

# Scenario-Based Object-Oriented Test Frameworks for Testing Distributed Systems

W. T. Tsai, L. Yu, A. Saimi

Department of Computer Science and Engineering  
Arizona State University  
Tempe, AZ 85287

Ray Paul

Department of Defense  
Washington, DC

## Abstract

Testing is difficult and expensive, and testing distributed system even more difficult due to issues such as interoperability, collaboration, synchronization, timing, and concurrency. A tester often needs to spend significant time in developing lengthy testing code to ensure that the System Under Test (SUT) is reasonably well tested. This paper proposes an Object-Oriented (OO) framework to test distributed system rapidly and adaptively by using scenario modeling, state modeling, verification patterns, design patterns, regression testing, ripple effect analysis, simulation, automated test execution, and remote testing using TCP/IP or SOAP. Finally, this paper uses a supply-chain example to illustrate the key concepts.

## 1. Introduction

This paper addresses testing distributed systems. Many issues such as interoperability, timing, fault tolerance, availability, reconfiguration, reliability, security, and performance are some important ones in deploying distributed systems. Each of them is related to quality of service (QoS). Even though significant research has been performed on formal specifications and model checking, most of V&V activities still focus on testing as the primary quality assurance technique. The cost of testing can be as high as 50% to 70% of total development effort and budget in various industries such as aerospace, medical devices, and telecommunication.

Testing is often costly and time consuming, and testing distributed systems is especially difficult because many processes or objects may be active at the same time causing significant problems in synchronization, critical section, and message communication.

The objective of this research is to devise an object-oriented (OO) framework to test distributed systems rapidly and adaptively. By using the framework, a tester does not need to write test scripts, but focuses on identifying system scenarios. In this way, the framework can significantly reduce the time and effort to perform distributed systems testing. Test framework generates test scripts based on test scenarios/cases stored in a database and performs testing by dynamically binding objects and runtime invoking methods. If system design or requirement is changed, tester just needs to re-specify the test scenarios/cases. Java source codes of JUnit [5] test program can be generated.

The principal techniques include scenario specification techniques [1][21], OO test frameworks [18][22], and verification patterns [30]. The research provides infrastructure for testers to specify the system based on requirements, perform static analysis, perform dynamic simulation, execute test remotely via TCP/IP or SOAP, and perform runtime verification and evaluation. The proposed approach also addresses timing issues and timing analysis. To validate the methodology, this paper uses a test framework to test a supply chain system implemented using web services.

This paper is organized as follows: Section 2 provides a brief survey on related work; Section 3 introduces scenario-based test frameworks for distributed system testing; Section 4 presents both static and dynamic analyses that can be performed on scenarios; Section 5 presents distributed testing including distributed test agents; Section 6 presents customization techniques used in the framework; Section 7 demonstrates an example of testing a distributed application developed using web services; and finally Section 7 concludes this paper.

## 2. Related Research

Testing distributed systems has been studied for decades focusing on different issues. Ulrich [26] suggested two test architectures for testing distributed systems: a global tester that has total control over the distributed system, and a distributed tester comprising several concurrent tester components. Coordination between tester components is done by redundant observation of internal interactions, or by using synchronization events between tester components. Mathur [12] and Hoffmann [9] addressed the problems that arise when testing CORBA-based distributed systems. While some papers discussed the theoretical aspects of testing component-based distributed systems, most focus on unit testing of single components only [11][27][29].

OO test frameworks have been initially proposed by Firesmith [7], McGregor [13], and Poonawala [15] independently. The frameworks proposed by Firesmith and McGregor often follow the OO structure such as inheritance graph to develop the test framework, Poonawala proposed a techniques to organize test scripts so that they can be reused to test safety-critical applications even though the System Under Test (SUT) does not have the OO structure, and this approach has been used at an industrial site with significant results. Recently, several operational test frameworks have been developed to test OO programs in Java, e.g., JUnit [5], HttpUnit [3], Cactus [2], and Mock Objects [10], and these frameworks use OO design patterns [7] extensive, and is often based on specific programming languages such as Java or specific applications such as Internet publishing. Most of these test frameworks address unit testing, and are platform and protocol dependent. We have developed several OO test frameworks to test real-time embedded systems [21][22] and they address integration testing.

