University of Alberta

Location-based Services Testing with A Scalable Test Framework

By

Jiang Yu ©

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of Master of Science

Department of Electrical and Computer Engineering

Edmonton, Alberta

Fall 2005

Your file Votre référence
ISBN: 0-494-09323-4
Our file Notre référence
ISBN: 0-494-09323-4

University of Alberta

Library Release Form

Name of Author: Jiang Yu

Title of Thesis: Location-based Services Testing with A Scalable Test Framework

Degree: Master of Science

Year this Degree Granted: 2005

Permission is hereby granted to the University of Alberta Library to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves all other publication and other rights in association with the copyright in the thesis, and except as herein before provided, neither the thesis nor any substantial portion thereof may be printed or otherwise reproduced in any material form whatsoever without the author's prior written permission.

Jiang Yu

Date: Sept 30, 2005

ABSTRACT

With the rapid growth of mobile commerce in recent years, Location-based services (LBS) have generated a lot of interests and attracted many new researchers and market players developing and offering numerous applications in recent years. Therefore, effective testing strategies need to be considered for LBS. However, various technologies and the complex architecture involved in LBS cause numerous challenges on the LBS testing from many perspectives, including functionality, usability, performance, scalability, interoperability and security. In this thesis, we investigate the principles of all types of LBS and define the primary challenges on the LBS testing, and present a new test framework for testing Location-based services. The key idea of this test framework is to introduce a scalable test framework that enables a rapid prototyping test environment and the easy integration of third party components. We implement our test framework with the TTCN-3 testing language and the SWANS wireless simulator, and evaluate the test framework with an empirical investigation to demonstrate the feasibility and effectiveness of our test framework on the LBS testing from six testing strategies, including functional testing, usability testing and server side testing, performance testing, security testing and interoperability testing.

Acknowledgements

For his indispensable guidance during my research at the University of Alberta, the efforts of Dr. James Miller are gratefully acknowledged. Additionally, I would like to thank Dr. Rimon Barr for his comments and advice during our many insightful conversations.

TABLE OF CONTENTS

List of Tables

List of Figures

Chapter 1 Introduction

1.1 Introduction and Definition

As one part of M-Commerce applications, Location-based services (LBS) have generated a lot of interest and attracted many new researchers and market players developing and offering numerous applications and research in recent years. According to the UMTS Forum, Location-based services daily traffic will grow from 1.3 Tbytes in 2012 to 160 Tbytes in 2020, caused by the combination of both subscriber's growth and frequency of use growth [UMTS 05]. In the US, the Federal Communication Commission (FCC) has mandated that, by the end of 2005, all operators will be able to locate the position of any emergency call placed by a mobile phone with an accuracy of 125 meters. However, delivering new services requires developing means and testing tools to assist the creation and support the quality of the service. Consequently, testing of Location-based services becomes one of important strategies for the success of location-based services.

In this chapter, we briefly address the problem of mobile commerce applications, and describe the software automation testing process and terms that are used in this thesis. Also, we present current research work on location-based services testing and the motivation for our research. Finally, we describe the scope and contributions of our research, and give the organization for the remainder of this thesis.

1.1.1 Mobile Commerce Applications and Limitations

Mobile commerce (M-Commerce) is an emerging discipline with various entities involved, i.e. mobile applications, mobile devices, service provider, content provider, and wireless networks, as shown in figure 1-1. The emerging mobile service includes object identification, health monitoring, location discovery, M-payment, digital content, mobile entertainment, corporate services, M-government, and M-education [UMTS 05].

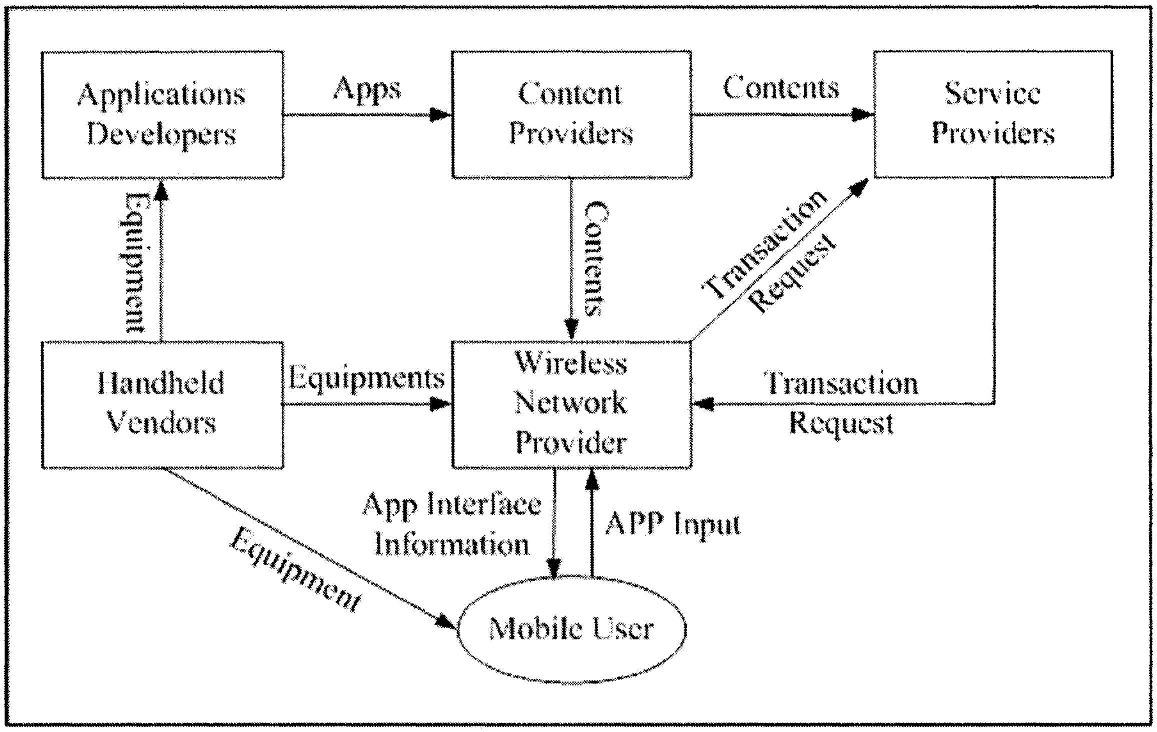


Figure 1-1: Mobile Commerce Life Cycle [Varshney+01]

While M-Commerce applications have grown explosively, the specific characteristics of M-Commerce application bring big challenges to improve software quality. Testing M-Commerce applications differs from the testing of traditional desktop client/server applications in many aspects. Table 1-1 shows the limitation category of M-Commerce applications. Generally, the limitations of M-Commerce applications can be concluded from the following aspects: mobile device limitations and wireless network limitations. The main difference between M-Commerce applications and desktop C/S applications is that, the client application is running on the micro web browser at mobile devices, i.e. PDA, Smartphone, BlackBerry, that have many restrictions that a desktop client application does not process. Besides, M-Commerce applications run on a wireless network with many uncertainties within that environment.

Mobile Device Limitations	**Description**
Small screen size	The screen size may be not big enough to properly display all content converted from the standard HTML web page.
Low CPU Speed	It may take more execution time of client application due to low CPU speed.
Limited RAM memory	The limited RAM memory prevent the mobile device running complex applications.
Short battery life dependencies	It may cause data loss because of loss of power during the execution of the client application.
Limited secondary storage	Less data storage capability than desktop applications
Different OS platforms**	The client application is platform dependent and may fail when run on different OS platform, i.e. Windows CE, Symbian, and Palm OS.
Different micro browsers**	The client application may not be suitable for each type of micro browsers, i.e. IE2002, Blazer, and Wapaka, which might cause incompatibility problems.
Wireless Network Limitations	**Description**
Poor network bandwidth	Poor network bandwidth limits the capacity of data transferred over the wireless network.
Different network infrastructure	The mobile client application may fail when running across wireless networks provided by different network provider.
Message size limitation	The input data may be lost if it is larger than the maximum size allowed to be transmitted over the wireless network.

** See Appendix B for current available browsers for each OS on the mobile device.

Table 1-1: Limitation Category of M-Commerce Applications

1.1.2 Software Automation Testing Process

Software testing is the process of exercising or evaluating a system or system component by manual or automated means to confirm that it satisfies requirements or to identify differences between expected and actual results. According to [Tassey+03], it is reported that bugs in the final products cost the United States economy $59.5 billion per year. It has been known that, in a typical programming project, approximately 50% of the elapsed time and over 50% of the total cost are expended in testing the program or system being developed [Myers 79]. The increasing focus has been put on how to design effective test cases and improve the test efficiency, and various methodologies and approaches for testing have become mature and been adopted by testing professionals.

Software test automation has been widely used in the software industry. Software test automation refers to the activities and efforts that intend to automate engineering tasks and operations in a software test process using well-defined strategies and systematic solutions [Li+04]. The main objective of software automation test is to reduce manual testing activities and redundant test operations using a systematic solution to achieve a better testing coverage and improve the software quality.

Building a suitable test framework is one of the key factors for the success of software test automation. To achieve this, a cost-effective test process should be established to support engineers to carry on various test automation activities with the provided test framework and develop required test automation solutions. Figure 1-2 shows a typical software automation test process. According to [Li+04], the following steps are necessary in a typical test automation process:

- Test automation planning
- Test automation design
- Test framework development
- Test framework deployment
- Review and evaluation

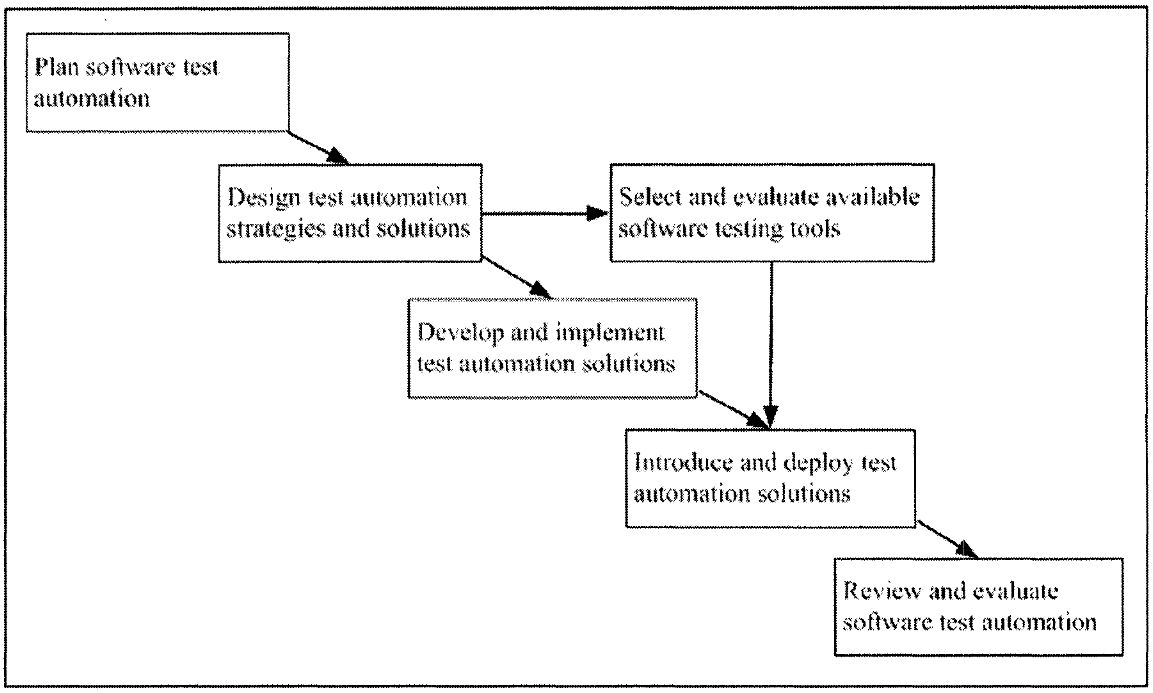

Plan software test automation
Design test automation strategies and solutions
Select and evaluate available software testing tools
Develop and implement test automation solutions
Introduce and deploy test automation solutions
Review and evaluate software test automation

Figure 1-2:Software Test Automation Process [Li+04]

Test Automation Planning

Test automation planning is the first step of software test automation. The main activity of test automation planning is to build up a comprehensive plan that covers the specification of test automation objective, scope, strategies, requirements, schedule and budget.

Test Automation Design

At this step, the detailed test automation solutions needs to be developed to achieve the major test automation objectives and meet the given requirements in the test automation plan. The test environment must be provided to support the execution of applications under test. Therefore, suitable testing tools, i.e. commercial and open-source tools, simulators and emulators, need to be selected to support the test automation process. Also, detailed design for the required test automation solutions needs to be conducted in this stage.

Test Framework Development

The primary objective of this step is to develop a qualified framework and facilities to satisfy the requirements of test automation solutions. As many test automation projects fail due to poor quality and documentation, we must ensure that the developed framework is reliable and reusable with good documentation.

Test Framework Deployment

At this step, the developed test framework and facilities must be introduced and deployed into a project or onto a product line. Basic user training and user support activities are necessary in this step.

Review & Evaluation

After a new test framework is deployed, a review process should be conducted to identify its issues and limitations, and evaluation criteria need to be set up to fully evaluate the provided test framework features. The review's results will then return valuable feedback to the test automation group for further improvements and enhancements.

1.1.3 Definition

Most of the following definitions are taken from [Li+04], except those terms that are otherwise referenced.

Component

A *component* is any software aggregate that has visibility in a development environment, for example, a method, a class, an object, a function, a module, an executable component, a task, a utility subsystem, or an application subsystem. This includes executable software entities supplied with an application programmer interface (API) [Binder 00].

Functional Testing

Functional testing is generally required for all products. The purpose of functional testing is to reveal defects related to the functionality of products or components, and conformance to any documented functional requirement specification [Kaner+99].

Stress Testing

Stress testing is the testing conducted to evaluate a system or component at or beyond the limits of its specified requirements with the goal of causing the system to fail [IEEE 90].

Performance Testing

Performance testing measures the response times of a system to complete a task and the efficiency of the algorithms under various conditions. Therefore, performance testing also takes into consideration possible hardware platforms, operating systems, and other applications used by the customers.

Usability Testing

Usability testing ensures that the presentation, data flow, and general ergonomics of the application meet the requirements of the intended users. This testing phase is critical to attract and keep customers. Usually, manual testing methods are inevitable for this purpose.

Security Testing

Testing which confirms that the program can restrict access to authorized personnel and that the authorized personnel can access the functions available to their security level.

Interoperability Testing

Interoperability Testing determines that a product implementation of an Implementation Specification interoperates with other product implementations of the same Implementation Specification, different but related Implementation Specification(s), or within a particular computing environment [OGC 05].

Grey box testing

Grey box testing is a combination of black box and white box testing to reveal omissions and surprises. Information from requirements, design, and source code are used to develop test cases.

Emulator

A device, computer program, or system that accepts the same inputs and produces the same outputs as a given system [Alliance 05].

Simulator

The programs developed to simulate the functions and behaviours of external software and hardware entities, components, or subsystems.

1.2 Statement of the Research Problem and Motivation

1.2.1 Related Work and Existing Problems

Based on the fact that the mobile device usually has a less powerful processor with limited memory, a small screen and a restricted keyboard, it is very hard to build and debug the software on the mobile device itself. Moreover, the mobile device has to be carried to different locations in order to test location change scenarios, which is laborious and inefficient.

Currently a lot of research work has been proposed on testing context-aware applications or location-based services. One popular way to test the software application in the mobile device is to use a software-based emulator for the mobile device. Most mobile device providers offer their own device emulators, such as BlackBerry JDK, Motorola iDEN SDK, Palm PDA emulator as well as Java Wireless Toolkits provided by Sun. These emulators and toolkits are usually designed to run on the workstation and simulate the application-level execution environments of the mobile device, such as transmission, radio and battery. As a result, wireless applications can run on the simulated mobile device and work in a simulated wireless network. These emulators are very convenient and widely used in the development of software for mobile devices. However, as mentioned above, they are focusing on application-level simulation and therefore are not suitable for system-level testing, such as interoperability and scalability testing, for the whole location-based service (LBS).

Ichiro Satoh [Satoh 03] describes a system for testing mobile computer software. The system offers a mobile agent-based emulator which performs application transparent emulation of the mobile device for the application and allows the application to travel across the network and connect to the local server to simulate its movement and execution within the network. This testing framework distinguished itself by introducing logical mobility as a methodology for building and testing applications requiring the physical mobility. Specifically, each local server has its own access point host, which provides a runtime system for executing and migrating mobile agent based emulators. In other words, the physical movement of a mobile computing device from one network and attachment to another is simulated by the logical mobility of a mobile agent-based emulator with the target applications from an access-point computer in the source network to another access-point computer in the destination network. Therefore, the mobile application can keep executing its process after it is located at another host, based on the code and state (Networked running state, Isolated running state and Suspended state) of the application carried by the emulator. This testing framework supports a runtime system for the execution of the test application. It, however, does not support any network disconnection operations or addressing schemes for the mobile device. Moreover, it can only be deployed and executed within the domain of the current sub-network and doesn't have network simulations for wireless protocols, such as GPRS and wireless LAN.

Another approach, UBIWISE, is a simulator used for ubiquitous computing systems design. This simulator concentrates on computation and communication devices situated within their physical environment [Barton+03]. It simulates the physical environment, including environment interaction and multi-user experience, surrounding devices and users with a 3-D rendering engine. It also simulates prototypes of new devices, device interaction, services and protocols. The core of the simulation is creating the virtual counterparts of devices and the picture frames present in the physical environment. The UbiWise simulator has excellent support for interaction, providing the user with graphical feedback and 3D virtual representation of the simulated environment on the simulated devices and environment. Also, the user can utilize the UbiWise simulator for the rapid prototyping of ubiquitous computing applications. However, compared with other location based testing frameworks, UbiWise supports connecting simulated devices to applications but doesn't support network emulation for application packets. Thus, the developer cannot compare their implementation's behaviour over different networks and protocols.

A hybrid test and simulation environment, provided by Ricardo Morla and Nigel Davies from the Lancaster University, describes a new approach to the test and evaluation of location-based applications without the extensive resource investment necessary for a full application implementation and deployment, focusing on component interaction, networking, location changes, and multiple component instances [Morla+04]. It introduces network simulators to provide different network protocols, such as TCP/IP, wireless LAN and GPRS, and integrates a third-party context simulator [Dey 00] into the test environment in order to simulate context change events. As shown in figure 1-3, the test environment consists of several components, including the system manager, application daemons, and simulators. These components cooperate in a distributed system environment to test the selected location-based application, interacting with each other using a web service interface. The system manager conducts the experiment, setting up the simulators, starting the applications under test via the daemons, logging data throughout the experiment, and supplying feedback to the user. By using different network and context simulators, this testing environment can achieve a less resource-demanding evaluation and easily change the prototype to support other applications. It, however, does not support scalability testing for Location-based services (LBS) when multi-mobile devices try to concurrently interact with a server.

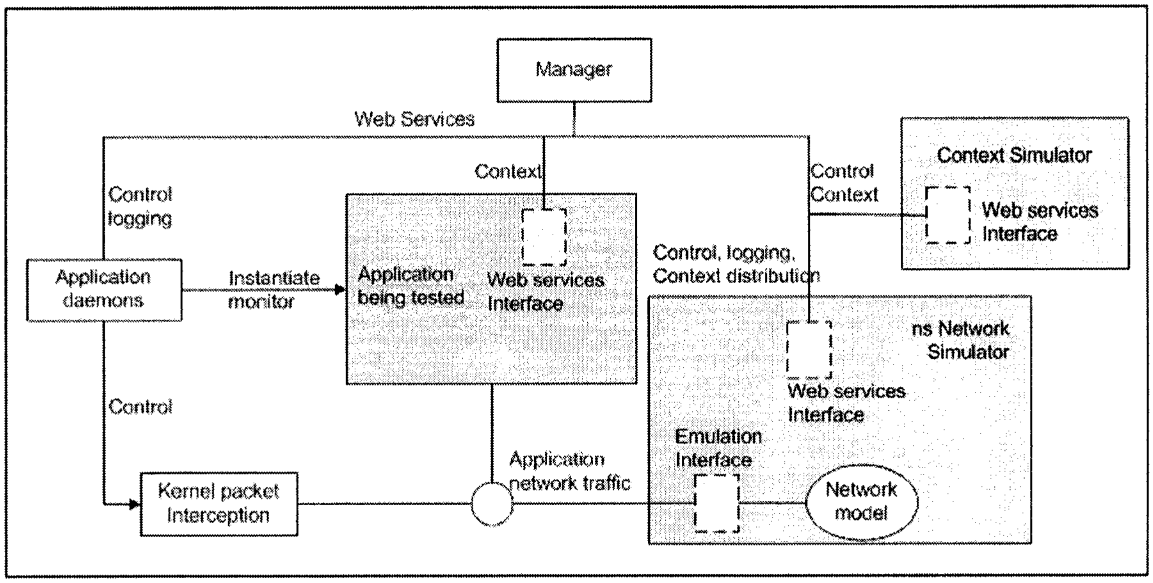


Figure 1-3: Ricardo Morla's Test Environment [Morla+04]

1.2.2 Motivation

Most of current research is mainly focused on the network infrastructure and system security for the mobile applications. Location-based services (LBS) distinguish themselves from other software applications by the changes in location, which must be considered in the testing strategy. One direct way to test location change is to carry the handheld by the tester and get the local position by interacting with the local network, which is almost impossible to do because of excessive demands upon manpower.

Alternatively, many researchers and handheld manufacturers tend to provide either mobile device/application emulators or network and context simulators to simulate the physical environment and test location-based applications. Although these methodologies can reach certain levels of testing the application, they are either limited to specific handhelds, or too sophisticated to adapt to the requirements of the LBS industry, where its market is evolving very fast.

Location-based service (LBS) is a mobility service that discovers the derived location of a mobile user to provide services that have a geographic context. With a number of core components in the LBS systems, it is essential to ensure each component's correct functionality in any LBS transaction. Therefore, from the testing perspective, it is necessary to test, not only location-based applications, but also other components in LBS, such as sever side applications and the location management platform (See chapter 2 for details about LBS and its components).

From the above section, we can discover that current research has focused on either building a simulator for the location-based application at the application-level, or simulating the current physical environment at the network-level and system-level. None of them can evaluate the location–based service from both client side and server side. In addition, each testing framework presented above built the test environment with its unique methodology, which is application-dependent and platform-dependent. As a result, it increases the difficulty of applying the test framework to various location-based services running on different platforms.

Motivated by the above considerations, we intend to propose one test framework which can be used to evaluate location-based services with a systematic method, and test components contained in both the client side and the server side, including network performance, web sever performance, application server components, and functions of location-based applications (LBA). Also, this test framework should have a standard interface to integrate third-party test components and expand the test framework to apply the test framework to future location-based services.

1.3 Scope of the Thesis and Overview of Contributions

This thesis presents a new scalable test framework for testing location-based services (LBS). Specifically, the key contribution of this thesis is the design and prototype of a new approach on the LBS testing. This approach intends to provide a test framework which can perform most of test strategies that are essential for the success of the LBS system (See section 4.6 for the details of test strategies on LBS with our test framework). Instead of the application-level emulation, which is the main approach of [Satoh 03], this test framework focuses on the *system-level testing* for the LBS system, which primarily focuses on the verification of interactions among the location-based application (LBA), server-side applications, and other third party components, such as the content provider and the location management platform (See chapter 2 for more details about third party components). For instance, can LBA receive correct service content from the server? How does the sensitive data transmitted within the wired and wireless network communication? How does the server response to multiple concurrent user requests? How well is the LBS system on integrating the components developed by different vendors. Such issues must be covered in the system-level testing for the LBS system.

In the system-level testing, we need to consider the typical challenges brought by the specific characteristics of the LBS system, including the functionality, the usability, the performance of wireless network and server side applications, the interoperability and the security (See chapter 2 for more details about challenges of testing LBS). The system-level testing, however, does not cover the testing specific functions, i.e. graphical user interface, local storage mechanism, for the individual components in the LBS system, such as the location-based application (LBA) running on the mobile device, which can be done by most of mobile device emulators. Compared with our test framework, either the emulator in [Satoh 03], UBIWISE, or the test environment in [Morla+04] can only provide limited test strategies, i.e. functional testing, for the location-based application (LBA), and cannot perform system-level testing for the location-based service (LBS), such as verifying the security mechanism and interoperability of the system, etc.

Rather than defining a test system architecture around specific application to be tested, this thesis emphasizes on building a standardized test framework based on a cost-base solution to provide the *scalability* and *extensibility* for the system-level testing of LBS that contains different components, and within different vendors. Therefore, we need to define one scalable test framework for the system-level testing of LBS. The scalable test framework should be extensible both on test components, i.e. the test plan generator and the context simulator, that implement specific test functions, and on the test system console that manipulates the test process and dynamic connections to various components, i.e. mobile subscribers, services provider, within the location-bases service, a typical distributed system.

To achieve this scalable test framework, the test framework should be designed with an *open distributed architecture* and it can dynamically connect to the components in the LBS system and manipulate the test process operations with the corresponding components in LBS via the *well-defined interface*. Instead of the web service interface, which is employed in [Morla+04], we define three types of standard interface (system interface, external interface and internal interface, see section 3.4 from the details of interfaces) via which the test framework can communication with the system under test (SUT), the third party components, which is used to provide test environment, and the internal test components, which is used to provide specific test functions. These interfaces can be implemented in Java or C language depending on the implementation of related system under test. The definition of standard interface distinguishes our test framework with the test framework in [Morla+04] and allows the hierarchical structure and more test functions being implemented in our test framework. In addition, it enables the ease of integration of third party components, i.e. context simulator to provide context information, in our test framework and makes it possible to apply the cost-based solution during the implementation of our test framework.

Another essential step of building a scalable test framework is to *modularize* test components that implement different test functions, i.e. creating the test plan, analyzing test results, for the test system. As our test framework is designed with an open distributed architecture, these test components can be connected to the test system console via standard communication protocols and interfaces. With these standard interfaces and test components, it is possible to reuse test components and test functions in different test scenarios and projects. Thus, modularizing test components helps to reduce initial test development costs and increase the reuse and scalability of our test framework, and leads to the cost-based solution when building our test framework. Moreover, our test framework separates the test definition from the test implementation, so that testers can concentrate on writing test cases and test functions in scripts language, and do not need to consider how to implement test cases in certain programming language. By doing this, the test framework can significantly improve the efficiency of LBS testing.

Compared with our scalable test framework, other test approaches, i.e. UBIWISE, [Satoh 03], and [Morla+04], both provide their unique test frameworks or simulation environment on context-aware application testing, which can provide the application-level testing to some extent. However, from the testing viewpoint, none of their testing approaches is scalable and defined in a standard process. They cannot provide a clear test definition and test descriptions that are essential in the test process. In other words, testers have to struggling on understanding their unique approaches and creating test cases and functions for each project, and have to rework on creating test cases and functions whenever there is any change in the system under test (SUT). Consequently, it is inefficient to apply their approaches on the testing of location-based services, which is fast changing both on technologies and on methodologies.

Building a simulated wireless environment in our test framework is another contribution in this thesis. Instead of NS2 wireless network simulator [Fall+05], which is used by [Morla+04], we employ SWANS [Barr 05] to provide simulated wireless network to our test framework. SWANS is an open source wireless network simulator built on the Java Virtual Machine (JVM), and it supports running standard Java network applications over simulated networks. However, as it is mainly used to simulate the ad-hoc wireless network, SWANS does not contain the signal propagation model suitable for simulating the signal propagation mechanism within the location-based service. Therefore, we develop our signal propagation model using Hata Model [Hata 80] and embed it into the SWANS wireless network simulator. Therefore, it can propagate the signal and enables our test framework to test corresponding scenarios for the LBS system, such as multiple services selection mechanism.

In addition, we implement our test framework with the TTCN-3 test specification and implementation language, which enable testers to rapid prototype their test environment

[Grabowski+03]. In addition, we develop corresponding standard interfaces to connect the TTCN3 test system with components in SUT and third party components, such as Context Simulator and the SWANS wireless simulator.

Finally we demonstrate the feasibility of our test framework with experimental studies, in which 45 test cases are developed to perform LBS testing from various testing strategies, including functional testing, usability testing, network performance testing, server side testing and security testing. Interoperability testing strategy is also investigated at the end of experimental studies.

1.4 Organization of the Thesis

The remainder of this thesis is organized as follows:
Chapter 2: Describes background information about the location-based service, and challenges on testing the location-based service.

Chapter 3: Specifies the test scope of the system under test and test framework requirements, and presents the prototype of the test framework.

Chapter 4: Presents the architecture of our test framework, implements the test framework with the TTCN-3 test system and the SWANS wireless simulator, and evaluates the test framework with typical testing criteria.

Chapter 5: Applies our test framework to some experimental settings, and evaluates our techniques.

Chapter 6: Gives conclusion and suggestions for future work.

Chapter 2 Location-based Service

2.1 Introduction

Location-based services (LBS) can be defined as mobility services that exploit the derived locations of a user (specified by user, network or handset) to provide services that have a geographic context [Mitchell+03]. LBS differentiates itself from other software applications with the following unique elements:

- **Convenience** since they are available through mobile devices (i.e. Smartphones and PDAs) which are expected to provide users with the information about their immediate situation.
- **Personalized** by their very nature in encouraging users to better define their mobile activity, such as check the traffic situation on the route and notify drivers of the nearest gas station.
- **Real Time** in their transaction to link dynamic information related to the user's location.
- **Relevant** through filtering only that information proximal to a user's location and simplifying an already complex user experience.

In this chapter, we will describe the core components of the location-based service, and location positioning technology, and present the challenges of testing location-based service at the end of the chapter.

2.2 Location-based Service

As one type of mobile commerce applications, Location based service (LBS) involves many stakeholders and complicated business relationships, such as consumers, service providers (to brand and run the service), network operators (provider of mobile network connectivity), mobile phone vendors (provider of mobile terminals), GPS technology vendors, network positioning vendors, map providers and developers (vendors and produces of client-server application). LBS applications include emergency and safety-related services, entertainments, navigation, directory and city guides, traffic updates, and location-specific advertising and promotion, etc.

Figure 2-1 shows an overall architecture of a typical Location-based service (LBS), which is conceptually a Client/Server distributed system, consisting of following three tiers:

- Mobile Client tier
    - Location-based applications (LBA)
- Middleware tier
    - Location measure technology
    - Location management platform
- Service Server tier
    - Application/Geo-Spatial platform
    - Application/Services.

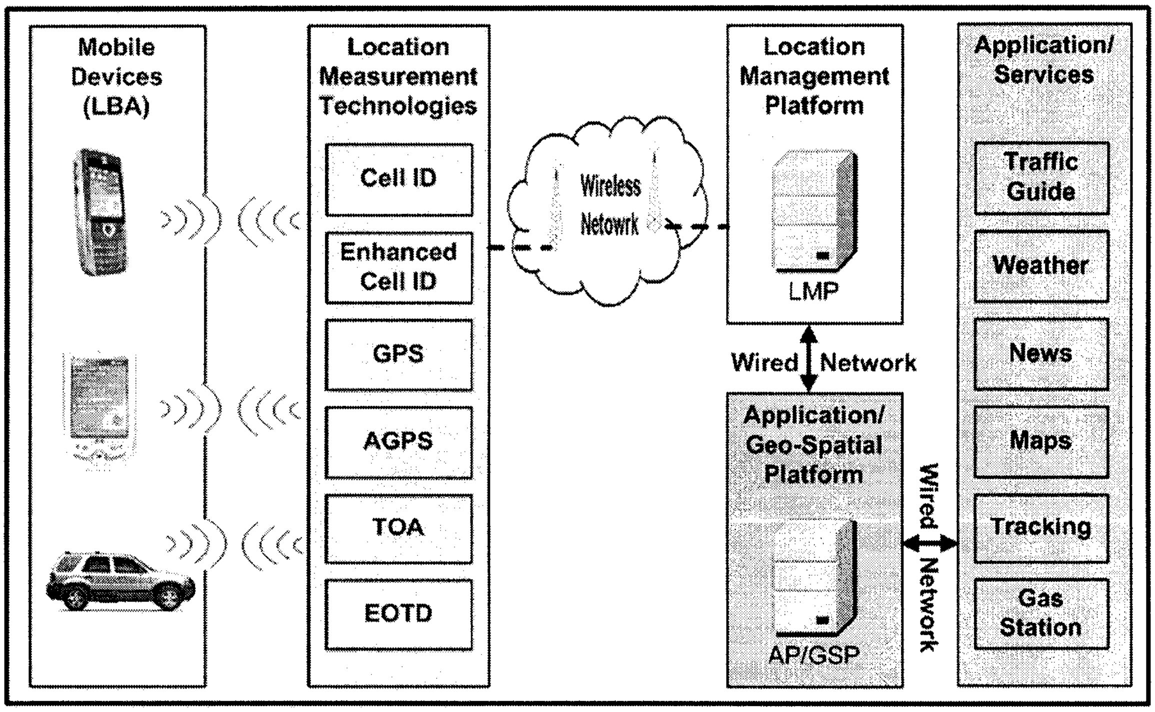


Figure 2-1: Core Components of the Location Based Service [Mitchell+03]

2.2.1 Location-based Application (LBA)

The location-based application (LBA) on the mobile client tier is the application running on the mobile station (device), responsible for sending the user's service request and receiving the response (location-based service) from the service provider (or Web Sever). Although a wide range of mobile stations are available in the market, the operating systems, the core component of mobile stations, are dominated by just three major brands, Palm OS, Pocket PC, and Symbian OS [Lee+03].

Currently there have three prevalent technologies for developing applications on mobile handset:

- SMS (Short Message Service)
- Browser-based programming (WAP and NTT DoCoMo i-Mode)
- Java 2 Platform, Micro Edition (J2ME)

Short Message Service (SMS) is a globally accepted wireless service that enables the transmission of alphanumeric messages between mobile subscribers and external systems. Using SMS to deliver location-based services is one of approaches to build location-base services. In this approach, the subscriber sends the service request in the form of a SMS message to the SMS center, which will delivers the message to the application platform. The application platform consists of an SME (Short Message Entity) emulator and an application server for the application. The SME emulator is used to parse the incoming message and send it to the application server. The application server will generate location information and send it to the location provider. Then, the Application server will request the service content by passing the current location to the content provider along with the request text. The context provider processes the request in the context with the provided location and sends the result to the application server. The SME emulator in the application platform formats the result in the form of a DMD message and sends it to the mobile device [Krishnamurthy 02].

SMS allows users to send and receive text messages to and from cell phones, containing up to 160 characters in each SMS message. The application uses SIM toolkit technology and is ideal for one-to-one or one-to-few information pushing. However, SMS is not ideal for implementing a location-based application, since it can only support simple applications. The application can only be equipped with very limited logic, mainly text processing, and does not cater to complex m-commerce solutions, such as map and tracking services. In addition, the current SMS messaging technology does not guarantee that the packet will arrive at a certain time [GSM Association 05].

Wireless Application Protocol (WAP) is the open standard for information services on wireless terminals like mobile phones. WAP is designed for micro browsers, a small piece of software that makes minimal demands on hardware, memory and CPU. WAP uses the Wireless Markup Language (WML) to create pages to be displayed in a micro browser. WAP enables easy access and delivery of information and services to mobile devices. Inherited from Internet standards (HTML, XML and TCP/IP), WAP is intended to address the need to access the Internet from handheld devices such as mobile phones and PDAs by optimizing these Internet standards and complying them with the unique constraints of wireless environment (i.e. low bandwidth, high latency and unstable connection). It defines a set of standard components (a WML language specification, a WMLScript specification, and a Wireless Telephony Application Interface (WTAI specification) that enable communication between mobile terminals and network servers.

Figure 2-2 shows the WAP programming model, which is very similar to the WWW programming model [WAPForum 05]. The mobile device set with the WAP browser communicates with the web server via a WAP Gateway, which includes HTML to WML filters, the HTTPs interface to a web server as well as interface to the mobile device via the wireless network (i.e. GSM, CDMA). Requests from the mobile device are sent as a URL through the operator's network to the WAP Gateway. Responses are sent from the web server to the WAP Gateway in HTML, which are then translated in WML and sent to the mobile terminals.

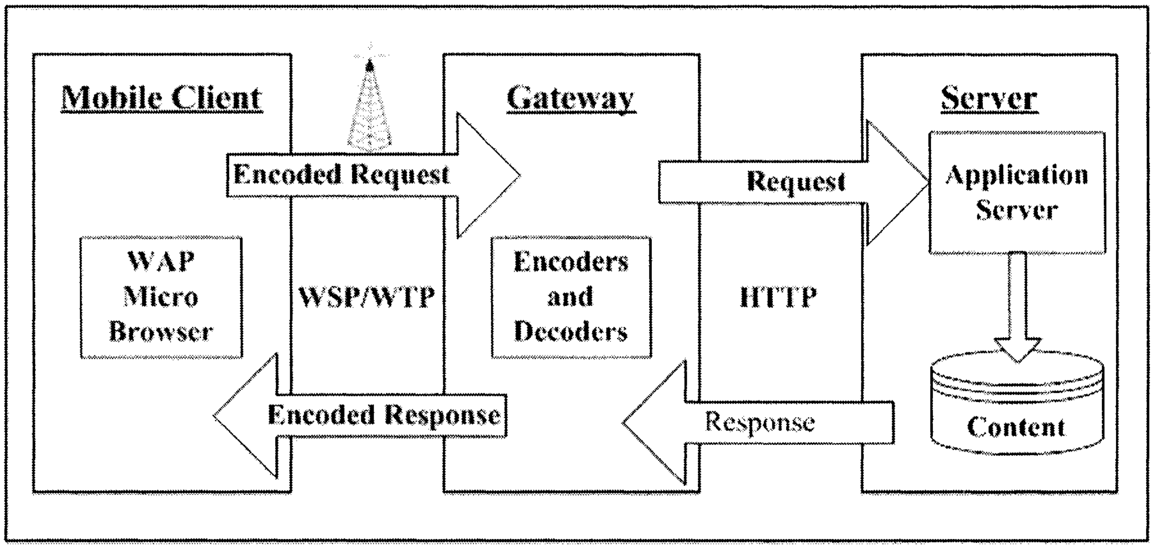


Figure 2-2: WAP Programming Model

Since founded in 1997 by Ericsson, Motorola, Nokia, and Unwired Planet, WAP has received wide support in the wireless industry [W3C Schools 05]. However, as WAP logic executes on the server-side (with the exception of some simple scripting that can run in the device), WAP is not quite suitable for graphics intensive applications such as games, interactive charting, etc. WAP also has known security issues in the WAP gateway, where protocol conversion occurs.

NTT DoCoMo i-Mode. i-Mode was first introduced by NTT DoCoMo in Japan. This technology competes with WAP [Eurotechnology.com 05]. It offers a similar mechanism to allow users to access the Internet from their wireless devices over a packet-switched network. However, i-Mode uses cHTML, a subset of HTML, as a page description language to develop content in the cell phone browsers, while WAP uses WML. i-Mode is a complete wireless internet service at present covering almost all of Japan with 42 million i-Mode subscribers in Japan and about 4 million users outside Japan (as of Summer 2004).

For the time being, the development platform of location-based applications is dominated by the Sun's **Java 2 Platform, Micro Edition (J2ME)** platform. J2ME is designed to run on consumer devices and electronic appliances, including wireless devices such as cell phones and Palm PDAs [Raju 00]. J2ME offers a way to exploit the processing power on the mobile device better by running the code on the device itself. Therefore, it provides a better network implementation, a better graphical user interface, and local database management. Figure 2-3 shows an overall map of Java Technology.

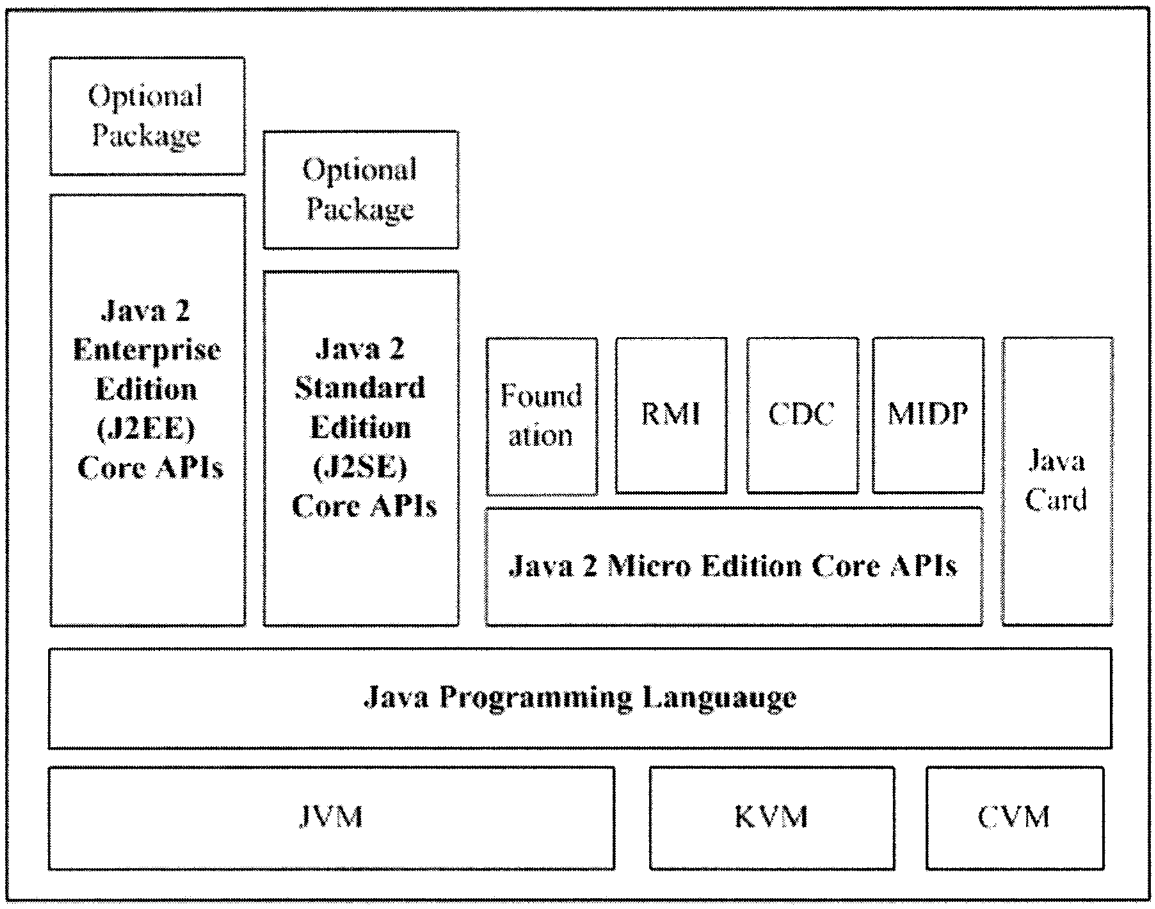

Optional Package
Optional Package
Java 2 Enterprise Edition (J2EE) Core APIs
Java 2 Standard Edition (J2SE) Core APIs
Found ation
RMI
CDC
MIDP
Java Card
Java 2 Micro Edition Core APIs
Java Programming Languauge
JVM
KVM
CVM

Figure 2-3: Java Technology Map

J2ME is divided into *configurations*, *profiles*, and *optional packages* [Sun Microsystems 05]. Configurations are specifications that detail a virtual machine and a base set of APIs that can be used with a certain class of device. The virtual machine is either a full Java Virtual Machine (JVM) or some subset of the full JVM, such as Kilo Virtual Machine (KVM), which is designed to be small, with a static memory footprint of 40-80 kilobytes. The set of APIs is customarily a subset of the J2SE APIs. Current J2ME standard configuration is the Connected Limited Device Configuration (CLDC).

A profile builds on a configuration but adds more specific APIs to make a complete environment for building applications. While a configuration describes a JVM and a basic set of APIs, it does not by itself specify enough detail to enable you to build complete applications. Profiles usually include APIs for application life cycle, user interface, and persistent storage. The Mobile Information Device Profile 2.0 (MIDP 2.0) is the latest version of the J2ME profile specification, including new features such as an enhanced user interface, multimedia and game functionality, greater connectivity, over-the-air (OTA) provisioning, and end-to-end security.

An optional package provides functionality that may not be associated with a specific configuration or profile. One example of an optional package is the **Location API (JSR 179)**, which provides a standardized APIs that enable developers to write wireless location-based applications and services for resource-limited devices like mobile phones, and can be implemented with any common location method. The compact and generic J2ME location APIs provide mobile applications with information about the mobile device's present physical location and orientation, and support the creation and use of databases of known landmarks, stored in the device (See Appendix A for all J2ME configurations, Profiles and APIs).

The J2ME CLDC and KVM have been embedded into many different platforms and devices, including Motorola phones and two-way pagers, Research in Motion (RIM) wireless handsets, and Palm PDAs. Compared with WAP and i-Mode technologies, J2ME has the following benefits:

- *Local computing power*: J2ME is able to access the device's resources (i.e. device's processor and memory) more by running application locally while Browser-based applications is often good at building contents in the web pages with minimal configuration.
- *Multithreading*: J2ME allows an application to use multithreading, thus provides a mean for application developers to create a more complex logic.

