapter and coding/decoding code to transfer data from an internal representation to the real world representation. This can be achieved by using ETSI standard tri and tci classes interfaces and a set of APIs. Sample code can be viewed at www.site.uottawa.ca/ bob/ECTesting. Once the adapter and codec code compiled, they can be fed to the muTTman test case manager that shows a list of test cases and upon execution a test case execution trace as shown in Figure 8.

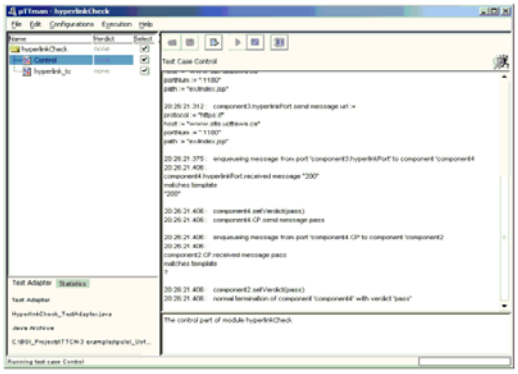

Fig. 8. Test Case Execution Result

5 Summary and Future Work

In this paper, we proposed an approach which realizes a life-cycle test process for e-commerce systems by specifying the ATS in OO-TTCN-3. The approach facilitates the reuse of test scripts. It also provides a unified and standard ATS interface to test tool vendors. This has significant potential to attract more support from the IT industry.
Making a complete object-oriented extension to TTCN-3 would be quite complex. In this paper we present a preliminary such extension. A formal description of this extension and a prototype tool that supports OO-TTC-3 will be considered in future work.

References

1. Kit, E.: Software Testing in the Real World. Addison-Wesley (1995)

2. Bashir, I., Goel, A.L.: Testing Object-Oriented Software: Life-Cycle Solutions. Springer-Verlag (1999)
3. Probert, R., Sims, D.P., Ghazizadeh, B., Li, W.: A Risk-Directed E-Commerce Test Strategy. In: Proc. of Quality Week Europe 2000 Conf. (QWE). (2000) 388–401
4. Jim, C.: Modeling Web Application Architectures with UML. Communications of the ACM **42** (2003) 63–77
5. Li, J., Chen, J., Chen, P.: Modeling Web Application Architecture with UML. In: Proc. of 36th Intel. Conf. on Technology of Object-Oriented Languages and Systems. (2000) 265–274
6. Yang, J.T., Huang, J.L., Wang, F.J., Chu, W.: An Object-Oriented Architecture Supporting Web Application Testing. In: Proc. of 23rd Annual Intel. Computer Software and Applications Conf. (1999) 122–127
7. D.C., K., Liu, C.H., Hsia, P.: An Object-Oriented Web Test Model for Testing Web Applications. In: Proc. of First Asia-Pacific Conf. on Quality Software. (2000) 111–120
8. Lucca, G.D., Fasolino, A., Faralli, F., Carlini, U.D.: Testing Web Applications. In: Proc. of Intel. Conf. on Software Maintenance. (2002) 310–319
9. Probert, R.L., Chen, Y., Ghazizadeh, B., Sims, D.P., Cappa, M.: Formal Verification and Validation for E-Commerce: Theory and Best Practices. Information and Software Technology **45** (2003) 763–777
10. ETSI: The Testing and Test Control Notation version 3, Part1: TTCN-3 Core Language, V2.2.1. European Institute Standards Telecommunication (2003)
11. ETSI: The Testing and Test Control Notation version 3, Part3: TTCN-3 Graphical Presentation Format (GFT), V2.2.1. European Institute Standards Telecommunication (2003)
12. Schieferdecker, I., Pietsch, S., Vassiliou-Gioles, T.: Systematic Testing of Internet Protocols - First Experiences in Using TTCN-3 for SIP. In: Proc. of 5th IFIP Africom Conf. on Communication Systems. (2001)
13. Dai, Z., Grabowski, J., Neukirchen, H.: Timed TTCN-3?A Real-Time Extension for TTCN-3. Testing of Communicating Systems **14** (2002)
14. Schieferdecker, I., Stepien, B.: Automated Testing of XML/SOAP Based Web Services. In: Proc. of 13th Fachkonferenz der Gesellschaft fur Informatik (GI) Fachgruppe Kommunikation in verteilten Systemen? (2003)
15. Schieferdecker, I., Vassiliou-Gioles, T.: Tool Supported Test Frameworks in TTCN-3. Electronic Notes in Theoretical Computer Science **80** (2003)
16. McGregor, J.D., Sykes, D.A.: A Practical Guide to Testing Object-Oriented Software. Addison-Wesley (2001)
17. Xiong, P.: Life-Cycle E-Commerce Testing with Object-Oriented TTCN-3. Master's thesis, University of Ottawa (2004)
18. Testing Technologies IST GmbH: The TTthree and uTTman TTCN-3 Tool Chain (2004) http://www.testingtech.de.