One common feature associated with all the existing testing frameworks is that they need the tester to understand/master the framework techniques before they can develop the test script. Unfortunately, it may take a long time for an inexperienced to master these frameworks before they can become proficient in testing applications using these frameworks.

## 3. Testing Distributed Systems

This paper proposes a scenario-based test framework to test distributed systems, where individual SUTs are connected through a network, such as LAN, Internet, satellite dishes or wireless networks. The main features of the test framework are:

1. **Scenario-based testing:** Testing scripts are developed based on specifying system scenarios, and scenarios are formalized as a sequence of events, actions and associated pre-/post-conditions [1][24][21].
2. **Automated test script generation from system scenarios:** Tester does not need to write any test source codes or test scripts. Instead, the framework generates test scripts based test scenarios/cases stored in the database. Currently, the framework generates test script in Java [23].
3. **Distributed automated test execution:** The framework can also execute tests by issuing test commands to distributed systems, because sub-systems are connected by a network, the framework also automatically generate communication commands to send test scripts/cases to appropriate systems for test and evaluation [23].
4. **Rapid test script generation by verification patterns:** The framework supports rapid test script generation by classifying system scenarios into patterns, and each pattern has a corresponding test script that can be parameterized to test all the system scenarios belong to the pattern. This approach promotes test script reusability and reduces the cost of test script generation significantly [30].
5. **Customization:** A tester can customize the test framework for their specific purposes [22][24]. This feature makes the test framework flexible to test a variety of distributed applications.
6. **Multi-level testing:** Test framework supports unit testing, integration testing, and system testing.
7. **Statistical modeling:** The framework evaluates the test performed by using a statistical model assurance-based testing (ABT) originally developed for Y2K testing [17].
8. **Uniform usage:** The test framework follows a simple process: *setup*, *execute* and *verify*. A tester has the freedom to add new testing strategies such as a new regression testing selection algorithm, but the tester is not allowed to change the overall testing strategy, i.e., *setup*, *execute*, and *verify*. In this way, test scripts developed can be reusable while at the same time allowing new testing strategies or techniques to be introduced into the framework. This also minimizes misuses and mistakes.
9. **Accommodation to changes:** When the SUT is changed, the user just needs to modify/update the corresponding scenarios specification [21][22][24]. The framework will re-generate the test script automatically to accommodate the change and

execute the new test script generated. This is supported by design patterns Strategy, Template Method, Composite, and Command [8]. The framework also supports regression testing to ensure that changes will not introduce undesirable ripples.

### 3.1 Framework Architecture

Test framework follows the three-tier architecture: front-end, middle and target system tiers (Figure 1).

*Front-end tier:* This provides a user-friendly interface (GUI and/or Web browser), and allows the tester to specify the test scenarios/cases derived from requirements using the textual/graphical representations, query the database, and review test results.

*Middle tier:* This is divided into two internal tiers: front-middle and back-middle tier. *The front-middle tier*

- ?? Organizes scenario specifications in an OO fashion: such as creating testing scenario objects, test case objects, input data objects, method signature objects, and complex scenario objects;
- ?? Performs a variety of analysis, such as completeness and consistency check, dependency analysis;
- ?? Executes tests (such as regression tests, functional tests) by sending commands cross the network using TCP/IP or SOAP;
- ?? Performs runtime verification;
- ?? Performs dynamic simulation.

The *back-middle tier* facilitates access to the database for storing test specifications and results.

*Target system tier:* In this tier, test agents act as proxies of the test master and perform tests on the target system on behalf of the master. The systems can be existing ones or prototypes. Test agents carry out test execution by collaborating with each other, and report the test results to the test master.