- *Connectivity with distributed system*: J2ME allows the application to talk to back-end systems in XML, a language that is suitable for cross-platform and distributed system application.
- *Lower network usage and server load*: The MIDP client's interface is resident within the mobile device, so it won't consume the network and server resources when it is run offline. However, in a WAP-based solution, the server is responsible for generating display pages, which results in a roundtrip load every time the interface changes.
- *Offline operation*: J2ME provides a mechanism that allows the application to be run on or off the network, while a browser-based application can achieve this in a very limited way with local browser page caching.
- *Security*: J2ME provides an enhanced security model and end-to-end connection from the client device to web servers and does not need gateways as in WAP.
- *User personalization*: With the ability of manipulation of mobile devices, J2ME can take better advantage of local operation system than Browser-based applications does on developing complicated user interfaces.

However, J2ME and WAP are complementary technologies. Both the WAP technology and i-Mode technology are comparable to HTML and Web browsers on desktop computers, while J2ME is comparable to desktop Java applications.

2.2.2 Location Measurement Technology

Today there is a variety of so called 'positioning' technologies that are involved with the calculation of position in a space or grid, based on some mathematical model. Generally, a mobile device is allowed to be aware of its location with different degrees of precision and accuracy, based on the different capabilities from the device, such as power consumption, cost, autonomy, speed, networking, processing and overall system complexity. Currently, there have four main categories of technologies for positioning the location of the mobile device and user:

- User-defined
- Network-defined
- Network-oriented
- Handset-defined

The **User-defined** technology is widely used in the early location-based services. In this kind of service, the system requires users to define their own position, such as the address, phone number, street, city or postcode, etc. and services required will be delivered to the WAP browser of the mobile device. Therefore it doesn't need position information of the handset being locatable by the wireless network operator. In Tokyo, for instance, J-Phone's J-Navi service lets users enter a phone number, address or landmark, and then searches the area within a 500-metre radius. This makes it possible to find the subway station nearest to a particular shop, or a particular kind of restaurant within walking distance of a particular office building. The service user can also download a full-colour map of the area, which points out the route to the user's destination.

The **Network-defined** method is based on the thousands of base stations in the mobile phone networks. The method is to use Cell ID information plus some other radio signal measurements. These include plain cell ID, cell ID combined with base station TA (Timing Advance) information or cell ID with RTT (Round Trip Time). These are techniques based on measurements of signal delays, round trip delays or signal jitter combined with Cell ID information. As Cell ID has already been used in the GSM radio network, it can be deployed very cheaply on modern handsets.

However, the positioning accuracy using Cell ID technology depends on the cell size or the number of base stations in a given area. In urban areas, it can provide higher accuracy services as urban areas generally have a higher density of base stations. In short, accuracy can range from 10m (where micro cells are installed, such as in an airport) up to 2km. In a typical urban area, the achievable accuracy is often sufficient for proximity services, such as finding the nearest restaurant or gas station.

The **Network-oriented** method uses some form of triangulation techniques, such as the ones found in TDOA (Time Difference of Arrival), Mobile Assisted TOA (Time of Arrival), AOA (Angle of Arrival), which use information from the base stations. These approaches are popular for deployment with 2G and 2.5G operators since most of the times involve an evolutionary upgrade to their network.

The Time Difference of Arrival (TDOA) technique is based on estimating the difference in the arrival times of the signal from the source at multiple receivers. This is usually accomplished by taking a snapshot of the signal at a synchronized time period at multiple receivers. The cross-correlation of the two versions of the signal at pairs of base stations is done and the peak of the cross-correlation output gives the time difference for the signal arrival at those two base stations. A particular value of the time difference estimate defines a hyperbola between the two receivers on which the mobile may exist, assuming that the source and the receivers are coplanar. If this procedure is done again with another receiver in combination with any of the previously used receivers, another hyperbola is defined and the intersection of the two hyperbolas results in the position location estimate of the source. This method is also sometimes called a hyperbolic position location method. One such system called "Cursor", which is based on the GSM technology, is already available [Clarke 97].

The Time of Arrival (TOA) Technique involves listening to the handset access burst across three or more mobile base stations. The base station can indirectly determine the time that the signal takes from the source to the receiver on the forward or the reverse link by measuring the time in which the mobile responds to an inquiry or an instruction transmitted to the mobile from the base station. After the measurement of different time of arrival data at each base station, the resultant information is then compared with the time reading from a GPS absolute time clock, and the coordinate of the mobile device can be calculated based on the above compared results. This method can increase positioning accuracy by up to 50%. It, however, requires the modification of the handset, which makes it costly to implement.

The Angle of Arrival (AOA) method utilizes multi-array antennas and tries to estimate the direction of arrival of the signal of interest, and then it calculates the angles at which a signal arrives at two base stations. Once the angles are obtained, a simple triangulation calculation is performed to determine the coordinate solution. Similar with the TOA method, this method can also increase positioning accuracy, but needs the modification of handset.

The most straightforward **Handset-defined** positioning method is to use the Global Positioning System (GPS) technique, which has been widely used in the vehicle navigation system and dedicated handset devices for years. With at least three satellites and the GPS receiver involved, GPS is able to position the mobile device to as accurate as a resolution of 1m. However, the accuracy of GPS positioning would be affected greatly by the surrounding environment because GPS signals are very weak in the building, bridges, tunnels, etc. Consequently, it will reduce the suitability of GPS positioning in urban area. In addition, the severe weather conditions may have side impacts to the accuracy of GPS positioning. To overcome these impacts and increase the accuracy of GPS positioning, some enhanced GPS solutions have been developed to combine GPS signal information with cellular handset position information.

The Network-assisted GPS (A-GPS) [Djuknic+01] is one of such enhanced GPS solutions, which involves installing fixed GPS receivers every 200km to 400km to fetch data to complement the readings by the mobile handset. With assistance from these fixed receivers, the mobile handset can make timing measurements without having to decode the GPS information, greatly improving the calculation time. Measurement results will be sent to a location information management framework, where the position of the handset is calculated from measurement results using differential positioning. A-GPS can potentially provide high positioning accuracy. However, it requires the modification of the network as well as the handset, making it costly to implement.

Another solution of Handset-defined methods is Estimated Observed Time Difference (EOTD), which relies only on software in the mobile handset to listen to bursts from multiple base stations. The time difference in the arrival of these bursts is used to triangulate where the mobile handset is located. This technique needs to know the exact location of the base stations as well as the sending of data from each base station to be synchronized. Although potentially not as accurate as AGPS, EOTD does not rely on a clear view of the sky. With the same characteristics to A-GPS, E-OTD requires the modification of both the network and handset, which makes it expensive to implement.

We can see from table 2-1 that AGPS is currently most accurate positioning technologies, but it could not be used for the indoor positioning. Cell ID, on the other hand, is the cheapest positioning method although its positioning accuracy is not as ideal as others.

Techno logy	Typical accuracy	Device requirement	Network	Indoor positioning
Cell ID	Depends of the size of cell 3000m – 100m	None	All	If cell phone coverage
AGPS/ GPS	30m – 5m	Integrated GPS receiver	All	May not always work as it requires free line-of-sight GPS satellites
TOA	300m – 50m	Yes	All	If cell phone coverage
AOA	200m – 100m	None	All	If cell phone coverage
TDOA	200m – 100m	None	All	If cell phone coverage
EOTD	200m – 50m	EOTD software	GSM	If cell phone coverage
Note: “All” means it can adapt to GSM, CDMA as well as TDMA networks				

Table 2-1: Basic Data – Selected Positioning Technologies

Outside the remit of 2G, 2.5G and 3G cellular networks, exist other families of positioning technologies that are often referred to as ‘local positioning’, which usually make use of **short range** networks such as 802.11, Bluetooth, RFID, Ultrasound, UWB or IrDA. Furthermore, recently there have been attempts to use position technologies based on TV radio signals.

2.2.3 LBS Operation Scenario

Despite of diverse positioning methods employed, the raw data of the mobile device's location information will be firstly retrieved by the Location Management Platform (LMP), which is normally operated by the wireless network operator, such as Sprint, Nextel, etc. Then, location information may be used by a service provider to personalize the service, or to improve the user interface by reducing the need to interact with a small device while on the move [Basso 02]. Here the third party service component, such as a GIS system in the Geo-Spatial Platform (GSP), may be employed according to the required service by the user. Finally, the service provider deploys the user's location information as well as required service to the mobile device from the Application Platform (AP).

Generally, we can define two main categories from various location-based services (LBS): User-requested and Triggered.

In a **User-requested** scenario, the mobile user is responsible for sending request to retrieve the position and then use it on subsequent requests for location-dependent information. This type of service usually involves either personal location (i.e. navigation system of finding where you are) or services location (i.e. direction service of finding where is the nearest petrol station). For instance, figure 2-4 illustrates a case of simple user-requested LBS application with the following operation procedures:

1. The user accesses the service with a login account from the WAP browser and sends requests (i.e. find the nearest petrol station) via the WAP Gateway to the Application Platform (AP). When the mobile client application is developed in

the J2ME MIDP structure, the mobile device could talk to the AP without the WAP Gateway.

2. A service provider sends a user coordinate query to the Location Management Platform (LMP), and LMP will then verify the authorization of the service provider to ensue only authorized services have rights to access user's coordinate information.

3. After successful verification of the service provider, LMP queries the wireless network elements for user's raw position data. This may involve different location measurement technologies, such as Cell ID, AGPS, EOTD, etc.

4. LMP will calculate the raw position data and send the coordinate result (X, Y) to the relevant Application Platform (AP) of the service provider.

5. According to the user request, the Application Platform (AP) may send requests to employ a Geo-Spatial Platform (GSP), which will convert the coordinates (X, Y) into detailed location information (i.e. street address or point on a map).

6. The Geo-Spatial Platform (GSP) provides Application Platform (AP) with list of address results ranked closest by travel time.

7. Application Platform (AP) deploys the service results (address list) to the mobile user via WAP Gateway.

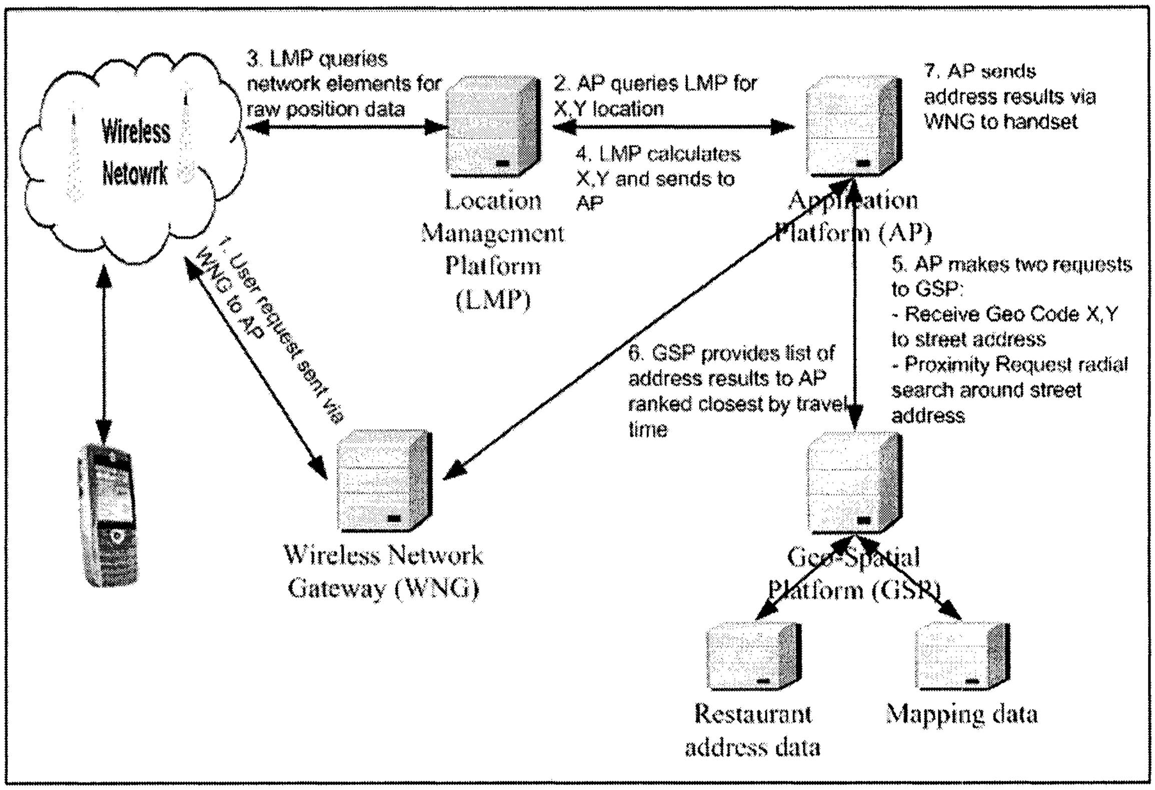

3. LMP queries network elements for raw position data
2. AP queries LMP for X,Y location
4. LMP calculates X,Y and sends to AP
7. AP sends address results via WNG to handset
Wireless Netowrk
Location Management Platform (LMP)
Application Platform (AP)
5. AP makes two requests to GSP:
- Receive Geo Code X,Y to street address
- Proximity Request radial search around street address
1. User request sent via WNG to AP
6. GSP provides list of address results to AP ranked closest by travel time
Wireless Network Gateway (WNG)
Geo-Spatial Platform (GSP)
Restaurant address data
Mapping data

Figure 2-4: LBS Operation Scenario [Mitchell+03]

Triggered LBS by contrast relies on a condition set up in advance that, once fulfilled, retrieves the position of a given device automatically. The Triggered LBS has similar scenarios with the User-request LBS except that the user position is automatically tracked by the Application Platform (AP) without user interference. One example is with regard to emergency services, where the call to the emergency centre triggers an automatic location request from the mobile network. Another example can be found in Xora's GPS Time Tracking system [Xora 05] where mobile workers are tracked by the GPS-enable phones they carry during the day. Location information is captured from a GPS satellite. By having employees' location and job shift information, the system can track hours worked and enforce overtime policies and radically improve timesheet accuracy. A geographic boundary, known as **Geofence**, is also introduced to monitor job sites, routes, and safety zones by triggering alerts when the worker being tracked crosses the Geofence perimeter. Employers and managers can view the locations and activities of their field force – all through a web browser. They can also download detailed time and job reports into a spreadsheet.

2.3 Challenge of Location-based Service Testing

Clearly, Location-based services bring enormous business opportunities for the each player in LBS, together with various technologies involved, such as wireless network, positioning technologies (i.e. GPS, EOTD), WAP, J2ME, Database, and GIS etc. Regardless of various positioning methods employed, the raw data returned is usually a co-ordinate, perhaps with an error parameter, for example '(52 03.50N, 001 16.89E) ± 5 m'. While this is by far the most accurate representation, to most users the raw data is of little use and only becomes valuable when interpreted in different ways. In other words, identifying the X, Y coordinates for the location of a mobile user is only the first step. The interaction of a number of core systems is required to create further real value to the customer. Therefore, rather than the accuracy of different positioning technologies, application concepts and market requirements are the biggest issue to test for the LBS testing. To improve the degree of LBS to meet the market expectations, the following primary issues need to be considered from the testing viewpoint:

- Functionality
- Usability
- Performance
- Scalability
- Security and Privacy
- Interoperability

2.3.1 Functionality

Like any other software, the location-based service (LBS) must be tested to ensure correct functionality under all working conditions. Testing is even more critical in the LBS applications as they face more sophisticated working conditions than most software applications, and various software components and technologies are involved in their system, making them more complicated than many other software applications.

A primary difference between traditional e-commerce scenarios and their mobile counterparts is the role of the infrastructure. In m-commerce applications, the mobile device may suffer from fundamental restrictions, such as limited form-factor, less powerful processor with limited computational resources, small screens, less memory and high-latency connectivity, etc. Due to these limitations, the application frameworks are fundamentally different. For instance, the location-based application (LBA) in the client tier can be developed with different technologies, including SMS, WAP, NTT DoCoMo i-Mode and J2ME, as mentioned earlier. The LBS applications developed with such technologies have their own architecture framework, which causes different considerations in building the test strategies.

Taking the WAP-based LBS application as an example, its client tier application is basically a WAP micro browser running in the mobile device. The WAP micro browser allows to display page content written in WML, and access a web server via WAP Gateway, which can translate the HTML sent from the web server to the WML, and send request from the mobile client to web server in HTML. Therefore, how to validate the WML and WAP protocol becomes one of most important issues in the WAP-based LBS testing strategy.

The J2ME-based LBS application, on the other hand, employs a J2ME MIDlet [Sun Microsystems 05] application in the client device, which enables it to hook up Java Servlets [Sun Microsystems 05] on the web server. So the MIDlet application connects to the web server via HTTP directly without any intermediate machine like a WAP gateway. Apart from the primary consideration for testing traditional Client/Server Java applications (i.e. HTTP socket connection), it is necessary to take the MIDlet application testing over the air, a "real" wireless environment.

Aside from the LBA, applications on other entities of the LBS architecture, such as the Location Management Platform (LMP), Application Platform (AP) and Geo-Spatial Platform (GSP), if required by LBS, should be tested to validate their functionalities. Some of these software applications may be commercial off-the-shelf (COTS) products from third party companies (i.e. Geo-Spatial Platform, Location Management Platform). Therefore, how to test the functions of COTS software components will be one of biggest challenges for LBS testing.

2.3.2 Usability

Usability and performance are the foremost reasons for the lack of success of LBS in its current format. The client location-based application usually residents on the mobile device typically with a small screen and keyboard. Location-based applications have to limit the user interface to the extent that is adapted to most of the mobile devices. This will definitely affect the readability and usability of most applications.

Even if they have enough functionality, most location-based services are too slow and too cumbersome to use. This is caused by both poor UI design and poor browser performance in most mobile devices. So it is not enough just to make a successful service demand and a willingness to pay for the service. The service must also be delivered in a way that meets the end users' expectations.

As an example, consider a People Tracking service where people can find the location of their friends and families. Obviously, most people don't like to allow their friends to track their locations all the time. Therefore all People Tracking services should include the option to turn off the being-positioned function. Default is that positioning is not allowed, which means that the being-positioned function is disabled unless the user switches it on. Theoretically this scenario is pretty simple. In practice, however, it requires 4-5 clicks of buttons and more than one-minute response time to turn this functionality on or off, which is enough to make most users reluctant to use it, and hence loose interest and leave it in the disabled model.

Therefore, ease of use, personalization and focus on the needs of the consumer are crucial elements in the success of location aware applications. We need to test the GUI navigation of the entire system and focus on the external interface and the relationships among the user interfaces. How to organize the consumer's operation in a transaction? Is the navigation of the user interface appropriate for each particular function? How flexible is the application? Is it easy to upgrade the software and hardware? Unfortunately, many applications are abandoned by the consumers because of poor usability and flexibility.

2.3.3 Performance

Performance is one of the critical elements for the success of location-based services. Performance issues of location-based services can be affected by a variety of sources, including wireless network, positioning system, client location-based application, web server as well as databases.

Low bandwidth rate is the typical performance problem for the wireless network. Currently most of location-based services are offered through GSM, CDMA or a GPRS wireless network, or the third generation (3G) networks, such as WCDMA, CDMA2000 and TD-SCDMA. However, the growing demand on bandwidth is still one of biggest challenges in wireless applications as the service provider extends more attractive features in order to maximize the number of its consumers. How does the location-based service respond to unexpected slow transaction speeds because of slow infrastructure? Such questions should be considered in the performance testing strategy.

Reliability is one of the issues that should be considered for the performance of the positioning system, such as GPS, Cell ID. Although GPS systems can accurately locate the object, its accuracy is still suffered by some factors, like the weather conditions and surrounding environment. The positioning accuracy of the GPS system in clear weather is much better than that in cloudy weather, and the GPS system can position the object more accurate in rural areas than in skyscrapers. For the positioning system with Cell ID technology, it also contains reliability concerns. For instance, the mobile device could be temporarily disconnected from the wireless network when it passes through a tunnel. How does client application response when it cannot be positioned by the positioning system? Do applications save the transaction data into the device's cache when its connection to the network is temporarily unavailable? Furthermore, for LBS services that provide high reliable services, it should provide consistent, high-performance system output and take advantage of fault management and recovery mechanisms to protect applications from client, server system, or network failures.

Client application bottlenecks also drive the decrease of location based service performance. The performance of location-based applications running on mobile devices is very important for the entire system since the customer may not have enough patience to wait for any extended execution time. One of the reasons for the client application bottlenecks is the limitation of hardware and software in the mobile device (i.e. Low powerful CPU, Less memory). This is especially critical for J2ME-based LBS applications as they provide customized User Interface and other computational programs on the mobile devices and demand more resources from mobile devices. WAP client application may also be one of the potential bottlenecks for WAP-based LBS as the WAP browser is the only client application (User Interface) on the mobile client and all other applications are resident in the web server, which may cause the high demand of network resources and long waiting time for the mobile client. Such situations should be considered in the design and testing of the location-based service.

2.3.4 Security and Privacy

Security and Privacy concerns are critical issues for the success of a LBS service. Users are very sensitive about their personal location information. To win the customer's trust, it is important to prevent the third party's knowing the physical location of users other than the service provider. Also, for most location based services (i.e. People Tracking Service), it is always necessary to switch on/off the positioning function by the mobile user to decide whether he/she should be positioned by others.

In addition, some security contributions can be found from the architecture design of LBS to prevent user's location from threads of unauthorized parties. For instance, the position information would be firstly retrieved by the Location Management Platform (LMP), which is normally operated by the wireless network operator. To protect the user's privacy from other parties, LMP can add security mechanisms to ensure that only authorized service providers are allowed to access the mobile user's position information.

Data transmitted over the wireless network (i.e. password, account number) could be another security risk, as it can be captured using digital RF scanning equipment [Srivatsa 02]. Unfortunately, most wireless protocols do not have built-in encryption mechanisms. Additional security measures, such as secure connections and cryptography, are definitely needed, especially for those services transmitting sensitive data.

Furthermore, the security risk can come from other entities in the LBS architecture, such as mobile devices, the Internet, and databases. The SIM cards found in mobile devices can be cloned and used in a different device, which may cause identity verification security problems for the application based on user authentication to the device (from the SIM) [Srivatsa 02]. Insecure Internet connections in the middle tier (i.e. Location Management Platform, Application Platform, Geo-spatial Platform) can also create serious security problems to the LBS service, such as *Denial of Service*, in which the intermediate entity can suppress messages meant for the other parties, and *More Points of Attack*, in which more complex message paths typically bring in more points of failure and more points of attack [Surech+01]. These security problems should be presented in the security testing scenarios.

2.3.5 Scalability

Location-based service (LBS) is implemented in a distributed application environment with mass-market scalability ability. With the increasing subscribers of the LBS service, the growing demands of network resources (i.e. bandwidth) and server machine resources (i.e. router, database server, websites) drive high performance requirements for the web server. It is important to gauge the performance and load capability of sever applications and evaluate hard performance issues about how much traffic a given site will be able to handle and make intelligent choices of hardware, software and often times even design approach of the application to make sure the application will be able to handle an onslaught of customers on the LBS applications.

To accommodate the growth of service subscribers, Load/Stress testing should be undertaken to evaluate the scalability and performance of LBS and ensure that the web applications sustain a high volume of simultaneous users and/or transactions, while maintaining adequate response times. Because it is comprehensive, load testing is the only way to accurately test the end-to-end performance of a web server application prior to going live.

2.3.6 Interoperability

Software interoperability describes the ability of locally managed and heterogeneous systems to exchange data and instructions in real time to provide services. Interoperable systems are generally *distributed* (i.e., at different places on the network), and interoperability also applies to different types of systems or similar systems from different vendors communicating while running on the same computer [McKee 05].

Like other M-commerce applications, many players and shareholders, such as mobile network operators, mobile manufacturers, content providers, service providers, middleware providers, are involved in the business of Location-based services (LBS).

Depending on the characteristics of the business model, each entity can be active in many of these roles, or in none of them. With these various entities, there should be mature standards defined for location-based services to achieve a high degree of integration capability for data services and consumers, and easy access to external systems.

What is needed for location-based services to become successful is a simple, consistent, managed, end-to-end solution, which requires a high degree of vertical integration to external data and systems, making sure that technology, business models and market requirements are well aligned. In other words, the service should employ an architecture that maintains interoperability and flexibility required for dynamic system elements.

Taking the Open Geospatial Consortium (OGC) as an example with high interoperability, OGC is a non-profit, international, voluntary consensus standards organization that is leading the development of standards for geospatial and location based services [OGC 05]. OpenGIS specifications support interoperable solutions to make spatial data accessible from multiple technologies and software vendors and making spatial data and spatial functionality available to other IT systems such as customer response management, logistics, location-based services for wireless devices, etc. The benefit is that users can thus access, combine, and disseminate geospatial information from distributed and varied information sources. Integration streamlines workflow and reduces costs of information production, maintenance and dissemination [McKee 05].

Thus, interoperability testing plays a key role to ensure the service maintaining high interoperability and flexibility with software components from different parties.

Chapter 3 Test Framework Requirement and Prototype

3.1 Introduction

From the previous chapter, it is evident that the complex architecture of location-based services causes numerous challenges for location-based service testing since so many entities and technologies are involved in the system. When considering the task of testing location-based service, it is necessary to provide a prototype of the test system, and a definition of the system under test.

As a result, specific test scope and test requirements need to be set up in order to build an effective test framework to test and evaluate the location-based service. In this chapter, we will describe the scope of system under test (Location-based Service) and test framework requirements that are necessary to deal with problems of testing LBS. The test framework will be prototyped at the end of this chapter.

3.2 Scope of System Under Test

In Chapter 2, we describe three methodologies on developing the location-based service application: SMS-based, WAP-based, and J2ME-based LBS. The SMS-based LBS sends the location service information in the form of test messages, imposing the following limitations:

- SMS messages contains 160-character in maximum, which will definitely limit the service function and cause inconveniency to the service subscriber.
- SMS message can only deliver test messages, not suitable for many graphics intensive applications like tourism service and traffic guide service that require map and graphic information to make them useful.
- SMS message does not guarantee that the packet will arrive at a certain time

Based on these limitations, SMS technology is not suitable for building graphics location-based service, and it could mostly be involved in LBS as a supplementary way to communicate between the mobile device and service provider. Thus, SMS-based LBS is not considered in our test framework.

WAP-based LBS has similar architecture with J2ME-based LBS on the web server and the application server, and WAP browsers use HTTP, the same network protocol that MIDP requires [Mahmoud 00]. However, WAP technology differs from J2ME application architecture by adding the WAP Gateway in the system, which acts as a bridge between the mobile client and the webs server, and offers the function of WAP services like encoding of WML pages, end-user authentication system, and WML script compiling. Since the wireless protocol employs WML for application contents instead of the Hypertext Markup Language coding (HTML), the WAP Gateway contains HTML to WML filters to translate the WML from the micro browser on the mobile device into HTML read by the web server, as mentioned earlier in Chapter 2.

Therefore, testing the client side application of WAP-based LBS is mainly to validate the correctness of WML sending to / received from the WAP Gateway. There exists some testing tools for the validation of WML and WAP testing, such as Testing Service for WAP [The Open Group 05], and Validate WML [W3C Schools 05]. *Testing Service for WAP* is a comprehensive certification and interoperability testing program that covers device testing, content verification, and a set of authoring guidelines to assist developers in providing interoperable WAP applications and services. *Validate WML* is a tools provided by W3Schools to validate WML with a WML parser. Thus, WAP client side WML validation is not considered in our test framework at the current step. However, WAP server side testing is similar with J2ME-based server side testing and could be tested in our test framework.

J2ME is the latest technology introduced in the design of location-based service (LBS). Figure 3-1 shows the high level logical architecture of a Java wireless enterprise application implemented with a J2ME device and a J2EE application server. From this figure, we can see that the J2ME client application consists of a User Interface, a MIDlet application, RMS (Record Management System) and Network communication entity. The MIDlet Client application provides a user interface on the mobile device and communicates with the Java Servlet usually via HTTP, and over a secure channel (HTTPS) when necessary. The servlet interprets requests from the MIDlet and then dispatches client requests to EJB components [Adatia 01]. When the requests are fulfilled, the servlet generates a response for the MIDlet application.

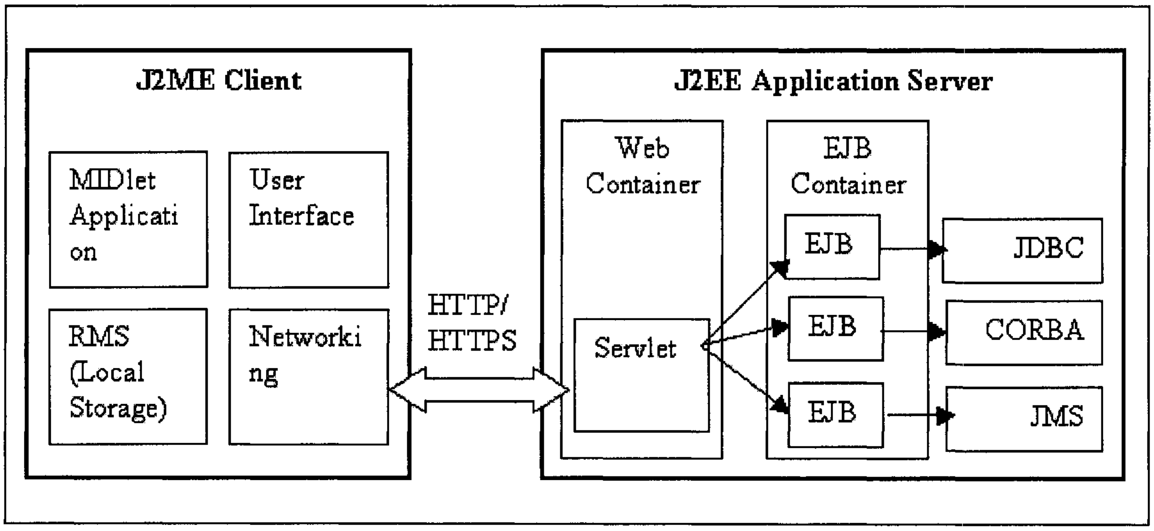


Figure 3-1: High-level Architecture of a Java Wireless Enterprise Application

Currently, the primary method to develop and test J2ME MIDlet applications is to utilize the handheld emulator provided by the handheld vendor, such as Sun's J2ME Wireless Toolkit, Motorola's iDEN SDK, etc. Once the application has been tested on an emulator, it can be moved on a real device, and be tested in a live network.

These handheld emulators support KVM runtime environment for the J2ME MIDlet application, and provides the tester with great advantages and convenience on testing functionality and usability for the J2ME MIDlet application. However, as stated earlier, they focus only on application-level simulation and are not suitable for system-level testing (i.e. scalability testing), which is one of challenging issues on LBS testing.

To solve this problem, our test framework focuses on the system-level testing of location-based service and, in turn, limits the test of J2ME client application only on its **networking function component**, which interacts with the Application Server under the J2SE runtime environment. By applying this strategy in our test framework, we can minimize the impact caused by the changes in the applications we are testing, and changes in the tools we use to test them.

Another important reason that makes us switch the platform is based on the consideration of "future" location-based service. It is well known that J2ME is designed specifically for mobile devices. However, existing Java implementations for such information appliances, using Sun's J2ME platform, is inconvenient for developers because of the restricted set of capabilities they offer in order to cater to the hardware limitations of many of these smaller devices. As the exploration of networked information devices, there was a natural evolution for developers to directly access the J2SE platform at the advanced end of such devices.

Some mobile operation systems have been developed to support J2SE directly on the mobile devices. For instance, SavaJe OS [SavaJe.com 05] is an open standard, Java™ platform for mobile phones, which brings J2SE functionality and security to the mobile space. Therefore, the developer can take full advantage of advanced Java technologies such as the Java Foundation Classes/Swing (J.F.C./Swing) components, the Abstract Window Toolkit (AWT), Java 2D extension, CORBA, Java Database Connectivity (JDBC) API, Jini network technology, Java Remote Method Invocation (RMI), and the full Java 2 Security Model.

Based on the above considerations, our goal is to create a test framework that can be adaptable and extended to accommodate future implementation and future version of the location-based service. As a result, we define the scope of system under test (location-based service) with following requirements:

- The System Under Test we target in our test framework builds with a generic architecture and includes necessary components, including the client application, running on the mobile device, and the server application. For brevity, we treat the application server and the web server as one entity and only use the term Server in the rest of thesis.

- The mobile device fully supports J2SE platform.
- The client application on the mobile device is developed under J2SE runtime environment, connecting the Java Servlet in the web server via HTTP/HTTPS protocols.
- The server application consists of necessary components for a generic LBS architecture, such as servlets and a database. Other components, like EJB, JMS, are not considered at the current stage due to our desire to construct an application-independent test framework that we try to build (See section 4.2.4 for more details).

3.3 Test Framework Requirements

As mentioned above, the application we test (Location-based service) has a complex architecture. Even after we minimize the scope of system under test, there are still many potential changes in the architecture because of different user requirements and technologies. As a result, when developing our test framework, we must minimize the impact caused by changes in the applications we are testing, and changes in the tools we use to test them.

Therefore, the following requirements need to be defined for the test framework to create an execution environment for the location-based service testing:

- Open distributed architecture
- Simulated application environment
- Test automation

3.3.1 Open Distributed Architecture

The test framework should be constructed in an open, distributed architecture that can be adapt to the evolution of location-based service (LBS) and allows third-party components to be easily integrated into the test framework with a standard communication protocol.

As we see from the Chapter 2, LBS involves various elements in the system, including mobile client applications, positioning technologies, services, location contexts, GIS contents, etc. These elements could all be independently developed and managed. As a result, the architecture of LBS may have a considerable potential for evolution in terms of the sources of information used, the types of information supported, or the underlying infrastructure.

Therefore, the test framework should be designed as an open distributed architecture, which can be better adapted to the evolution of LBS. Even though there is still a lack of established standards for most of entities in LBS, this limitation is merely intermediary; and in the long term, like other web-based applications, location-based service will adopt open standards for automatic information exchange. At that time, it will be more obvious of the advantages of an open approach in the test framework.

Another benefit of employing open distributed architecture for the test framework can be seen from easy integration of third-party components. In order to test functionality of the location-based service, test components need be developed to perform certain function of entities in LBS. Some of these components, such as a context simulator to provide the location information, and a wireless simulator to offer wireless network environment, have already be produced by researchers or industry companies. In order to reuse these external modules, the test framework should be designed in an open structure to make it flexible enough to integrate new third-party components into the system. By doing this, less code needs to be designed and implemented by the developers and the test framework has the potential to save both time and money in the test framework development process.

To achieve the goal of being an open distributed architecture, a standard interface to testing should be used by all external components and internal components that are necessary to the test framework. This leads to more consistent and uniform structure for the test framework. In addition, the test framework should employ a test management component to monitor the interactions between all the components and control the workflow in the test framework. The internal test components in the test framework, such as performance gauge to analyze the performance of the test framework, should be designed as independent modules with a standard interface, via which they can connect to test management component in the test framework.

3.3.2 Simulated Infrastructure

We must ensure that the test framework accommodates the conformance policy defined in the specification of location-based service testing. The application execution environment is one of most important issues in a software specification. Therefore, the test framework needs to support the system under test with a simulated application environment so that the system under test can be executed in this simulated application environment as if it is executed in the real environment. To achieve this, the following issues should be considered when building a simulated application environment:

- Wired and wireless network simulation
- No modification of application code
- Location contexts acquisition
- Service selection mechanism

3.3.2.1 Wired and Wireless Network Simulation

In Chapter 2, we mentioned that the success of LBS relies on success for application concepts and business logic, rather than the accuracy of different positioning technologies. The interactions of numerous components in the location-based service (LBS) are the biggest issue to be tested for a LBS. This includes the interaction of components within a wired network and wireless network environment. As a result, we should consider these two situations when building the test environment for LBS testing.

Specifically, the test framework should have the interface to connect the wireless simulator, which is responsible for providing a simulated wireless communication test environment whenever the application under test needs to access the wireless network. Moreover, this wireless simulator should be developed as a potable component, being managed independently so that it can be hooked up to the test framework according to the test requirements. In other words, the test framework provides options to either: test under a wireless network environment when the wireless simulator is hooked up; or test (functional and performance testing, etc.) for the components of LBS that run under a wired network, such as Servlet, Database, EJB components in the web server, after the wireless simulator is disconnected from the test framework.

The wireless simulator is required to support the transition of wireless data (i.e. TCP/UDP sockets) in a transparent manner with a predefined interface (API), and support various network communication protocols, such as TCP/IP, 802.11b, wireless LAN, GPRS, etc., for the location-based service. Moreover, for LBS applications in production, performance is affected by many external network infrastructure factors that are independent of application behaviour, such as unexpected communication traffic load in the wireless network, As a result, the test framework should be able to simulate these typical network infrastructure factors via the wireless simulator, i.e. mobility model, packet loss, delay and drop with predefined probability in the process of testing LBS applications.

One benefit of using a wireless simulator is that, the problem with testing in a live wireless network is affected by many factors and it is almost impossible to record the test results and repeated them later. So, testing the performance in an emulated wireless network becomes a critical method used both in the development of wireless applications, and in academic research.

3.3.2.2 No Modification of Application Code

In order to provide a simulated test environment that is similar to the real environment, the application code of system we test, or at least the main components of the system under test, should be kept unmodified when interacting with the test framework. To achieve this, the test framework should offer the standard interface and communication protocol via which the test data or requests can be sent from the test system to the system under test, and required responses can also be received from the SUT back to the test system.

3.3.2.3 Location Contexts Acquisition

No matter what kind of positioning technology is used, location contexts will normally be retrieved by the location provider from the wireless network or GPS satellites according to the request of authorized service provider, and then delivered to the mobile device via the wireless network. This context information is presented in various formats. For example, the GPS location context mainly consists of data of latitude, longitude, timestamp etc. In addition, the overlaps of location contexts, which imply that a mobile device can effectively be located in two location contexts at the same time, may exist in the scenario of the location-based service. However, for brevity, this situation is not considered in our test framework and we assume one mobile device can only apply with one location context at any time.

Therefore, it is necessary for the test framework to have the mechanism dealing with location contexts acquisition. This can be fulfilled either by the integration of third-party context simulators, i.e. Context Toolkit [Dey+00], or simply by accessing the Database that stores the context information. As a result, the test framework should support the system under test by providing an interface (i.e. JDBC API) to connect the context simulator, so that the location-based application can retrieve the location context from the context simulator as if it interacts with the service provider in the real environment.

3.3.2.4 Service Selection Mechanism

Service selection is one of main functions in the specification of a location-based service (LBS). It allows mobile applications to dynamically select services that are specifically associated with their particular location. In other words, mobile applications should be able to discover and select local services that are available in their physical location. Supporting service selection mechanisms makes it possible for the test framework to test service selection functions for the location-based service.

To support this mechanism, a particular service selection model should be established to discover and distinguish available services. An important issue when modeling the association between services and physical location is which geographical criteria to use when determining the "nearby" services that a client should select. Rui José [José+03] described two models of location-based service selection: distance-based and scope-based models. A distance-based model means that the mobile client can select servers providing services located within some distance from its own position, while a scope-based model means that each service is associated with a service scope that explicitly represents the usage context of that service as a region in physical space. While the distance-based model emphasizes the location of the server offering the service, the scope-based model focuses on the geographical area defined for the service usage.

In our test framework, we deal with the service selection mechanism by supporting both models during the service selection process as they usually happen simultaneously in a location-based service. Specifically, the mobile client application will discover the available service when it is physically in the scope of server of providing the service. The scope of the server is determined by the characteristics of base stations [Stüber 01], which are responsible for transmitting the service content to the mobile client, such as frequency, maximum allowable path loss, antenna height of the transmitter, and by the surrounding physical environment of mobile devices, such as urban area, open rural area, etc. Alternatively, if the mobile client falls in the space where multiple networks exist, it will discover and select service according to the distance between the mobile client and servers of providing services. In Chapter 4.4.3, we will describe this service selection model in detail and implement it in Java.

3.3.3 Test Automation

Test automation becomes an increasingly important and critical testing strategy as it can improve the reliability and quality of software products, reduce or eliminate the drudgery and repetition associated with manual testing, decrease the time required to test and make tests more consistent and repeatable. This is even more important for wireless applications like location-based services that are hard to reproduce the possible problems with manual tests.

Therefore, we need to consider one automation test framework for LBS testing and it should reusable and manageable. To achieve this, the following characteristics are essential to be considered when developing automation test framework:

- Application-independent
- Extensibility and Maintainability
- Hierarchical structure
- Ease of Use
- Platform-independent
- Manual Test
- Test Results Verification
- Test Results Reporting

3.3.3.1 Application-independent

As we have seen in Chapter 2, the location-based service consists of various independent components and technologies. Thus, our test framework should be focused on common components that make up the application under test, such as a client/server networking module, and a database module. Only then can we make the system under test application-independent and reusable through the whole automation testing process, taking away all application-specific contexts from the framework.

Another method to make the test framework application-independent is to separate test scripts from hard coded components of the system under test. In an automated test process, test scripts are written to drive the application under test, i.e. execute keyboard stroke, mouse actions and background processing, and verify application responses and behavior just as a human would. Generally, most of test scripts create function calls to the system under test by using application-specific, hard coded values, which will greatly reduce the degree of application independency. To deal with this problem, like many other commercial tools, we use variables, rather than hard coded values, to provide application specific data to our test framework. By doing this, the test framework can greatly improve effectiveness of being application-independent.

Take the data driven testing as an example, data driven testing is an automation test framework where the automated tests read the test data from an external source, such as a file/table, rather than having the values hard coded into the scripts. The test data should be written in parameters (Variables) to make them data driven. These parameters define the way in which the test functions, allowing the user to test a variety of scenarios with the same automated test scripts by just changing the data. As a result, data driven testing is application-independent and very efficient mechanism for automating testing.

3.3.3.2 Extensibility and Maintainability

Because the system under test (Location-based service) will expand over time and additional functionality may be added in the system under test, the test framework should support extensibility in the design of a scalable infrastructure, making is possible to add new test material. Generally the test framework should be a highly modular and maintainable framework and each module should be independent and communicate with the test management with a standard interface.

With this modular black-box approach, the functionality available within each module can be readily expanded without affecting any other part of the system. This makes code maintenance much simpler and the complexity of any one module will likely be quite minimal. In addition, the test framework can imporve the maintainability by defining a consistent interface used by all testers. This leads to more consistent and uniform results which aid in more uniform conformance testing.

3.3.3.3 Hierarchical Structure

To improve test efficiency and reusability, the test framework should be constructed in a hierarchical structured manner, while the test module definition and the test management are separate from the implementation of the test cases. Ideally, the test module should be framework-independent with complete specification, and can be parsed and compiled as a separate entity. The test module definition specifies the high-level definitions of the modules in the test framework, while the test management is used to manipulate the interaction between all test modules and the system under test. Test cases and test functions are defined in the test module definition, and are called in the test management part. All the test module definition and management should be written in script language, such as Perl, Tcl, Python, Ruby, etc., to ensure the ease of use by the testers.

Test implementation, however, residents on the low-level of the test framework architecture. It consists of an execution runtime interface and a control interface that realizes the test system. Here, a standardized adaptation of the test system for communication, management, component handling, external data and logging, are defined and implemented usually in same language as the system under test. The test implementation should also provide an interface to connect the system under test and this interface allows itself to be distributed into different components of the system under test to satisfy the condition of scalability testing for the location-based service.

3.3.3.4 Ease of Use

The test framework should be easy to use as usability is a critical requirement for the test framework. Good test frameworks should provide sufficient documentation on how to add new test cases, how to execute tests, how to monitor and analyze the test results.

3.3.3.5 Platform-independent

The test framework should be designed to be platform-independent by using open standards. Ideally, the test framework should be platform-independent in terms of test execution and test reports. Platform-dependent test frameworks have several drawbacks with respect to time and resources, and it requires additional working in creating these platform-specific frameworks as well as ascertaining their quality.

3.3.3.6 Manual Test

While our goal is to maximize the automation testing for all the scenarios in the system under test, there are still very likely that some of scenarios cannot be testautomated to some extend. In this situation, we need to consider manual testing for the system under test. Therefore, the test framework should not only facilitate test automation, but also support manual testing.

3.3.3.7 Test Results Verification

Results verification is a common testing activity used to determine if a test passes or fails by verifying the actual test result against the expected result. As a critical part of the test framework, it should be considered in the construction of test framework.

3.3.3.8 Test Results Reporting

The test framework should have a results reporting mechanism. Ideally, it should be able to generate the reports regarding all the requirements for the test framework, such as applicability to any platform and implementation, ease of use, test results, and failures.

In addition, the test reports should be exported in a self-contained format suitable to be published on the web, such as XML format. The reports should also provide test logs and sufficient information about failures for future investigation.

3.4. Test Framework Prototype

In order to satisfy the requirements we described above, we designed our prototype for the test framework. This prototype is mainly focused on the system-level testing for the location-based service that is constructed under a J2SE runtime environment. It is constructed in an open, distributed architecture with a standard interface connecting test components in the test system. It also provides the location-based service with a simulated execution environment by employing a context simulator and a wireless simulator, and can undertake major tests for the location-based service, such as functional testing, performance testing and server seide testing.

In paticular, the test framework is prototyped with serveral components: Context Simulator, Wireless Simulator, Test Modules, Test System Console, UI, Standard Interface and System Under Test, as shown in figure 3-2.