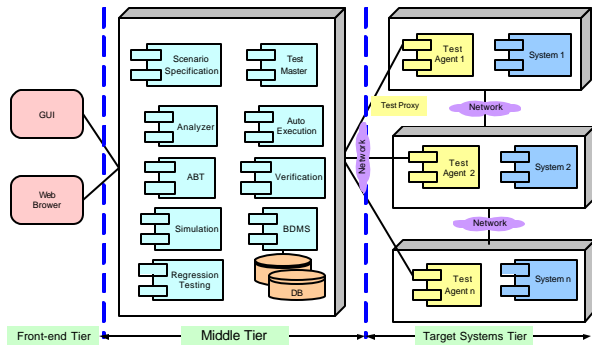


Figure 1. Overall Architecture of Test Framework

### 3.2 Scenario Specification

One of key activities in testing distributed systems is functional testing, which often involves specification of system behavior scenarios and development of test cases/scripts based on the specified scenarios. Distributed systems often have clear interface through which they interact with each other. For example, in a supply chain system, entities (such customers, retailers, manufacturers, and suppliers) interact with each other through well-defined API (Application Programming Interface). The Web Services Interoperability (WS-I) defines three interaction scenarios in supply-chain web services [28]:

- ?? One-Way: a consumer sends a request message to a provider without response from the provider.
- ?? Synchronous Request/Response: a consumer sends a request message to a provider. The provider receives the message, processes it and sends back a response.
- ?? Basic Callback: At runtime a consumer sends the initial SOAP request in a request/response sequence to the provider, which in turn sends back an immediate acknowledgement. At a later time the provider will initiate the final request/response sequence to the consumer containing the response data for the initial request sent by the consumer.

To derive scenarios of distributed systems, a tester can use the following steps:

- ?? Derive scenario specification for each sub-system, and formalize the scenario specification by annotating each scenario as a sequence of events, actions, and associated pre-/post-conditions;
- ?? Specify the interaction between each pair of sub-systems;
- ?? Derive the overall scenarios for the distributed system by combining the scenarios for individual sub-systems with the interaction.

#### Derive Scenarios for Each Sub-System

This step derives scenario specification from sub-system requirements. Each scenario can be classified as an atomic scenario, a sub-scenario, or a complex scenario. The derived scenarios are organized into a tree structure with each sub-tree represent a group of functionally related scenarios that the tester can analyze them together in a hierarchical manner. Scenarios are annotated with pre-conditions, events, actions, and post-conditions, and specified using OCL (Object Constraint Language) [24] and XML. The information specified is useful in various analyses such as dependency analysis, consistency analysis and concurrency analysis. For example, “A customer accesses to the retailer system with a valid customer ID” is a scenario in Retailer

system. The following sample shows an XML-based scenario.

```

<Scenario_retailer>
  <stateName>A customer accesses to the retailer system with a valid customerID </stateName>
  <event id="d">
    <name>Customer::login()boolean </name>
    <inputMethod>getCustomerID</inputMethod>
    <precondition>
      <pre stimulus="false">Retailer ::getCurrentStatus() == Ready</pre>
      <pre stimulus="true">Retailer ::verifyCustomerID() == TRUE</pre>
    </precondition>
    <postcondition>
      <post response="false">Retailer ::getCurrentStatus() == AcceptingOrder </post>
      <post response="true">Retailer ::login() == TRUE</post>
    </postcondition>
  </event>
  <State>
    ...
  </Scenario_retailer>

```

### Specify the Interaction between Systems

Individual distributed systems have their particular functionality, and provide API to the rest of the system. Although they may work independently, most of time they need to collaborate with each other to perform the mission. Systems interact with each other using communication patterns: one-way, synchronous and callback. For example in a supply chain system, Retailer provides to Customer three services (or API): *login*, *getCatalog*, and *submitOrder*; to Manufacture one service: *submitShippingNotice*. On the other hand, Manufacturer provides to Retailer one service: *submitPurchaseOrder*. *submitPurchaseOrder* is one way interaction; and *login*, *getCatalog*, and *submitOrder* is synchronous interaction; while *submitShippingNotice* is callback interaction. The services provided have detailed description, e.g., services provided by Retailer have:

- ?? *login*: The customers login the SCM system with customer ID.
- ?? *getCatalog*: The customer requests the list of products.
- ?? *submitOrder*: The customer submits an order to specify the product and quantity, and then receives a response indicating goods will be shipped.