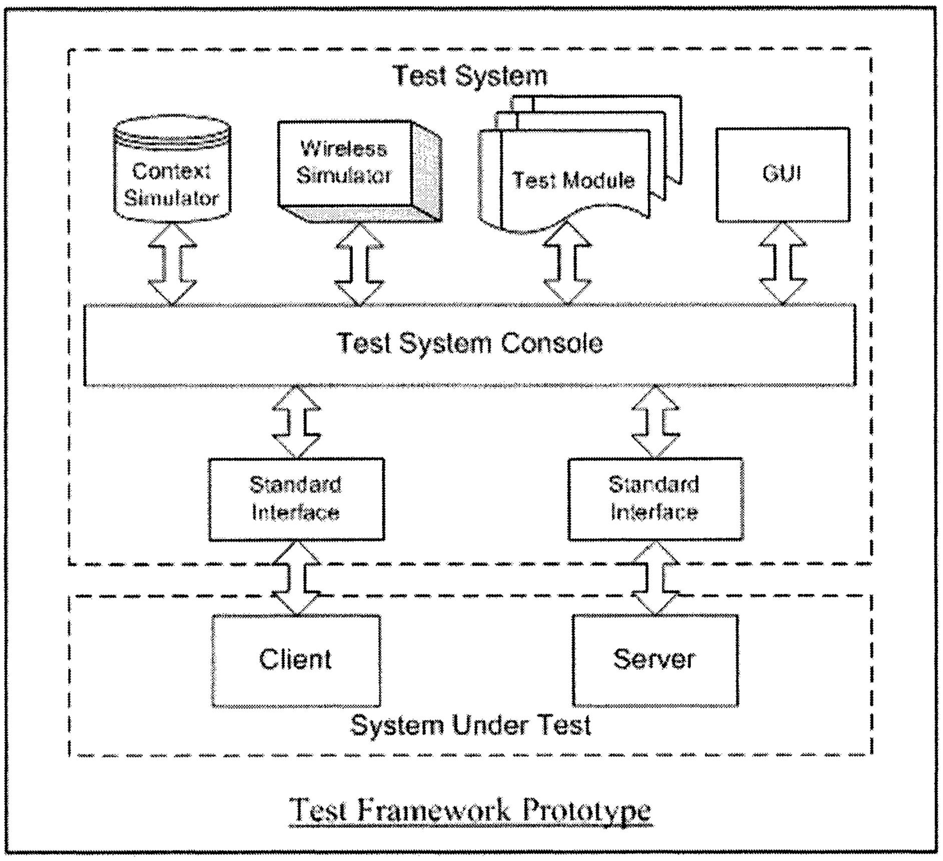


Figure 3-2: Prototype of The Test Framework

To build an open distributed architecture, the framework should define standard interfaces used to connect each test component in the test system. With well-defined formats, these interfaces can be classified into the following categories:

- **System Interface**, also called the **System Adapter**, which is the communication interface between the test system and system under test (SUT). To fulfill this fucntion, the system interface should be deployed dynamically to the test device where the components of SUT (the client application and the server application) resident.

- **External Interface** which is used to connect external third-party components, i.e. Context Simulator, Wireless Simulator, and extract the information about execution environment, such as context information and data packets that need to be transffered within the simulated wireless network.

- **Internal Interface** which links internal test components, i.e. test management, test modules, and deliver messages between the test modules and the test management.

The main component in the test system, test system console, conducts the monitor and management of the interactions and behaviours among the test components in the test system. The test modules, also called test components in the test system, mainly specifies the definitons and activities that are necessary for the test process and behaviours, i.e. test cases, test functions. They also coordinate with the test system console during the test process. Finally, the test system exchanges the event and information with the system under test via the system interface defined by the tester.

Chapter 4 Test Framework Design, Implementation and Evaluation

4.1 Introduction

As mentioned in the last chapter, several requirements must be satisfied in order to present an effective test framework for testing a given location-based service (LBS). To achieve this, we need to design a suitable open distributed test framework that is able to provide simulated infrastructure for the execution of the location-based service (LBS), and run the test cases in an automated manner.

In this chapter, we will introduce our approach of building the test framework for testing the location-based service, and apply our test framework with specified test methodologies (the TTCN-3 testing language and the SWANS wireless simulator). At the end of the chapter, we will evaluate the test framework with different criteria.

4.2 Test Framework Architecture

4.2.1 Overview

The test framework we present is motivated by Ricardo Morla's test environment [Morla+04], where the location-based application can be tested under a simulated wireless environment. However, as we mentioned earlier, Morla's test environment is application-dependent, which is not easy to be extended and maintained.

To solve this problem and fulfill the test framework requirements we previously identified, we present a test framework designed with an open distributed structure. The test framework is able to integrate test components via the well-defined standard interfaces (See section 3.4 for the category of standard interfaces). The test components include third-party components, such as context and wireless simulators, and other associated test components, i.e. test analyzer and test system console, which are necessary to apply different test strategies in an efficient and automatic way. All of the test components are designed as independent modules, connecting the test system with pre-defined interface. The system under test (LBS) hooks up the test framework via the test adapter, which is implemented with certain programming language, i.e. Java, C++, and can be dynamically distributed into various test devices according to the structure of system under test. By doing this, the test framework is capable of being extended and maintained easily whenever additional functionality is added in the system under test (SUT) (*Requirement 3.3.3.2*). Finally, certain structured test scripts should be employed in the test framework to support automation test (*Requirement 3.3.3.1*), and it is used by the tester to define the test cases and the entire test process from the given user interface.

4.2.2 Overall Test Framework Architecture

As shown in figure 4-1, our test framework is designed with a hieratical structure to satisfy the requirement 3.3.3.3 identified in chapter 3. In particular, it is functionally divided into three layers: Function layer, Control layer and Target system layer.

The **function layer** includes several functional test components supporting the test framework, including the Test Plan Generator (TPG), the System Centre (SC), the Context Simulator (CS), the Test Analyzer (TA), the Wireless Simulator (WS), and the User Interface (UI). These components mainly provide test infrastructure and simulated execution environment to the automation test framework.

For instance, the Context Simulator is introduced to provide the context event to the LBA via the Test System Console. The Wireless Simulator provides the simulated wireless communication environment for the test system. The Test Analyzer is responsible for generating the test result report, in which we can analyze the test result and measure the performance of LBA and Server. The Test Plan Generator is used to analyze the information units probed in the target system and create the test cases and test functions automatically.

The **control layer** consists of the Test System Console (TSC), and the System Adapter (SA). While the Test System Console is mainly used to monitor and manage the test process and test behaviour generated from/required by the functional test components in the function layer, the System Adapter operates as an intermediate tier to connect the test system and the system under test, encoding/decoding the test data to and from the target system under test.

The **target system layer**, also called the System Under Test (SUT) in our framework, consists of sever side applications and location-based applications (LBA). The client application and server application connect the test system via the System Adapter, and are executed under the given simulated wireless network environment as if they are running in the real environment.

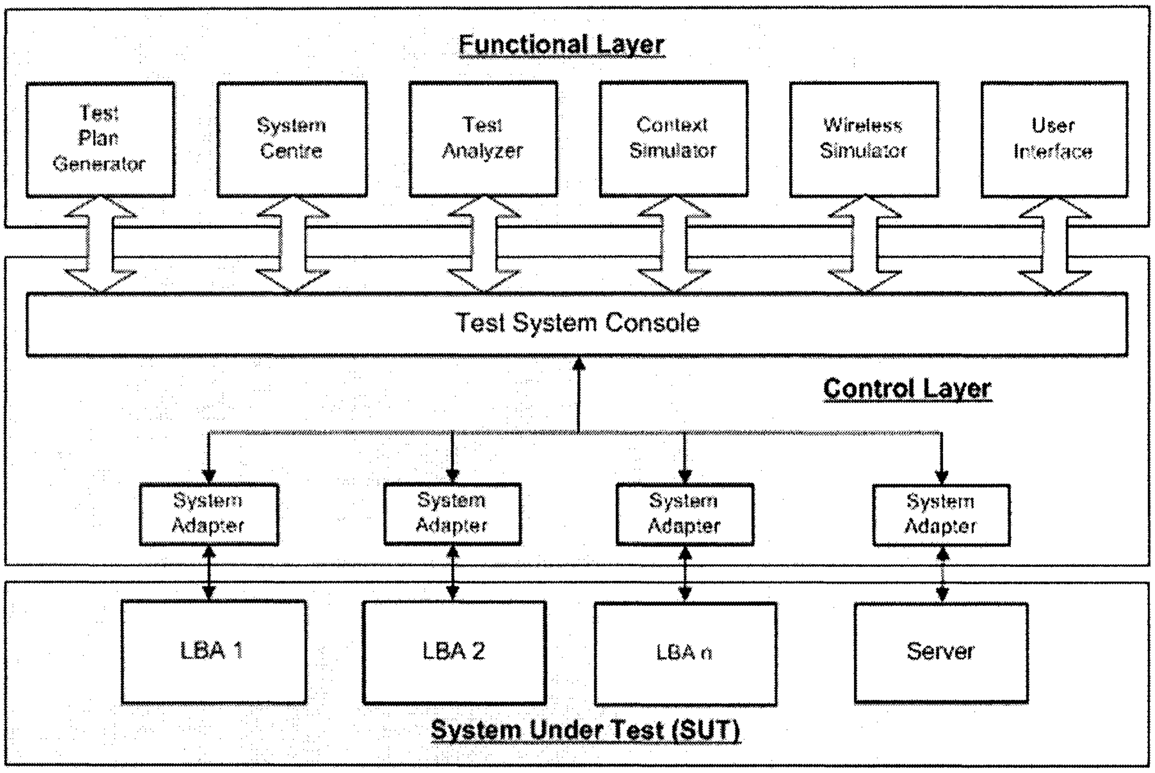


Figure 4-1: Architecture View of The Test Framework

With the standard communication protocol, each component within three layers communicates with other components by sending and receiving events and information. In the following sections, we will describe the functionality and the specification of each component in detail from the conceptual point of view.

4.2.3 Location-based Application

Most of the existing Location-based applications (LBA) are normally developed using the J2ME MIDlet structure and run on the Sun's J2ME platform. However, as mentioned in the *Scope of System Under Test* (Section 3.2), J2ME is designed specifically for small devices with limited hardware and software resources and in turn has many restrictions in the functions it offers, which results in much inconvenience to the developers. Besides, the future trend of location-based services shows the possibility of supporting J2SE platform in the mobile device, i.e. SavaJ2OS [SavaJe.com 05]. Therefore, we assume that the location-based application we test in our framework is running on the J2SE platform instead of J2ME platform.

To make our test framework application-independent (*Requirement 3.3.3.1*), our test framework needs to focus on the common components in the location-based service. As a result, the network communication component in the location-based application is considered as the core component that will be tested in our test framework since it is the common component in each location-based application, and is essential for the system-level testing of LBS. Other components in the location-based application, i.e. GUI, are not considered in our test scope at the current stage since they could be application-dependent, developed with different technologies (See Appendix B for various technologies used in the mobile device). Alternatively, the user interface of a LBA can be evaluated in the handheld emulator provided by the handheld manufacturer.

The LBA connects the Test System Console (TSC) via the System Adapter (SA), one of the standard interfaces defined in the test system, and interacts with the Server under a simulated wireless network environment, provided by the component Wireless Simulator (WS).

4.2.4 Server

As we mentioned in section 2.2, there are conceptually three tiers in the typical architecture of the location-based service: the mobile client tier, the middleware tier and the service server tier. While the mobile client tier only contains location-based applications (LBA), there are many entities involved in the middleware tier and the service server tier, such as the location measurement technology and the location management platform in the middleware tier, and application/Geo-Spatial platform and service applications in the service server tier. In this thesis, however, we do not aim at testing exhaustively interactions within all of entities in the above three tiers. Instead, we only consider "typical" scenarios in LBS, which have direct interaction with the mobile client application (LBA) from the system viewpoint. Therefore, the interactions between entities in the middleware tier and entities in the service server tier are out of scope in this thesis and would not be considered in our test framework.

As a result, we define the Server in our test framework as a *combination* of the web server and the application server. Generally, depending on the specific system requirement, there may have various software components involved in the web server and application server, such as Servlet, JSP, JMS, EJB, and Database, etc. Again, we need to focus on the common components in the server in order to build an application-independent test framework (*Requirement 3.3.3.1*). Thus, *Servlet* and *Database* components are selected as the core components tested in our test framework, as the Servlet application is commonly used in the communication between the web client and the web server, and Database is one essential part to store the data.

Specifically, the Server retrieves information (i.e. context events, request of service) from the Context Simulator (CS) via the Test System Console (TSC), and then provides the required service back to the client over a simulated wireless network provided by the Wireless Simulator (WS), or saves the client's context information in a database, which can be accessed by other external applications, such as an ERP system. Similar to the LBA, the Server interacts with other test components in the test framework via the System Adapter (SA).

4.2.5 Test System Console

The test system console (TSC) is the core component in the test framework. The main function of the test system console is to supervise the test activities in the test process and manage the interactions among all the test components and entities within the test framework.

In particular, TSC coordinates the test activities (or behaviours) and exchanges the events and information via the standard interfaces that interact with other test components and the system under test. As we mentioned in section 3.4, these standard interfaces can be an external interface, an internal interface or a system interface. From a functional point of view, these interfaces can be the pre-defined ports that are either message-based port for an asynchronous message exchange, or procedure-based communication to call procedures in remote entities, or mixed port for a message-based and a procedure-based port with the same name. For instance, the service request sent from the LBA is usually message-based information, and can be retrieved by TSC via the message-based port. The context request from the Server, however, can be one type of procedure-based communication, and will eventually invoke the Context Simulator via a procedure-based port and TSC.

The test behaviour that TSC manipulates mainly include:

- The Initialization of the test framework according to structure rules of SUT stored in the System Center.
- Execution of test cases and test functions generated by the Test Plan Generator. Also, the test result is verified here in TSC with actual test result, and thus the test framework can satisfy the test requirement of Test Results Verification (*Requirement 3.3.3.7*).
- Creation, distribution and removal of test components and system adapters dynamically according to specific test requirements. By doing this, TSC enables the test framework to be extended dynamically whenever there is any change of functionalities in the system under test. Thus, the test framework can fulfill the requirement of Extensibility and Maintainability (*Requirement 3.3.3.2*).

4.2.6 System Adapter

One of the criteria to evaluate the performance of the test framework is the ability of the test framework to monitor the status and the behaviour of the system under test (SUT). Since SUT has a dynamic architecture in terms of the number of its client applications, the test framework should contain such mechanisms to monitor and control this dynamic structure. Therefore, the System Adapter (SA) is introduced in our test framework to monitor and interact with SUT.

As one of the standard interfaces defined in the test framework, SA acts as the intermediate tier to connect the test system console with the system under test. SA can be dynamically created and distributed into the components of the SUT. Alternatively, it can be removed from the test framework after its connected component is disconnected from SUT. As we mentioned earlier, the employment of SA makes our test framework fulfill the requirement of Extensibility and Maintainability (*Requirement 3.3.3.2*).

The primary function of SA is to extract events and information from the system under test and deploy the message to SUT during the execution of SUT. It helps us to determine and control exactly what is happening within SUT at any time during its operation. To achieve this, the application under test should provide certain external interface that uses same communication protocol with SA. Therefore, some modification of the application under test may be conducted to accommodate that interface.

Another function of SA is to encode/decode the message and test data transmitted between the test system and the system under test. As the test framework introduces the test script language to develop the test cases and associated test functions, which may be in a different language from the language of SUT. Therefore, SA must be able to translate the message into relevant formats that can be understood and read by both the test system and SUT.

4.2.7 Test Plan Generator

Similar to the current commercial automation testing tools, our test framework introduces a test script language to describe the test process and define the test cases. With a hierarchical architecture, the test framework employs the Test Plan Generator (TPG) component to separate the definition of test cases and other test functions from the implementation of the test process (*Requirement 3.3.3.3*). As a result, the tester does not have to know how the test process is realized with the implementation language, and only concentrates on the design of efficient and reusable test cases.

Specifically, TPG produces the test plan which will then be executed by the test system console (TSC). The test plan includes two parts: the test definition, (i.e. test data types, test cases, test functions), and the test process control (Defining how the test runs sequentially). In addition, the test plan can be generated automatically by TPG or manually by the tester.

In the automatic method, TPG will retrieve the rules of the system under test (SUT) from the System Centre (SC) and generate relevant test definitions according to these rules.

These rules are pre-defined in the SC regarding to the specific characteristics (state and number of components) and the behaviour (relationships between these components) of SUT. As we mentioned earlier, our test framework only focus on the system under test with the generic architecture. Distributed systems with complicated architectures are not considered at the current stage in our framework. Therefore, it is feasible to describe the behaviour and characteristics of SUT in certain language understandable by TPG. However, even though SUT has generic architectures, it may bring some inconvenience to the tester who prefers to write test cases directly. To solve this problem, our test framework also allows the manual generation of the test plan by the tester. By doing this, the tester can create the test cases and test process written in the test script language. As a result, the usability of the test framework is greatly improved to satisfy the requirement of Ease of Use (*Requirement 3.3.3.4*).

4.2.8 System Centre

One of the criteria to evaluate the performance of the automation testing process is the ability of recording the states and behaviours of the system under test (SUT). To achieve this, we design the System Centre (SC) in our test framework to store the information about SUT. However, all the description regarding to SUT is specifically defined in certain presentation formats by the tester, instead of being recorded automatically. One reason of using this mechanism is that, the test framework is focusing on the system-level testing and common components in SUT, such as networking communication components and database. These components are not suitable for using the record/replay technique that is widely used in the GUI testing by most commercial automation tools.

In particular, the main function of SC is to describe the specific characteristics (state and number of components) of SUT, which will be used by the test system console (TSC) during the initialization of the test process. For instance, TSC can configure the necessary test components that are associated with the particular structure of SUT at the initialization stage of the test process. In addition, the dynamic relationships among the components of SUT are defined in SC, which will be retrieved by the test plan generator (TPG) in the creation of the test plan. Therefore, the addition of SC enables the automatic testing process and increases the usability of the test framework to reach the requirement of Ease of Use (*Requirement 3.3.3.4*).

Another function of SC is to store the test information collected by TSC during the execution of the test process, i.e. test verification results, response time, etc. The test information is stored in a certain format, i.e. XML format, and can be extracted by the test analyzer (TA) to create testing reports, make historical analysis and evaluate the SUT's performance. This function is also one of critical criteria to evaluate the performance of the test framework.

4.2.9 Test Analyzer

The Test Analyzer (TA) is a software component that can be developed as an internal component within the test framework. Alternatively, it can be done by integrating third-party components via the external standard interface. The main function of TA is to generate the test reports and analyze the historical data stored in the system centre (SC). The analysis generates a series of graphs and reports to help summarize and present the end-to-end test results. In this way, the test framework can reach the requirement of Test result Reporting (*Requirement 3.3.3.8*)

In addition, TA allows the tester to view and monitor the performance of components in the SUT, i.e. the clients, the network and the server, at any time during the test. This can be done directly by interacting with the test system console (TSC). Real-time monitoring allows for early detection of performance bottlenecks during test execution. As a result, the test framework helps identifying the pitfall existing in SUT and makes a precise evaluation for the SUT's performance. It can then accelerate the test process and achieve a more stable application.

4.2.10 Context Simulator

In order to satisfy the Requirement 3.3.2.3 – Location context acquisition, the test framework employs the Context Simulator (CS) component. CS is a third-party software component and can be integrated into the test framework via the external standard interface. From the functional point of view, CS acts as the location management platform (LMP) in the typical location-based service (LBS) (See section 2.2.3 for the function of LMP), and the primary function of CS is to distribute context events to the Server whenever the Server sends any location request during the execution of the test. Therefore, the introduction of CS helps the test framework creating a simulated execution environment for the system under test (SUT).

The test framework provides two ways to build this context simulator. The first method is to integrate a third-party context tool, i.e. Context Toolkit [Dey+00], to produce the context events. The Context Toolkit is an open source context tools used in the academic area. It consists of context widgets and a distributed infrastructure that hosts the widgets. Context widgets are software components that provide applications with access to context information while hiding the details of context sensing. The Context Toolkit supports the transmission with the test system by using HTTP and XML, which enable communication across a wide variety of platform and devices.

Another way of building the CS can be done by integrating a database that stores the context information. Specifically, the context data is firstly pre-defined and stored in the database. What the test framework does is developing an external interface (i.e. JDBC API) to access the database any time during the test. By doing this, LBS can retrieve the location data from the database via the test system.

4.2.11 Wireless Simulator

The Wireless Simulator (WS) is another third-party component that can be integrated in the test framework via the external standard interface. It enables the test system to simulate a wireless network of multiple nodes, in which the multiple mobile client applications and Server applications (nodes) communicate with each other under certain network protocols, such as GRPS, 802.11b etc (*Requirement 3.3.2.1*). In addition, the test framework enables the service selection function under a wireless environment with the association of the WS to reach the requirement of Service selection mechanism (*Requirement 3.3.2.4*).

In order to provide a wired network and a wireless network simulation, WS should be designed as a portable component and can be loaded in/unloaded from the test framework any time according to the specific test requirements. In this way, the execution of test cases can be undertaken in a wired or wireless network environment. Thus, the location-based service can be tested firstly in the wired network to ensure the functionality in the client tier, middle tier and server tier applications. After that, the whole system under test (SUT) can move on to the wireless communication network and verify its functionalities under relevant wireless communication protocols.

4.3 Test Framework Implementation

One of the important criteria to evaluate the performance of a test framework is the ability of simulating the behaviour of the system under test (SUT). The typical scenario of LBS is that, the context information can be retrieved from LMP (Location Management Platform, also see Section 2.2.3 for details) either after the mobile user sends a location request from the mobile device (*User Requested LBS*), or after LBS is triggered when the mobile user is out of pre-defined geographic boundary (*Triggered LBS*). No matter what kind of LBS is executed, the context information and required service will be sent from the server to the client application (LBA).

To simulate the typical LBS behaviour, the tester will act as the mobile user by firstly sending the service request from the client application (LBA) to the server. After the server detects the service request, it will retrieve the user's current position from the Context Simulator (CS) via the test system console (TSC) and then submit the user's context information and any required service to the LBA, such as the nearest restaurant or gas station.

In particular, the primary role of a tester in the initialization phase of the test process is the encoding of the behaviour and the rules that apply to the system under test (SUT). The required behaviour is injected into the system using a scripting language and stored inside the system center (SC). With the initial rules and behaviour encoded, the test plan generator (TPG) will retrieve the information from SC and extract a set of test cases along with other relevant information, such as the number of test adapters and specific test components required to execute the associated test cases.

After the client application (LBA) sends the location request, it will be forwarded to the Server application via the simulated wireless network. Then, the context simulator (CS) is invoked to produce the context event (test data) and send it to the Server via the test system console (TSC). Alternatively, test data can be manually input from the user interface (UI). This function increases the flexibility of the test framework as the tester can type in any test data from UI and process the manual test for some specific test cases and enable the test framework to fulfill the requirement – Manual test (*Requirement 3.3.3.6*).

All generated test cases are executed by the test system console (TSC) after associated system adapters are deployed into related client applications (LBAs) as well as the Server application. The test framework can also accept the manual creation of test cases, which allows for very specific test case generation, for example, writing test cases in a test-first manner. Upon execution, the test framework will automatically bring together all relevant components including the Context Simulator, Wireless Simulator and Test Analyzer. The test result will be shown on the user interface, logged and stored in the System Centre in the certain format for further test analysis and report by the Test Analyzer.

4.4 Testing and Test Control Notation (TTCN-3)

As mentioned earlier, our test framework should be constructed with standard interfaces to all test components connecting to the test framework. The Testing and Test Control Notation (TTCN-3) [Grabowski+03] is one testing implementation language to satisfy all of test framework requirements we defined in chapter 3. In this section, we will introduce the concept of the Testing and Test Control Notation (TTCN-3) [Grabowski+03], which is a new test specification and test implementation language. Later, we will apply it to our test framework.

4.4.1 Introduction of TTCN-3

TTCN-3, is conventionally driven by key players with the telecommunication industries, and is a new test specification and test implementation language that supports all kinds of black box testing of distributed systems. TTCN-3 is applicable to the specification of all types of reactive system tests over a variety of communication interfaces. Typical areas of application for TTCN-3 are protocols, services, APIs, software modules, etc. TTCN-3 is not restricted to conformance testing or protocol testing. It can be used in many areas, such as interoperability / inter-working testing, robustness testing, performance testing, regression testing, system testing, integration testing, load/stress testing, etc.

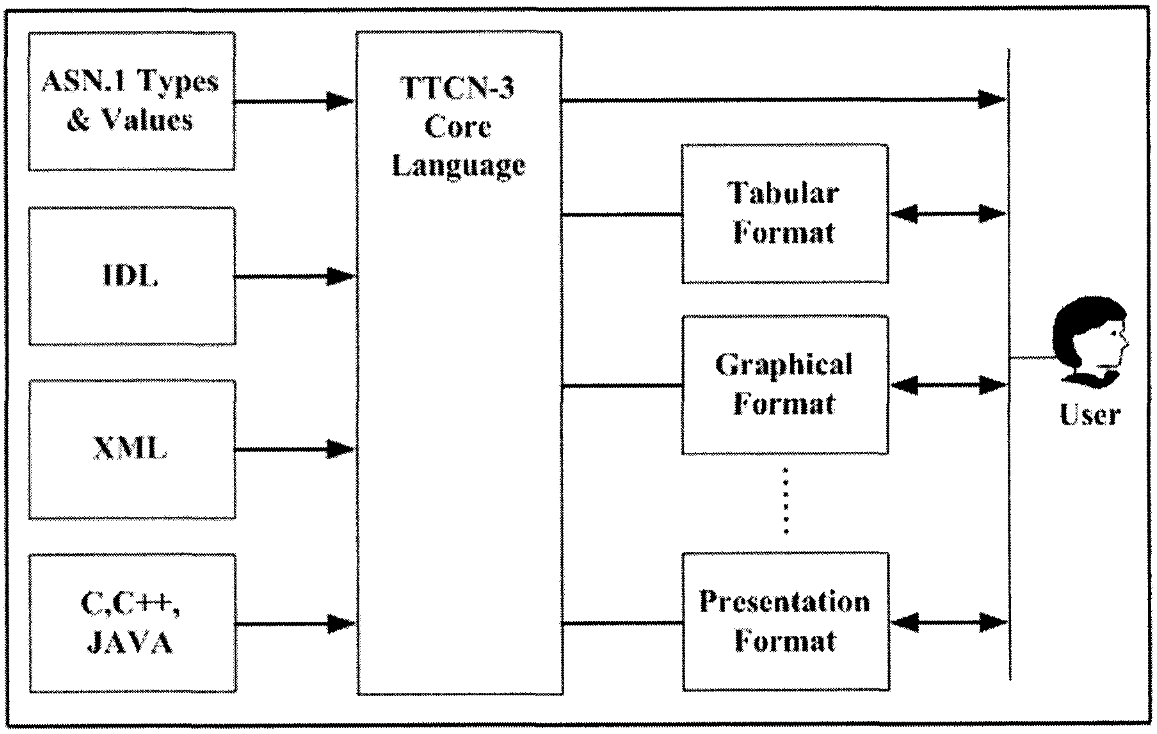


Figure 4-2: TTCN-3 Standard Presentations

As shown in figure 4-2, TTCN-3 is built from a textual core language that provides interfaces to different data description languages, including ASN.1, IDL, XML, C++, Java, etc., and TTCN-3 can be represented with different presentation formats, such as tabular format and graphical format, etc. In other words, different TTCN-3 presentation formats provide various alternative ways of specifying test scenarios visually or in a context-specific manner. All presentation formats will be eventually converted into the core notation while still preserving their meaning, which allows the same compiler and runtime execution environment to be used regardless of which presentation format the different tests are specified in. Some examples are shown in Appendix C and Appendix D for the tabular presentation format and graphical presentation format, respectively.

TTCN-3 allows an easy and efficient description of complex distributed test behaviour in terms of sequences, alternatives, loops and parallel stimuli and responses. Stimuli and responses are exchanged at the interfaces of the system under test, which are defined as a collection of ports being either message-based for asynchronous communication or signature-based for synchronous communication. The test system can use any number of test components to perform test procedures in parallel. Likewise for the interfaces of the system under test, the **interfaces** of the test components are described as **ports** [Schieferdecker+03]. The well-defined interfaces of TTCN-3 enable a set of operations independent of the target, processing platform, implementation language, etc. Communication between TTCN-3 components is either *message* or *procedure* based.

As shown in figure 4-3, the TTCN-3 test system can be distributed among several test devices, and the entire test system is synchronized by the Test Management (TM) and the Component Handling (CH) components [ETSI 04]. The CH entity is responsible for distributing parallel test components. This distribution might be across one or many physical systems. The CH entity allows the test management to create and control distributed test systems in a manner which is transparent and independent from the test execution.

The TTCN-3 Executable (TE) entity is responsible for the interpretation or execution of the TTCN-3 abstract test suite. The External Codec (ECD) entity is responsible for encoding and decoding data associated with message based or procedure based communication within the TE. The SUT Adapter (SA) is responsible for adapting message and procedure based communication of the TTCN-3 test system with the SUT to the particular execution platform of the test system. The Platform Adapter (PA) implements TTCN-3 external functions.

A TTCN-3 test system contains two interfaces: the TTCN-3 Runtime Interface (TRI) and the TTCN-3 Control Interface (TCI). While the TRI defines the interactions between the TE, SA and PA entities within a TTCN-3 test system implementation, the TCI defines the interaction between the TE, CH, TM and Coding/Decoding (CD). It provides means for the TE to manage test execution, distribute the execution of test components among different test devices and encode and decode the test data.

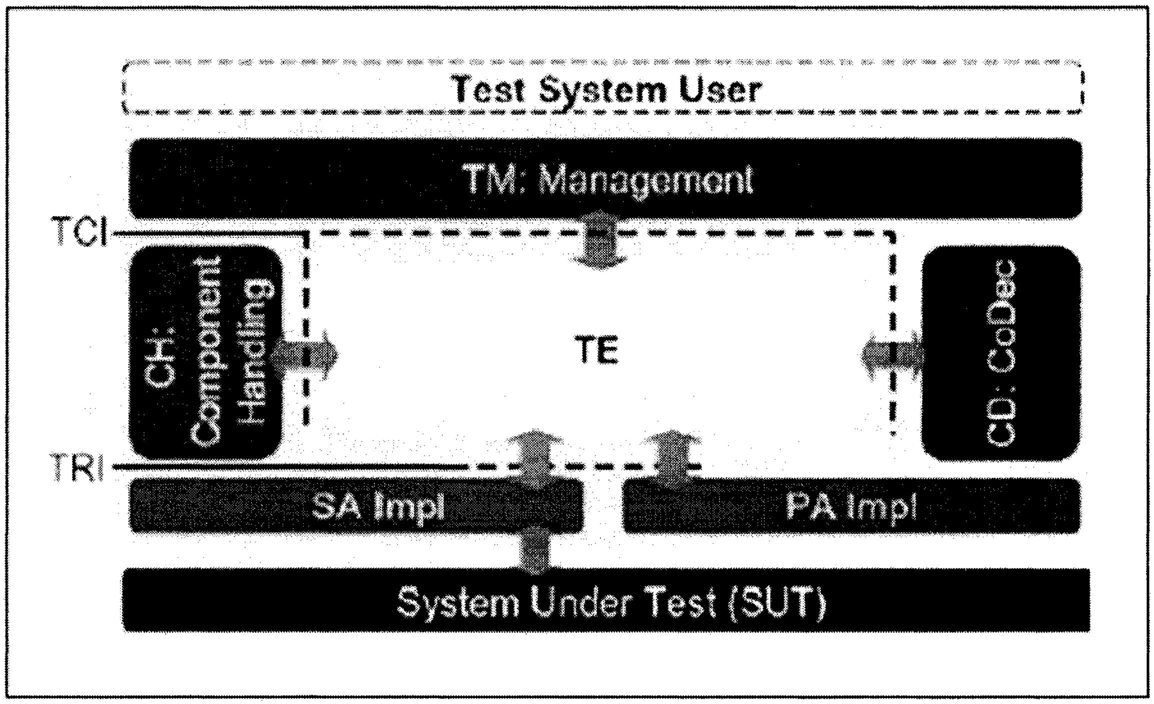


Figure 4-3: General Structure of A TTCN-3 Test System [ETSI 04]

In short, the benefits of using TTCN-3 on testing applications include:

- Re-use of existing conformance test environment, including communication ports, templates, implemented message, sequences
- Platform-independent specification of tests
- Well-define system interfaces, modularity, extensibility
- Allows repeated use of the same test scenario with different protocols
- Readable specification language, easy-to-scale and built-in load balancing

4.4.2 Applying Test Framework with TTCN-3

When applying TTCN-3 to our test framework, as shown in figure 4-4, the architecture is separated into three layers, including the TTCN layer, Adapter layer and the Connector layer.

In the TTCN layer, the functional components, such as the Test Plan Generator (TPG), the Wireless Simulator (WS), the System Centre (SC), the Context Simulator (CS), and the Test Analyzer (TA) can be realized by several independent test components, connected to the TTCN-3 test system via standard communication ports, i.e. Platform Adapter defined in the TTCN-3 structure in figure 4-3. The Test System Console should be defined as the main test component (MTC), responsible for the creation of test components, the starting of the execution of a test component, the verification of the distribution, as well as component termination indication. The definition and implementation of all of the components are written in the TTCN-3 testing script language.

In the Adapter layer, the test adapter and other components, including Wireless Simulator, Context Simulator and Test Analyzer, are implemented with Java language and zipped in one JAR archive. The components WS, TA and CS work as external functions in the TTCN-3 architecture, connecting the TTCN-3 testing system via External Function Container, provided by the Platform Adapter in the TTCN-3 structure in figure 4-3.

Finally, the TTCN-3 testing system connects with SUT via the System Adapter (SA) in the Connector Layer, which is implemented in Java. Here, SA matches the SUT Adapter defined in the TTCN-3 structure in figure 4-3. It will be distributed into the server application and client applications (LBAs) and encode/decode the messages from or to the system under test (SUT).

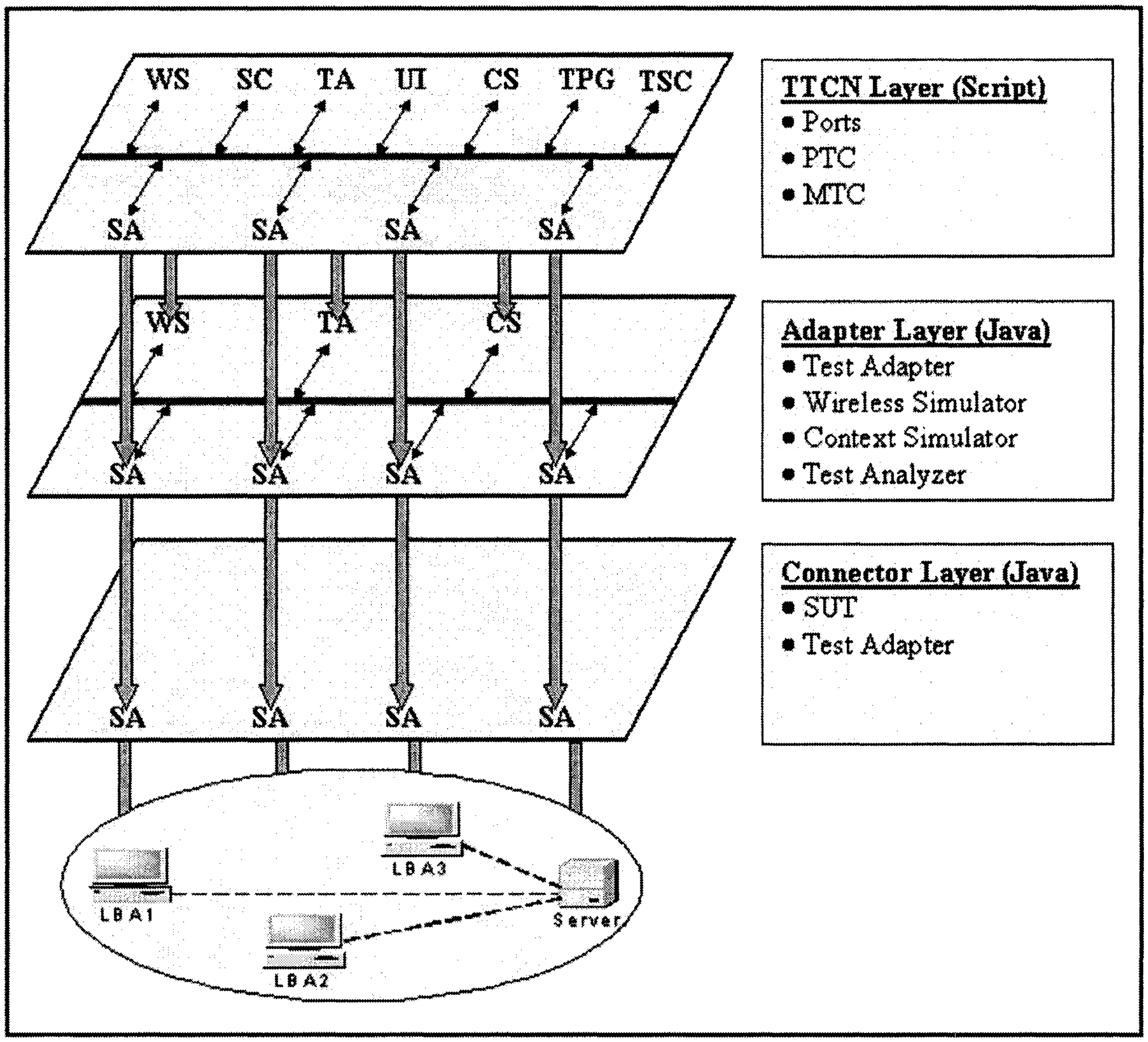
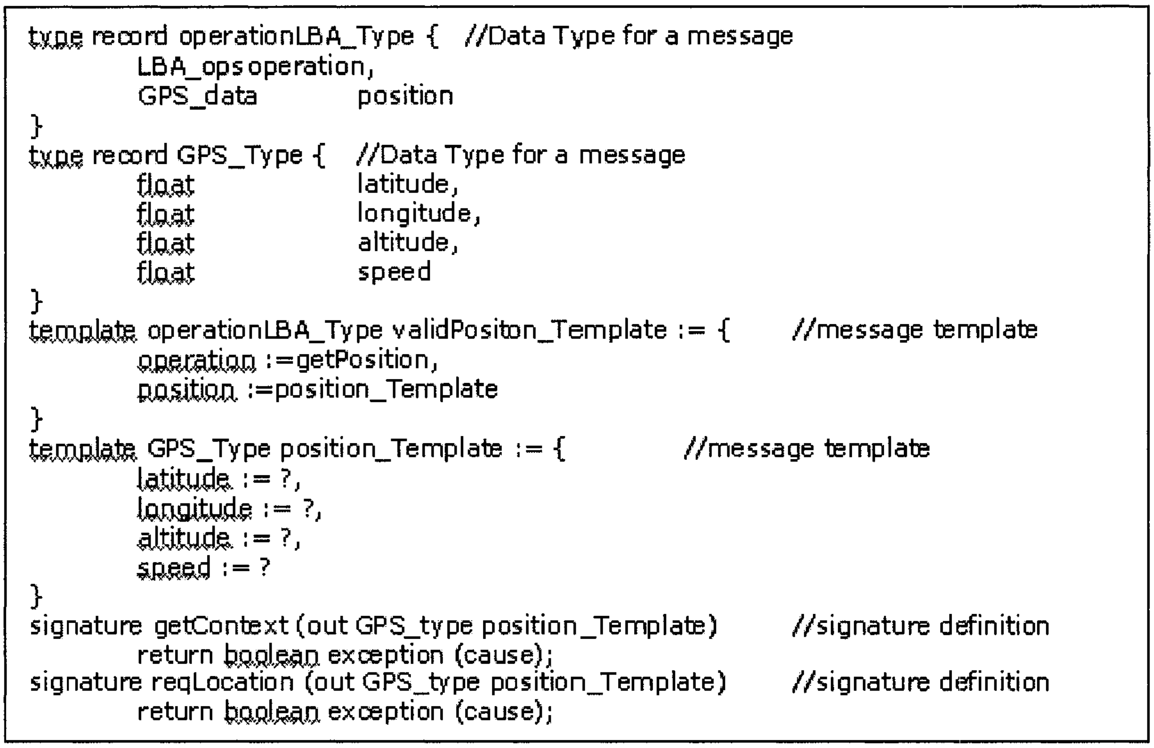


Figure 4-4: The Test Framework Architecture with TTCN-3 Test System

Depending on the specific testing demands, the test components can be integrated into the TTCN-3 testing system dynamically by using the external function, which is defined within the TTCN-3 test specification, and be implemented and executed in the test adapter. A TTCN-3 test specification consists of four main parts:

- Type definition for test data structure
- Templates definitions for concrete test data
- Function and test case definitions for test behaviour
- Control definitions for the execution of test cases

The test data is the information exchanged among the test components and between test components and SUT. We use two types of communication: one is *message-based communication* between the test system and LBA, and for the coordination among the test components; and the other is *procedure-based communication* between the Context Simulator and the test system. Message-based communication and procedure-based communication are realized by data type and procedure signatures, respectively. There are some data types required by SUT in the figure 4-5.



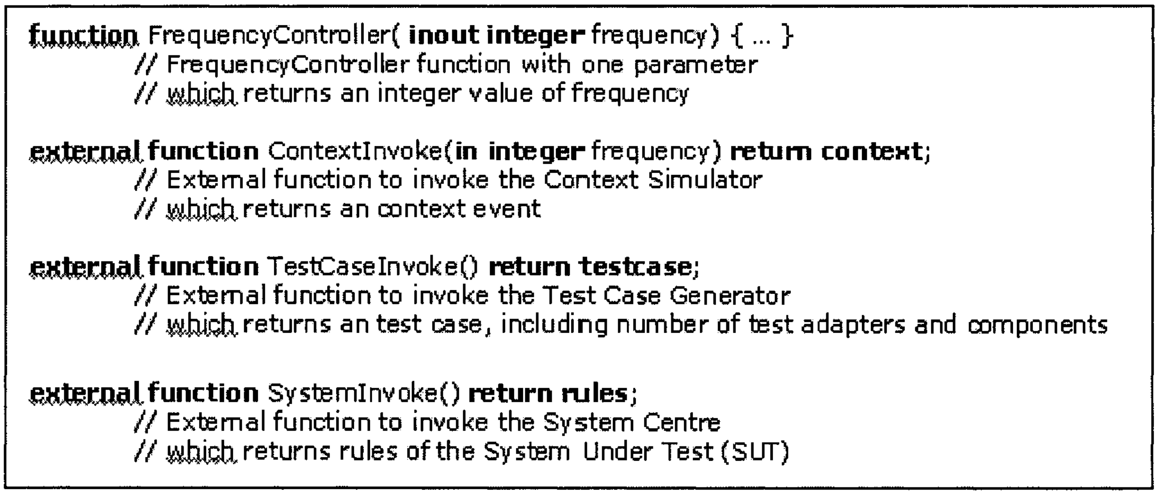
```
type record operationLBA_Type {  //Data Type for a message
        LBA_opsoperation,
        GPS_data            position
}
type record GPS_Type {  //Data Type for a message
        float               latitude,
        float               longitude,
        float               altitude,
        float               speed
}
template operationLBA_Type validPositon_Template := {      //message template
        operation :=getPosition,
        position :=position_Template
}
template GPS_Type position_Template := {          //message template
        latitude := ?,
        longitude := ?,
        altitude := ?,
        speed := ?
}
signature getContext (out GPS_type position_Template)       //signature definition
        return boolean exception (cause);
signature reqLocation (out GPS_type position_Template)      //signature definition
        return boolean exception (cause);
```

Figure 4-5: Data Types, Procedure Signatures and Test Data

Apart from the *test data*, we also need to define the test system with the address, port and component types in the test specification. Among these types, the *address* is used to define the physical IP address of server and computers in the distributed system. In addition, *communication ports* should be defined to enable communication between test components and between the components and the test system interface. As a result, the ports of test components, i.e. TPG and SA, can be connected to the ports of the test system console (TSC) by means of the **map** operation provided by the TTCN-3 core languages.

Dynamic *test behaviour* is expressed as test cases as well as functions. *Functions* are used in TTCN-3 to express test behaviour, to organize test execution or to structure computation in a module, such as to calculate a single value, to initialize a set of variables or to check some condition. Functions may return a value. In our case, functions can be used to calculate response time of one test case in the Test Analyzer (TA). Likewise, we can create one internal function (Frequency Controller) to control the frequency in which the Context Simulator provides the location information. By doing this, we can simulate one scenario of the location-based service - dynamic movement of the mobile client when, for example, the mobile user is driving the car on the freeway.

A function may be defined within a module or be declared as being defined externally (i.e. **external**). For an external function only the function interface has to be provided in the TTCN-3 module. As shown in figure 4-6, Test Plan Generator (TCG), System Centre (SC) and Context Simulator (CS) would be realized by the external function in Java programs, and they will be invoked by the TTCN-3 module via the predefined interface. All of these components could be dynamically integrated into the test system according to the required components' information provided by the test case generator. The TCG would interpret the rules stored in System Centre and convert them into the corresponding test cases and number of test adapters required by SUT with the XML format. In the meantime, we need to develop one XML parser Java program to transfer the XML data into the data format readable by TTCN-3. Alternatively, the test case can be manually defined by the system administrator using the TTthree compiler tool. Finally, a test case would be called and executed in the control part, which may contain local definitions and describes the execution order.



```
function FrequencyController( inout integer frequency) { ... }
        // FrequencyController function with one parameter
        // which returns an integer value of frequency

external function ContextInvoke(in integer frequency) return context;
        // External function to invoke the Context Simulator
        // which returns an context event

external function TestCaseInvoke() return testcase;
        // External function to invoke the Test Case Generator
        // which returns an test case, including number of test adapters and components

external function SystemInvoke() return rules;
        // External function to invoke the System Centre
        // which returns rules of the System Under Test (SUT)
```

Figure 4-6: Function Definitions

4.5 Wireless Simulator

As mentioned earlier, we integrate a wireless simulator into our test framework. The purpose of using the wireless simulator is to provide the system under test with a simulated wireless communication network.

From the developer perspective, running the location-based application on the handheld emulator is one of popular approaches to communicate with the server under a simulated wireless network. Most of the current handheld emulators provided by mobile device manufactures contain the function of simulating a wireless network environment and can simulate different wireless network coverage conditions for transmitting data from the handheld, i.e. good coverage, delays and random success of transmission. However, most of them are not open source software and can't be easily combined as a third party component into the automated testing tools. Therefore, in the phase of integration testing and system testing, the tester must undertake manual process for each test case, which is very time-consuming.

Open source wireless simulators, however, provide an alternative method to build the simulated wireless network environment. Meanwhile, they provide related APIs to connect to third party software and thus work well with the automated testing tools.

In this section, we will introduce some of current open source wireless simulators and describe the Hata computation model [Hata 80], which will be applied as the mobility model in our wireless simulator.

4.5.1 Current Wireless Simulators

There are two approaches for wireless communication between two hosts [Altman+04]. The first is the centralized cellular network in which each mobile is connected to one or more fixed base stations, so that a communication between two mobile stations requires the involvement of one or more base stations. Currently most of the location-based services use the centralized cellular network. However, sophisticated simulation tools of the physical radio channel and the simulation of power control mechanisms are needed in order to model cellular networks, which makes it unfeasible to be used as the approach of simulating the wireless network in our test framework.

A second decentralized approach consists of a mobile ad hoc network (MANET) between users that wish to communicate between each other. Due to the more limited range of a mobile terminal (with respect to a fixed base station), this approach requires mobile nodes not only to be the sources or destination of packets but also to forward packets between other mobiles. In other words, wireless mobile ad hoc networks differ from centralized cellular networks because of their highly dynamic topologies and special routing protocols that have to be adapted to their dynamic topologies. Users can define arbitrary network topologies and create their own simulation scenarios.

There exist many discrete event simulators to simulate ad-hoc networks for commercial and research purposes, such as NS2 [Fall+05], GloMoSim [UCLAPCL 05], SWANS

[Barr 05] and OPNET [OPNET 05]. Due to their popularity and widespread utility, discrete event simulators have been the subject of much research into their efficient design and execution (surveyed in [Fujimoto 90], [Fujimoto 95], [Misra 86], [Nicol+94]). Next we are going to introduce some open source discrete event simulators used in the academic research area.

NS2 [Fall+05] is a discrete event simulator developed as part of the GINT project at the University of California at Berkeley. NS2 is extensively used by the networking research community. It provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks, etc. NS2 simulator is based on two languages: an object oriented simulator, written in C++, and an OTcl (an object oriented extension of Tcl) interpreter, used to execute user's command scripts. NS2 can simulate the main existing routing as well as transport layers that network applications use. In addition, it can take into account the MAC and link layers, the mobility, and some basic features of the physical layer.

GloMoSim [UCLAPCL 05], developed at University of California at Los Angeles, is a scalable simulation environment for wireless and wired network systems. The simulator is being designed using the parallel discrete-event simulation capability provided by Parsec, which is a programming language similar with C. GloMoSim is being built using a layered approach similar to the OSI seven layer network architecture. Simple APIs are defined between different simulation layers, including the channel, radio, MAC, network, transport, and higher layers. This allows the rapid integration of models developed at different layers by different people. GloMoSim source and binary code can be downloaded only by academic institutions for research purposed.

SWANS [Barr 05] is a scalable wireless network simulator built atop the JiST [Barr 04] platform. SWANS is organized as independent software components that can be composed to form complete wireless network or sensor network configurations. Its capabilities are similar to NS2 and GloMoSim, but is able to simulate much larger networks. SWANS leverages the JiST design to achieve high simulation throughput, save memory, and *run standard Java network applications* over simulated networks. In addition, SWANS implements a data structure, called hierarchical binning, for efficient computation of signal propagation.

As described earlier, the location based application tested in our test framework is the Java-based software developed on J2SE platform. SWANS is a wireless network simulator written purely in Java and it provides special API as a harness to run ***regular, unmodified*** Java network applications, such as web servers and peer-to-peer applications, over the simulated wireless network. This approach lowers the learning curve and is convenient to be used for modeling users own wireless communication. Based on the above consideration, we will use SWANS as the wireless simulator in our test framework.

4.5.2 SWANS Wireless Simulator

SWANS [Barr 05] is a **S**calable **W**ireless **A**d hoc **N**etwork **S**imulator built atop the JiST [Barr 04] platform, a general-purpose discrete event simulation engine, as shown in figure 4-7. Java networking applications are running over SWANS and all of the above components are running within the Java virtual machine (JVM).

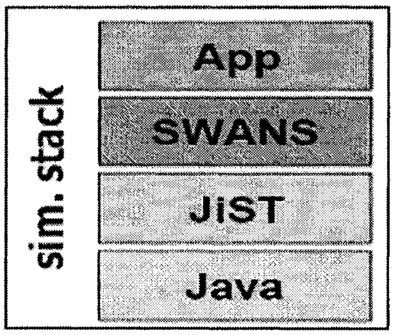


Figure 4-7: The Stack of SWANS Wireless Simulator [Barr 05]

The architecture of JiST, as shown in figure 4-8, is constructed with four distinct components: a compiler, a bytecode rewriter, a simulation kernel and a virtual machine. The process of JiST is that, the JiST simulation programs, written in plain, unmodified

Java, are firstly compiled to bytecode using a regular Java language compiler. After the modification of those compiled classes by the bytecode-level rewriter, they are then run over a simulation kernel within a standard, unmodified Java virtual machine (JVM).

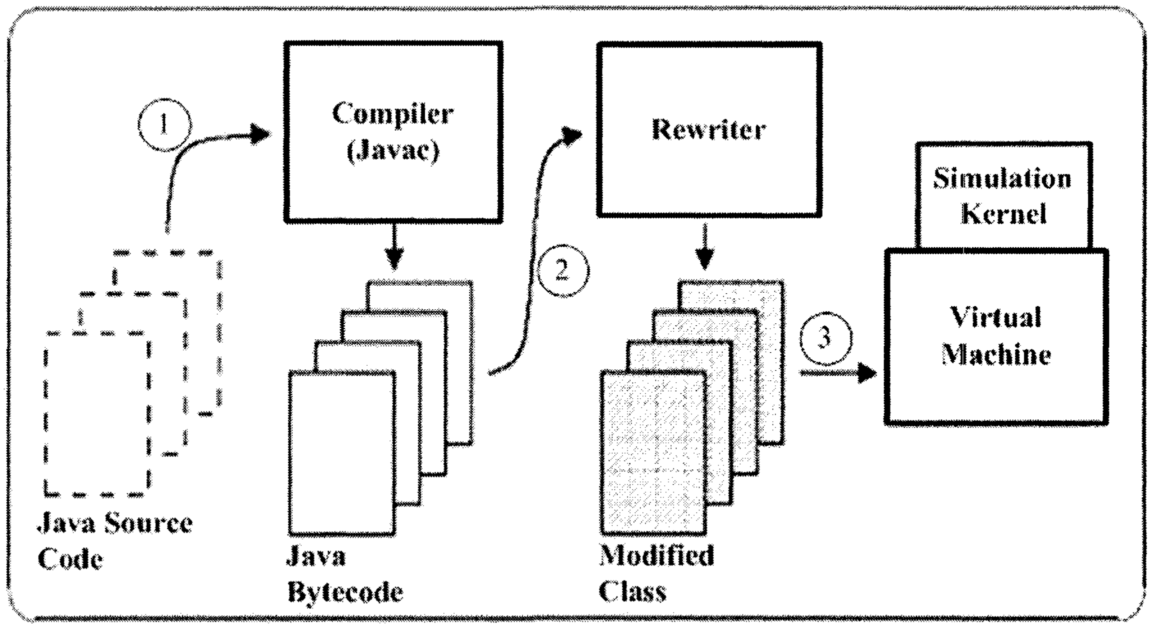


Figure 4-8: JiST System Architecture [Barr 04]

The SWANS software consists of several independent components, as shown in figure 4-9. Each component is designed as an individual JiST entity and provides different functions for building the wireless network, such as networking, routing and media access protocols, radio transmission, reception and noise models, signal propagation and fading models, and node mobility models. Thus, SWANS allows the user to set up specific wireless network characters via the relevant interface and class provided by SWANS, such as the **Mobility** entity in the **Field** component and the **RadioInterface** entity in the **Radio** component.

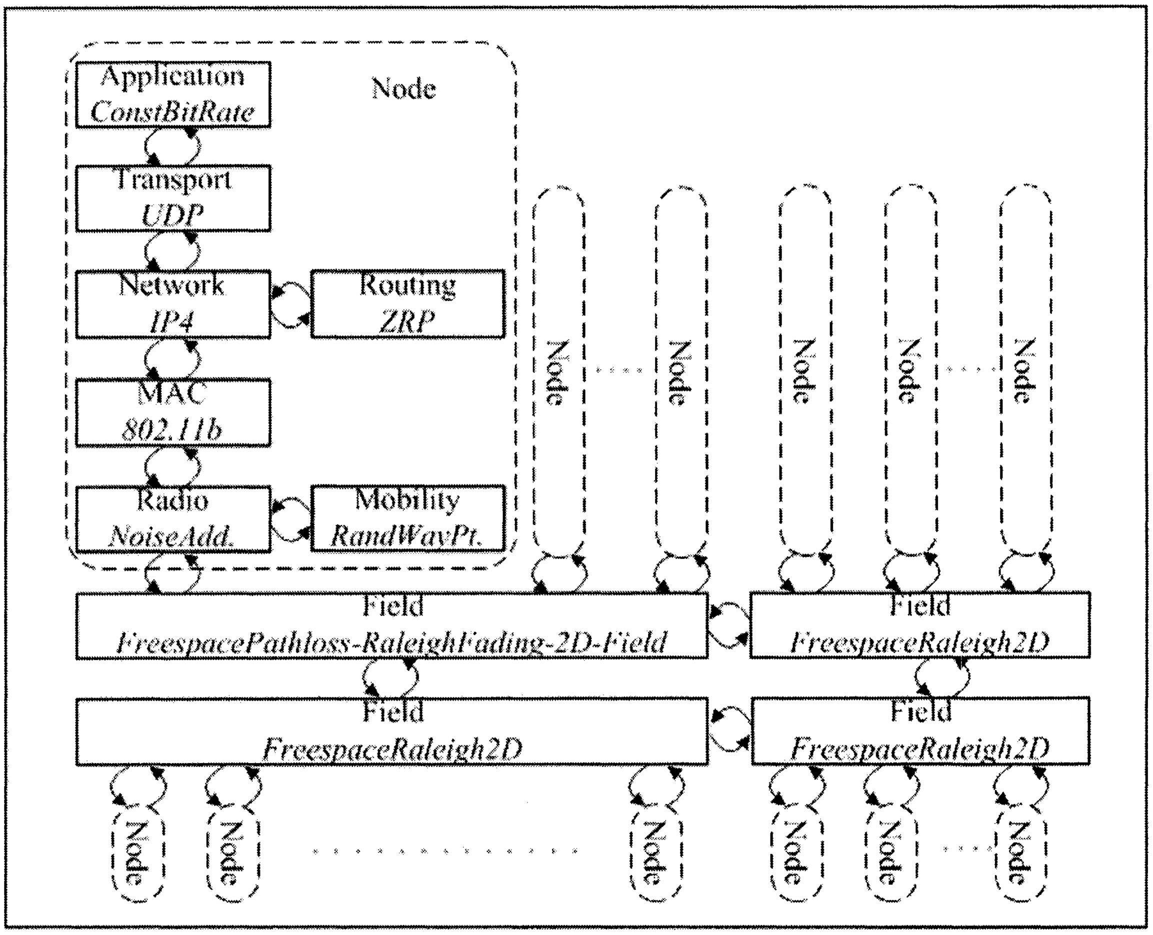


Figure 4-9: SWANS Wireless Simulator Structure [Barr 05]

SWANS offers the unique feature of running regular, unmodified Java network applications over the simulated network, which is being achieved via the ***AppJava*** [Barr 05] application entity in the **Application** layer. As a harness for Java applications, the ***AppJava*** entity inserts an additional rewriting phase into the JiST kernel, which substitutes SWANS socket implementations for any Java counterparts that occur within the application. Therefore, Java network applications can open regular communication sockets, which will actually transmit packets from the appropriate simulated node, through the simulated network.

4.5.3 Applying Test Framework with SWANS

4.5.3.1 Mobility Model

There are some situations that need to be considered when applying the test framework with the SWANS wireless simulator. The first consideration is the mobility model among mobile nodes. SWANS defines several mobility models in the ***Mobility*** entity of the Field layer, including Static, Random Waypoint, Random Walk, and Teleport models. Each model has specific mobility behaviour. For instance, the static model means no mobility among the nodes in the pre-defined area. The Random Waypoint model allows the node to pick a random "waypoint" and walk towards it with some random velocity, then pause and repeat. Thus, the network topologies among the mobile nodes are very dynamic and each mobile node needs to send/receive as well as forward packets between other mobile nodes.

However, all of the above mobility models are too sophisticated to simulate the mobility of mobile devices (clients) and web servers in the location-based service. Normally, the communication between the client and web server is very straightforward, as shown in figure 4-10. Instead of forwarding the packets, each mobile client would directly send/receive packets to/from the nearest web server that is available to it. Each mobile client does not share information with each other.

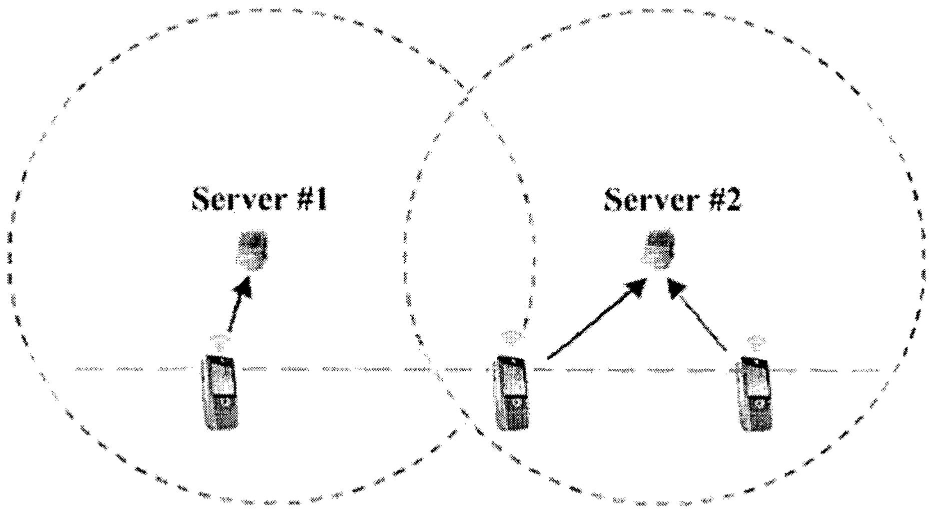


Figure 4-10: LBS Service Selection Scenario

SWANS allows users to define specific mobility models in the Mobility entity. To simulate the scenario of location-based servers, the mobile client application (location-based application) and the web server application are treated as two independent Java network applications running on the Application entity in two separated nodes, connecting the wireless network via the ***AppJava*** harness. Thus, the mobile client and web server can communicate with each other via TCP/UDP socket over the simulated wireless network.

There exists another scenario that more entities are involved in the location-based service. For instance, the mobile client may receive two different services provided by two web servers at same time. In this situation, two servers and one mobile client can be added to three different mobile nodes in the simulated wireless network, as shown in figure 4-11. The mobile client allows the access of the server's service when it is in the range of the server.

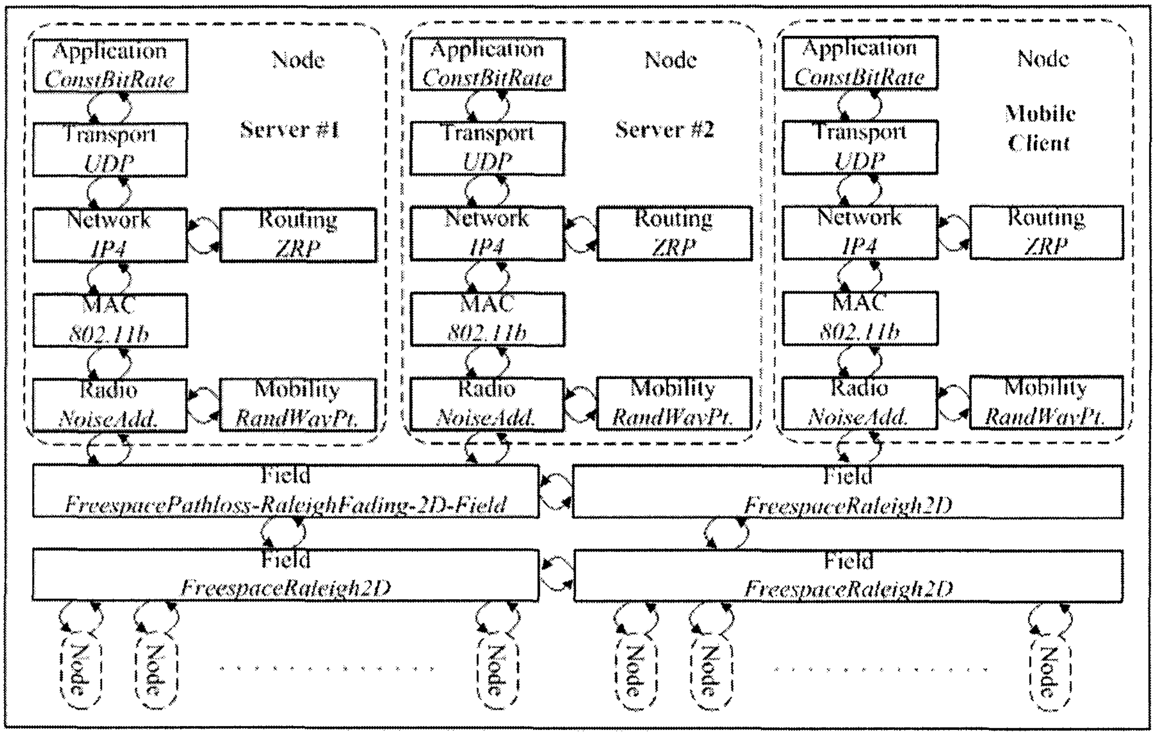


Figure 4-11: The Location-Based Service on SWANS

4.5.3.2 Propagation Models

Another issue that must be considered is the effective radio range of the wireless base station, which is used to broadcast the radio. Because of the separation between the mobile device (receiver) and the wireless base station (transmitter), attenuation of the signal strength occurs. In addition, the signal propagates by means of diffraction, scattering, reflection, transmission, refraction, etc [Neskovic+00]. Thus, a propagation model has to be defined to represent the radio characteristics of a given environment to predict the base station's effective radio range.

Generally, the prediction models can be either empirical (also called statistical) or theoretical (also called deterministic), or a combination of these two. While the empirical models are based on measurements, the theoretical models deal with the fundamental principles of radio wave propagation phenomena [Neskovic+00]. The following are some outdoor propagation models used in the academic research area:

- **Okumura Model** [Okumura+68]

  It is one of the most widely used models for signal prediction in urban areas. It is an empirical model in the frequency range of 150 MHz to 1920 MHz & distances from 1 to 100 Km. It can be extrapolated up to 3 GHz.
- **Hata Model** [Hata 80]

  It is an empirical formulation of the path loss data model to match the Okumura model, and is valid from 150 MHz to 1500 MHz for urban area.
- **PCS Extension to Hata Model** [Rappaport 95]

  This is an extension of the Hata model up to 2 GHz for Personal Communication Systems which have cells of the order of 1 Km to 20 Km radius.

Although the Okumura model is accurate for predicting path loss of cellular and land mobiles, it is not good in rural area and shows slow response to rapid changes in terrain. Likewise, the PCS extension to Hata model is mainly used for urban areas and is not suitable for rural areas. Therefore, the propagation model utilized in our wireless simulator is the Hata model, which presents our test framework with a wide simulated environment, such as large cities, suburban areas and rural areas as well.

The standard formulas for the path loss in three areas are given by:

1. Path Loss in Urban areas

Path Loss = 69.55 + 26.16*log(f) - 13.82*log(h_{te}) -a(h_{re}) +(44.9-6.55*log(h_{te}))*log(d)

Where:

f = Frequency (in MHz) from 150 MHz to 1500 MHz

h_{te} = Effective Transmitter Height, from 30 to 200 meters

h_{re} = Effective Receiver Height, from 1m to 10 meters.

d = Transmitter - Receiver separation (in Km)

a(h_{re})= Correction factor for effective mobile antenna height, which is a function of the size of the coverage area.

=(1.1*log f - 0.7)h_{re} -(1.56*log f - 0.8) dB		For medium sized city
=8.29(log1.54h_{re})^2 - 1.1 dB	f <= 300 MHz	For large city
=3.2(log11.75h_{re})^2 - 4.97 dB	f>= 300 MHz	For large city

2. Path Loss in Suburban Areas

Path Loss (Suburban) = Path Loss (Urban) -2*[log(f/28)]^2 - 5.4

3. Path Loss for Open Environment

Path Loss (Open Rural) = Path Loss (Urban) - 4.78(log f)^2 +18.33*(log f) - 40.94

As shown in figure 4-12, the Path Loss data is increased gradually with the increment of Distance. Thus, we can calculate the radio range of one base station given the following parameters.

- Maximum Allowable Path Loss of the Base Station (dB)
- Frequency (150...1500MHz)

- Effective Transmitter Height (30m...200m)
- Effective Receiver Height (1m...10m)
- City Size (Medium sized City / Large City)
- Surrounding Environment (Urban area / Suburban area / Open Rural area)

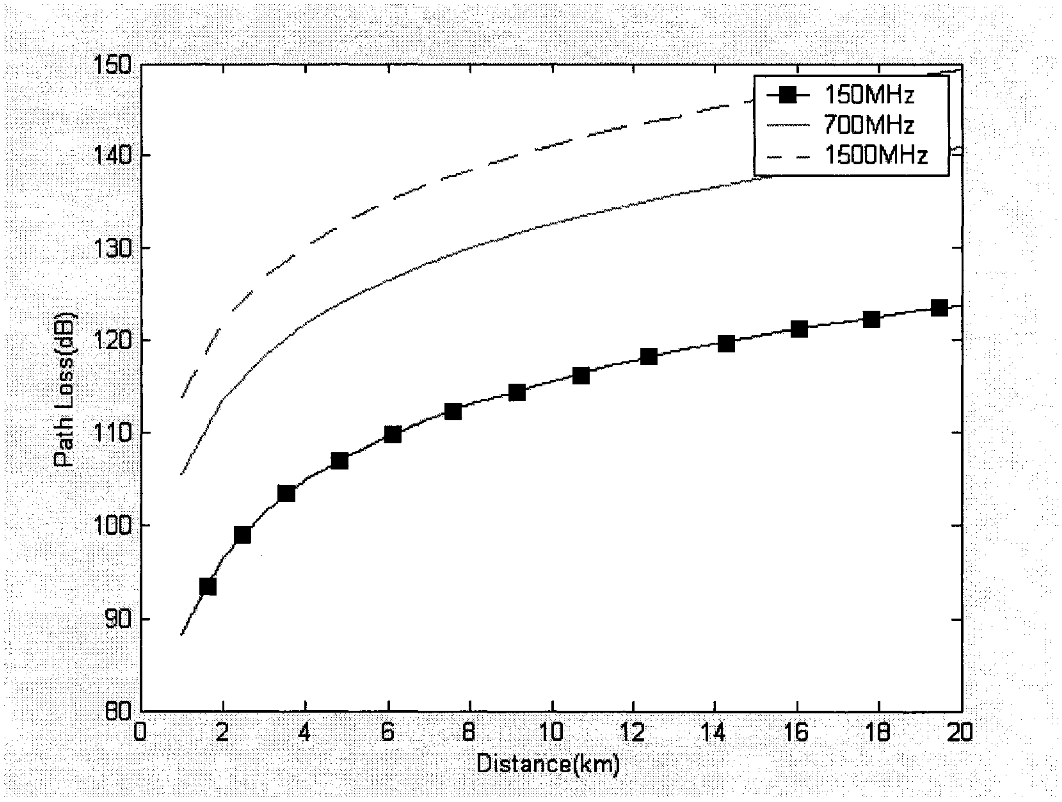


Figure 4-12: The Path Loss/Distance Trend of Hata Model

4.5.3.3 Integration of SWANS and TTCN-3

Since SWANS is open source software written in Java language, it is very easy to integrate it into the TTCN-3 test system. Specifically, SWANS will be connected to the TTCN-3 test system via the External Function Interface, which contains all the implementations of the external function calls, such as an external database and test case generator. All the test results generated under the SWANS wireless simulator will be sent back the TTCN-3 test system.

After the integration of SWANS and TTCN-3, it is possible to implement all the test scenarios of the location-based service (LBS) under two situations, as shown in figure 4-13:

- Situation 1: Testing each LBS components in the wired network using TTCN-3
- Situation 2: Testing each LBS components in the wireless network using TTCN-3 and SWANS

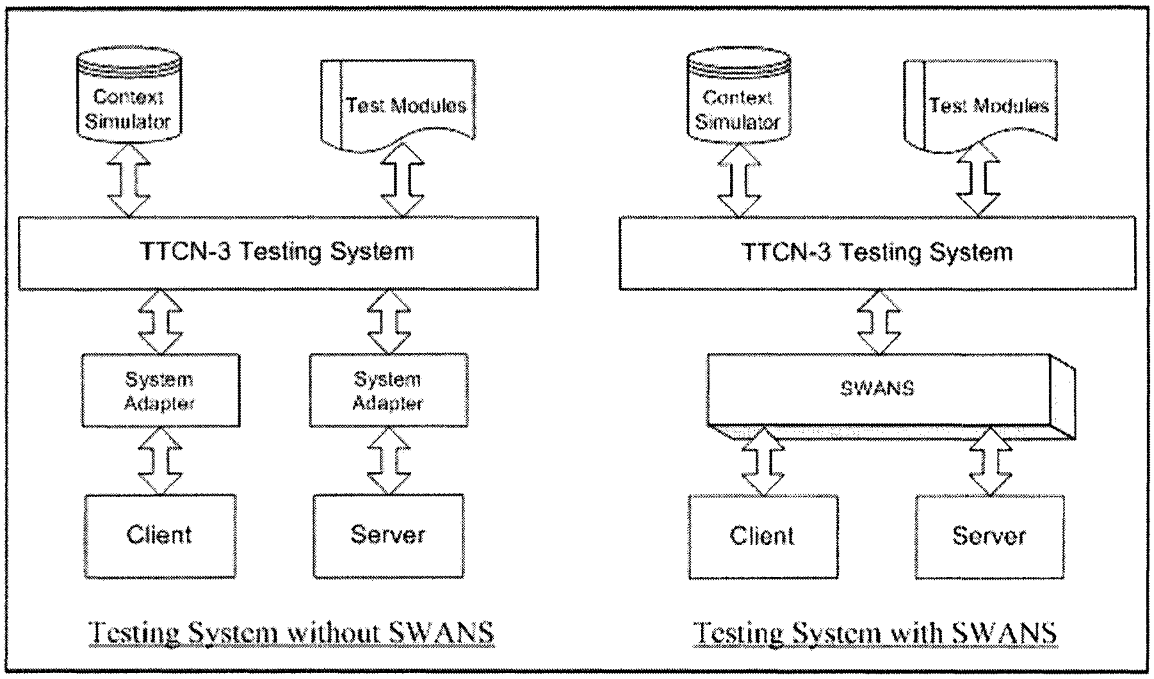


Figure 4-13: Integration of TTCN-3 and SWANS

4.6 Test Framework Evaluation

In this section, we will evaluate our test framework of implementing Location-based services testing via TTCN-3 and SWANS. As we mentioned in the above section, Location-based services (LBS) must be tested to ensure functionality, usability and scalability under all working conditions. With the integration of TTCN-3 and the SWANS wireless simulator, our test framework can provide the evaluation of Location-based services in the following areas:

Test Strategy	**Test Issue**
Functional testing	Can the LBA receive the correct service information from the Server in terms of new context events? What's the breaking point for the LBA when it is affected by quickly changing contexts?
Usability testing	Does the LBA perform the execution of service manually or automatically when the new context occurs?
Network performance testing	How is the system affected over weak wireless network signal or even temporary disconnection?
Server-side testing	How does the server handle a high volume of requests by increasing numbers of subscribers? What's the response time from the server?
Interoperability testing	Can all application components in LBS interact correctly with each other?
Security testing	Can users switch off the ability of being tracked when they don't want to be exposed to others? Does LBS have the security mechanism in terms of sensitive contents?

Table 4-1: Summary of Test Issues for Location-based Services

The above questions and issues cover most of what LBS developers are concerned with. Next, we will evaluate our test framework from the following set of criteria:

- Functional Testing – Location Changes
- Usability Testing – Auto/Manual Tracking Services
- Network Performance Testing – Poor Wireless Network Performance
- Server Site Testing - Multiple Instances
- Interoperability Testing - Multiple Software Components
- Security Testing – Personal Integrity and Secured Contents

4.6.1 Functional Testing - Location Changes

TTCN-3 distinguishes itself on supporting all kinds of black box testing for distributed systems. In our case, we want to test how the location-based application behaves over changing context events. In other words, how is the application affected by mobility? The direct way for the developer is to actually carry a device to run software and to attach it to sub-networks in the current location, which is extremely laborious and inefficient. Instead, our test framework can generate simulated context events, which are sent to the server after the location-based application (LBA) requires the location service. Then, the server forwards the context information and other service information to the location-based application within the simulated wireless network as if the location-based application were physically moved to interact with the real environment. What we need to achieve this is to monitor the data sent back to LBA to see if the location-based application responses correctly to the context events.

Specifically, the test system would call the Test Plan Generator (TPG) to deploy test cases and related configuration information (i.e. initialization parameters) to the main test component in order to invoke components in the system under test (SUT) and other functional components (i.e. Context Simulator, SWANS). The test cases generated by the TPG can be predefined manually by the user, or created automatically by interpreting the rules of SUT behaviour stored in the System Centre. Once LBA sends any location service request, the Context Simulator will be invoked to deploy the context event to the sever, and then the server will forward the received context information and corresponding service back to LBA through the simulated wireless network provided by the SWANS wireless simulator. Finally, the test system would monitor and compare the context information received by the server as well as LBA via probes (System Adapters) injected in the targeted system. If the actual result is same as the expected result, the test case is passed, otherwise it is failed.

Alternatively, the test system could invoke the Frequency Controller component to control the frequency of context events sent by the Context Simulator. This function can be used to simulate the situation like a user holding a GPS-enabled mobile phone while driving his car on the freeway. How is LBA affected by the quickly changing events? With the Frequency Controller, we can implement stress testing on the location-based application and find out its breakpoint. The response time can be recorded by the Test Analyzer during the execution of test cases, which helps to measure the update time of the LBA on the handheld.

4.6.2 Usability Testing – Auto/Manual Tracking Services

There are currently several methods of categorizing the Context-Aware Application. The first was by Schilit *et al.* [Schilit+94], who produced a taxonomy of context-aware applications containing four categorizations (*proximate selection applications, automatic contextual reconfiguration, contextual command applications, context-triggered actions*) based on 2 orthogonal dimensions: whether the task is to get information or to execute a command; and whether the task is executed manually or automatically. Another taxonomy of context aware applications was produced by Dey and Abowd [Dey+99], in which they defined three categorizations: 1) *presentation* of information and services to a user; 2) automatic *execution* of a service; and 3) *tagging* of context to information for later retrieval. No matter what taxonomy is used to define the context-aware application, it is necessary to distinguish whether the application gets the information manually or automatically, and whether the application performs the automatic execution of a service. In other words, applications that retrieve information for the user automatically based on available context are classified as *automatic contextual reconfiguration*, and vice versa.

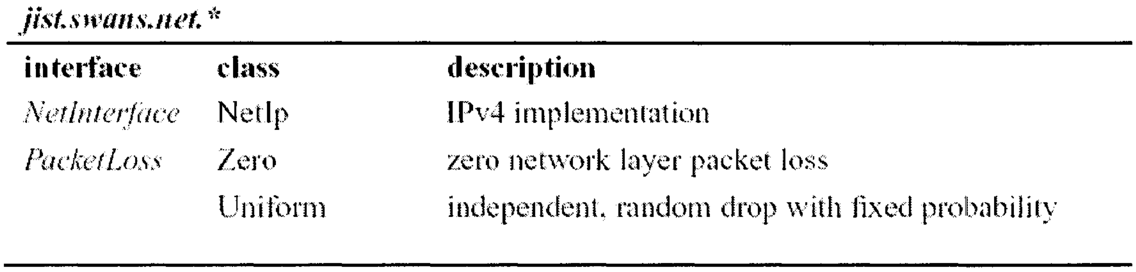
As one type of context-aware applications, Location-based services could also be categorized from the above two aspects. For instance, Mike is getting ready to leave for work in the morning. He uses his mobile terminal for a quick check of the local weather forecast (presentation of information to user manually). While driving his car to the office, he may search available information about the gas station service in order to find out the cheapest gas station around him (presentation of information to user manually). Also, his personal navigation system on the mobile terminal verbally prompts him that there has been an accident on his normal route to work (automatic execution of a service). Upon the arrival at work, his mobile web browser automatically shows him the news headlines from the site he has linked to this location (automatic execution of a service). These situations should be investigated in the LBS usability testing.

To test these scenarios, at least one LBA and two servers should be run within the SWANS wireless simulator. Each server provides an independent service within the certain range of area (predefined by the tester). After the start of test cases, the Context Simulator would send specific location data to the mobile client, so that the mobile client (LBA) would be simulated to roam in the area where servers are available or unavailable to the LBA. The main test component (MTC) would then observe the activity of the LBA and check it can detect and execute the available service.

4.6.3 Network Performance Testing – Poor Wireless Network Performance

Poor network performance is one of the key issues that need to be tested in a Location-based service. It may cause context errors as well as temporary network disconnections. In the first situation, potential errors might exist in the context information interacting with the location-based application because of the complicated surrounding environment, GPS satellite signal conditions and packet data availability. Thus, we can simulate this situation by embedding specified mutants, i.e. incorrect format and lack of parameters, into the context event generated by the Context Simulator. After the erroneous context information is pushed into LBS, the test system monitors the behaviour of the LBA and the server in terms of those mutants in the target system. In this way, we can check if the LBS service contains the mechanisms of detecting and handling the errors in the context information.

Our test framework supports wireless network simulation and emulation by integrating the SWANS wireless simulator into the framework. The weak wireless network performance would highly impact the probability of successfully transmitting packets. As shown in figure 4-14, SWANS provides a ***PacketLoss*** interface in the **Network** layer, which allows the server only receive a percentage of packet data successfully according to the predefined packet loss probability. For instance, if PacketLoss.Zero() is used, all packets sent from the mobile client will be successfully transmitted to the server. If Packet.Uniform(0.4) is used, it means that 40% of packet data sent from the mobile client will be lost during the transition of packets in the simulated wireless network. If Packet.Uniform(1.0) is set up, all of packets will be blocked in the wireless network. This can simulate the scenario that the handheld is Out of Coverage of wireless network.



*jist.swans.net.**

interface	class	description
NetInterface	NetIp	IPv4 implementation
PacketLoss	Zero	zero network layer packet loss
	Uniform	independent, random drop with fixed probability

Figure 4-14: Interfaces of The Network Layer in SWANS

In addition, the delays and random success of transmission are supported in our test framework. By inputting different transmission rates from the GUI, users can measure the transmission performance of LBS during the various wireless network coverage conditions.

4.6.4 Server Site Testing - Multiple Instances

Server site testing is one of biggest challenges in the testing of location-based services (LBS) since many components are involved in the server operation. Generally, there have two main issues that need to be considered during the server site testing: Time-out and Resource issues.

The server's time-out problem is usually caused by incorrect server application design and the database problem. For instance, if the application is not architected and implemented well, time-out conditions may lead to the loss of connection (i.e. the user must login again). In addition, if the database receives an excessive number of requests for data, it may cause long delays when responding to these requests. It is also possible that the database has been completely offline and cannot provide any feedback to the server.

The resource problem is quite common in distributed systems, including LBS services. All software applications require server's resources during their execution, such as RAM, disk space, CPU, bandwidth, open connections, etc. Does the server contain the mechanism to handle the lack-of-resource conditions?

Such issues need to be tested in the server site testing. Generally, load and stress testing are the effective techniques to measure the server performance. The purpose of load/stress testing is to discover under what conditions the application's performance becomes unacceptable by changing the application inputs to place a heavier load on the application and measuring how performance changes with various inputs. In our test framework, the server application's performance can be evaluated through load/stress testing by increasing number of clients (mobile nodes) and loading high volumes of requests.

The direct way to undertake stress testing on the server is to manually vary the inputs, i.e. number of clients, frequency of requests, mix of requests, and measure the variety of performance. This is feasible for some small client/server applications. However, with the increasing service request from multiple subscribers, it is almost impossible to run manual tests consistently and it can be difficult to accurately reproduce the set of tests for regression testing. Also, it is difficult to implement scalability testing for the server application.

Another solution is to use an automated tool. With the integration of TTCN-3, our test framework enables the performance, load and scalability testing for component-based distributed test systems via the dynamic creation and termination of test components including dynamic connections between test components and to the system under test (the SUT). The load conditions for mobile clients (LBAs) can be realized by an ensemble of parallel test components (PTCs), which can be distributed to the remote nodes of a network constituting a distributed test system and managed by the main test component.

Finally, some assumptions need to be made for the server side testing. As we focus on the load testing of the server, any other factors that may influence the response time of the server must be eliminated from the test environment in order to get a real performance evaluation on the server. Thus, the SWANS wireless simulator is not included in the serve load testing as it may increase the server response time by the wireless communicate characteristics, i.e. the time delay and the packet loss. Hence, the load testing on the server is presented only within the TTCN-3 test framework.

4.6.5 Interoperability Testing - Multiple Software Components

Interoperability testing is the activity of providing end-to-end functionality between more than two communicating systems as required by those base systems' standards. Interoperability testing should be performed at the end points and at functional interfaces. It is one of the important issues for testing Location-based services (LBS) because various software components, i.e. Map database, LBA, Web Container applications, and Databases, may be involved in the whole system. These components could be run on different platforms and physical hardware. Therefore, it is necessary to undertake interoperability testing to evaluate the LBS's capability of adapting itself on different application components that are developed by various vendors and run on different operation systems and hardware platforms.

The following are the key factors that characterize interoperability testing [ETSI 05]:

- The system under test (SUT) and the qualified system (QS) define the boundaries for testing;
- The SUT and QS come from different suppliers, or at least different production lines;
- Interoperability tests are performed at interfaces that offer only normal user control and observation;
- Interoperability tests are based on functionality as experienced by a user. In this context, a user may be human or a software application;
- The tests are performed and observed at functional interfaces such as Man-Machine Interfaces (MMIs), protocol service interfaces and Application Programming Interfaces (APIs).

TTCN-3 is good at testing the distributed system for generic LBS systems. With the TTCN-3 test system, our test framework is able to connect and communicate with all types of components in the SUT that can apply to the interfaces given by the TTCN-3 test system. Currently the TTCN-3 test system can be implemented in C++ and Java, which means, theoretically, the TTCN-3 can apply tests to any vendor's LBS components, i.e. MS SQL 2000, MySQL database, Location-based Application, that support the connection via the interface written in above two languages (C++ and Java). Therefore, it is feasible to use our test framework to make the interoperability testing for the location-based service (LBS).

4.6.6 Security Testing – Personal Integrity and Secured Contents

As mentioned in chapter 2, the security aspect is a critical factor to realize the opportunities presented by the location-based service (LBS) and it brings one of the biggest challenges for the security testing of LBS as the security and privacy issues may be considered from different aspects in LBS, such as the location-based application (LBA), the wireless network protocol, the various server software components, etc.

Some of the key security issues with the location-based service security include:

- **Privacy** – The user's privacy should be protected from the access by unauthorized parties. This is particularly significant for LBS as there is a potential that the user can be illegally tracked. Suitable business models and requirements can help the protection of the user's privacy. For instance, the service provider can remove the user's personal information (User name and ID) and only submit the user's position data (Latitude and Longitude), to the location content provider when it wants to retrieve the service content related to the user's current location, i.e. address and name of nearest restaurant.
- **Personal Integrity** – Positioning someone always interferes with the personal integrity of the user being tracked, which means a people tracking service needs to be based on the voluntary participation of the individual being positioned. As a result, LBS should always enable users to switch off the ability of being tracked when they do not want other people reading their positions.
- **Confidentiality** – confidential and sensitive data, i.e. password and PIN, should be accessed and transmitted in a secured way. According to [Yuan 05], network and data security can be guaranteed by securing either connections or contents. Securing connections can be done by establishing point-to-point secure connections with security protocol, i.e. SSL/TLS (Secure Socket Layer/Transport Layer Security). Among e-commerce applications, SSL-based secure HTTP (HTTPS) has become the standard protocol for transferring sensitive data. Securing contents can be fulfilled by building a suitable end-to-end security model with flexible encryption schemes to meet different requirements. Most of data are transferred in XML data format in e-commerce applications. J2ME can also support the communication with back-end servers using XML data format. Thus, XML security protocols can be used to improve the security of contents, such as the Web Services secure XML protocol family (WS-Security), Security Assertion Markup Language (SAML), XML digital signatures, etc. [Yuan 05].

Testing general privacy concerns and personal integrity issues can be well done in our test framework by verifying the relevant functionalities in LBS during the functional testing phase, such as checking the functionality of LBS to see if it contains the mechanism to allow users to switch off the being-positioned function from the mobile devices.

Testing the confidentiality issue can be divided into two aspects: connections and contents, as we mentioned above. As J2SE provides excellent and transparent support for HTTPS in its Generic Connection Framework, and J2ME also supports HTTPS in the MIDP2.0 specification, secured connection is already included in any location-based service which is built on J2SE or J2ME platform. Thus, secured connection testing is not considered our test framework.

Testing secured contents is eventually the testing of XML security protocols. With the integration of TTCN-3, it is feasible to testing XML based messages and protocols in our test framework. There was already some research on the automated testing of XML/SOAP based web service with TTCN-3 [Stepien+03]. Basically, the unique characteristic of XML testing with TTCN-3 is generation of the test data, which is created by mapping the XML DTD and Schemas to TTCN-3. Other steps, such as the test configuration (test components and test ports), and the test execution are similar to traditional testing procedures defined in Section 4.4.2.

4.7 Comparison between our test framework and other testing methodologies

The table 4-2 shows the comparison between our test framework (TTCN-3 & SWANS) and other test methodologies from the different testing issues. We can find out that our test framework outperforms the other testing methodologies on covering most of necessary testing strategies, including functional testing, usability testing, network performance testing, server-side testing, interoperability testing and security testing.

Test strategies Test Framework	**Functional Testing**	**Usability Testing** - Ease of use - Navigation	**Network Performance** - Low bandwidth rate - Network disconnection	**Server side Testing** - Scalability - Database	**Interoperability Testing**	**Security Testing** - Personal Integrity - XML secured contents
Handheld Emulator	Yes, only for LBA	Yes	Yes	No	No	No
Flying Emulator Framework by [Satoh 03]	Yes, only for LBA	No	No	No	No	No
UBIWISE by [Barton+03]	Sort of, only for LBA	Yes	No	No	No	No
Simulation Environment by [Morla+04]	Yes, only for LBA	No	Yes	No	No	No
Our Test Framework	Yes, for LBA & server side applications	Yes, focusing on system level	Yes	Yes	Yes	Yes

Table 4-2: Comparison of LBS Testing Methodologies

Chapter 5 Experimental Studies

In this chapter, we present experimental studies in which we apply our test framework with TTCN-3 and SWANS wireless simulator to the typical location based service and illustrate the feasibility of our test framework.

5.1 Overview of Mobile Service Tracking

A Mobile Service Tracking (MST) system is one typical Location-based service (LBS) to track the user's location and provide users with local available services from a GPS-enabled mobile phone. Users simply carry their GPS-enabled mobile phones throughout their regular activities and their locations are automatically tracked and recorded on demand. When users want to access local services, they enter that information directly into the phone and activate the GPS tracking function using a simple, one-click interface. Then, the timestamp and location are captured automatically and all available mobile services can be accessed from the interface of a user's GPS-enabled mobile phone.