### Derive the Overall System Scenarios

Based on scenarios for each individual sub-systems and interaction among these sub-systems, a tester can now derive the overall system scenarios from the end user point of view. For example, end-user initiates a request to system A, which does certain tasks and sends back results to the user. In some circumstances, system A may need cooperation from system B, and in turn sends a request to system B to trigger a B's service. Finally, system B returns the results to system A or directly to the end-user. Such information can be viewed a scenario of distributed systems by combining scenarios for individual sub-systems with interaction scenarios. For example, scenario "Customer Successfully Purchases Goods" consists of "A customer accesses to the retailer system with customer ID", and "purchases the electronic product to specify the

product and quantity", then "requested product is shipped, and the retailer system returns confirmation to customer".

### Generate Test Scripts from Scenarios

Scenario-based testing follows the three-step process: *setup*, *execution* and *verification*, and thus test scripts can be generated based on these three aspects.

- ?? The *setup* aspects of test scripts can be obtained by issuing commands to set the SUT to the state specified in the preconditions of the formalized scenario, and generating inputs events as specified in the formalized scenario.
- ?? The *execution* aspects can be generated by issuing method calls corresponding to the actions specified in the scenarios with appropriate method signatures to the SUT either locally or remotely. In case of remote service invoking, information such as protocol type and remote location are necessary.
- ?? The *verification* aspect can be obtained by looking up the verification pattern associated with the scenario following the pattern approach [30].

Once obtaining the test scripts, test framework can automatically execute the test (see Section 5).

## 4. Analyses

Once scenarios specifications are obtained, a tester can perform both static and dynamic analyses.

### 4.1 Static Analyses

The static analysis includes completeness and consistency checking of scenarios, dependency analysis, timing analysis, pattern analysis, risk analysis, and usage analysis [24].

- ?? Completeness analysis – this ensures that all the needed scenarios are specified and there are no missing scenarios.
- ?? Consistency analysis – this ensures that all scenarios specified are consistent with each other.
- ?? Dependency analysis – two scenarios may have dependency relationship if they share input or outputs. A scenario may be control dependent on another scenario if the output of later scenario is used to control the decision in the first scenario. Dependency analysis is useful in regression testing, impact analysis after system modifications. Several types of dependency are possible and include input/input dependency, input/output dependency, output/output dependency, and control dependency (an output of a scenario is used as a control condition for another scenario).



messages among sub-systems including requests and their responses, and each agent sends information back to master asynchronously.

- ?? **Request Change Configuration:** The master requests agents to change configuration of the system to start a new round of testing. Each agent resets its own state and reports to the master.
- ?? **Notice State Change:** The test master adds listeners (this is Observer design pattern) to the agents. When the system state is changed either triggered by internal or by external events, the agent notices this change to the master asynchronously.
- ?? **Verify Results:** The master receives the information such as exchange messages and state changes from agents asynchronously, and collects the data related to specific test scenarios. Then the master verifies the test results with the expected output.

## 6. Customization

The test framework can be configured to test both individual sub-systems and distributed systems. To test an individual sub-system, the test master uses scenarios related to that system only. By designing different scenarios, different aspects of the system can be tested. For example, a tester may design scenarios that address module testing, integration testing, or distributed testing aspects of the system. Furthermore, test scripts generated by the framework use design patterns such as Strategy and Template Method so that the test scripts can be easily customized to address a specific purpose. For example, a tester may develop a new script by defining new atomic methods used in a test script with a Template Method. Also, the framework binds communication commands at runtime using Java's reflection technology [4] during distributed testing, and thus the framework can be used in a variety of network environments by changing the binding.

## 7. An Application: Testing Web Services

This section illustrates the proposed techniques in testing a Supply Chain Management (SCM) application developed using web services. In general, testing web services is difficult due to the following reasons:

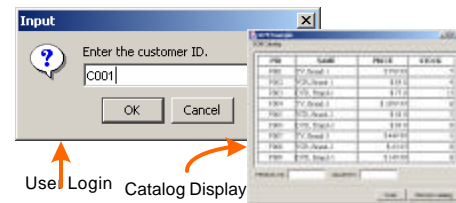
- ?? Web services often runs on loosely coupled architecture but still demand high assurance;
- ?? Web services can be invoked by unknown parties with unpredictable requests, among service providers and service clients;
- ?? Web services have runtime behavior including dynamic discovery and binding with multi-parties including middleware and other web services;
- ?? Web service has performance issues including concurrent accessing and object sharing;

This SCM application follow the SCM specification of Web Services Interoperability (WS-I) Organization [28]. The entities in a supply chain are Suppliers, Manufacturer, Wholesaler, Retailer, and Customer. The retailer system contains the following web services (Figure 3):

- ?? login: The customers login the SCM system with their customer ID.
- ?? getCatalog: The customer requests the list of products.
- ?? submitOrder: The customer submits an order to specify the product and quantity, and then receives a response indicating the time good will be shipped.
- ?? submitShippingNotice: After manufacturer finishes processing an order, it submits the shipping notice.