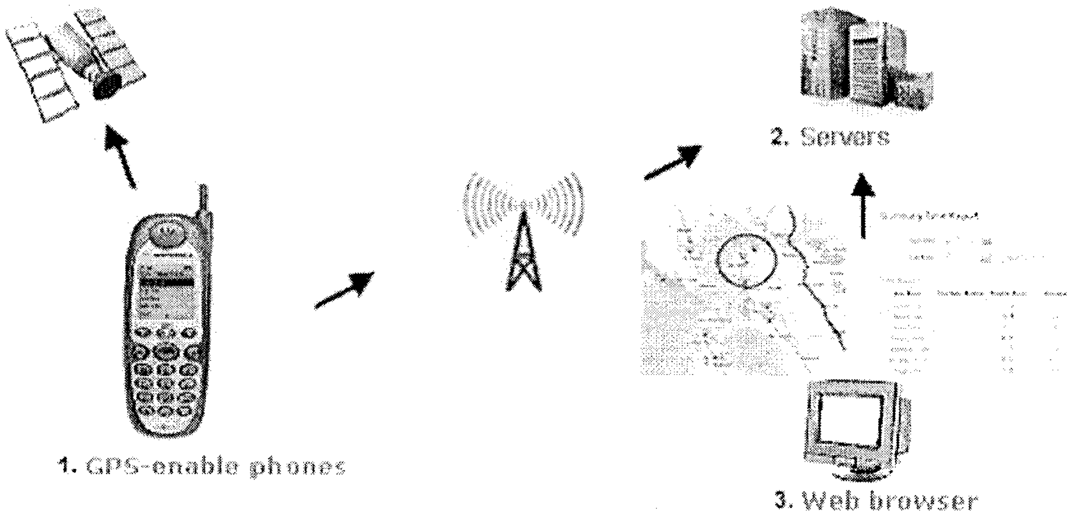


Figure 5-1: The Architecture of Mobile Service Tracking System

As shown in figure 5-1, the Mobile Service Tracking (MST) system is one generic location-based service (LBS) with four entities in its architecture – GPS satellites, GPS-enable phones, Servers and web browsers. The primary scenarios of the system are that, the mobile users turn on GPS-enable phones with them and check the local weather forecast when they get ready to leave for work in the morning. The system would then automatically report the traffic load information to the users on their normal routes to work. Upon the way to work, the system provides the fuel price in the petrol stations around you according to user's commands. In the above scenarios, location information is captured from a GPS satellite. The location and other information data, i.e. speed and timestamp, will be wirelessly transmitted to the MST's server from the Location Management Platform (not shown in figure 5-1), which is normally operated by the wireless network provider, e.g. Telus, Nextel, Verizon, etc. Once the server receives the context information, they can send it to the Content Provider (not shown in figure 5-1) in order to retrieve location content information from there. Finally, the MST's server will forward user's location information associated with required services to the mobile user. In addition, it may store the user's location information into the database on demand.

The mobile client application is developed using a J2ME MIDlet structure and is responsible for receiving current location context from GPS-satellites and tracking any available service around the mobile device. The server is built with one JBoss application server and contains one database to store service content information and one Servlet application to retrieve the requests from and send the responses to the client application.

5.2 Test Environment

When testing this location-based service (LBS), one of the key issues is to build the test environment required by the system. Here we introduce the SWANS wireless simulator as a simulation tool to simulate the real wireless communication environment on the desktop machine. Therefore, the client and server applications can implement the TCP/UDP communication upon the SWANS wireless platform. The TTCN-3 tool for testing LBS is the TTCN-3 to Java compiler TTthree [Testing Technologies 04]. TTthree is a test development and execution environment based on TTCN-3, the international standardised testing language, which includes the full range of features needed for test specification, execution and analysis. All the test scripts and test configurations are implemented through the test manager uTTman [Testing Technologies 04]. The test adapters are written in Java and loaded by the test manager. In addition, the Context Simulator is represented by the MySQL open source database [MySQL 05], providing the test data (location information) to the test system. The whole test system is run on Windows 2000 as well as the Linux operation system.

5.3 System Analysis

There are two types of interactions among entities in the system under test (SUT). One is the interaction between LBA and the server via the simulated wireless network. The other is the interaction between the server and customer web browser through the Internet, in which customers can access web services provided by the server regarding the customer' account information. Testing the functionalities between the web server and the customer web browser can be treated as the testing of web services, or in other words, the testing of e-commerce applications, which is not within the scope of our thesis. Therefore, we do not consider this situation in this experiment and limit our scope of the system under test (SUT) to the sub-system between the server and the location-based application.

5.4 Test Scenario

In this case study, we are going to apply various testing strategies, i.e. functional testing, network performance testing, server site testing, usability testing and interoperability testing for the MST system. Table 5-1 shows the typical issues of our test strategies:

Testing Strategies	**Testing Issues**
Functional Testing	- Correct data transmission under fast changing context situation - Incorrect data handle mechanism
Network Performance Testing	- Successful data transmission rate under poor wireless network performance
Usability Testing	- Automatic tracking available service
Server Side Testing	- The server response time under normal load
Security Testing	- Privacy concern and Personal Integrity
Interoperability Testing	- Compatibility with various databases by difference vendors

Table 5-1: Typical Test Issues of The Experimental Study

The test scenario can be generated from the above system analysis and test issues that will apply tests to the system under test. In order to cover the above test issues, we will consider the following test scenarios:

- *Incorrect context data.* It is quiet normal that an error may exist in the context information received by the mobile device because of its complicated surrounding environment. By means of injecting specific faults in the context data sent from the Context Simulator, such as an incorrect format and missing element of the current context, it is very efficient to detect whether location-based applications contain such error handling mechanisms.

Test Scenario	TS01
Description	Incorrect context data
Pre-conditions	Invalid context data is input into the context simulator No packet loss in the wireless communication network
Action	Inject the invalid context data into the system under test

Post-conditions	Error context information page is displayed.

- *Fast changing context information.* This scenario can be used to test the situation in which the position of the mobile client is changing quickly, i.e. driving the car in the freeway. It can be simulated in our test framework by using the Frequency Controller internal function, which enables users to control the frequency of context events sent to the system under test (SUT). Therefore we can implement stress testing on the server.

Test Scenario	TS02
Description	Fast changing context information
Pre-conditions	1. Different context data is input into the Context simulator 2.Input the frequency value 3. No packet loss in the wireless communication network
Action	Start the context simulator and test system, and check the response from the client application.
Post-conditions	The client should receive exact same context information sending from context simulator

- *No corresponding network traffic.* The network performance of the wireless network may vary in terms of the number of service subscribers and surrounding environment. For instance, if the mobile client is in the tunnel where no wireless signal exists, then all transmitting packets might be lost in the communication. It is necessary to set up different network circulated traffic to find out the influence to entities in the location-based service (LBS). We conduct this experiment by setting up different packet loss rates in the SWANS wireless simulator. If the traffic does not conform to these situations, it means potential errors may exist within the SUT.

Test Scenario	TS03

Description	No corresponding network traffic
Pre-conditions	1. Different context data is input into the Context simulator 2. Input the load times of test cases 3. Input the packet loss rate
Action	Start the context simulator and test system, and check the number of packets received by the client.
Post-conditions	The client should receive corresponding number of packets from the mobile client.

- *Services auto detection.* This scenario can be happened when the mobile device is moving in the wireless network coverage. The LBA can automatically detect all available services around it. In our test framework, we can change the radio range of the server (or the base station) by setting up its relevant characteristics, such as frequency, receiver's height, maximum allowable path loss rate and the surrounding environment (large city or medium city, urban area or rural area) of the mobile client. Thus, the mobile device can be simulated either in coverage or out of coverage from the server (base station) according to the distance between the server and the mobile client.

Test Scenario	TS04
Description	Services auto detection
Pre-conditions	1. Input the characteristics for servers, e.g. frequency, effective height and maximum allowable path loss. 2. Pre-defined context data to put the mobile client in or out of the range of servers
Action	Start the context simulator and test system, and check the connection between the mobile client and servers.
Post-conditions	The radio range of servers (base stations) should be varied according to varied pre-conditions. The mobile client can move in and out of the server's range, and detect the service

	when it is in the range of the server.

- *Concurrent user requests.* This scenario is used to simulate the situation in which multiple users request the service simultaneously. The dynamic creation of parallel test components in the TTCN-3 enables our test framework to create a number of virtual mobile users, which can then send service requests at the same time and make load testing to the server. Therefore, we can evaluate the server's performance by checking the response time caused by the requests from a different number of virtual clients.

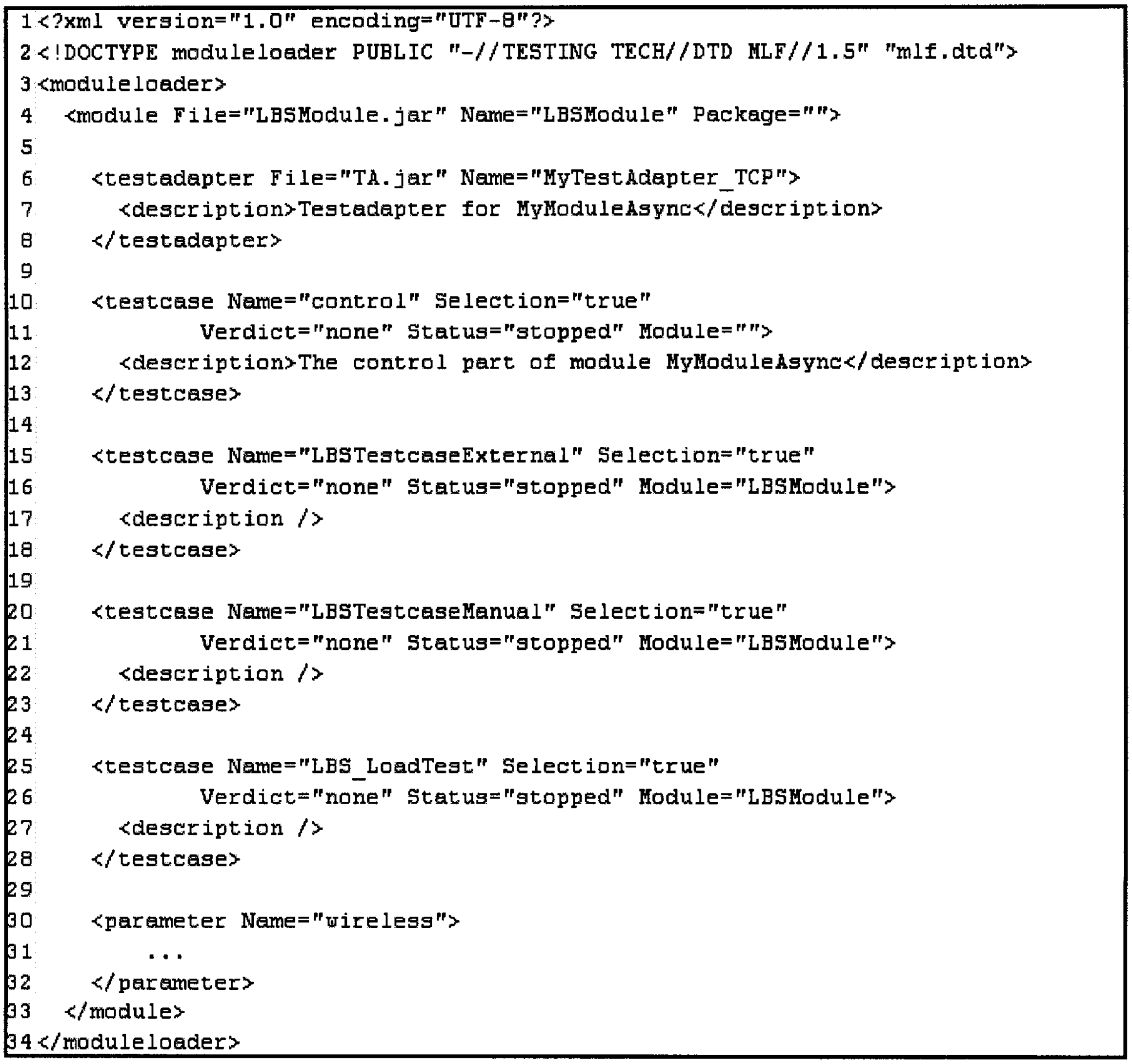
Test Scenario	TS05
Description	Concurrent user requests
Pre-conditions	1. Input the number of virtual mobile users 2. Disable the SWANS wireless simulator
Action	Start the context simulator and the test system, send service requests from virtual mobile clients and monitor the response time from the server.
Post-conditions	The virtual user receives the requested context information, and the server's performance (response time) varies according to the increasing number of virtual users.

5.5 Test Generation

The interactions of the test process consist of the test generation, the test execution and evaluation. The test generation starts from the user interface of the test system and includes the generation of test definition, test configuration, test modules, test cases as well as test data.

5.5.1 Test Definition

As mentioned before, we use the TTCN-3 to Java complier TTthree [Testing Technologies 04] as the automated TTCN-3 testing tool in our test environment. TTthree contains an XML to TTCN-3 conversion tool (*uTTman*) and can generate a test definition through one module load file (MLF). This module loader file is a XML file, responsible for performing test parameterization and configuring test cases as well as test adapters.



```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE moduleloader PUBLIC "-//TESTING TECH//DTD MLF//1.5" "mlf.dtd">
<moduleloader>
 <module File="LBSModule.jar" Name="LBSModule" Package="">

  <testadapter File="TA.jar" Name="MyTestAdapter_TCP">
   <description>Testadapter for MyModuleAsync</description>
  </testadapter>

  <testcase Name="control" Selection="true"
      Verdict="none" Status="stopped" Module="">
   <description>The control part of module MyModuleAsync</description>
  </testcase>

  <testcase Name="LBSTestcaseExternal" Selection="true"
      Verdict="none" Status="stopped" Module="LBSModule">
   <description />
  </testcase>

  <testcase Name="LBSTestcaseManual" Selection="true"
      Verdict="none" Status="stopped" Module="LBSModule">
   <description />
  </testcase>

  <testcase Name="LBS_LoadTest" Selection="true"
      Verdict="none" Status="stopped" Module="LBSModule">
   <description />
  </testcase>

  <parameter Name="wireless">
    ...
  </parameter>
 </module>
</moduleloader>
```

Figure 5-2: The Module Load File of TTCN-3

In figure 5-2, we define the module load file in our case study, in which the <testadapter> tag references a test adapter, which should be used for test execution. It also defines the name (MyTestAdapter_TCP) of a Java file, which is stored in the JAR file (TA.jar) and contains the implementation of TTCN-3 test cases.

In addition, four test cases (Control, LBSTestcaseExternal, LBSTestcaseManual and LBSLoadTest) are defined in the module (line 10- line 28). In the attributes of test cases, the *Selection* attribute sets the selection state of the respective test case or test group. Possible values are true and false. Only selected test cases can be executed. The Verdict attribute stores the current test verdict. Possible verdict values are: **none, fail, pass, inconc** and **error**. When a test case is instantiated, its local verdict object is created and set to the value **none**. The **fail/pass** means the failure/pass of the test case and **inconc** means an inconclusive verdict. The **error** verdict is a result of an executed test case operation and indicates that a test case (i.e. run-time) error has occurred. The *Verdict* attribute is set by the test management as a result of the test execution. The Status attribute stores the current test execution status and should only be modified by the test management. Possible values are stopped, running and error. The *Control* test case is used to control the implementation of *LBSTestcaseExternal, LBSTestcaseManual and LBSLoadTest* test cases, which will be discussed in the later chapter.

There are eight parameters defined with the <parameter> tag (line 30-line 32). They are initiated at the beginning of the test and include latitude, longitude, LoadTimes, packetLossRate, serverIP_ADDR, ExternalDB, wireless, LoadTesting and MAX_Vuser (See Appendix E for the definition and interface of all parameters).

The *latitude* and *longitude* parameters are the coordinator of the mobile phone. For brevity, we treat the latitude and longitude as the integer type, and ignore other related GPS information, e.g. timestamp and speed, in our test system. These parameters should be input by the tester for the manual testing. Alternatively, the coordinator data can be provided from an external database (MySQL in our case), or a Context Toolkit.

The *LoadTimes* parameter defines the execution times for the test case. The tester can enter the load times from the user interface, which defines the number of execution times for the test case and is used to undertake load testing on specific components, i.e. the server application.

The *packetLossRate* parameter is used to set up the packet loss rate for the test data transferred in the simulated wireless network. Therefore, it is very straightforward for the tester to adjust this rate to create different wireless network traffic situations and create test conditions for corresponding tests.

The *serverIP_ADDR* parameter is used to define the IP address for the system under test (SUT). Eventually, the administrator can take functional testing to the remote SUT by deploying the test adapter separately. In our case, we set up the server and client both at same computer and use the local host address 127.0.0.1.

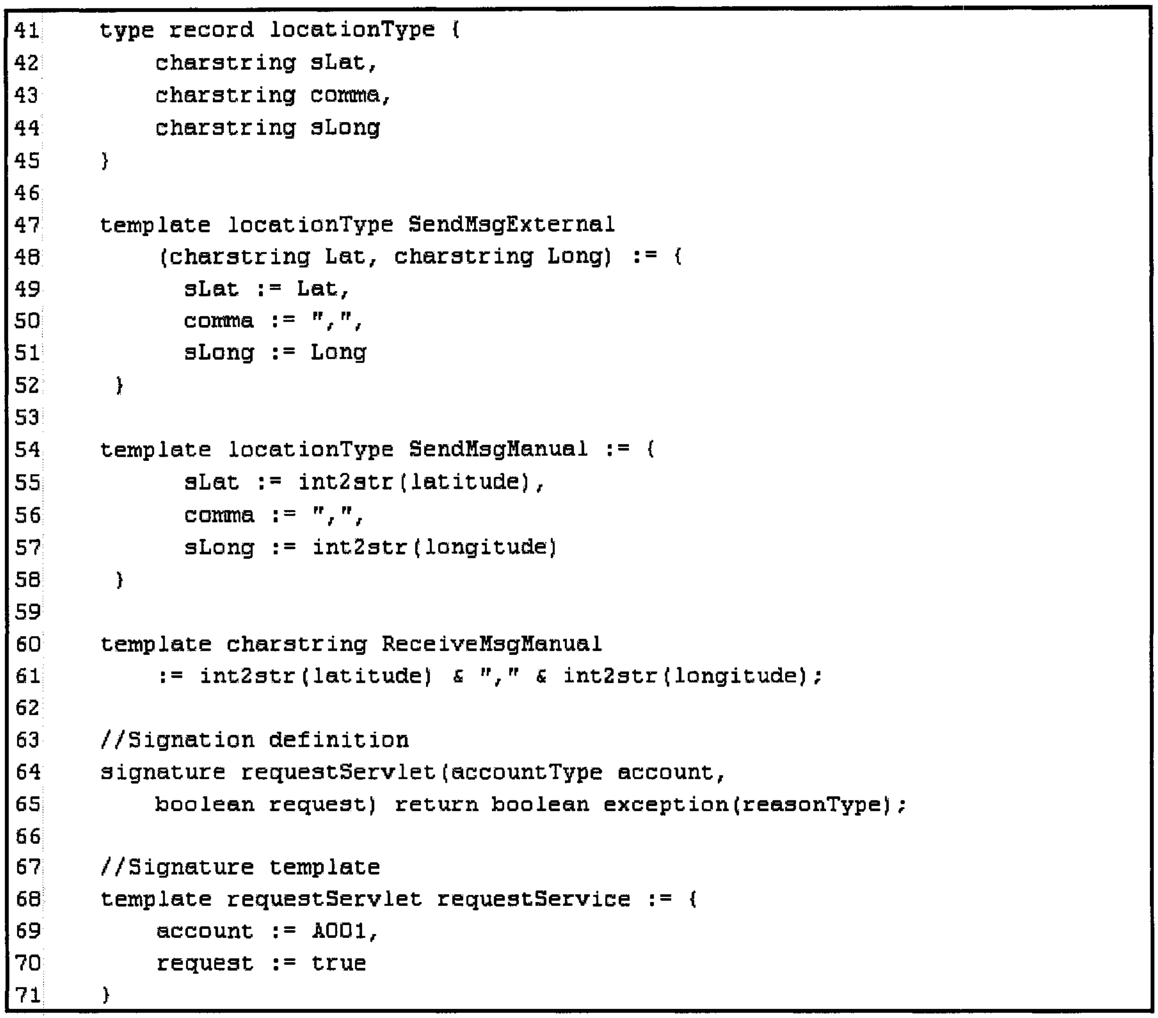
The *ExternalDB* parameter is a Boolean value, used to define the source of the test data. If it is true, the test data (latitude and longitude) will be provided automatically from an external database. Otherwise, the users have to manually input the test data from the user interface.

The *wireless* parameter is used to control the execution of wireless simulator. In our test system, it is possible to run the test case in the wired or wireless mode. Therefore, the SUT can be tested under wired network and wireless network according the system requirement.

The *LoadTesting* parameter is a Boolean value, being set up when we want to make load testing on the server and evaluate its performance. The *MAX_Vuser* parameter is used to set up the number of virtual mobile users that will be created during the load testing on the server. The virtual clients will run concurrently and send service requests to the server simultaneously.

5.5.2 Test Configuration

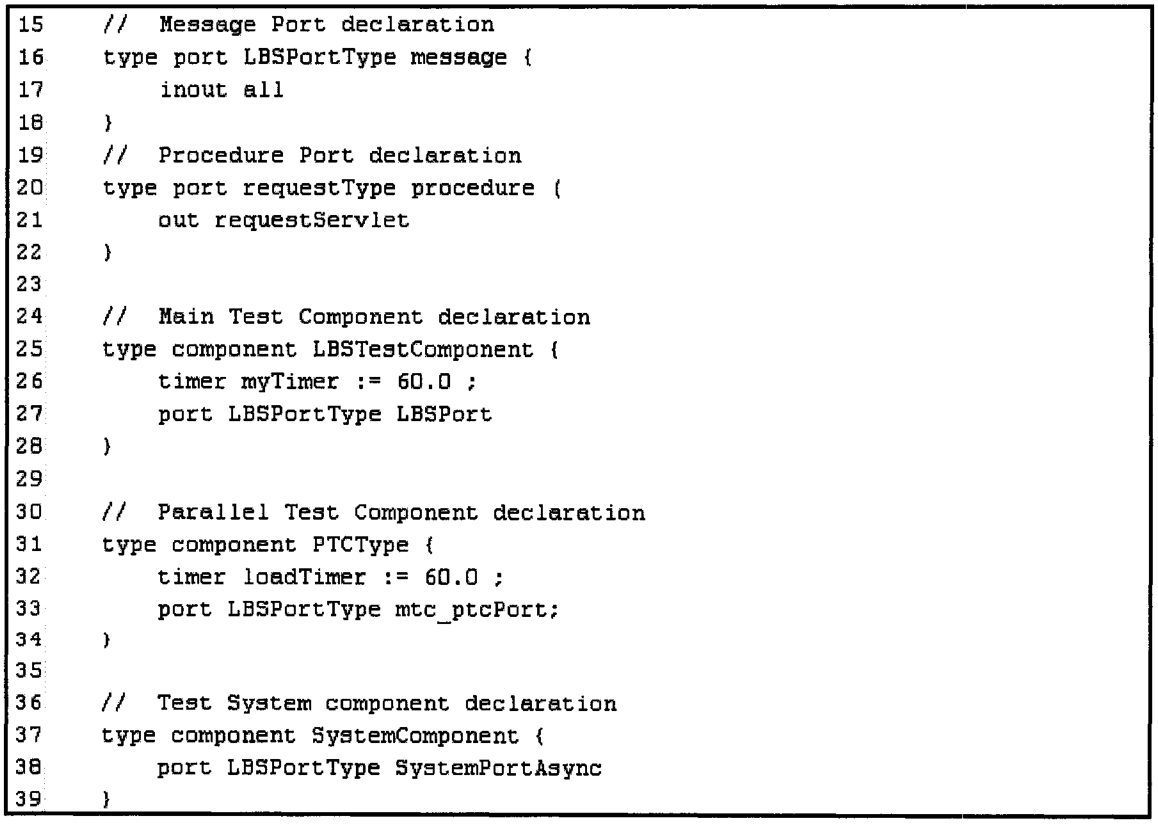
After the test definition in the module load file (MLF), test configuration need to be developed in the abstract test suit (ATS). In TTCN-3, the ATS can be represented in various formats, including the *tabular presentation format for TTCN-3* (TFT), the *graphical presentation format for TTCN-3* (GFT) (See Appendix C/D for the example of TFT and GFT) and *TTCN-3 core language*. In our test system, we use the TTCN-3 core language presentation format in the abstract test suit. The test configuration consists of data type, procedure signature, test data, port & component definition and external function definition.



```
type record locationType {
    charstring sLat,
    charstring comma,
    charstring sLong
}

template locationType SendMsgExternal
    (charstring Lat, charstring Long) := {
      sLat := Lat,
      comma := ",",
      sLong := Long
 }

template locationType SendMsgManual := {
      sLat := int2str(latitude),
      comma := ",",
      sLong := int2str(longitude)
 }

template charstring ReceiveMsgManual
    := int2str(latitude) & "," & int2str(longitude);

//Signation definition
signature requestServlet(accountType account,
    boolean request) return boolean exception(reasonType);

//Signature template
template requestServlet requestService := {
    account := A001,
    request := true
}
```

Figure 5-3: The Data Type and Template Definitions

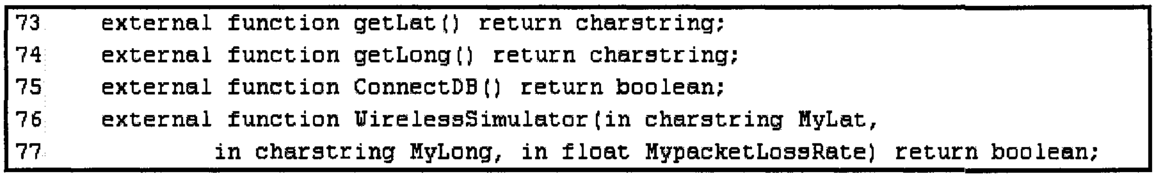
For testing the mobile service tracking system (MST), the information exchanged among the test components and between the test components and the SUT has to be defined. As shown in figure 5-3, the message-based communication and the procedure-based communication are both used among the test components in our test case. While the *locationType* is defined for the message-based communication, the *requestServlet* is employed for the procedure-based communication. In addition, the test data in our test system is represented in several templates, in which *SendMsgExternal* template is the test data defined for the *LBSTestcaseExternal* test case, which is used to perform tests with an automatic data source (via external database). The *SendMsgManual* and *ReceiveMsgManual* templates are test data definitions for the *LBSTestcaseManual* test case, used to implement the test case with manual input test data. The *requestService* template is defined for the *LBSLoadTest* test case, in which the Servlet application in the server is invoked by the procedure calls from the virtual users.



```
//  Message Port declaration
type port LBSPortType message {
    inout all
}
//  Procedure Port declaration
type port requestType procedure {
    out requestServlet
}

//  Main Test Component declaration
type component LBSTestComponent {
    timer myTimer := 60.0 ;
    port LBSPortType LBSPort
}

//  Parallel Test Component declaration
type component PTCType {
    timer loadTimer := 60.0 ;
    port LBSPortType mtc_ptcPort;
}

//  Test System component declaration
type component SystemComponent {
    port LBSPortType SystemPortAsync
}
```

Figure 5-4: The Port and Component Type Declaration

As shown in figure 5-4, the *LBSPortType* port type is a message-based port, used for communication by means of message exchange. The *requestType* port type is a Procedure-based port, used in the function calls on the server. The *SystemComponent* component type is used to perform the communication between the test component and SUT, and the *LBSTestComponent* component type is the main test component, defined for the communication among test components. The PTCType is the parallel test component dynamically created by the main test component and it is used to connect remote test components (Servlet applications) in the test of distributed system.



```
external function getLat() return charstring;
external function getLong() return charstring;
external function ConnectDB() return boolean;
external function WirelessSimulator(in charstring MyLat,
        in charstring MyLong, in float MypacketLossRate) return boolean;
```

Figure 5-5: The External Function Declaration

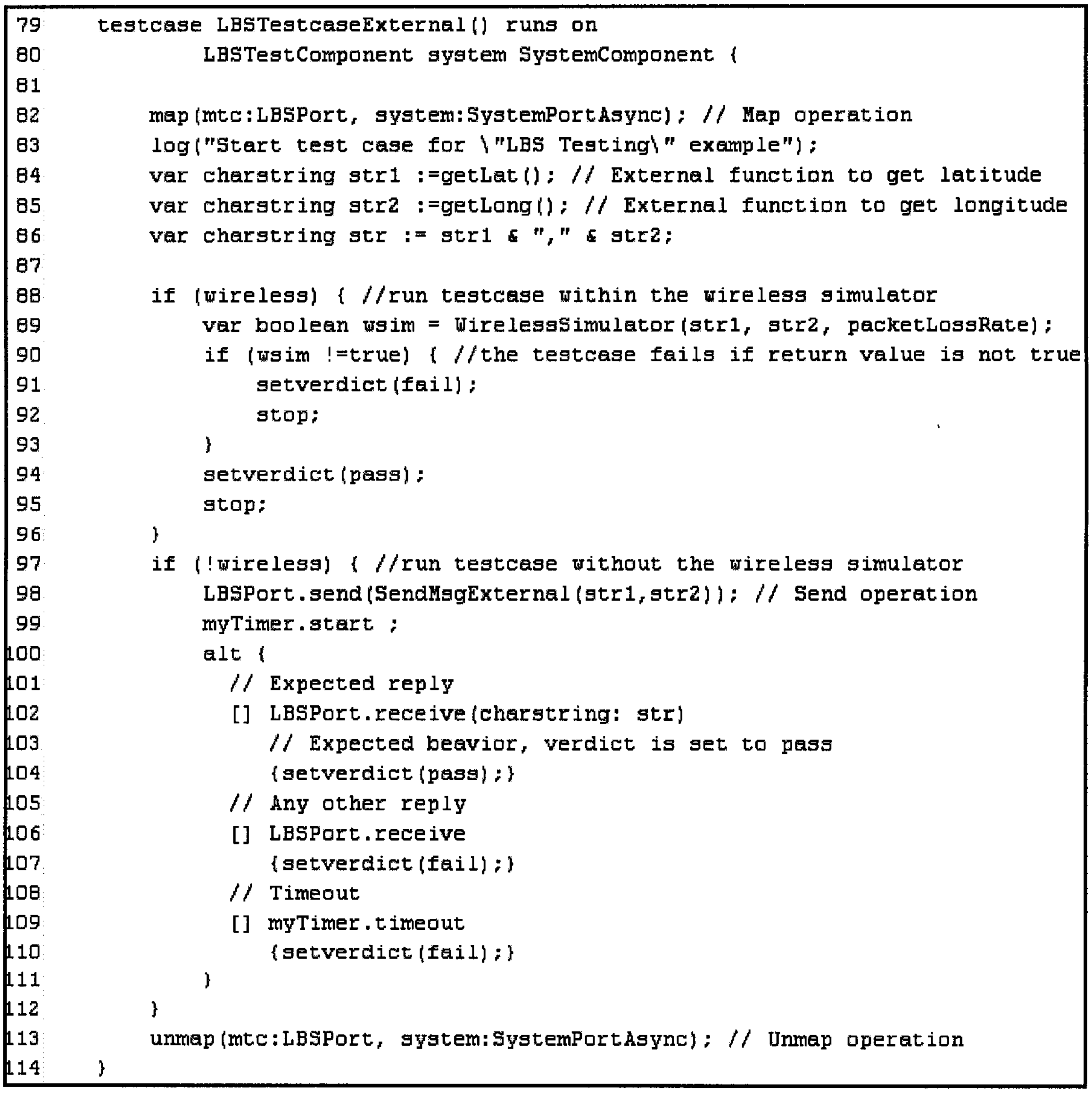
Our test system defines four external functions, as shown in figure 5-5. Among these functions, *ConnectDB()* is responsible for building the connection between the external database and the test system, implemented by JDBC interface. After the successful connection with the external database, the *getLat()* and *getLong()* functions enable the test system to invoke the external database and provide the test system with predefined test data. The *WirelessSimulator* function is invoked by the test system to run the test case within the SWANS wireless simulator and return the test result to the TTCN-3 test system.

5.5.3 Test Module

After test definition and configuration, test cases need to be developed to cover designed test scenarios. In our test system, we developed three generic test modules: LBSTestcaseExternal, LBSTestcaseManual and LBSLoadTest. Combined with other input parameters (LoadTimes, ExternalDB and packetLossRate, etc.), these generic test modules can generate various test cases to cover all test scenarios (TS01-TS05) and test strategies defined in section 5.4.

5.5.3.1 LBSTestcaseExternal Test Module

The first generic test module LBSTestcaseExternal enables us to test the mobile service tracking system (MST) with test data provided by an external database, as shown in figure 5-6.



```
testcase LBSTestcaseExternal() runs on
        LBSTestComponent system SystemComponent {

    map(mtc:LBSPort, system:SystemPortAsync); // Map operation
    log("Start test case for \"LBS Testing\" example");
    var charstring str1 :=getLat(); // External function to get latitude
    var charstring str2 :=getLong(); // External function to get longitude
    var charstring str := str1 & "," & str2;

    if (wireless) { //run testcase within the wireless simulator
        var boolean wsim = WirelessSimulator(str1, str2, packetLossRate);
        if (wsim !=true) { //the testcase fails if return value is not true
            setverdict(fail);
            stop;
        }
        setverdict(pass);
        stop;
    }
    if (!wireless) { //run testcase without the wireless simulator
        LBSPort.send(SendMsgExternal(str1,str2)); // Send operation
        myTimer.start ;
        alt {
          // Expected reply
          [] LBSPort.receive(charstring: str)
             // Expected beavior, verdict is set to pass
             {setverdict(pass);}
          // Any other reply
          [] LBSPort.receive
             {setverdict(fail);}
          // Timeout
          [] myTimer.timeout
             {setverdict(fail);}
        }
    }
    unmap(mtc:LBSPort, system:SystemPortAsync); // Unmap operation
}
```

Figure 5-6: The LBSTestcaseExternal Test Module

The test module starts with a map operation (line 82) to set up the test system and connect the master test component (MTC) with the system under test (SUT). Line 84 and line 85 define two variables which store location data provided from the external database by invoking external functions (getLat() and getLong()).

Afterwards, the test module will be run within the SWANS wireless simulator if the *wireless* parameter is set to true. All current test data and relative conditions, i.e. *packet loss rate*, will be transferred to the WirelessSimulator() external function, in which test data will be sent to the SUT and the received result will be verified. If the result is true, the WirelessSimulator() will return true and the test case is then passed, otherwise it is failed.

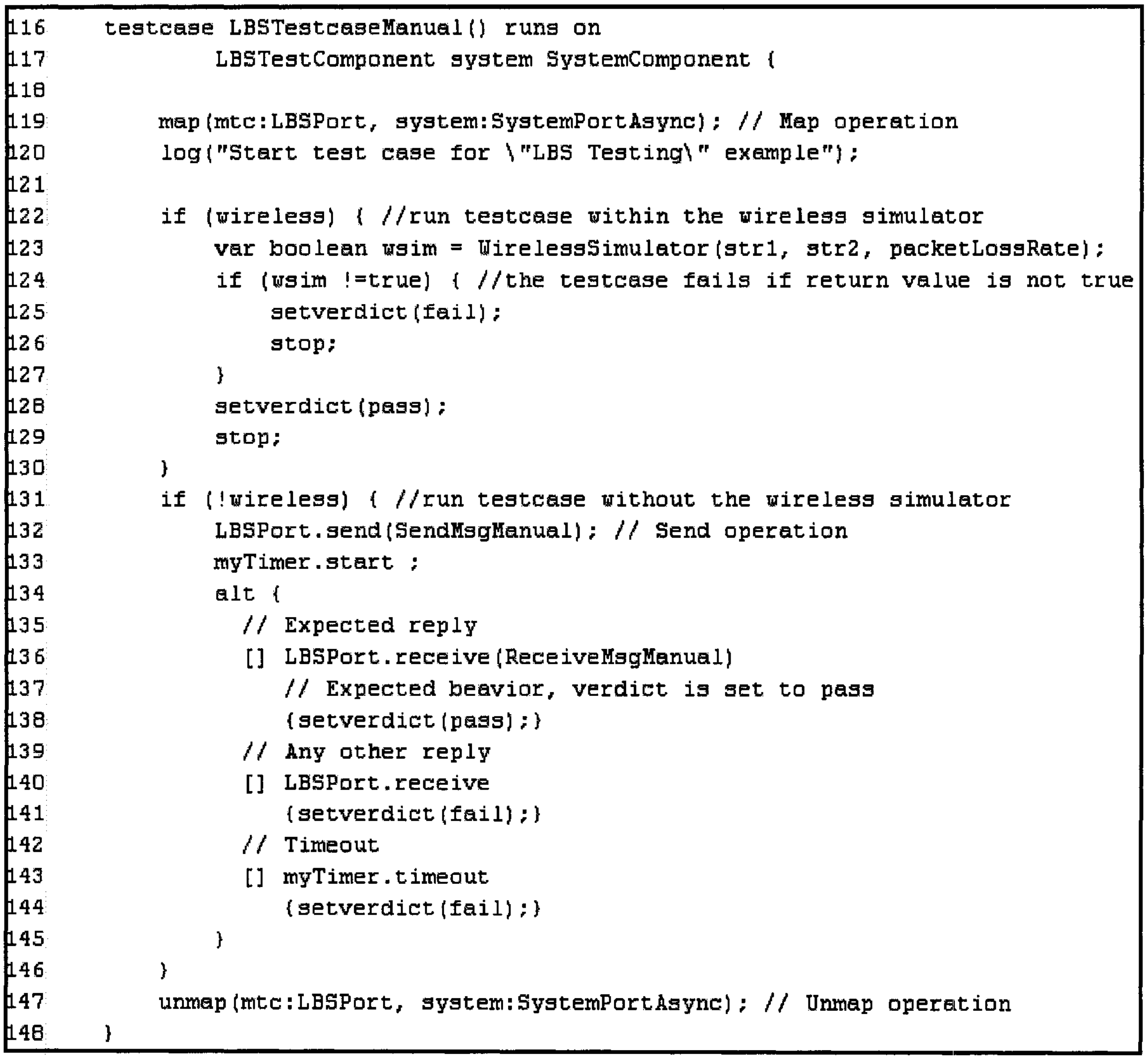
In addition, if the *wireless* parameter is set to false, the test module will be run without the SWANS wireless simulator. Similarly, all the test data (location information) will be sent to the SUT and the received data will be compared with the expected result. If they are same, the test case will be passed, vice versa.

Finally, the *unmap* operation is executed to break the connection between the MTC and the SUT.

5.5.3.2 LBSTestcaseManual Test Module

Similar to the test module *LBSTestcaseExternal*, the test case *LBSTestcaseManual* in figure 5-7 is responsible for testing the mobile service tracking system (MST) with manual input test data. After the tester inputs the location data from the GUI, it will be stored in *sLat* and *sLong* variables, and then deployed to the wireless simulator via *WirelessSimulator()*. The test data will be received and verified by the system under test (SUT) within SWANS and the corresponding test result will be sent back to the TTCN-3 test system.

Alternatively, the user can perform specific functional testing to the LBA and server application without the selection of a wireless simulator. In this situation, the whole SUT is running under a wired network. The test data, stored in *SendMsgManual* variable (see figure 5-3 for the data definition), will be sent directly to the client application and the received data will then be compared with expected result to get the test result (pass/fail).



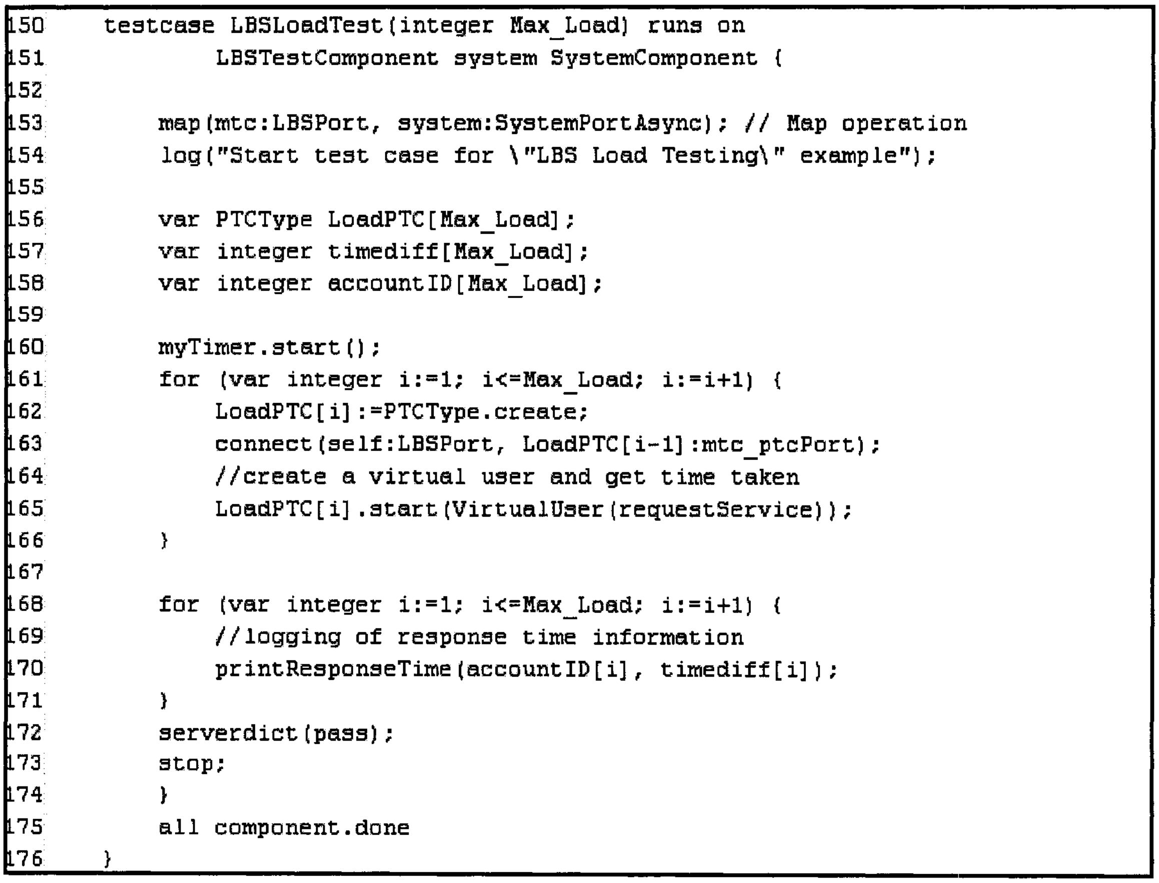
```
testcase LBSTestcaseManual() runs on
        LBSTestComponent system SystemComponent {

    map(mtc:LBSPort, system:SystemPortAsync); // Map operation
    log("Start test case for \"LBS Testing\" example");

    if (wireless) { //run testcase within the wireless simulator
        var boolean wsim = WirelessSimulator(str1, str2, packetLossRate);
        if (wsim !=true) { //the testcase fails if return value is not true
            setverdict(fail);
            stop;
        }
        setverdict(pass);
        stop;
    }
    if (!wireless) { //run testcase without the wireless simulator
        LBSPort.send(SendMsgManual); // Send operation
        myTimer.start ;
        alt {
          // Expected reply
          [] LBSPort.receive(ReceiveMsgManual)
             // Expected beavior, verdict is set to pass
             {setverdict(pass);}
          // Any other reply
          [] LBSPort.receive
             {setverdict(fail);}
          // Timeout
          [] myTimer.timeout
             {setverdict(fail);}
        }
    }
    unmap(mtc:LBSPort, system:SystemPortAsync); // Unmap operation
}
```

Figure 5-7: The LBSTestcaseManual Test Module

5.5.3.3 LBSLoadTest Test Module

The *LBSLoadTest* test module is developed specifically for server load testing. As mentioned in section 4.6.4, the SWANS wireless simulator is not included in the server side testing and thus is not used in this test module.

As shown in figure 5-8, the test module defines the array *LoadPTC*[] (line 156), which is used to load the parallel test components (PTCs) dynamically created by the main test component (line162- line 165). These PTCs can act as virtual mobile users by starting the function *VirtualUser*(), associated with the test data (*requestService* signature template, see figrure 5-3 for the template definition). The number of virtual users is determined by the *Max_Load* variable, which can be input by the tester from the user interface. The *VirtualUser* is responsible for the interaction with the server by sending service requests, receiving the requested service (location information) and recording the corresponding response time. After all virtual users send requests to the server, the response time and user account ID is logged in the system and printed via the function *printResonseTime*() (line 170).

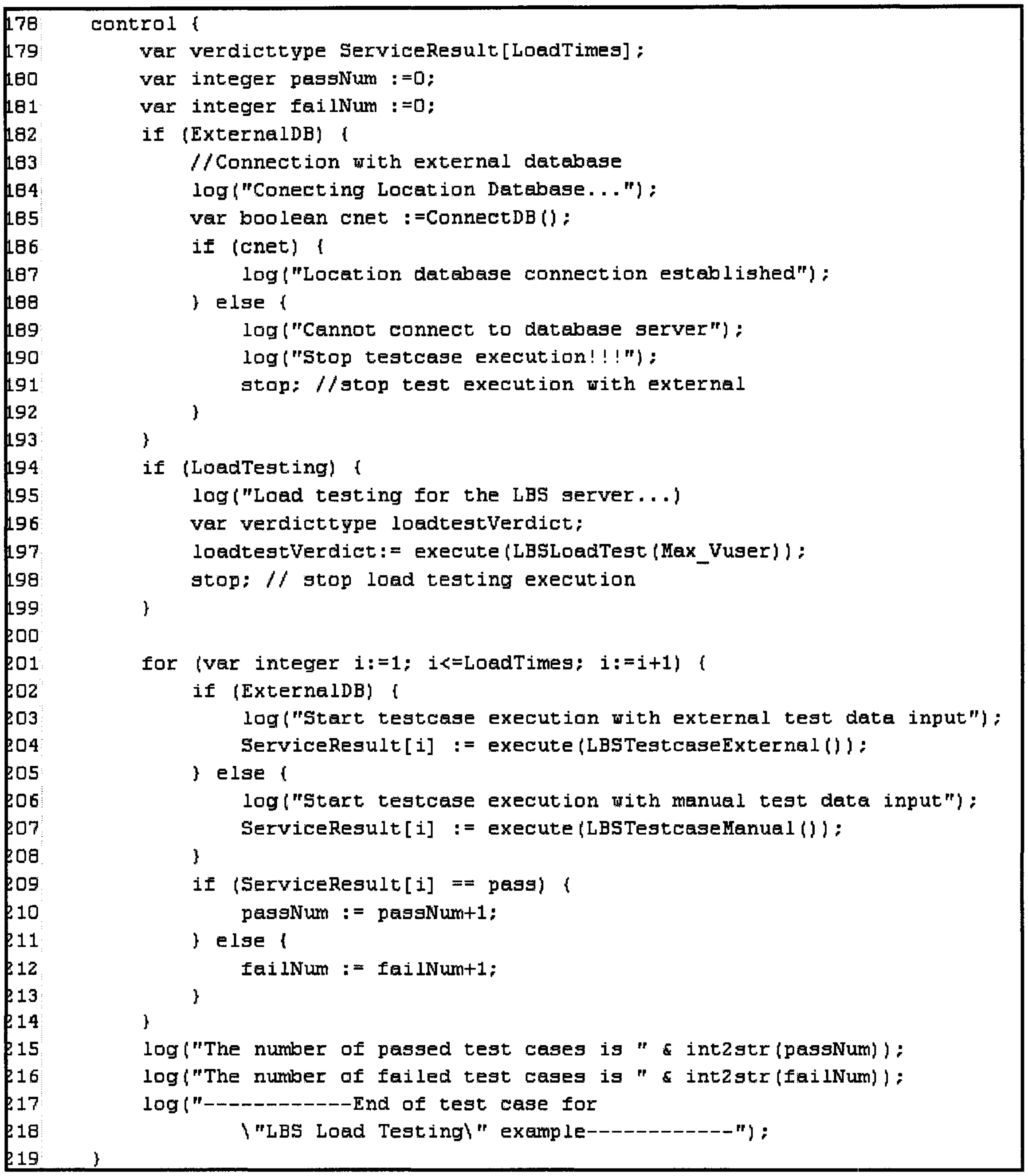


```
150   testcase LBSLoadTest(integer Max_Load) runs on
151           LBSTestComponent system SystemComponent {
152
153       map(mtc:LBSPort, system:SystemPortAsync); // Map operation
154       log("Start test case for \"LBS Load Testing\" example");
155
156       var PTCType LoadPTC[Max_Load];
157       var integer timediff[Max_Load];
158       var integer accountID[Max_Load];
159
160       myTimer.start();
161       for (var integer i:=1; i<=Max_Load; i:=i+1) {
162           LoadPTC[i]:=PTCType.create;
163           connect(self:LBSPort, LoadPTC[i-1]:mtc_ptcPort);
164           //create a virtual user and get time taken
165           LoadPTC[i].start(VirtualUser(requestService));
166       }
167
168       for (var integer i:=1; i<=Max_Load; i:=i+1) {
169           //logging of response time information
170           printResponseTime(accountID[i], timediff[i]);
171       }
172       serverdict(pass);
173       stop;
174       }
175       all component.done
176   }
```

Figure 5-8: The LBSLoadTest Test Module

5.5.3.4 Control Test Module

The execution of above three generic test modules is controlled with the *control* test module as shown in figure 5-9.



```
control {
    var verdicttype ServiceResult[LoadTimes];
    var integer passNum :=0;
    var integer failNum :=0;
    if (ExternalDB) {
        //Connection with external database
        log("Conecting Location Database...");
        var boolean cnet :=ConnectDB();
        if (cnet) {
            log("Location database connection established");
        } else {
            log("Cannot connect to database server");
            log("Stop testcase execution!!!");
            stop; //stop test execution with external
        }
    }
    if (LoadTesting) {
        log("Load testing for the LBS server...)
        var verdicttype loadtestVerdict;
        loadtestVerdict:= execute(LBSLoadTest(Max_Vuser));
        stop; // stop load testing execution
    }

    for (var integer i:=1; i<=LoadTimes; i:=i+1) {
        if (ExternalDB) {
            log("Start testcase execution with external test data input");
            ServiceResult[i] := execute(LBSTestcaseExternal());
        } else {
            log("Start testcase execution with manual test data input");
            ServiceResult[i] := execute(LBSTestcaseManual());
        }
        if (ServiceResult[i] == pass) {
            passNum := passNum+1;
        } else {
            failNum := failNum+1;
        }
    }
    log("The number of passed test cases is " & int2str(passNum));
    log("The number of failed test cases is " & int2str(failNum));
    log("------------End of test case for
            \"LBS Load Testing\" example------------");
}
```

Figure 5-9: The Execution Control Test Module

The control module starts from the connection with the external database. The test cases will be stopped if any error occurs when connecting to the external database. Afterwards, the *LBSLoadTesting* test module will be invoked if the tester selects the load test scenario from the user interface. Also, the *LBSTestcaseExternal* or *LBSTestcaseManual* generic test modules will be selected according the value of the *ExternalDB* parameter, and executed in number of times defined by external constant *LoadTimes*. The test result of every test case will be recorded and logged in a XML file. Finally, the test system will report the number of test cases that are passed and failed.

5.5.4 Test Case and Test Data

As shown in table 5-2, table 5-3 and table 5-4, we generate the following test cases (TC001 – TC007) according to the above three generic test modules and test scenarios we defined in section 5.4.

Test Case	**TC001**	**TC002**	**TC003**
Test Scenario	TS01	TS01	TS02
LBSTestcaseExternal	Invalid	Yes	Yes
LBSTestcaseManual	Yes	Invalid	Invalid
Load Times	1	1	10
Packet Loss Rate	0.0	0.0	0.0
External Database	Invalid	Yes	Yes
SWANS WS	Yes	Yes	Yes
Test Data (latitude, longitude)	(Null, 200)	(100,200) from Database	Correct patch data from Database
Expected Result	Error coordinator data	(100,200)	All of correct test data received
Comment	Incorrect context data with manual	Correct context data with auto test data	Fast changing context information

	test data input	input	

Table 5-2: Test Case List (1)

Test Case	**TC004**	**TC005**	**TC006**
Test Scenario	TS03	TS03	TS03
LBSTestcaseExternal	Yes	Yes	Yes
LBSTestcaseManual	Invalid	Invalid	Invalid
Load Times	10	10	10
Packet Loss Rate	0.7	1.0	0.0
External Database	Yes	Yes	Yes
SWANS WS	Yes	Yes	Yes
Test Data (Latitude, Longitude)	Correct data from Database	Correct data from Database	Correct data from Database
Expected Result	30% of test data received	All of test data lost	Some correct test data received
Comment	Weak network performance, 70% packets lost	Mobile client is out of coverage of wireless network	Random success of packets transition after setting random success rate.

Table 5-3: Test Case List (2)

TC007 is a test case for the server side load testing. It is used to evaluate the server performance under various numbers of concurrent virtual users. The response time represents the time that one complete transaction takes under certain conditions. It can be affected by different factors, including the size of the database, number of concurrent users and the characteristics of running environment (hardware, network and software configuration). For brevity, we only consider the size of database and the number of concurrent users in this test case, as shown in table 5-4.

Test Case ID	**TC007**

Test Scenario ID	TS05
LBSLoadTest	Yes
SWANS WS	Invalid
External Database	Yes, 1000 records (location context)
Test Data (MAX_Vuser)	1, 3, 5, 7, 9, 10
Expected Result	Corresponding response time in seconds
Comment	It should vary according to different number of virtual users

Table 5-4: Test Case List (3)

These test cases are created for specific functions, covering most of test scenarios defined earlier, including Incorrect context information (TS01), Fast changing context information (TS02), No corresponding network performance (TS03) and Concurrent user requests (TS05). Apart from these basic test cases, we can also reuse the test modules defined earlier and combine these test cases to create more comprehensive test cases for the mobile service tracking (MST) system. After the combination of above test cases, we generate 33 test cases (CA01-CA33), shown in Appendix H, I, J, to cover 16 test scenarios (SA01-SA16) defined for the following use cases in the MST system:

1. Activation of the MST
2. Deactivation of the MST
3. Directory service - Where am I?

The test scenario *Service auto detection* (TS04) cannot be covered with above test cases because of test environment it requires. In order to simulate this scenario, we need to build two server applications and one client application running on the SWANS wireless simulator. Each server connects with one Base Station and provides the mobile client with one independent service, e.g. weather service and news service. Therefore, according to the distance between the server and the mobile client, the corresponding service may be available or unavailable to the mobile client when he or she is moving between two servers. Specifically, there are five test cases generated to cover the test scenario TS04, as show in the table 5-5.

Test Case	Test Scenario	Pre-conditions	Expected result
TC008	TS04	D1 <= Rangel && D2 > Range2	Mobile client is in the range of server #1
TC009	TS04	D1 <= Rangel && D2 <= Range2 && D1<=D2	Mobile client is in the range of server #1
TC010	TS04	D1 <= Rangel && D2 <= Range2 && D1>D2	Mobile client is in the range of server #2
TC011	TS04	D1 > Rangel && D2 <= Range2	Mobile client is in the range of server #2
TC012	TS04	D1 > Rangel && D2 > Range2	Mobile client is out of the range of two servers
D1/D2 – the distance between the mobile client and the server (base station) #1 / #2 Rangel/ Range2 – the radio range for the server (base station) #1 / #2			

Table 5-5: The Test Case List for Test Scenario TS04

As mentioned in section 4.5.3.2, we introduced the Hata computation model [Hata 80] in the SWANS wireless simulator. Thus, the server (base station)'s wireless communication range (maximum distance between the transmitter and receiver) is determined by the following parameters:

- Frequency of the Base Station (150 ... 1500 MHz)
- Maximum allowable path loss (dB)
- Antenna height of the transmitter (base station) (30...200 m)
- Antenna height of the receiver (mobile phone) (1...10 m)
- City size (Medium sized city / Large city)
- Surrounding environment (Urban area, Suburban area or Open rural area)

Each parameter can be set up from the interface in the SWANS wireless simulator. In our case study, we use the following initial parameters for two base stations.

Base Station	City Size	Surrounding Environment	Frequency (MHz)	Maximum allowable path loss (dB)	Transmitter Height (m)	Receiver Height (m)	Range (KM)
BS1	Medium	Suburban	150	135	200	2	121
BS2	Medium	Suburban	150	135	200	2	121

Table 5-6: Initial Parameters for the Base Station

As shown in table 5-6, the range of base station is 121 KM for both the BS1 and the BS2. According to the radio ranges of two base stations, we design a set of following test data stored in the external database, as shown in table 5-7. By means of loading the test data (mobile client's coordinators) sequentially from the database, it is possible for the test system to simulate the movement of mobile client in/out of the range of server #1 and server #2.

	Server#1	Server#2	Mobile Client						
Coordinator	C01	C02	C03	C04	C05	C06	C07	C08	C09
Latitude	300	500	160	180	200	250	300	350	380
Longitude	100	100	50	50	50	50	50	50	50
	Mobile Client								
Coordinator	C10	C11	C12	C13	C14	C15	C16	C17	C18
Latitude	400	410	430	480	550	600	620	640	660
Longitude	50	50	50	50	50	50	50	50	50

Table 5-7: The Test Data for Test Scenario TS04

5.6 Test Execution and Result Analysis

Through the execution of all the test cases (TC001-TC0012), we can get the test results as shown in following table:

	TC001	TC001	TC001
Test Data	(Null, 200)	(100,200)	(100,200)
Expected Result	Error data...	(100,200)	(100,200)
Actual Result	Error data...	Error data...	(100,200)
Test Result	Pass	Fail	Pass
Notes	Manual input the test data from the user interface		

Table 5-8: The Test Result of Test Case TC001

The table 5-8 shows the test results of TC001, in which two of them are passed and one is failed. The first test result means the system under test (SUT) contains the error handling mechanism to handle the error data. When this function is disabled, we can get the second test result (fail). With the manual input, our test framework enables the tester to take negative tests by typing in specific mutants.

	Test Data	Expected Result	Actual Result	Test Result
TC002	(100,200)	(100,200)	(100,200)	Pass
	(Null,200)	Error data...	Error data...	Pass
Notes	Test data is automatically loaded from the database			

Table 5-9: The Test Result of Test Case TC002

The table 5-9 shows the test results of TC002, in which the test data is loaded from the database automatically. By doing this, the tester can effectively implement the tests with predefined test data.

	Test Data	Expected Result	Actual Result	Test Result
TC003	(100,30)	(100,30)	(100,30)	Pass

	(150,30)	(150,30)	(150,30)	Pass
	(200,30)	(200,30)	(200,30)	Pass
	(240,30)	(240,30)	(240,30)	Pass
	Repeat...	Repeat ...	Repeat ...	Repeat ...
Comments	4 sets of test data are retrieved from the database and the test case is executed for 10 times to simulate TS02 (fast changing context information)			

Table 5-10: The Test Result of Test Case TC003

The table 5-10 shows the test results of TC003, in which 4 sets of test data are loaded in the test for 10 times in order to simulate the scenario of fast changing context information. As shown from the table, the client received exact same data from the server. None of test cases are failed and the whole test case TC003 is passed.

	Test Data	Expected Result	Actual Result	Test Result
TC004	(240,30)	(240,30)	Error data...	Fail
	(240,30)	(240,30)	Error data...	Fail
	(240,30)	(240,30)	Error data...	Fail
	(240,30)	(240,30)	(240,30)	Pass
	(240,30)	(240,30)	Error data...	Fail
	(240,30)	(240,30)	Error data...	Fail
	(240,30)	(240,30)	(240,30)	Pass
	(240,30)	(240,30)	Error data...	Fail
	(240,30)	(240,30)	(240,30)	Pass
	(240,30)	(240,30)	Error data...	Fail
Notes	After the packetLossRate is set to 0.7, 70% of packets are lost in the transaction.			

Table 5-11: The Test Result of Test Case TC004

The table 5-11 shows the test results of TC004, in which the system loads the same test data from the database for 10 times. 7 of 10 packets are lost due to the poor wireless communication. The system shows "Error data..." when the client does not receive any data from the server and the transaction time is out. The corresponding test result becomes fail.

We do not list the test results of TC005 and TC006 in this thesis, as they are similar with the results of TC004, except that the packet loss rate is randomly generated at the beginning of the test execution in TC006, and the packet loss rate is set to 1.0 in TC005. As a result, all of data is lost during the execution of TC005. Appendix F shows the snapshots of test execution for TC004 and TC005. Therefore, the test results of above test cases (TC004-TC006) indicate that our test framework is capable of validating the correctness of the system under test under varied wireless network performance.

Vuser	1	3	5	7	9	10
Response Time (sec)	1.352	3.11	5.728	10.546	14.196	15.82
	1.34	3.156	5.448	10.624	15.036	16.201
	1.572	2.843	5.168	12.576	14.301	15.954
Average	1.421	3.306	5.548	11.249	14.511	15.992

Table 5-12: The Test Result of Test Case TC007

The table 5-12 shows the test result of TC007, in which the test system generates a different number of virtual users and records the amount of time it takes for each transaction to be completed. The whole transaction starts from the service request from the virtual user and ends when the virtual user receives the response context information from the server. Figure 5-10 shows the relationship between the number of virtual users and their corresponding response time.

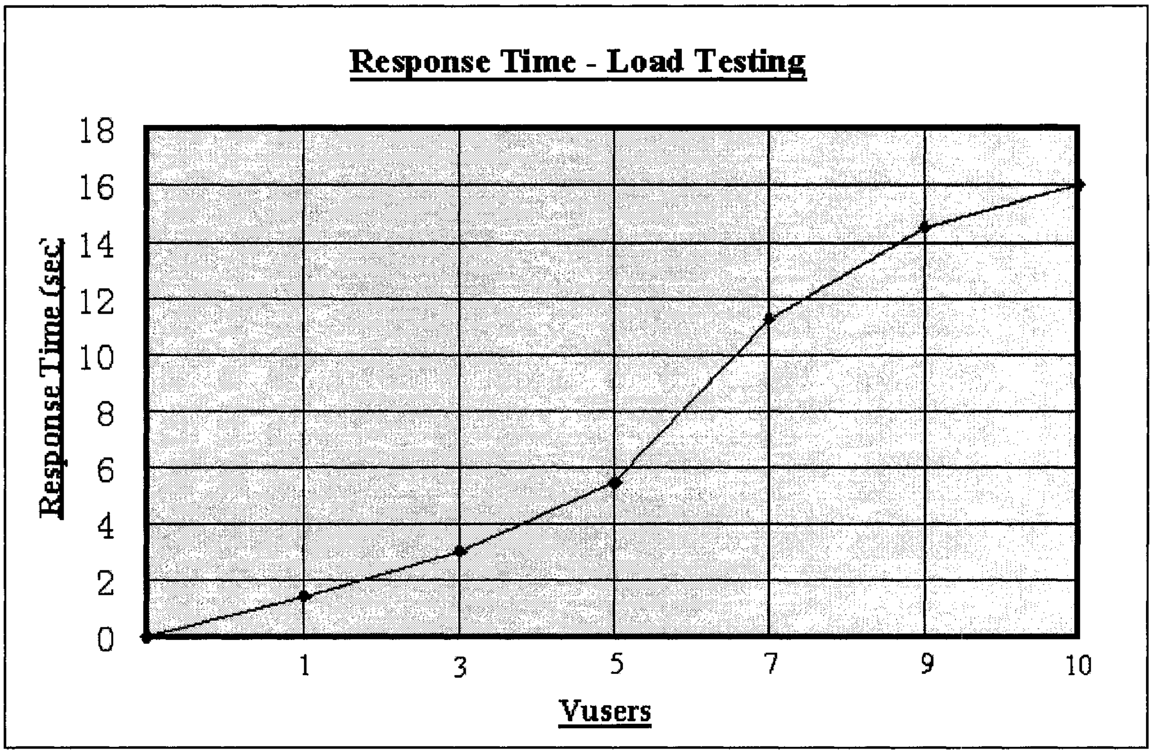


Figure 5-10: The Average Response Time with Different Virtual Users

With the load testing, our test framework can measure end-user response time, checking how long it takes for the users to perform a business process and receive a response from the server. For example, suppose that the system requires that the end users receive response to all requests within 10 seconds. From the above figure, we can see that the response time is increasing with the growth of virtual users that the system generates. When there are 7 concurrent virtual users' requests, the server response time exceeds dramatically from 5.5 seconds to 11 seconds, which means the server performance degrades significantly in terms of the response time when there have over 5 concurrent virtual users. In other words, we can check the system's capacity to determine how much excess capacity can handle without performance degradation. In addition, we can check system's reliability to determine the level of system stability under heavy or continuous work loads, i.e. over a period of weeks or months, identify bottlenecks of the system, and define system's optimal hardware configuration, i.e. CPU, memory usage, cache, adaptors, by performing load testing in our test framework. However, we do not investigate these situations in this experimental study.

Appendix G shows the test results of test cases (TC008 – TC012), which are generated from the SWANS wireless simulator interface. We can find out, that the mobile client roams from the sever #1 to the server #2 and connects with its closest server according to the distance between each server and the mobile client. For instance, when the mobile client is in the position of (350,50), it is in the range of the server #1 and detects the service from the server #1 (TC008). When it moves to the position of (410,50), it switches the connection to the server #2, as the mobile client is closer to the server #2 than to the server #1 (TC010). Therefore, the test scenario of service auto detection (TS05) can be simulated by implementing above test cases (TC008-TC012).

5.7 Interoperability and Security Testing Investigation

As mentioned in section 2.3.6, interoperability is one of the critical characteristics for location-based services (LBS) because of the heterogeneous issues involved in its environment, including different network technologies, different operation systems and programming languages, different service content vendors and location providers, etc. The location services market demands technology that subscribes to the principal of simplicity and interoperability so that these services and technologies will be widely adopted throughout the mobile commerce realm. To achieve interoperability and the portability of components within the LBS environment, the conformance between the implementation of components and the specification must be analyzed to ensure that the component's behaviour is in accordance with the specification.

From a testing viewpoint, the system under test (LBS) must be designed and developed under a specification in order to take the interoperation testing viable under our test framework. Therefore, the above case of the Mobile Service Tracking (MST) system that we use in the validation of the functional testing, the performance testing as well as the server-side testing is not suitable for being a system under test in interoperability testing because it lacks of the interoperable components.

Although there are a number of location-based services in use throughout the world today, most of them lack interoperability requirements. However, some organizations have undertaken efforts to define, develop and promote the specifications in order to deliver interoperability among the location-based services. For instance, the Location Interoperability Forum (LIF, lately OMA), founded in October 2000, is devoted to build interoperability specifications and create interoperability test concepts for location-based services [LIF 05]. The Open Location Services (OpenLS) is another program that is provided by the Open Geospatial Consortium (OGC) organization. The OpenLS [OGC 05] is devoted to the development of interface specifications that facilitate the use of location and other forms of spatial information in the wireless Internet environment. The purpose of the OpenLS is to produce open specifications for interoperable location application services that will integrate spatial data and processing resources into telecommunications and Internet services infrastructure.

Because of the lack of relevant systems under test for the LBS interoperability testing, we will demonstrate the feasibility of our test framework on the LBS interoperability testing by firstly introducing one CORBA interoperability testing framework that is implemented in TTCN-3, and then we will investigate how the OpenLS interoperability testing can be theoretically conducted by using the CORBA interoperability testing methodology.

5.7.1 CORBA Interoperability Testing

The Common Object Request Broker Architecture (CORBA) provides a set of specifications for the development of distributed object-oriented applications in heterogeneous environments. Using the Internet Inter-ORB Protocol (IIOP), a CORBA-based program from any vendor, on almost any computer, operating system, programming language, and network, can interoperate with another CORBA-based program from the same or another vendor, on almost any other computer, operating system, programming language, and network [OMG 05]. Similar to the location-based service, interoperability testing is an essential testing issue for CORBA-based programs.

Currently there are much research focusing on the interoperability testing of the CORBA-based programs [OMG 05]. Among them, Mang Li [Li+01] presented a test framework that focuses on the CORBA interoperability testing. The description languages used in this test framework are CORBA IDL, ODMG ODL and TTCN-3.

According to [Li+01], there are two different interfaces defined in the CORBA middleware platform: the horizontal interface and the vertical interface, as shown in figure 5-11. The horizontal interface is defined in the description languages, i.e. Interface Definition Languages (IDL), and it enables *portability* of the application by separating the application from the middleware. The vertical interface resides between two instances of a CORBA implementation and is defined through the General Inter-ORB Protocol (GIOP) or the Internet Inter-ORB Protocol (IIOP). The specification of the vertical interface enables the *interoperability* of CORBA-based applications.

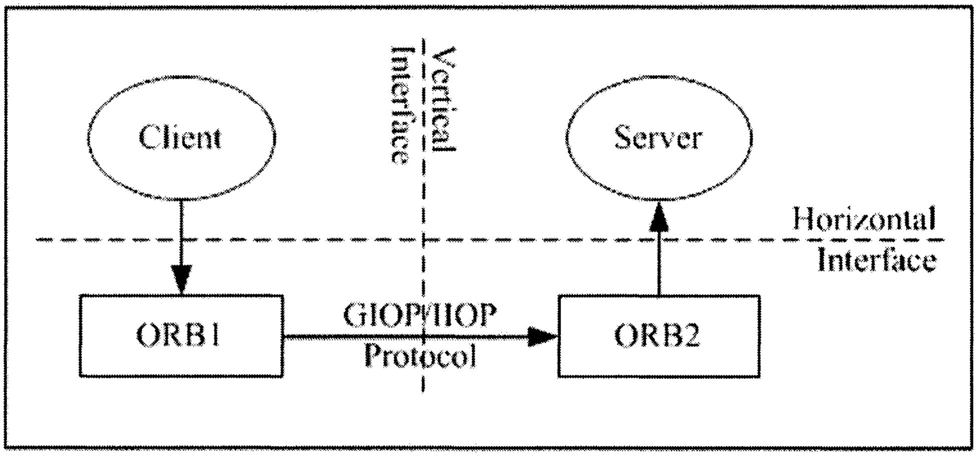


Figure 5-11: Vertical and Horizontal Interfaces in CORBA [Li+01]

Therefore, interoperability testing in CORBA-based systems is mainly the validation of the vertical GIOP/IIOP interface. For the GIOP tests, the test framework creates a test configuration with three test components in TTCN-3, as shown in figure 5-12. These test components are a test component for the vertical interface (TestClient), a test component for the horizontal interface (TestServer), and a master test component (MTC) for the overall control of the other two test components over the coordination points or ports (CPs). The TestClient and TestServer components are defined as parallel test components (PTC) in TTCN-3. The interface of the ORB under test (OUT) is defined by a separate component type definition for OUT containing the vertical (VG) and horizontal (HG) interfaces. With these well-defined interfaces, any ORB provided by different vendors can be connected to the test framework and performs interoperability testing within the test framework.

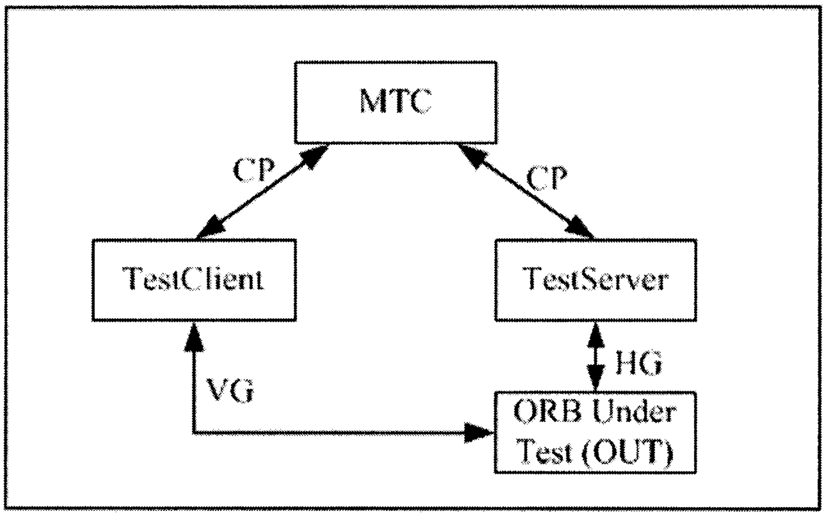


Figure 5-12: TTCN-3 Test Component Configuration for CORBA [Li+01]

The test case specification and test functions of the test framework are defined in TTCN-3 to describe and manage the exchange of test events within the test components. The test data transferred in the test framework is constructed in the IDL description language and consists of the test server IDL and the message header IDL. The message header IDL consists of the GIOP interface that is defined in the CORBA specification. The conformance between the test data and the CORBA specification is another key factor to perform interoperability testing for CORBA-based applications. By doing this, the test framework can exchange the test data and messages with any CORBA-based application provided by different vendors.

In short, the key issue to CORBA interoperability is the conformance of the CORBA specifications. The test framework presented in [Li+01] gives us one example of using TTCN-3 and the IDL description language to perform interoperability testing for CORBA-based applications. The IDL description language describes all the test data structures according to CORBA specifications while TTCN-3 is used to specify the behaviour of the test cases and test functions.

5.7.2 OpenLS Interoperability Testing

5.7.2.1 Overview of OpenLS

The Open Location Services Initiative (OpenLS) [OGC 05] is a non-profit organization that is devoted to building open specifications for location service interfaces and related protocols. The goal of OpenLS is to specify standard ***interfaces*** and ***protocols*** which developers can use to integrate geospatial data and geoprocessing resources into location services and telecommunications infrastructure and to demonstrate these capabilities for a wide variety of applications for consumers, businesses and governments. As an Open Geospatial Consortium (OGC)'s open architecture for location services, OpenLS can support the following "Core Services" via well-defined interfaces and protocols:

- Gateway Services that integrate OpenLS location application services with position determination equipment in the Mobile Positioning Center (MPC) /Gateway Mobile Location Center (GMLC), which is the place in the network that manages the location of devices.
- Directory services for searching yellow pages, green pages, travel guides, etc.
- Route determination services for navigation.
- Geocode (address to X,Y) and reverse geocode (X,Y to address) services.
- Map/feature display services.

5.7.2.2 OpenLS Architecture and Typical Scenario

Figure 5-13 illustrates an example of sequence for a service request over an OpenLS environment. Generally, it has the same principle and similar sequences with the operation scenario of a LBS application (see Figure 2-4) that we described in section 2.2.3. However, the key characteristic of this scenario is that it employs the GeoMobility Server (GMS, the platform of OpenLS) as the core component of the OpenLS environment. The GMS is responsible for providing the OpenLS core services and the location content, and guiding the transactions among all the components in the OpenLS environment. In addition, all the communications (requests and responses) between the GMS and other components are defined according to the OpenLS interface and protocol specifications. By doing this, any components from different vendors can be easily integrated into the OpenLS architecture, increasing the interoperability capability.

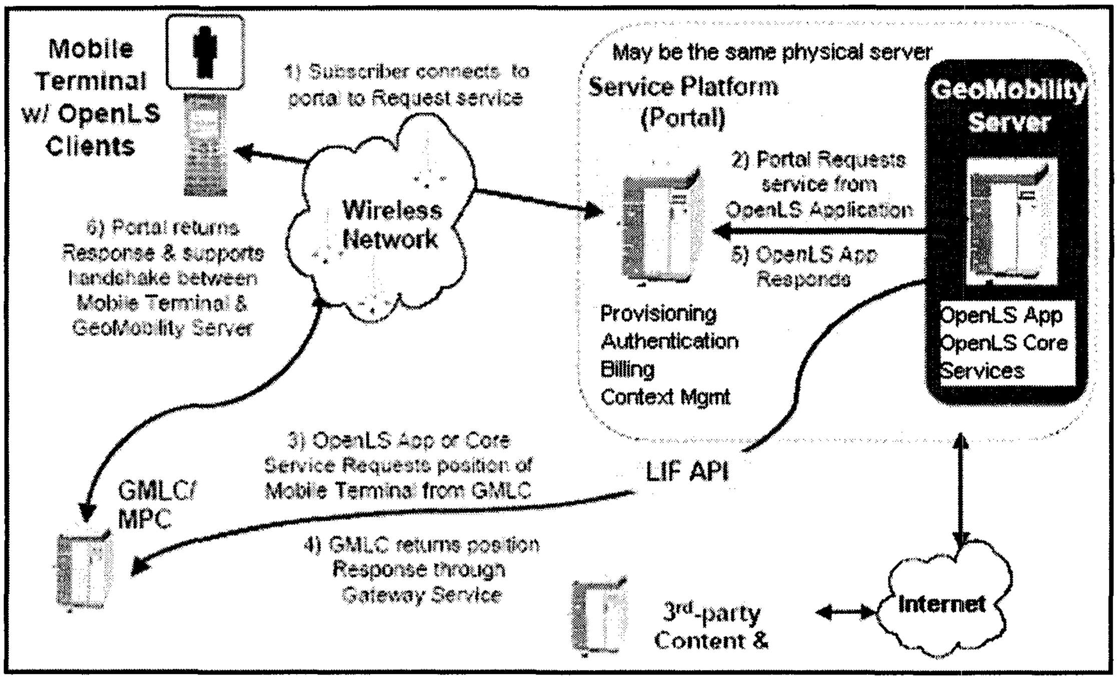

Mobile Terminal w/ OpenLS Clients
1) Subscriber connects to portal to Request service
Wireless Network
6) Portal returns Response & supports handshake between Mobile Terminal & GeoMobility Server
May be the same physical server
Service Platform (Portal)
GeoMobility Server
2) Portal Requests service from OpenLS Application
5) OpenLS App Responds
Provisioning
Authentication
Billing
Context Mgmt
OpenLS App
OpenLS Core Services
3) OpenLS App or Core Service Requests position of Mobile Terminal from GMLC
LIF API
GMLC/ MPC
4) GMLC returns position Response through Gateway Service
3rd-party Content &
Internet

Figure 5-13: OpenLS Typical Service Request/Response [Mabrouk 04]

Therefore, the key to the interoperability of the OpenLS architecture is the conformance to the interface and the protocol that are defined in the OpenLS specification. The

OpenLS Specification [Mabrouk 04] specifies all the necessary requirements for the interfaces and protocols of the XML location services (XLS), including encoding requirements, request requirements, response requirements, HTTP transaction protocol requirements and abstract data type requirements, etc.

[Mabrouk 04] also illustrates the general usage pattern for a XML based Request/Response in OpenLS, as shown in figure 5-14. A *Client Application* is any application that interacts with Core Services in the server, whether it resides on an end-user device or on a server. A *Request / Response* is a XML string that is passed from a Client Application to a Core Service (Request), or from a Core Service to a Client Application (Response).

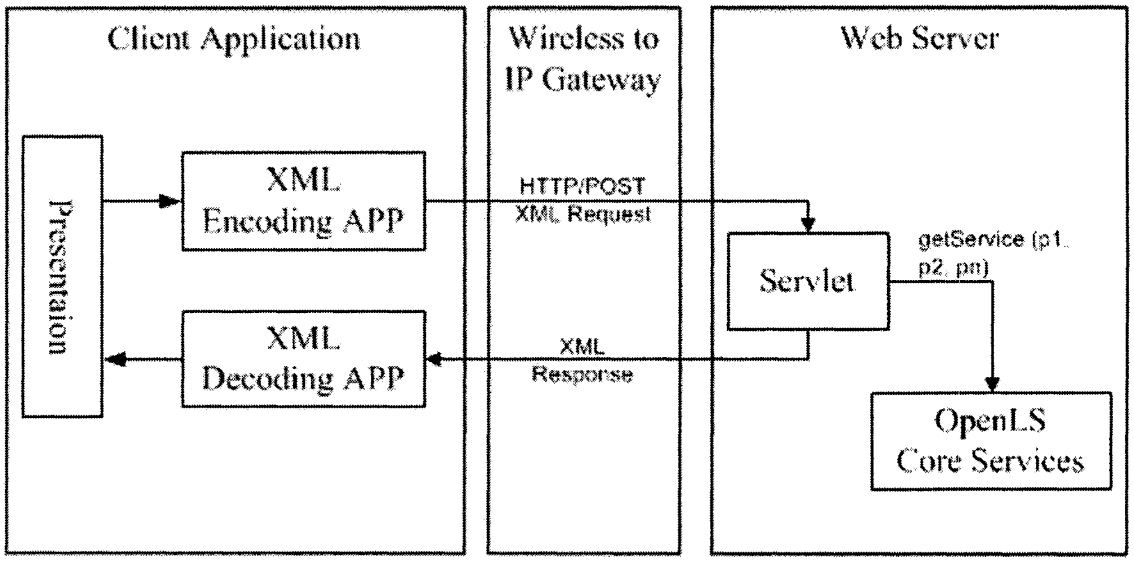
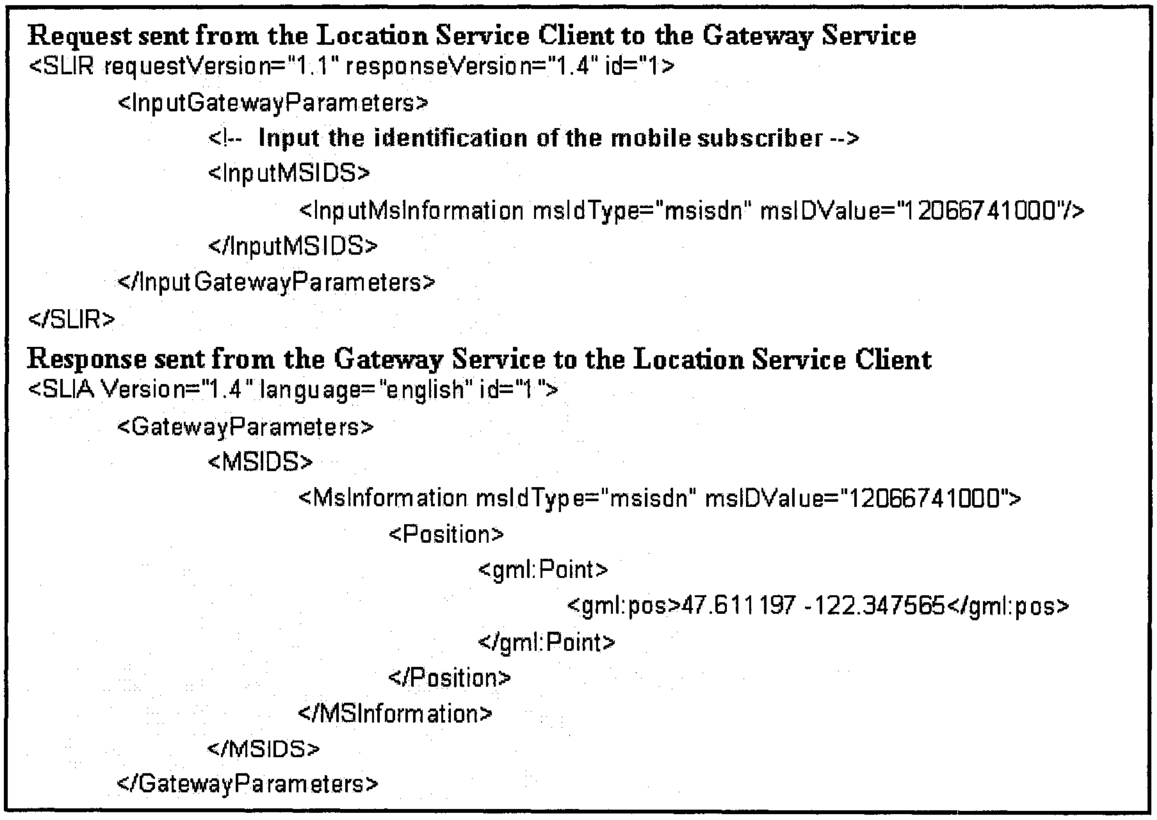


Figure 5-14: General Usage Pattern for OpenLS [Mabrouk 04]

The typical scenario of this general usage pattern is that, a Client Application firstly processes a user's request for service, which will in turn leads to a request for the use of a Core Service, e.g., a Directory Service. The request from the Client Application to the Core Service will be encoded in an XML message and sent to a Servlet on the web server using the HTTP/Post method. The Servlet in turn parses the XML Request, and generates the relevant function call to the Core Service. The Core Service processes the Request and sends back the Response to the Servlet, which will in turn encode the Response as an

XML Response and forward it to the Client Application. Finally, the Client Application will decode the XML Response and apply the proper presentation functions according to the XML Response tags.

Figure 5-15 shows the example of XML request/response messages exchanged between the GeoMobility Server (GMS) and Location Server that resides in the GMLC or MPC through which OpenLS services obtain position data for Mobile Terminals. In this example, the Gateway Service in the GMLC/MPC is employed to obtain the position of the subscriber's mobile terminal from the network. A Location Service Client in the GMS sends the request to determine a position to the Gateway. The Gateway calculates the position of the subscriber's mobile terminal and forwards to the Location Service Client, which may store it for as long as needed.



Request sent from the Location Service Client to the Gateway Service

```
<SLIR requestVersion="1.1" responseVersion="1.4" id="1>
        <InputGatewayParameters>
                <!-- Input the identification of the mobile subscriber -->
                <InputMSIDS>
                        <InputMsInformation msIdType="msisdn" msIDValue="12066741000"/>
                </InputMSIDS>
        </InputGatewayParameters>
</SLIR>
```

Response sent from the Gateway Service to the Location Service Client

```
<SLIA Version="1.4" language="english" id="1 ">
        <GatewayParameters>
                <MSIDS>
                        <MsInformation msIdType="msisdn" msIDValue="12066741000">
                                <Position>
                                        <gml:Point>
                                                <gml:pos>47.611197 -122.347565</gml:pos>
                                        </gml:Point>
                                </Position>
                        </MSInformation>
                </MSIDS>
        </GatewayParameters>
```

Figure 5-15: XML Request/Response Message in OpenLS

5.7.2.3 OpenLS Interoperability Testing Strategy

As mentioned earlier, the key to the interoperability of the OpenLS architecture is the conformance to the interface and the protocol that are defined within the OpenLS specification. From the above scenario of the general usage pattern, we can see that the OpenLS defines the HTTP protocol as its interface between GMS and the Client Application, and the XML message is the only format defined in the OpenLS between GMS and the Client Application. Therefore, the interoperability testing of OpenLS can be simplified to the validation of transactions between an arbitrary Client Application and GMS under the HTTP transaction protocol and other requirements defined in the OpenLS specification.

Because of the similar architecture between the CORBA-based application and the OpenLS application, it is very straightforward to adapt and apply the test methodology used in the CORBA interoperability testing [Li+01] to the OpenLS interoperability testing. As we implement our test framework with TTCN-3, it is in turn feasible to test the OpenLS interoperability under our test framework. Specifically, the HTTP transport protocol can be treated as the vertical interface (GIOP) as in the CORBA-based application. The horizontal interface is not included in the OpenLS. As shown in figure 5-16, the System Under Test (SUT) can be either the Client Application, or the GMS. The Main Test Component (MTC) is responsible for the creation of the test components and the overall control of test components over the CP ports. The TestClient and TestServer are two parallel test components defined by MTC. The TestClient connects to SUT via the HTTP protocol when the SUT is GMS, acting as the Client Application. The TestServer connects to SUT via HTTP protocol when SUT is the Client Application, acting as GMS.

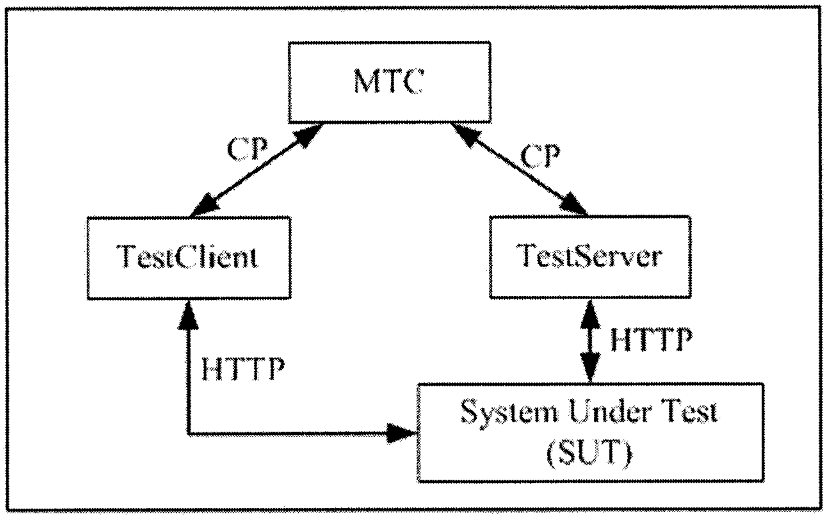
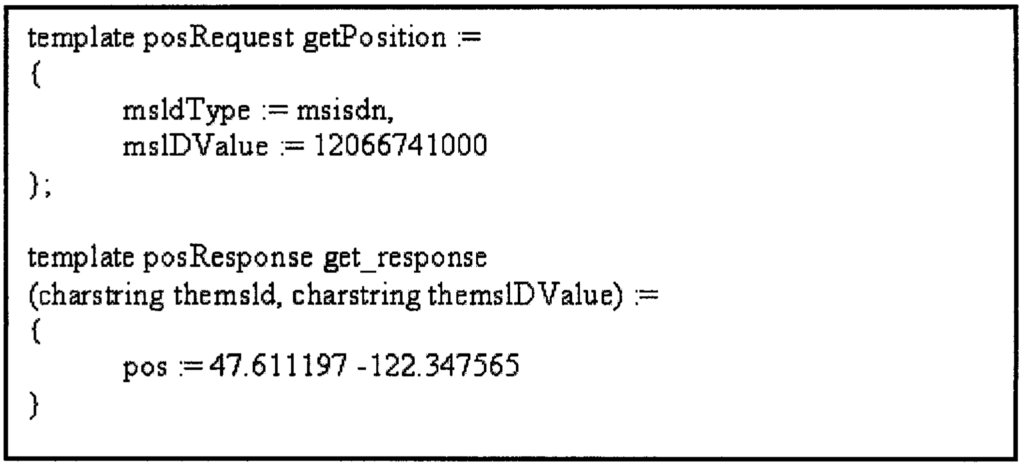


Figure 5-16: TTCN-3 Test Component Configuration for OpenLS

The test data sending to the system under test must represent the XML request/response messages. Instead of the IDL data structure defined in the CORBA interoperability testing, our test framework enables to define the XML data in the TTCN-3 data type format. As mentioned in section 4.6.6, TTCN-3 can test XML based messages and protocols, such as XML/SOAP based web service [Stepien+03]. Taking the XML messages in the figure 5-15 as an example, we can generate the following relevant test data in the TTCN-3 data type format by using the XML to TTCN-3 mapping approach defined in [Stepien+03]:



```
template posRequest getPosition :=
{
        msldType := msisdn,
        mslDValue := 12066741000
};

template posResponse get_response
(charstring themsld, charstring themslDValue) :=
{
        pos :=47.611197 -122.347565
}
```

Figure 5-17: OpenLS Request/Response Test Data

After the definition of the test data, the XML message can be encoded and decoded by the TTCN-3 compiler. The rest of test steps are similar to the traditional TTCN-3 test process we described earlier. Other requirements that are specified in OpenLS, i.e. encoding requirements, request requirements, response requirements, etc., can be defined within the IDL description language and then loaded by the TTCN-3 module, same as the test definition method used in the CORBA interoperability testing.