The manufacturer service provides one operation:

- ?? submitPurchaseOrder: The manufacturer places a purchase order when it finishes goods.



**Figure 3.** Client GUI

The example has multiple customers, one Retailer and multiple Manufacturers. To perform the testing, the experiment sets up the web services environment using the Java runtime environment (JRE) 1.4, Apache Tomcat 4.0, IBM web services toolkit (WSTK) 3.2.

Some typical scenarios in a SCM system are:

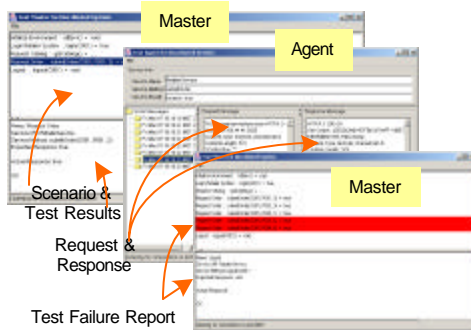
- ?? Customer successfully purchases goods.
- ?? Retailer orders products from a manufacturer.
- ?? Customer failed to purchase goods due to invalid customer ID.
- ?? Customer failed to purchase goods due to invalid order.
- ?? Customer failed to purchase goods due to out of stock.

A number of faults are seeded into the web services, the number indicating the number of bugs:

- ?? API (8): such as Input type, mismatching types, number of parameters, return type mismatching;
- ?? Internal services (7): malfunctions (e.g., can not update the inventory; can not find a proper manufacturer), the length of password, patterns of password;
- ?? Communication (3): using different protocols.

To test the SCM system, a tester does not need to write any testing code; instead the tester can focus on

specifying scenarios. Once the tester finishing scenario specification, the tool generates test scripts and detects all the seeded bugs in the system.



**Figure 4.** Roles of Test Master and Test Agent

If the SCM system is modified, it is necessary to test the modified feature as well as to perform regression testing to ensure that the modifications do not introduce new bugs. The former can be done by re-specifying scenarios that involved the modified feature, and generating test scripts based on these scenarios. Regression testing can be done by identifying those scenarios that are potentially affected by the change by dependency analysis. The framework then generates or re-generates test scripts corresponding to these scenarios for regression testing. The modified system scenarios can be further simulated to see if the runtime behavior is still valid.

## 8. Conclusion

This paper presents a scenario-based OO test framework for rapid distributed system testing. Using the framework, a tester does not need to write testing code, instead focuses on scenario identification and specification. The framework generates test cases/scripts, and executes them automatically. Whenever a change occurs, the tester just needs to re-specify the modified scenarios so that new test scripts can be generated to test the modified feature. The framework can also perform regression testing by identifying those affected scenarios by dependency analysis. This paper used a SCM application for illustration.

## References

- [1] X. Bai, W. T. Tsai, R. Paul, K. Feng, and L. Yu, "Scenario-Based Modeling and Its Applications to Object-Oriented Analysis, Design, and Testing", Proc. of IEEE WORDS 2002, pp. 140-151.
- [2] Cactus project, <http://jakarta.apache.org/cactus/index.html>.
- [3] HttpUnit, <http://httpunit.sourceforge.net/>.
- [4] Java.Sun.com, <http://java.sun.com/>.
- [5] JUnit.org, <http://www.junit.org/index.htm>.
- [6] M. Fayad, D. C. Schmidt, and R. E. Johnson, *Building Application Frameworks*, Wiley, New York, NY, 1999.
- [7] D. G. Firesmith, "Pattern Language for Testing Object-Oriented Software", Object Magazine, Vol. 5, No. 9, Jan. 1996, pp. 42-45.