Therefore, by utilizing the research from CORBA interoperability testing by [Li+01] and the relationship between the CORBA and OpenLS structures, we can theoretically demonstrate the feasibility of our test framework on performing the interoperability testing for both the Client Application and GMS provided by different vendors. However, the implementation of the test cases is not considered in this thesis because of the lack of relevant Client Applications and GMS at the current stage.

5.7.3 Security Testing

As we mentioned in section 4.6.6, many factors are related to the security issues in the location-based service, i.e. personal integrity and confidential data transmission. The personal integrity can be demonstrated by the test cases (CA02-CA04, CA09-CA14, CA19-CA20, see Appendix H, I, J for details of these cases) in the functional testing on the specific use cases, such as, activation (CA02-CA04) and deactivation (CA09-CA14) of the MST system and the authorization of subscriber's location request (CA19-CA20).

The confidential data transmission should also be considered in the interaction regarding to the sensitive contents, i.e. user's position and password. As mentioned earlier, the XML security protocols, i.e. Security Assertion Markup Language (SAML), XML digital signatures, can be one of methodologies to secure the confidential content during the transaction. Next we will take SAML as an example and investigate the principle of SAML and the feasibility of testing the SAML protocol under our test framework.

5.7.3.1 SAML Overview

Security Assertion Markup Language (SAML) is an emerging OASIS standard [Rouault 05], which defines an XML-based security specification for exchanging authentication and authorization information between online businesses. With SAML, users can carry entitlements across multiple sites, and different *trust domains*. From these entitlements, the Web Service will know whether to trust a user or not, and will also know the permissions to which a user is entitled.

As shown in figure 5-18, SAML is defined with the following four components [OASIS 05]:

- **Assertions**: An assertion is a package of information that supplies one or more statements made by a SAML authority. SAML assertions are encoded in an *XML schema*. SAML defines three different kinds of assertion statement that can be carried within an assertion:
    1. Authentication statements – The specified subject was authenticated by a particular means at a particular time. The authentication statement is typically generated by a SAML authority called an identity provider, which is in charge of authenticating users and keeping track of other information about them.
    2. Attribute statement – It contains specific details about the user, i.e. Gold member status. This specified subject is associated with the supplied attributes.
    3. Authorization decision statement – A request to allow the specified subject to access the specified resource has been granted or denied. In other words, is this subject allowed to access the specified resource in the specified manner, given this evidence?
- **Protocols:** SAML defines a number of request/response protocols, including authentication, attribute and authorization decision request/response protocols, etc. The protocol is encoded in an XML schema as a set of request-response pairs.

Appendix K lists one example of SAML Authentication request/response messages.

- **Bindings:** This describes exactly how SAML request-response protocols map into standard messaging or communication protocols. For example, the SAML SOAP Binding defines how SAML protocol messages can be communicated within SOAP messages.
- **Profiles:** The core of the SAML specification defines how the SAML requests and responses are transported. Profiles define how the SAML assertions, protocols and bindings are combined to support specific use cases. Several profiles are defined in SAML, such as Web Browser SSO (Single Sign-On) Profile, which defines how a Web Browser support SSO when using Authentication Request protocol messages, and Enhanced Client and Proxy (ECP) Profile, which is designed to support mobile devices front-ended by a WAP gateway.

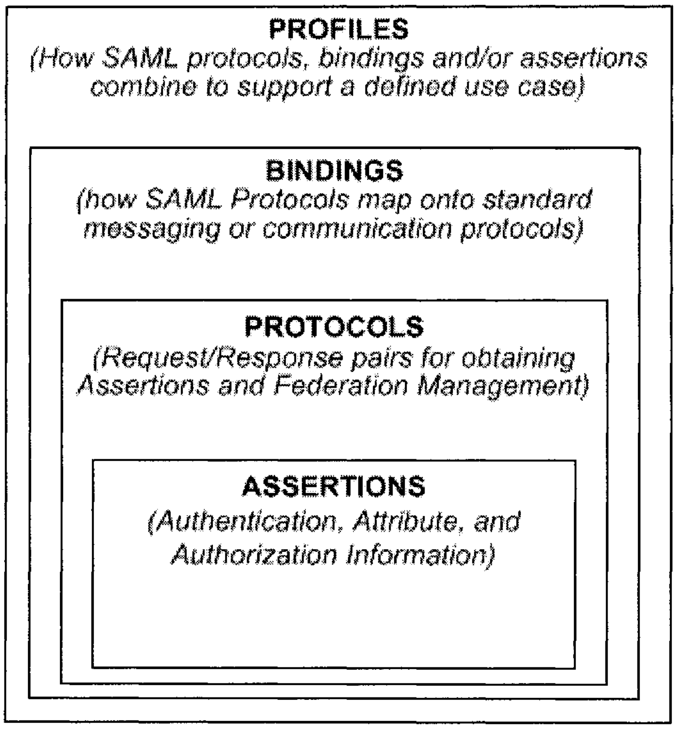

PROFILES
(How SAML protocols, bindings and/or assertions combine to support a defined use case)
BINDINGS
(how SAML Protocols map onto standard messaging or communication protocols)
PROTOCOLS
(Request/Response pairs for obtaining Assertions and Federation Management)
ASSERTIONS
(Authentication, Attribute, and Authorization Information)

Figure 5-18: SAML Components

5.7.3.2 SAML Use Case

The SAML standard components [OASIS 05] define a framework for exchanging security information between online business partners. SAML defines a common XML framework for creating, requesting, and exchanging security assertions between entities. These entities include the subject (or the user), the identity provider (IDP) and the service provider (SP). The identity provider (IDP) is a system that asserts information about a subject. For instance, it can assert that the user has been authenticated and has given associated attributes, i.e. username of "Jack Chen". The service provider (SP) is a system that relies on information supplied to it by the identity provider. SAML defines a number of mechanisms that enable the SP to trust the assertions provided to it from the IDP.

Figure 5-19 illustrates one typical SAML use case – Single Sign-On (SSO) Use Case. In this use case, a service subscriber login the source web site (Service.com) and searches for available services. For some reason, the selected service, i.e. weather service, is not available on the source web site. Thus, the identity provider (Service.com) asserts to the service provider (Weather.com) that the user is known to it and provides the user's name and session attributes (e.g. "Gold member"). As Weather.com trusts Service.com, it knows that the user is valid and creates a session for the user based on the user's name and/or the user attributes. This use case illustrates the fact that the user is not required to re-authenticate when directed over to the Weather.com site. Appendix L also lists another SSO use case – Web browser SSO Profile, in which the SAML assertion is transported to the Service Provider using the HTTP POST binding.

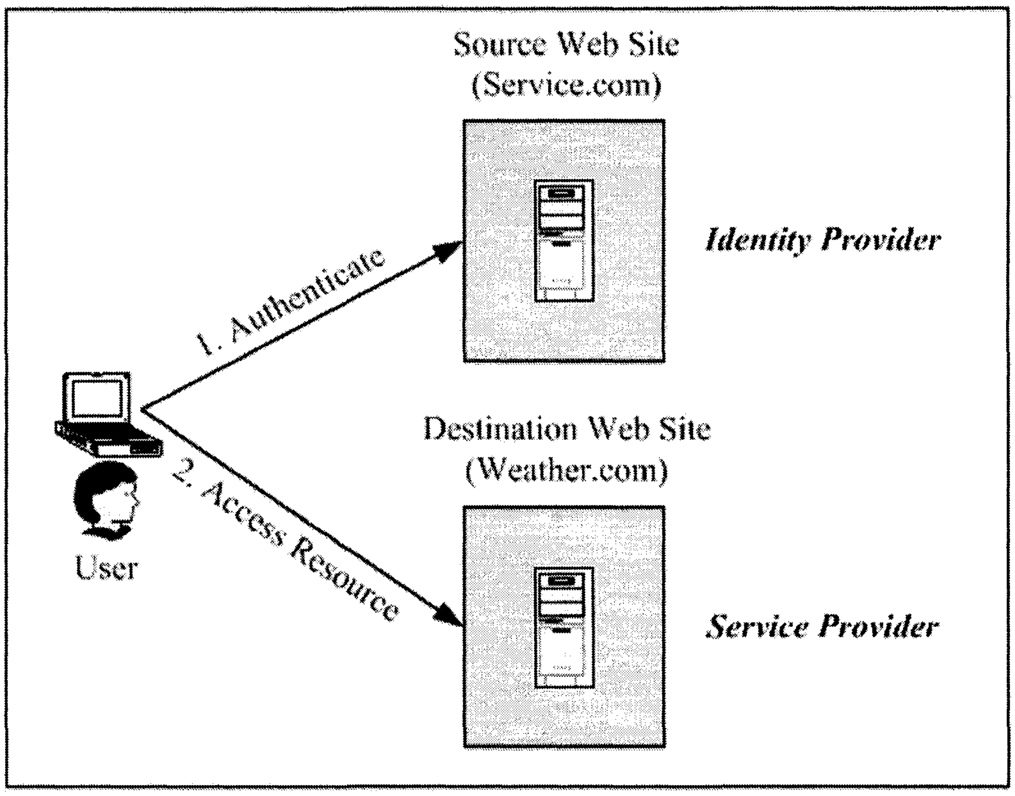


Figure 5-19: Single Sing-On Use Case [OASIS 05]

5.7.3.3 Apply SAML to Location-based Services

As mentioned earlier, the SAML protocol can be applied in the location-based service (LBS) to secure the confidential content, such as user's position and password information. There was some research about the location-based security with the SAML protocol, such as [Srivatsa 02], which embeds the location information into the SAML authentication request/response messages.

By using the authorization assertion request of SAML message protocol, we can build a security mechanism for the transaction between the location management platform (LMP) and the service provider (SP). Specifically, the service provider can send a SAML authorization assertion request to LMP for the subscriber's position information. After LMP check the authorization of the service provider, it can send SAML response back to the Service Provider.

For the authentication assertion request of the SAML message protocol, we can apply the SSO use case in the location based service and use it to secure the service subscriber's password and other private information transmitted between the subscriber and the service provider. Specifically, the service provider in LBS can be treated as the identity provider, which has a service category and can provide the service subscriber with all types of location based services, no matter whether they are available in the local server or not. If the selected service is available in the local server, the user will access the service via the traditional way we described earlier. If, however, the selected service is not available in the local server, the local server (the identity provider) will assert to another service provider that can offer user's needed service that the user is a legal service subscriber and provides the user's information to the second service provider. By doing this, the user's information is secured by the authentication assertion mechanism and the user can be transparently redirected to the new direction to access the available resources, resulting in the improvement of system usability.

5.7.3.4 Security Testing SAML Protocol in LBS

Since the SAML assertions and request/response protocols are encoded in the XML schema, and the messages exchanged between the entities within the system are an XML-based messages that has same structure with the XML request/response in the OpenLS, it is very straightforward to apply the testing methodology we employed on the OpenLS interoperability testing to the testing of the SAML message protocol. In other words, we can map the SAML assertion and protocol XML schema to the TTCN-3 data structure by using the XML to TTCN-3 mapping approach defined in [Stepien+03].

Taking the SSO use case as an example, we can use it in LBS to secure a user's private information and redirect the user request to a new service. To test this scenario, we need to generate the test component configuration as shown in figure 5-20. In this test component configuration, the system under test is the location-based application (LBA), which represents the user entity in the SSO use case. The Main Test Component (MTC) is responsible for the creation of the parallel test components (PTC) and coordinates the test behaviour among these components via the CP ports. The identity provider (IDP) and service provider (SP) are represented by the parallel test components PTC1 and PTC2, respectively. The communication between the parallel test components and the system under test is defined using the HTTP protocol, and the test data exchanged with these entities is defined with the SAML message protocol. The test process is initiated from the service request by the SUT. After the identity provider (PTC1) receives the SUT's request, it sends an authentication response message back to the SUT, which will then be forwarded to the service provider (PTC2) for verification with the expected data. SAML request XML messages are encoded from and SAML response XML messages are decoded into TTCN-3 data – used in the test specification by the TTCN-3 compiler.

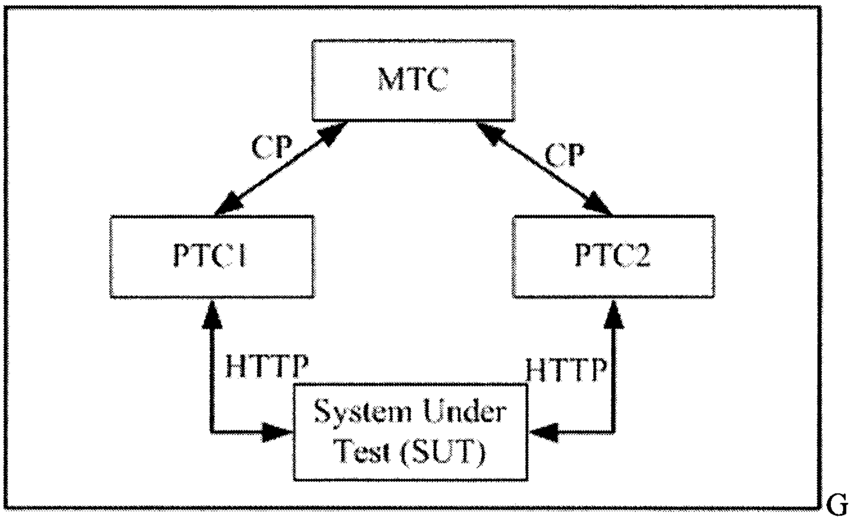
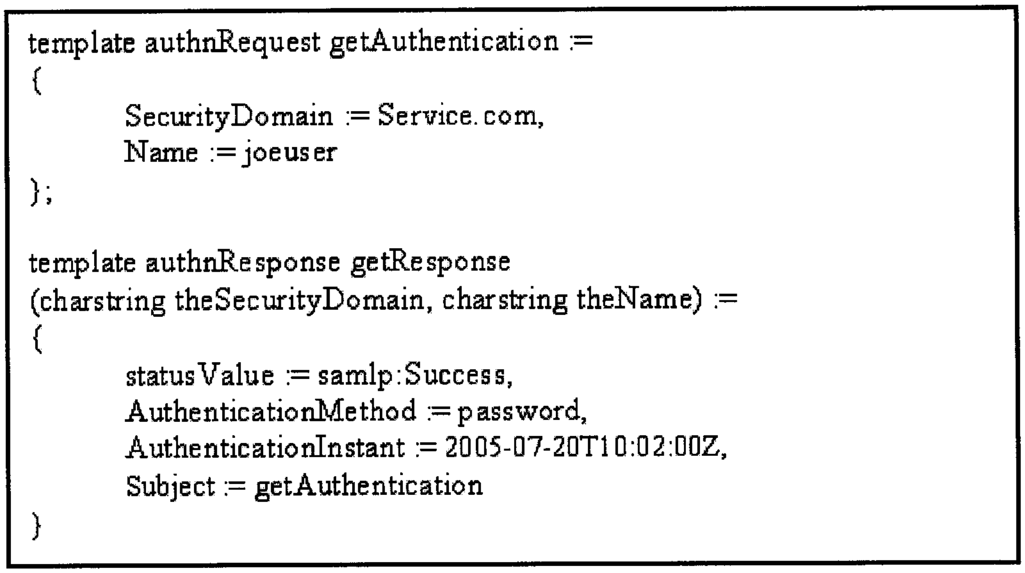


Figure 5-20: Test Component Configuration for SSO

Similar to the OpenLS interoperability testing, we can map the SAML message protocol to corresponding TTCN-3 data structures by using the XML to TTCN-3 mapping approach defined in [Stepien+03]. Figure 5-21 shows the test data definition that is generated from the authentication request/response XML messages listed in Appendix K. From this data definition, we can see that the user information, i.e. username and status, is combined in the authentication response message, which will be sent eventually to the service provider for the service request.



```
template authnRequest getAuthentication :=
{
        SecurityDomain := Service.com,
        Name := joeuser
};

template authnResponse getResponse
(charstring theSecurityDomain, charstring theName) :=
{
        statusValue := samlp:Success,
        AuthenticationMethod := password,
        AuthenticationInstant := 2005-07-20T10:02:00Z,
        Subject := getAuthentication
}
```

Figure 5-21: Authentication Request/Response Test Data

The rest of the test process for the SAML protocol testing, such as the test case and test function definition, is similar to the traditional TTCN-3 test process described earlier in the experimental study. Here we only investigate the feasibility of our test framework on the security testing of the SAML protocol; and hence, do not provide an implementation of the test cases and test functions for this situation. From what we described in this section, we can find out that it is possible to use our test framework to validate the LBS confidential data transmission through the testing of SAML message protocols.

5.8 Test Evaluation

Test Case	Test Scenario	Functional Testing	Usability Testing	Network Performance Testing	Server Site Testing	Security Testing
TC001	TS01	OK				
TC002	TS01	OK				
TC003	TS02	OK				
TC004	TS03			OK		

TC005	TS03			OK		
TC006	TS03			OK		
TC007	TS05				OK	
TC008	TS04		OK			
TC009	TS04		OK			
TC010	TS04		OK			
TC011	TS04		OK			
TC012	TS04		OK			
CA01-33		OK				
CA02-04		OK				OK
CA09-14		OK				OK
CA19-20		OK				OK

Table 5-13: The Evaluation Result of The Experimental Study

The Mobile Service Tracking experimental study has mainly exercised the testing of a generic infrastructure of location-based services. As shown on the table 5-13, all of test cases were designed and implemented to cover the test scenarios, which reflect most of evaluation strategies for the location-based servers (LBS), including functional testing, usability testing, network performance testing, server site testing and security testing. Interoperability is not covered by above test cases. However, we demonstrate the feasibility of our test framework on LBS interoperability testing by introducing the CORBA-based application and the OpenLS interoperability testing. Security testing on the confidential data transmission is also not cover by above test cases. But we investigate the principle of SAML message protocol and demonstrate that it is possible to make the SAML message protocol testing with our test framework. The overall results of the experimental study demonstrate the effective use of our test framework as an approach to test location-based services automatically with typical scenarios. It supports the fast prototype of test system for users and can be explored in other applications, such as distributed systems, component-based systems, and object oriented systems.

However, the Java dependency of our test framework is potentially one limitation if the exploitation of the test framework is applied on a target system written in other languages. The system complexity could also be one of limitations when all components are embedded into the test framework. Finally the test results have also raised the following issues:

- *Definition of Location Context.* Theoretically, the context specified in the location-based service can be any changing environment, i.e. location, user input and display. Dey and Abowd [Dey+99] defined context as any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves. Our typical approach has been focused only on general GPS location context, and does not consider other type of context events yet.

- *Domain application knowledge.* In order to embed the system under test into the test framework, we need to distribute the test adapter into the component under test. It means the tester must know the specific API provided by the SUT. In addition, the component under test needs to be modified on its API to increase its testability, being adapted to the standard interface provided by the test framework. Therefore, we consider our test framework is built on the gray-box testing methodology.

Chapter 6 Conclusion and Future Work

Through this thesis, we investigated the needs of testing the location-based service (LBS) and challenges of testing the LBS system. We also presented our test framework for testing LBS systems based on the TTCN-3 test system, supplemented by the context simulator, the SWANS wireless simulator in the test framework. Finally, we demonstrated the capability of the test framework with an experimental study on the testing of the LBS from four testing perspectives: functional testing, usability testing, performance testing, server site testing. Our test framework is aimed at providing users with rapid prototyping and an effective test environment by utilizing re-useable components and well-defined system interfaces.

6.1 Contributions

The contributions of this thesis can be summarized as follows:

- Investigate the principles of all types of location-based service, including SMS-based, WAP-based and J2ME based location-based service.
- Define and illustrate the critical challenges on the testing of the location-based service.
- Provide essential test requirements for building an effective test framework on the LBS testing.
- Present a new effective test framework for testing the location-based service. This framework provides simulated execution environment for the location-based service by integrating the SWANS wireless simulator and the context simulator.
- Provide a scalable test framework that allows rapid prototyping with the reusable test components and well-defined interfaces.
- Demonstrate a systematic approach for the location-based service testing from various perspectives, including functional testing, usability testing, network performance testing, server side testing, interoperability testing, and security testing.

6.2 Future Work

The testing framework described in this thesis can make the process of testing LBS system significantly easier and more thorough, especially with regarding to the performance testing and the functional testing. However, it does not make the testing process simple, straightforward, or flawless, especially on the implementation of the TTCN-3 test system. No matter how good the testing tool, someone still has to "drive" it. For us, our future work includes building more TTCN-3 test modules and test adapters, which may be slightly different according to different types LBS system. Once all these components are done, they can be reused to test further LBS systems and help us save time and effort.

TTCN-3, however, also has some major problems. For instance, it supports only static test interfaces and the representation of heterogeneous test data by means of unions, which makes the description of message content clumsy [Deussen+03]. We need to investigate the influence of these problems to our test framework. Moreover, interoperability testing and security testing need to be demonstrated deeply in our test framework later.

Bibliography

[Adatia 01] Adatia, R., et al., Professional EJB, UK: Wrox Press, 2001

[Alliance 05] Alliance Software Engineering, Software Testing Glossary, Retrieved June 2005 from http://www.sitetestcenter.com/software_testing_glossary.htm#S

[Altman+04] Eitan Altman and Tania Jiminez, NS Simulator for beginner, *Lecture notes, 2003-2004, Chapter 9 Mobile Networks, Page 111. Retrieved September 2005 from http://www.isi.edu/nsnam/ns/ns-contributed.html*

[Barr 04] Rimon Barr, JiST User Guide, *Retrieved December 2004 from http://jist.ece.cornell.edu/*

[Barr 05] Rimon Barr, SWANS User Guide, *Retrieved April 2005 from http://jist.ece.cornell.edu/*

[Barton+03] John J. Barton, V. Vijayaraghavan, UBIWISE, A Simulator for Ubiquitous Computing Systems Design, *HP Laboratories Palo Alto HPL-2003-93, April 29th, 2003*

[Basso 02] Basso, M. and Kreizman, G., Mobile location services for governments, *Gartner, September 2002.*

[Binder 00] Robert V. Binder, Testing Object-Oriented Systems, *Addison-Wesley Publishing Inc., 2000*

[Clarke 97] Peter Clarke, Cell phone positioned for new services, *Electronic Engineering Times, vol. 30, no. 25, pp. 6, Jan. 20, 1997.*

[Deussen+03] Deussen, P.H., Din, G., Schieferdecker, I., A TTCN-3 based online test and validation platform for Internet services, *Autonomous Decentralized Systems, 2003. ISADS 2003. The Sixth International Symposium on 9-11 April 2003 Pages:177 – 184*

[Dey+99] A. K. Dey, Gregory D. Abowd, Towards a Better Understanding of Context and Context-Awareness, *Lecture Notes In Computer Science, Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing, Karlsruhe, Germany, Pages: 304 – 307, 1999*

[Dey 00] A.K. Dey, Providing Architectural Support for Building Context-Aware Applications, *Doctoral thesis, College of Computing, Georgia Inst. Of Technology, 2000.*

[Dey+00] Anind K. Dey and Gregory D. Abowd, The Context Toolkit: Aiding the Development of Context-Aware Applications, *In the Workshop on Software Engineering for Wearable and Pervasive Computing, Limerick, Ireland, June 6, 2000.*

[Djuknic+01] G. M. Djuknic, R. Richton, Geolocation and assisted GPS, *Computer, vol. 34, pp. 123-125, Feb 2001*

[ETSI 04] European Telecommunication Standard Institute, TTCN-3 Control Interface, Version: 1.1.1, TTCN-3 Specification, *Retrieved September 2004 from http://www.etsi.org/ptcc/ptccttcn3.htm#*

[ETSI 05] European Telecommunication Standard Institute, Interoperability Testing Specification, *Retrieved June 2005 from http://portal.etsi.org/mbs/Testing/interop/interop.asp#TSS*

[Eurotechnology.com 05] Eurotechnology.com, The unofficial independent i-mode FAQ, *Retrieved March 2005 from http://www.eurotechnology.com/imode/faq.html*

[Fall+05] Kevin Fall, Kannan Varadhan, The NS Manual, *Retrieved April 2005 from http://www.isi.edu/nsnam/ns/doc/index.html*

[Fujimoto 90] R. M. Fujimoto, Parallel discrete event simulation, *CACM, 33(10): 30–53, Oct. 1990.*

[Fujimoto 95] R. M. Fujimoto, Parallel and distributed simulation, *In Winter Simulation Conf., pages 118–125, Dec. 1995.*

[Grabowski+03] Jens Grabowski, et al., An Introduction into the Testing and Test Control Notation (TTCN-3), *to appear in Computer Networks, 2003*

[GSM Association 05] GSM Association, Short Message Service, GSM World. *Retrieved March 2005 from http://www.gsmworld.com/technology/sms/index.shtml*

[Hata 80] M. Hata, Empirical Formula for Propagation Loss in Land Mobile Radio Services, *IEEE Trans. Vehic. Tech., vol. 29, no. 3, 1980.*

[IEEE 90] IEEE Std 610.12-1990, Standard Glossary of Software Engineering Terminology, Retrieved from http://standards.ieee.org/.

[José+03] Rui José, Adriano Moreira, Helena Rodrigues and Nigel Davies, The AROUND Architecture for Dynamic Location-Based Services, *Mobile Networks and Applications 8,377-387, 2003*

[Kaner+99] Cem Kaner, Jack Falk, Hung Q. Nguyen, *Testing Computer Software, 2nd Edition, Wiley, 1999*

[Krishnamurthy 02] Nandini Krishnamurthy, Using SMS to deliver location-based services, *Personal Wireless Communications, 2002 IEEE International Conference on 15-17 Dec. 2002 Page(s): 177 – 181.*

[Lee+03] Chung-wei Lee, Wen-Chen Hu, Jyh-haw Yeh, A System Model for Mobile Commerce, *Distributed Computing Systems Workshops, 2003. Proceedings. 23rd International Conference on 19-22 May 2003 Page(s):634 – 639*

[Li+01] Mang Li, Arno Puder, Ina Schieferdecker, A Test Framework for CORBA Interoperability, *Enterprise Distributed Object Computing Conference, 2001. EDOC '01. Proceedings. Fifth IEEE International 4-7 Sept. 2001 Page(s): 152 - 161*

[Li+04] Kanglin Li, Menqi Wu, Effective Software Test Automation: Developing an Automated Software Testing Tool, *Sybex 2004*

[LIF 05] Location Interoperability Forum, Location Services Interoperability Test Specification in GSM, *Retrieved September 2005 from the Open Mobile Alliance (OMA) webpage, http://www.openmobilealliance.org/tech/affiliates/lif/lifindex.html*

[Mahmoud 00] Mahmoud, Q., MIDP network programming using HTTP and the Connection framework, *Java Wireless Developer, Retrieved November 2001 from http://wireless.java.sun.com/midp/articles/network/*

[Mabrouk 04] Marwa Mabrouk, Open GIS Location Service (OpenLS): Core Services, OpenGIS Implementation Specification, January 2004, *Retrieved June 2005 from http://www.opengeospatial.org/functional/?page=ols*

[McKee 05] Lance McKee, The Importance of Going “Open”, OGC White Paper, *Retrieved June 2005 from http://www.opengeospatial.org/*

[Misra 86] J. Misra, Distributed discrete event simulation, *ACM Computing Surveys, 18(1): 39–65, Mar. 1986.*

[Mitchell+03] Kirk Mitchell, Mark Whitmore, Mobile Commerce: Technology, Theory and Applications, *Idea Group Publishing, 2003, Chapter 3*

[Morla+04] Ricardo Morla and Nigel Davies, Evaluating a location-based application: a hybrid test and simulation environment, *Pervasive Computing, IEEE, Volume: 3, Issue: 3, July-Sept.2004 Pages: 48– 56.*

[Myers 79] Glenford J. Myers, The Art of Software Testing, *John Wiley & Sons, Preface, 1979.*

[MySQL 05] MySQL Reference Manual, *Retrieved June 2005 from http://dev.mysql.com/doc/*

[Neskovic+00] A. Neskovic, N. Neskovic, and G. Paunovic, Modern Approaches in Modeling of Mobile Radio Systems Propagation Environment, *IEEE Commun. Surveys, vol. 3rd Quarter, Sept. 2000, pp. 2–12.*

[Nicol+94] D. M. Nicol and R. M. Fujimoto, Parallel simulation today, *Annals of Operations Research, pages 249–285, Dec. 1994.*

[OASIS 05] OASIS Security Service, SAML V2.0 OASIS Standard specification, Retrieved August 2005 from http://www.oasis-open.org/home/index.php

[Okumura+68] Y. Okumura *et al.*, Field Strength and its Variability in VHF and UHF Land-Mobile Services, *Review Elec. Commun. Labs., vol. 16, Sept.Oct., 1968, pp. 82537.*

[OGC 05] Open Geospatial Consortium Inc., Overview of OGC's Interoperability Program, *Retrieved June 2005 from http://www.opengeospatial.org/*

[OGC 05] Open Geospatial Consortium Inc., Resource-Compliance Testing, *Retrieved June 2005 from* http://www.opengeospatial.org/

[OMG 05] Objective Management Group, Common Object Request Broker Architecture (CORBA), Retrieved June 2005 from http://www.omg.org/gettingstarted/corbafaq.htm

[OPNET 05] OPNET Technologies, Inc, OPNET Modeler Brochure, *Retrieved April 2005 from http://www.opnet.com/products/modeler/home.html*

[Raju 00] Raju, S., Java programming for wireless devices using J2ME–CLDC/ MIDP APIs, *Sun Microsystems, 2002.*

[Rappaport 95] T. S. Rappaport, Wireless Communications: Principles & Practice, *Prentice Hall, 1995, Chapter 3*

[Rouault 05] Jason Rouault, Introduction to SAML, *Retrieved Auguest 2005 from http://www.simc-inc.org/archive0002/February02/devwed1015_rouault.pdf*

[Satoh 03] Ichiro Satoh, A testing framework for mobile computing software, *Software Engineering, IEEE Transactions on, Volume: 29, Issue: 12, Dec. 2003, Pages: 1112 – 1121*

[SavaJe.com 05] SavaJe.com SavaJe OS, *Retrieved March 2005 from http://www.savaje.com/*

[Schieferdecker+03] Ina Schieferdecker, et al., Realizing Distributed TTCN-3 Test Systems with TCI, *IFIP TC 6 / WG 6.1 15th International Conference on Testing of Communicating Systems - TestCom 2003, May 2003*

[Schieferdecker 04] Schieferdecker, I., TTCN-3 Tutorial, *Test & Testing Methodologies with Advanced Languages Seminar, 26th of March 2004, Oulu, FI.*

[Schilit+94] Schilit, B., Theimer, M., Disseminating Active Map Information to Mobile Hosts, *IEEE Network, 8(5) (1994) 22-32*

[Srivatsa 02] Harsha Srivatsa, Location-based security for wireless apps, *Retrieved May 2005 from http://www-106.ibm.com/developerworks/wireless/library/wi-loc/*

[Stüber 01] Gordon L. Stüber, Principles of Mobile Communication 2nd Edition, *Published by Kluwer Academic Publisher, Page 23, 2001*

[Stepien+03] Stepien, B., Schieferdecker, I., Automated Testing of XML/SOAP based Web Services, *Leipzig Conference, 2003.*

[Sun Microsystems 05] Sun Microsystems, Introduction to wireless Java. Wireless Java Technology, *Retrieved March 2005 from http://wireless.java.sun.com/getstart/*

[Surech+01] Surech C., Parviz K., Sean S., Leandros T., Security Issues in M-Commerce: A Usage-Based Taxonomy, *E-Commerce Agents, LNAI 2033, pp. 264-282, 2001. Springes-Verlag Berlin Heidelberg 2001*

[Tassey+03] Gregory Tassey, et al., The Economic Impacts of Inadequate Infrastructure for Software Testing, *Collingdale, PA: DIANE Publishing Co, 2003.*

[Testing Technologies 04] Testing Technologies, Inc., Ttthree Specification, *Retrieved September 2004 from http://www.testingtech.de/products/ttthree.php*

[The Open Group 05] The Open Group, Testing Service for WAP, *Retrieved March 2005 from http://www.opengroup.org/testingservices/wapts.html*

[UCLAPCL 05] UCLA Parallel Computing Laboratory, GloMoSim Manual,*Retrieved April 2005 from http://pcl.cs.ucla.edu/projects/glomosim/GloMoSimManual.html*

[UMTS 05] UMTS Forum, MAGIC MOBILE FUTURE 2010-2020, *UMTS Forum REPORT NO 37, Retrieved June 2005 from http://www.umts-forum.org/servlet/dycon/ztumts/umts/Live/en/umts/Resources_Reports_37_index*

[Varshney+01] Upkar Varshney, Ron Vetter, A Framework for the Emerging Mobile Commerce Applications, *Proceedings of the 34th Hawaii International Conference on System Sciences 2001*

[WAPForum 05] WAPForum, WAP 2.0 Technical White Paper, *Retrieved January 2005 from http://www.wapforum.org*

[W3C Schools 05] W3C Schools, WAP Tutorial, *Retrieved March 2005 from http://www.w3schools.com/wap/wap_intro.asp*

[W3C Schools 05] W3C Schools, Validate WML, *Retrieved March 2005 from http://www.w3schools.com/wap/wml_validate.asp*

[Xora 05] Xora, Inc., *Xora GPS Time Track System. Retrieved May 2005 from http://www.xora.com/timetrack/index.html*

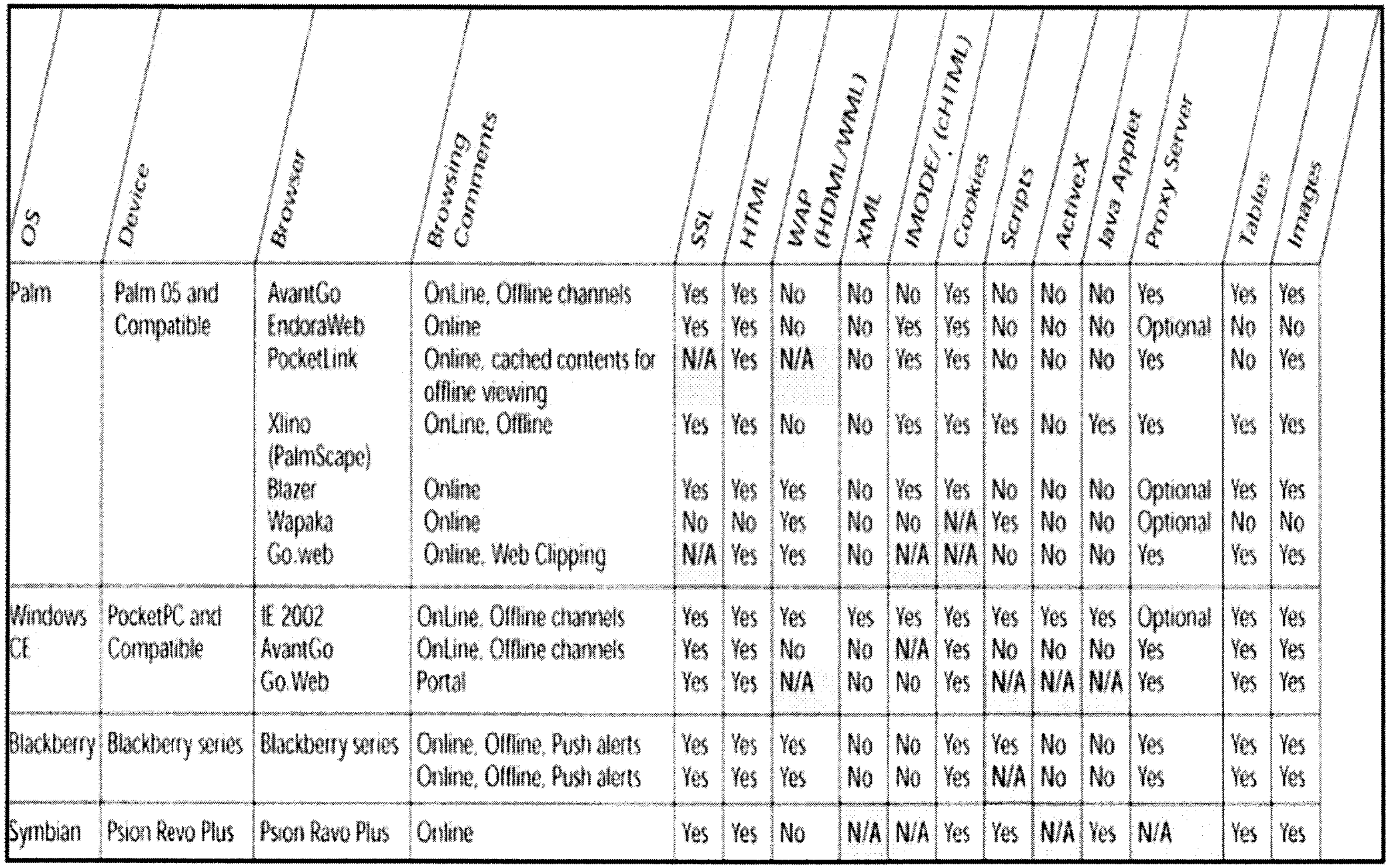
[Yuan 05] Michael Yuan, Ju Long, Securing wireless J2ME, *Retrieved June 2005 from IBM developerWorks, http://www-128.ibm.com/developerworks/library/wi-secj2me.html*

Appendix A: J2ME Profile and APIs

Configurations		
JSR 30	CLDC 1.0	Connected, Limited Device Configuration
JSR 139	CLDC 1.1	Connected, Limited Device Configuration 1.1
JSR 36	CDC	Connected Device Configuration
JSR 218	CDC 1.1	Connected Device Configuration 1.1
Profiles		
JSR 37	MIDP 1.0	Mobile Information Device Profile
JSR 118	MIDP 2.0	Mobile Information Device Profile 2.0
JSR 75	PDAP	PDA Profile
JSR 46	FP	Foundation Profile
JSR 219	FP 1.1	Foundation Profile 1.1
JSR 129	PBP	Personal Basis Profile
JSR 217	PBP 1.1	Personal Basis Profile 1.1
JSR 62	PP	Personal Profile
JSR 215	PP 1.1	Personal Profile 1.1
JSR 195	IMP	Information Module Profile
JSR 228	IMP-NG	Information Module Profile - Next Generation
Optional Packages		
JSR 75	PIM	PDA Optional Packages for the J2ME Platform
JSR 82	BTAPI	Java APIs for Bluetooth
JSR 120	WMA	Wireless Messaging API
JSR 205	WMA 2.0	Wireless Messaging API 2.0

JSR 135	MMAPI	Mobile Media API
JSR 164		JAIN SIMPLE Presence
JSR 165		JAIN SIMPLE Instant Messaging
JSR 172		J2ME Web Services
JSR 177	SATSA	Security and Trust Services API for J2ME
JSR 179		**Location API for J2ME**
JSR 180	SIP	SIP API for J2ME
JSR 184	3D	Mobile 3D Graphics API for J2ME
JSR 186		JAIN Presence
JSR 187		JAIN Instant Messaging
JSR 190		Event Tracking API for J2ME
JSR 209		Advanced Graphics and User Interface Optional Package for J2ME Platform
JSR 211	CHAPI	Content Handling API
JSR 213		Micro WSCI Framework for J2ME
JSR 214		Micro BPSS for J2ME Devices
JSR 226		Scalable 2D Vector Graphics API
JSR 229		Payment API
JSR 230		Data Sync API
JSR 232		Mobile Operational Management
JSR 234		Advanced Multimedia Supplements
JSR 238		Mobile Internationalization API
JSR 239		Java Bindings for OpenGL ES
JSR 246		Device Management API
JSR 253		Mobile Telephony API (MTA)

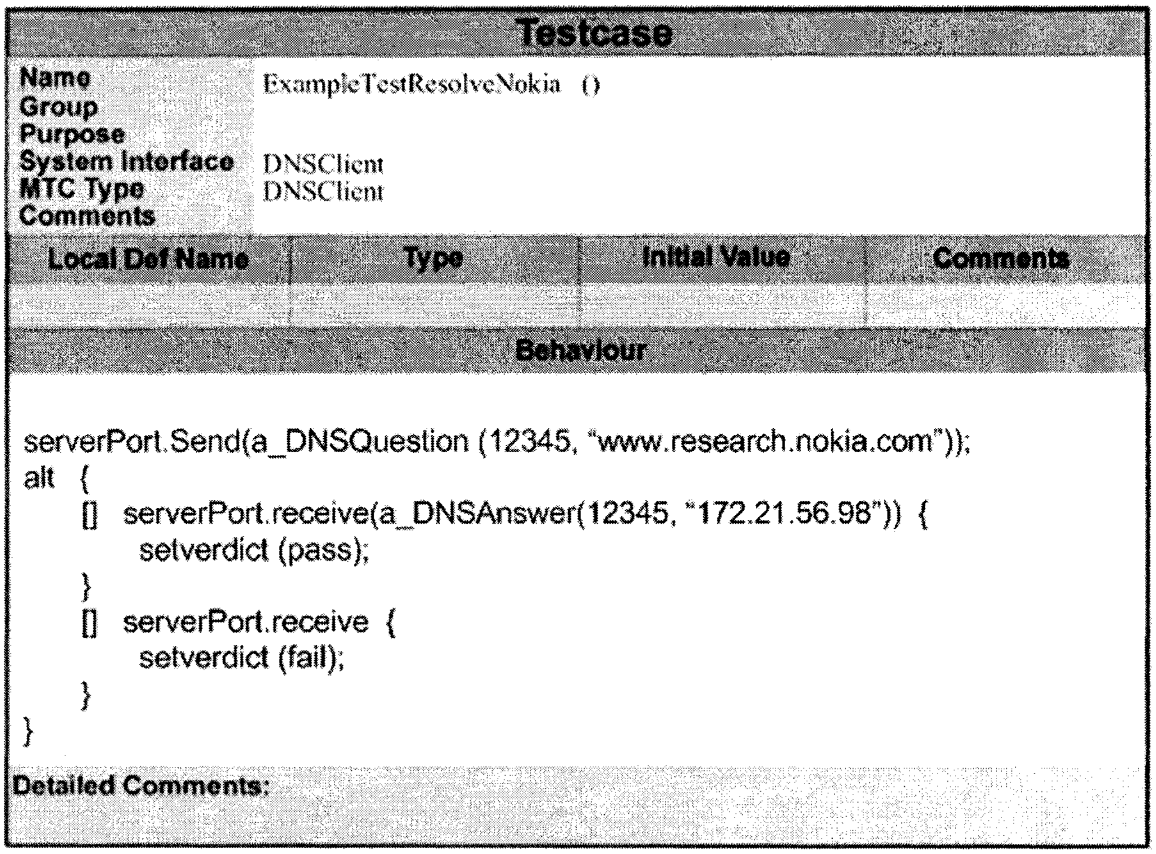
Appendix B: Current OS in the Mobile Devices



OS	Device	Browser	Browsing Comments	SSL	HTML	WAP (HDML/WML)	XML	IMODE/ (cHTML)	Cookies	Scripts	ActiveX	Java Applet	Proxy Server	Tables	Images
Palm	Palm OS and Compatible	AvantGo	OnLine, Offline channels	Yes	Yes	No	No	No	Yes	No	No	No	Yes	Yes	Yes
		EndoraWeb	Online	Yes	Yes	No	No	Yes	Yes	No	No	No	Optional	No	No
		PocketLink	Online, cached contents for offline viewing	N/A	Yes	N/A	No	Yes	Yes	No	No	No	Yes	No	Yes
		Xlino (PalmScape)	OnLine, Offline	Yes	Yes	No	No	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes
		Blazer	Online	Yes	Yes	Yes	No	Yes	Yes	No	No	No	Optional	Yes	Yes
		Wapaka	Online	No	No	Yes	No	No	N/A	Yes	No	No	Optional	No	No
		Go.web	Online, Web Clipping	N/A	Yes	Yes	No	N/A	N/A	No	No	No	Yes	Yes	Yes
Windows CE	PocketPC and Compatible	IE 2002	OnLine, Offline channels	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Optional	Yes	Yes
		AvantGo	OnLine, Offline channels	Yes	Yes	No	No	N/A	Yes	No	No	No	Yes	Yes	Yes
		Go.Web	Portal	Yes	Yes	N/A	No	No	Yes	N/A	N/A	N/A	Yes	Yes	Yes
Blackberry	Blackberry series	Blackberry series	Online, Offline, Push alerts	Yes	Yes	Yes	No	No	Yes	Yes	No	No	Yes	Yes	Yes
			Online, Offline, Push alerts	Yes	Yes	Yes	No	No	Yes	N/A	No	No	Yes	Yes	Yes
Symbian	Psion Revo Plus	Psion Ravo Plus	Online	Yes	Yes	No	N/A	N/A	Yes	Yes	N/A	Yes	N/A	Yes	Yes