- [8] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley, Reading, MA, 1994.
- [9] A. Hoffmann, A. Rennoch, I. Schubert, and A. Vouffo-Feuffio, "CCM Testing Environment", <http://www.fokus.fhg.de/tip>, 2001.
- [10] A. McCarthy, "Unit and Regression Testing", Dr. Dobbs Journal, Feb. 1997.
- [11] T. Mackinnon, S. Freeman, and P. Craig, "Endo-Testing: Unit Testing with Mock Objects", Proc. of Extreme Programming and Flexible Processes in Software Engineering - XP2000, 2000.
- [12] Martin et al., "Extending SUnit to Test Components", In K. Bauknecht et al. (editors): Informatik 2001, vol. 2, Sep. 2001, pp. 834 - 839
- [13] J. D. McGregor and A. Kare, "Parallel Architecture for Component Testing of Object-Oriented Software", Proc. of Annual Software Quality Week, Software Research Institute, May 1996.
- [14] A. P. Mathur, "Testing Distributed Systems", Status Report of NSF and SERC, 1999.
- [15] M. Poonawala, S. Subramanian, R. Vishnuvajjala, W. T. Tsai, R. Mojdehbaghsh, and L. Ellio, "Testing Safety-Critical Systems -- A Reuse-Oriented Approach", Proc. of 9th International Conference on SEKE, 1997, pp. 271-278.
- [16] A. Troelsen, *C# and the .NET Platform*, Addison Wesley, Reading, MA 2001.
- [17] W. T. Tsai, R. Paul, W. Shao, S. Rayadurgam, and J. Li, "Assurance-Based Y2K Testing", Proc. of IEEE HASE, 1999, pp. 27-34.
- [18] W. T. Tsai, Y. Tu, W. Shao and E. Ebner, "Testing Extensible Design Patterns in Object-Oriented Frameworks through Hierarchical Scenario Templates", Proc. of IEEE COMPSAC, 1999, pp. 166-171.
- [19] W. T. Tsai, X. Bai, R. Paul, W. Shao, and V. Agarwal, "End-to-End Integration Testing Design", Proc. of IEEE COMPSAC, 2001, pp. 166-171.
- [20] W.T. Tsai, X. Bai, R. Paul, and L. Yu, "Scenario-Based Functional Regression Testing", Proc. of IEEE COMPSAC, 2001, pp. 496-501.
- [21] W. T. Tsai, L. Yu, R. Paul, T. Liu, and A. Saimi, "Developing Adaptive Test Frameworks for Testing State-Based Embedded Systems", Proc. of IDPT, 2002.
- [22] W. T. Tsai, Y. Na, R. Paul, and F. Lu, "Adaptive Scenario-Based Object-Oriented Test Frameworks for Testing Embedded Systems", Proc. of IEEE COMPSAC, 2002, pp. 321-326.
- [23] W. T. Tsai, R. Paul, W. Song, and Z. Cao, "Coyote: An XML-Based Framework to Test Web Services Proc. of IEEE HASE, 2002, pp. 173-174.
- [24] W. T. Tsai, L. Yu, X. Liu, A. Saimi, and Y. Xiao, "Scenario-based Test Generation Tool for Embedded Systems", to appear in Proc. of IEEE IPCCC 2003.
- [25] W. T. Tsai, R. Paul, Y. Wang, C. Fan, and D. Wang, "Extending WSDL to Facilitate Web Services Testing", Proc. of IEEE HASE, 2002, pp. 171-172.
- [26] A. W. Ulrich, P. Zimmerer, and G. Chrobok-Diening, "Test Architectures for Testing Distributed Systems", Proc. of Quality Week 1999.
- [27] E. Weyuker, "Testing Component-Based Software - A Cautionary Tale", IEEE Software, Sep.-Oct. 1998, pp. 54-59.
- [28] WS-I: <http://www.ws-i.org/>, "WS-I Usage Scenarios", "Supply Chain Management Use case Model", "Sample Application Supply Chain Management Architecture".
- [29] Y. Wu, D. Pan, and M. Chen, "Testing Component-Based Software". International Conference Software & Systems Engineering and their Applications - ICSSEA '2002, December 2002.
- [30] F. Zhu, "A Requirement Verification Framework for Real-time Embedded Systems", PhD dissertation, Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN 55455, 2002.