N/A: Data was not available

Appendix C: TTCN-3 Tabular Presentation Format Example



Testcase

Name	ExampleTestResolveNokia ()
Group	
Purpose	
System Interface	DNSClient
MTC Type	DNSClient
Comments	

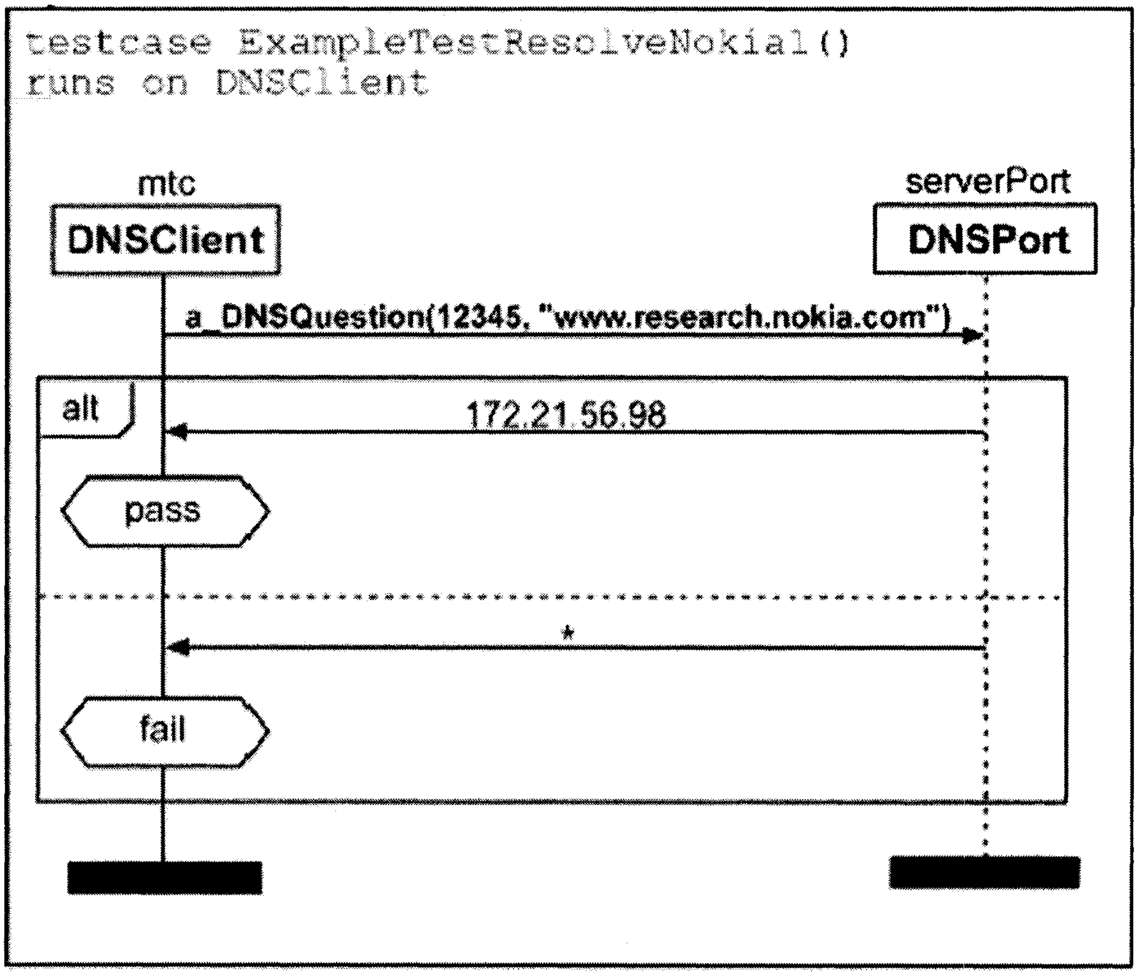
Local Def Name	Type	Initial Value	Comments

Behaviour

```
serverPort.Send(a_DNSQuestion (12345, “www.research.nokia.com”));
alt {
    [] serverPort.receive(a_DNSAnswer(12345, “172.21.56.98”)) {
         setverdict (pass);
    }
    [] serverPort.receive {
         setverdict (fail);
    }
}
```

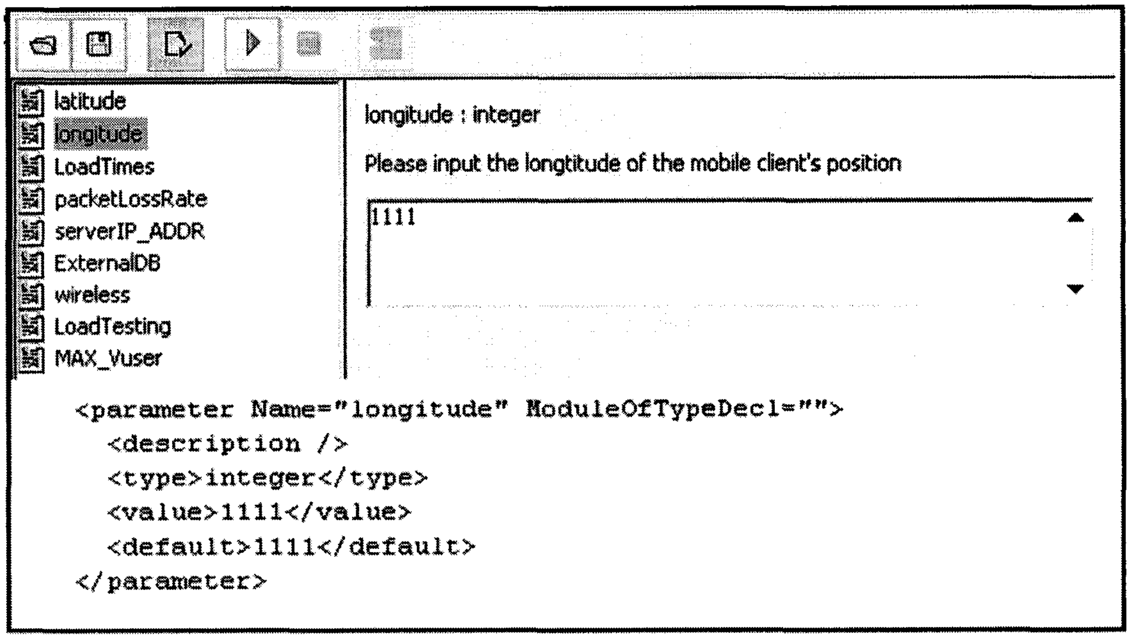
Detailed Comments:

Appendix D: TTCN-3 Graphical Presentation Format Example


testcase ExampleTestResolveNokia1()
runs on DNSClient
mtc
DNSClient
serverPort
DNSPort
a_DNSQuestion(12345, "www.research.nokia.com")
alt
172.21.56.98
pass
*
fail

Appendix E: The Parameter Definition and Interface

The latitude parameter definition and interface



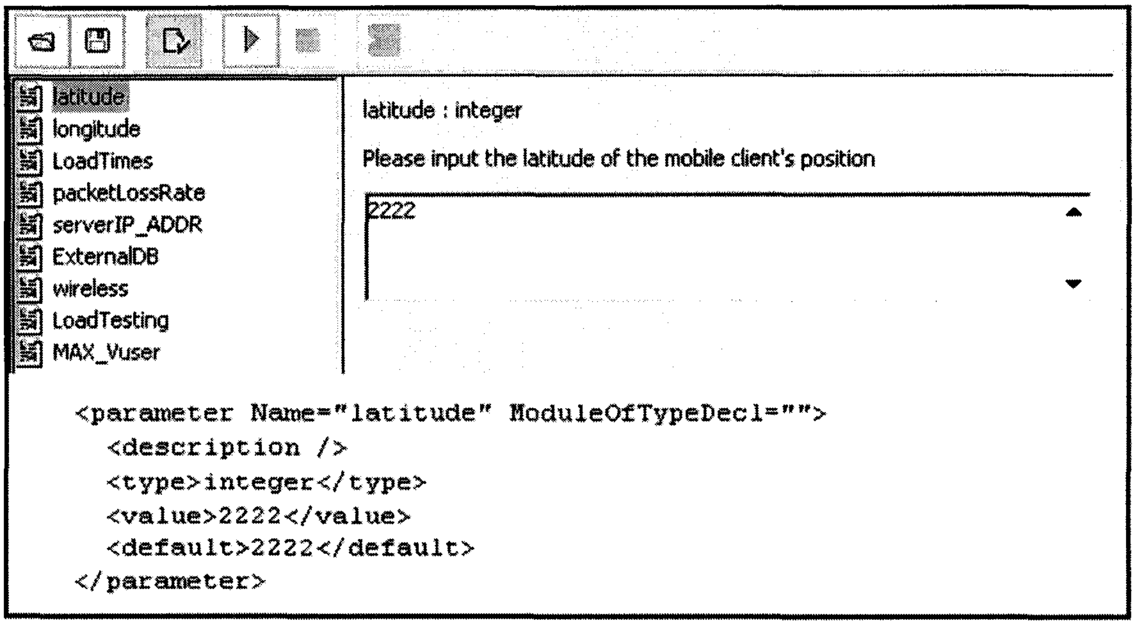
longitude : integer

Please input the longtitude of the mobile client's position

1111

```
<parameter Name="longitude" ModuleOfTypeDecl="">
  <description />
  <type>integer</type>
  <value>1111</value>
  <default>1111</default>
</parameter>
```

The longitude parameter definition and interface



latitude : integer

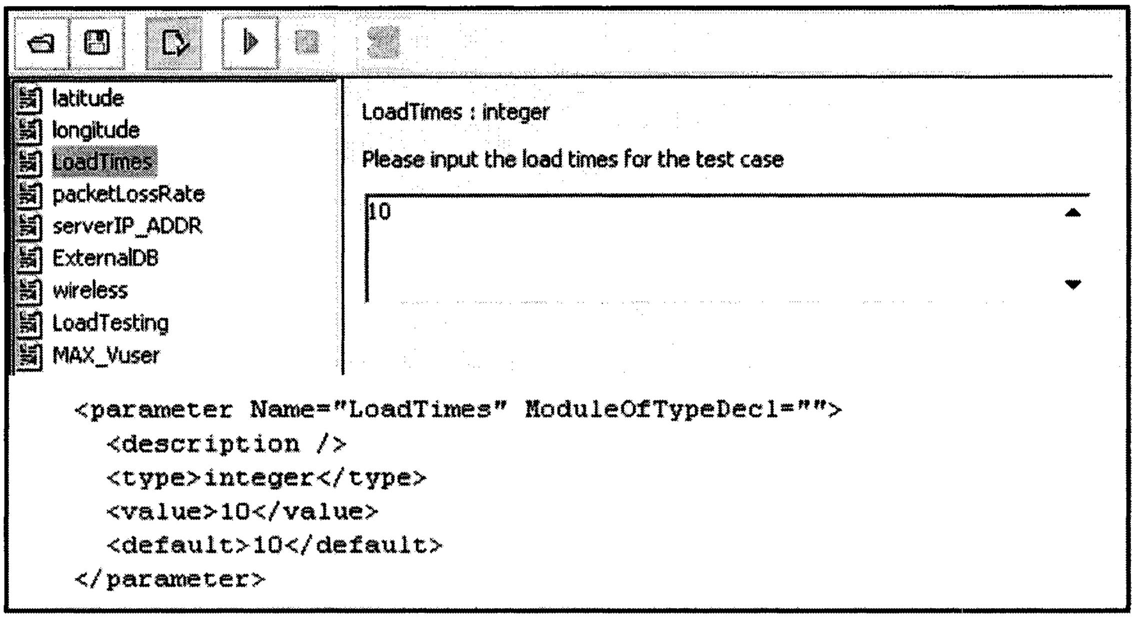
Please input the latitude of the mobile client's position

2222

```
<parameter Name="latitude" ModuleOfTypeDecl="">
  <description />
  <type>integer</type>
  <value>2222</value>
  <default>2222</default>
</parameter>
```

Appendix E: The Parameter Definition and Interface (2)

The LoadTimes parameter definition and interface



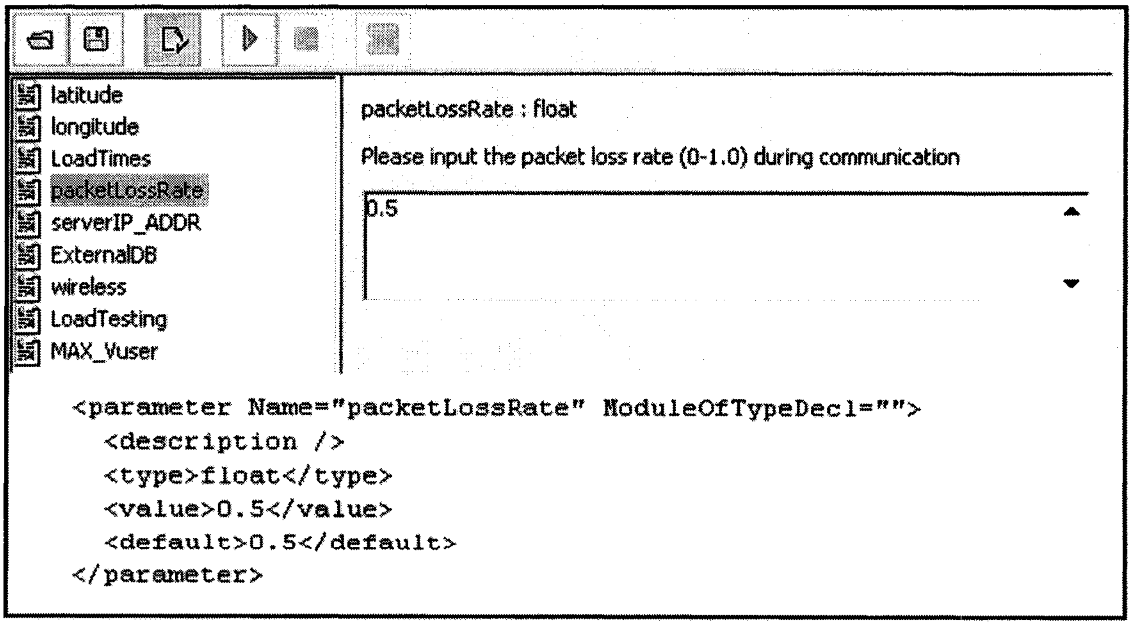
LoadTimes : integer

Please input the load times for the test case

10

```
<parameter Name="LoadTimes" ModuleOfTypeDecl="">
  <description />
  <type>integer</type>
  <value>10</value>
  <default>10</default>
</parameter>
```

The packetLossRate parameter definition and interface



packetLossRate : float

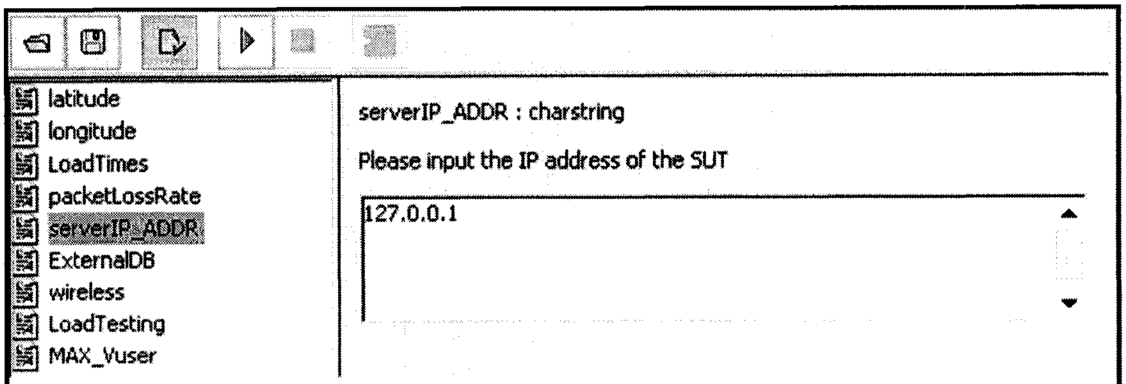
Please input the packet loss rate (0-1.0) during communication

0.5

```
<parameter Name="packetLossRate" ModuleOfTypeDecl="">
  <description />
  <type>float</type>
  <value>0.5</value>
  <default>0.5</default>
</parameter>
```

Appendix E: The Parameter Definition and Interface (3)

The serverIP_ADDR parameter definition and interface



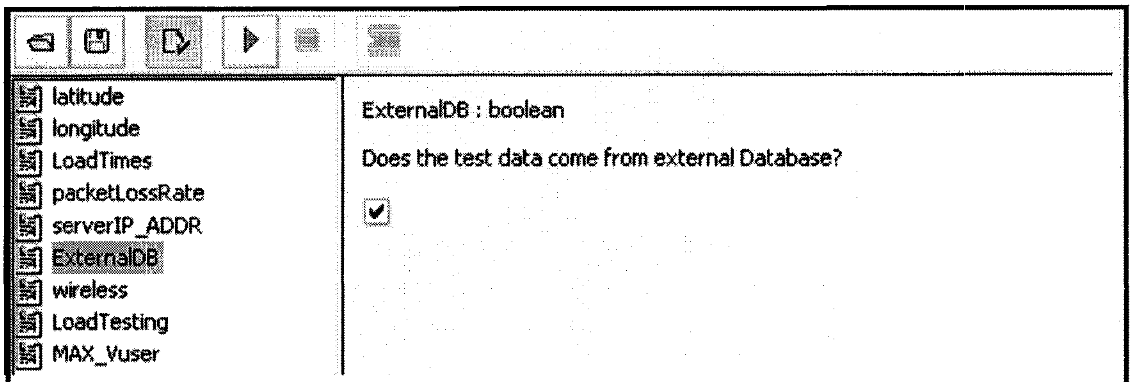
serverIP_ADDR : charstring

Please input the IP address of the SUT

127.0.0.1

```
<parameter Name="serverIP_ADDR">
    <description>Please input the IP
        address of the SUT</description>
    <type>charstring</type>
    <value>127.0.0.1</value>
    <default>127.0.0.1</default>
</parameter>
```

The ExternalDB parameter definition and interface



ExternalDB : boolean

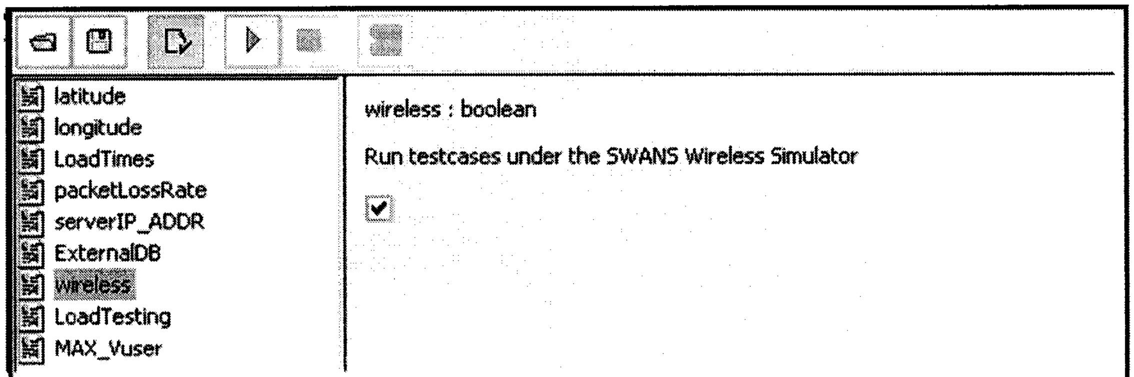
Does the test data come from external Database?

☑

```
<parameter Name="ExternalDB">
    <description>Does the test data come
        from external Database?</description>
    <type>boolean</type>
    <value>true</value>
    <default>true</default>
    <presentation>
        <boolean/>
    </presentation>
</parameter>
```

Appendix E: The Parameter Definition and Interface (4)

The wireless parameter definition and interface



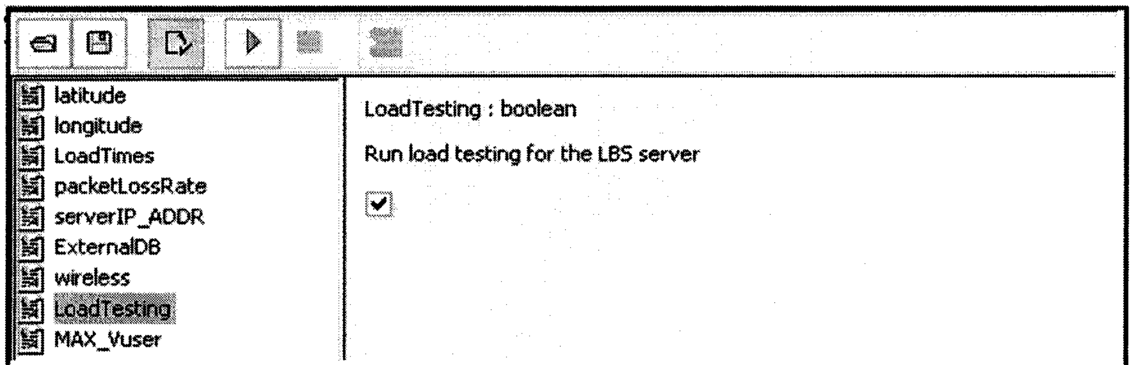
wireless : boolean

Run testcases under the SWANS Wireless Simulator

☑

```
<parameter Name="wireless">
    <description>Run testcases under the SWANS
        Wireless Simulator</description>
    <type>boolean</type>
    <value>false</value>
    <default>false</default>
    <presentation>
        <boolean/>
    </presentation>
</parameter>
```

The LoadTesting parameter definition and interface



LoadTesting : boolean

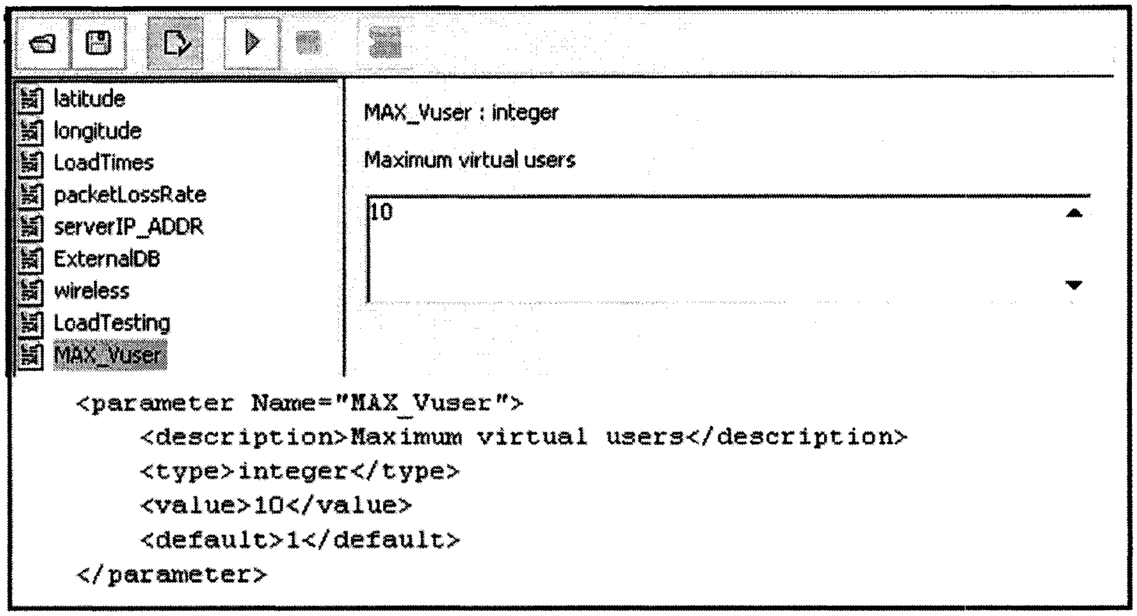
Run load testing for the LBS server

☑

```
<parameter Name="LoadTesting">
    <description>Run load testing for
        the LBS server</description>
    <type>boolean</type>
    <value>false</value>
    <default>false</default>
    <presentation>
        <boolean/>
    </presentation>
</parameter>
```

Appendix E: The Parameter Definition and Interface (5)

The MAX_Vuser parameter definition and interface



MAX_Vuser : integer

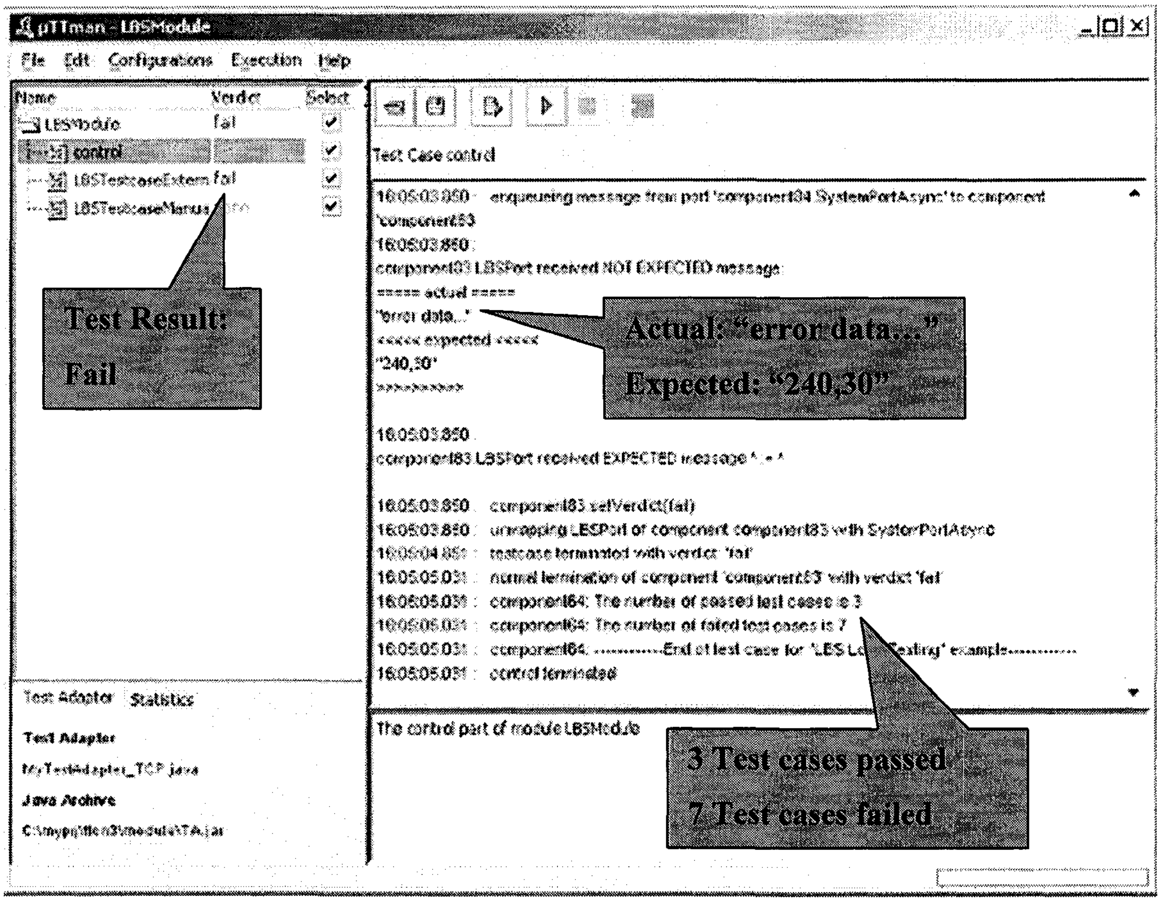
Maximum virtual users

10

```
<parameter Name="MAX_Vuser">
    <description>Maximum virtual users</description>
    <type>integer</type>
    <value>10</value>
    <default>1</default>
</parameter>
```

Appendix F: The Snapshots of Test Execution

The Test Result of Test Case TC004



Name	Verdict	Select
LBSModule	fail	✓
control		✓
LBSTestcaseExtern	fail	✓
LBSTestcaseManua		✓

Test Case control

```
16:05:03.850 :  enqueueing message from port 'component84.SystemPortAsync' to component
'component83
16:05:03.850 :
component83 LBSPort received NOT EXPECTED message:
===== actual =====
"error data..."
<<<<< expected <<<<<
"240,30"
>>>>>>>>>>

16:05:03.850 :
component83 LBSPort received EXPECTED message ^_^

16:05:03.850 :  component83.setVerdict(fail)
16:05:03.850 :  unmapping LBSPort of component component83 with SystemPortAsync
16:05:04.851 :  testcase terminated with verdict 'fail'
16:05:05.031 :  normal termination of component 'component83' with verdict 'fail'
16:05:05.031 :  component84: The number of passed test cases is 3
16:05:05.031 :  component84: The number of failed test cases is 7
16:05:05.031 :  component84: ............End of test case for 'LBS Lo[illegible] Testing' example............
16:05:05.031 :  control terminated
```

The control part of module LBSModule

Test Adapter

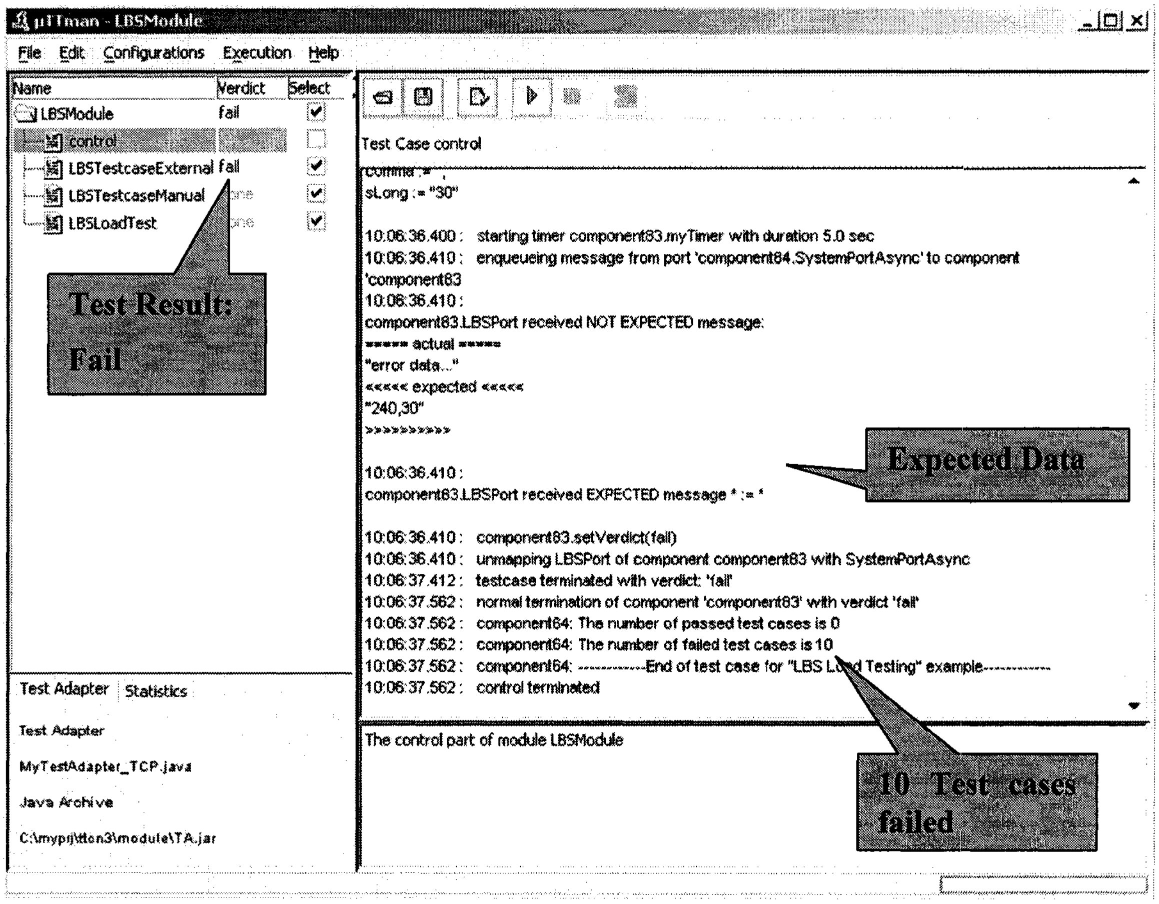
MyTestAdapter_TCP.java

Java Archive

C:\mypj\ttcn3\module\TA.jar

Appendix F: The Snapshots of Test Execution (2)

The Test Result of Test Case TC005



Name	Verdict	Select
LBSModule	fail	☑
control		☐
LBSTestcaseExternal	fail	☑
LBSTestcaseManual	one	☑
LBSLoadTest	one	☑

Test Adapter

MyTestAdapter_TCP.java

Java Archive

C:\myprj\tton3\module\TA.jar

Test Case control

```
comma := ,
sLong := "30"

10:06:36.400 :   starting timer component83.myTimer with duration 5.0 sec
10:06:36.410 :   enqueueing message from port 'component84.SystemPortAsync' to component
'component83
10:06:36.410 :
component83.LBSPort received NOT EXPECTED message:
===== actual =====
"error data..."
<<<<< expected <<<<<
"240,30"
>>>>>>>>>>

10:06:36.410 :
component83.LBSPort received EXPECTED message * := *

10:06:36.410 :   component83.setVerdict(fail)
10:06:36.410 :   unmapping LBSPort of component component83 with SystemPortAsync
10:06:37.412 :   testcase terminated with verdict: 'fail'
10:06:37.562 :   normal termination of component 'component83' with verdict 'fail'
10:06:37.562 :   component64: The number of passed test cases is 0
10:06:37.562 :   component64: The number of failed test cases is 10
10:06:37.562 :   component64: ------------End of test case for "LBS Load Testing" example------------
10:06:37.562 :   control terminated
```

The control part of module LBSModule

Test Result: Fail

Expected Data

10 Test cases failed

Appendix G: The Test Result of TC008-TC012

```
Database connection established

The Frequency of BaseStation1 is: 150.0 MHz
The Effective Height of BaseStation1 is: 200.0 M
The Receiver Height of BaseStation1 is: 2.0 M
The Maximum Allowable Path Loss of BaseStation1 is: 135.0 dB
The Range of Base Station #1 is: 121 KM

The Frequency of BaseStation2 is: 150.0 MHz
The Effective Height of BaseStation2 is: 200.0 M
The Receiver Height of BaseStation2 is: 2.0 M
The Maximum Allowable Path Loss of BaseStation2 is: 135.0 dB
The Range of Base Station #2 is: 121 KM

Starting the SWANS Wireless Simulator...

Server1 starting at t=0 on position (300,100)
Server2 starting at t=1 on position (500,100)
Mobile Client starting at t=2 on position (160,50)

Mobile Client sent at t=2; Position: 160,50
             distance1: 149KM; distance2: 344KM        TC012
The Mobile Client is out of range to two servers

Mobile Client sent at t=1000002; Position: 180,50
             distance1: 130KM; distance2: 324KM        TC012
The Mobile Client is out of range to two servers

Mobile Client sent at t=2000002; Position: 200,50
             distance1: 112KM; distance2: 304KM        TC008
Server1 received No.1 packet at t=2005002 : 200,50

Mobile Client sent at t=3000002; Position: 250,50
             distance1: 71KM; distance2: 255KM         TC008
Server1 received No.2 packet at t=3005002 : 250,50

Mobile Client sent at t=4000002; Position: 300,50
             distance1: 50KM; distance2: 206KM         TC008
Server1 received No.3 packet at t=4005002 : 300,50

Mobile Client sent at t=5000002; Position: 350,50
             distance1: 71KM; distance2: 158KM         TC008
Server1 received No.4 packet at t=5005002 : 350,50

Mobile Client sent at t=6000002; Position: 380,50
             distance1: 94KM; distance2: 130KM         TC008
Server1 received No.5 packet at t=6005002 : 380,50

Mobile Client sent at t=7000002; Position: 400,50
             distance1: 112KM; distance2: 112KM        TC009
Server1 received No.6 packet at t=7005002 : 400,50
```

```
Mobile Client sent at t=8000002; Position: 410,50           TC010
             distance1: 121KM; distance2: 103KM
Server2 received No.1 packet at t=8005002 : 410,50

Mobile Client sent at t=9000002; Position: 430,50           TC011
             distance1: 139KM; distance2: 86KM
Server2 received No.2 packet at t=9005002 : 430,50

Mobile Client sent at t=10000002; Position: 480,50          TC011
             distance1: 187KM; distance2: 54KM
Server2 received No.3 packet at t=10005002 : 480,50

Mobile Client sent at t=11000002; Position: 550,50          TC011
             distance1: 255KM; distance2: 71KM
Server2 received No.4 packet at t=11005002 : 550,50

Mobile Client sent at t=12000002; Position: 600,50          TC011
             distance1: 304KM; distance2: 112KM
Server2 received No.5 packet at t=12005002 : 600,50

Mobile Client sent at t=13000002; Position: 620,50          TC012
             distance1: 324KM; distance2: 130KM
The Mobile Client is out of range to two servers

Mobile Client sent at t=14000002; Position: 640,50          TC012
             distance1: 344KM; distance2: 149KM
The Mobile Client is out of range to two servers

Mobile Client sent at t=15000002; Position: 660,50          TC012
             distance1: 363KM; distance2: 168KM
The Mobile Client is out of range to two servers


terminate packet sent at t=15000002
```

Appendix H: Test Cases of Use Case 1 – Activation of the MST

1. Basic Flow and Optional Flows

<table>
<tr><th>Step</th><th>Basic Flow 1 (BF1) - User activates the Mobile Service Tracking system</th></tr>
<tr><td>1</td><td>User wants to activate the Mobile Service Tracking (MST) system.
- Input Password</td></tr>
<tr><td>2</td><td>Server validates user's information to see if it is the authorized user.</td></tr>
<tr><td>3</td><td>Server activates the user's MST system.</td></tr>
<tr><td>4</td><td>Server sends one response message back to the user.
- "MTS is activated"</td></tr>
<tr><td>5</td><td>The available services are displayed in the screen.</td></tr>
<tr><td colspan="2"><b>Optional Flow 1 (OF1) - Incorrect password at step 1</b>
- Server sends validation information "Incorrect Password" to the user.
- User has three chances to input the password. If there is still chance to input the password, the flow will go back to Step 1 of the Basic Flow 1.
- If the third type of password is wrong, Server sends “MST will be locked” and the case is finished.</td></tr>
<tr><td colspan="2"><b>Optional Flow 2 (OF2) – Weak wireless signal when the user sends the request at step 1</b>
- The user’s request will be discarded after time out. The case is finished.</td></tr>
<tr><td colspan="2"><b>Optional Flow 3 (OF3) – Weak wireless signal when the user receive the response at step 5</b>
- Server will repeat step 4 after time out for 3 times.</td></tr>
</table>

2. Test Scenarios

Test Scenario	Flow	
SA01 - Successfully activate the MST	BF1	
SA02 - Incorrect password (user still has input chance)	BF1	OF1
SA03 - Incorrect password (no input chance)	BF1	OF1
SA04 - Weak wireless signal when user sends request	BF1	OF2
SA05 - Weak wireless signal when user receives response	BF1	OF3

3. Test Cases

Test Case	Test Scenario	Password	Packet Loss Rate	Expected Result
CA01	SA01	4829	0	MST is activated, and available services are displayed in the mobile device screen.
CA02	SA02	4800	0	“Incorrect Password”, go back to step 1
CA03	SA02	4801	0	“Incorrect Password”, go back to step 1
CA04	SA03	4802	0	“MST will be locked”. The test case is finished
CA05	SA04	4829	1.0	Time out. The test case is finished
CA06	SA05	4829	1.0	Time out and go back to step 4
CA07	SA05	4829	1.0	Time out and go back to step 4
CA08	SA05	4829	1.0	Time out and the test case is finished

Appendix I: Test Cases of Use Case 2 – Deactivation of the MST

1. Basic Flow and Optional Flows

<table>
<tr><th>Step</th><th>Basic Flow 1 (BF1) - User disables the Mobile Service Tracking system</th></tr>
<tr><td>1</td><td>User disables the MST when he would not like to be tracked by other people.
- Select "Disable MST" from the menu on the screen</td></tr>
<tr><td>2</td><td>Server sends the message to confirm the user’s request – “Do you want to disable MST?”</td></tr>
<tr><td>3</td><td>User selects “Yes” from the menu to confirm his decision.</td></tr>
<tr><td>4</td><td>Server stops the MST and the user’s position is then concealed against other people.</td></tr>
<tr><td>5</td><td>Server sends one response message "MST is disabled" back to the user.</td></tr>
<tr><td>6</td><td>The "MST is disabled" is displayed in the screen.</td></tr>
<tr><td colspan="2"><b>Optional Flow 1 (OF1) – User changes his decision at step 3</b>
- User selects “No” at step 3
- Server receives “No” message from the user and still leaves MST working. The case is finished</td></tr>
<tr><td colspan="2"><b>Optional Flow 2 (OF2) – Weak wireless signal when the user sends the request at step 1</b>
- The user’s request will be discarded after time out. The case is finished.</td></tr>
<tr><td colspan="2"><b>Optional Flow 3 (OF3) – Weak wireless signal when the user sends the request at step 3</b>
- The flow will repeat step 3 after time out, waiting for the user’s confirmation</td></tr>
<tr><td colspan="2"><b>Optional Flow 4 (OF4) – Weak wireless signal when the user receives the response at step 2</b>
- The flow will repeat step 2 after time out for 3 times. After 3 times, the Server will discard the user request.</td></tr>
<tr><td colspan="2"><b>Optional Flow 5 (OF5) - Weak wireless signal when the user receives the response at step 6</b>
- The flow will repeat step 5 after time out until the user gets the message.</td></tr>
</table>

2. Test Scenarios

Test Scenario	Flow	
SA06 - Successfully disable the MST	BF1	
SA07 – User changes his decision during the confirmation of deactivation of the MST	BF1	OF1
SA08 - Weak wireless signal when user sends request at step 1	BF1	OF2
SA09 - Weak wireless signal when user sends request at step 3	BF1	OF3
SA10 - Weak wireless signal when user receives response at step 2	BF1	OF4
SA11 - Weak wireless signal when user receives response at step 6	BF1	OF5

3. Test Cases

Test Case	Test Scenario	User Request	Request Confirmation	Packet Loss Rate	Expected Result
CA09	SA06	Disable MST	Yes	0	MST is disabled
CA10	SA07	Disable MST	No	0	The MST keeps activated and the test case is finished
CA11	SA08	Disable MST	N/A	1.0	Time out. The test case is finished
CA12	SA09	Disable MST	Yes	1.0	Time out and go back to step 3
CA13	SA10	Disable MST	N/A	1.0	Time out and go back to step 2. Still have two more chances of sending confirmation
CA14	SA 10	Disable MST	N/A	1.0	Time out and go back to step 2. Have one last

					chance of sending confirmation
CA15	SA 10	Disable MST	N/A	1.0	Time out and the test case is finished
CA16	SA 10	Disable MST	N/A	0	The wireless signal changes to strong status at the3rd sending request. The user receives confirmation message “Do you want to disable MST?”. Go back to step 3
CA17	SA 11	Disable MST	Yes	1.0	Time out and go back to step 5

Appendix J: Test Cases of Use Case 3 – Where am I?

1. Basic Flow and Optional Flows

Step	**Basic Flow 1 (BF1) - User wants to know his/her current location.**
1	User accesses "Directory Service" - Select "Where am I" from the menu
2	Server sends a location request to the location management platform (LMP). - The context simulator (CS) is used as LMP to provide user's location coordinate (X,Y). - Send "User ID", "Service Provider ID" to the CS for the location request
3	The CS validates the authorization of the user's location request from the service provider.
4	The CS sends the user's location coordinate (X,Y) and timestamp to the Server. - The timestamp is the time at which the object is positioned
5	Server validate the correctness of user's location information from the CS.
6	Server forwards the location coordinate (X,Y) to the content provider. - In reality, the location coordinate (X,Y) must be translated into the real address, i.e. 10021 Jasper Ave., by third party content providers.
7	The content provider maps the location coordinate to the real address and sends it back to the Server
8	Server forwards the location information and timestamp to the user.
9	User receives the location information and timestamp on the screen.
Optional Flow 1 (OF1) – Unauthorized location request at step 3 - The Server cannot pass the authorization of the user's location request by the CS because of incorrect "User ID" or "Service Provider ID". - The Sever receives one message from LMP – "Unauthorized user xxx's location request by Service Provide xxx" - Go back to step 2.	

Optional Flow 2 (OF2) – Invalid location information at step 5

- LMP sends incorrect location coordinate, i.e. latitude/timestamp is missed, to the Server.
- The Server sends response message to the CS – “Invalid location coordinate” or “Invalid timestamp”. Go back to step 4.

Optional Flow 3 (OF3) – Weak wireless signal when the user sends the request at step 1

- The use request will be repeated for 3 times after time out and the request will be discarded after 3 times repeat. The case is finished

Optional Flow 4 (OF4) – Weak wireless signal when the user receives the response from the Server at step 9

- The flow will go back step 8, and the Server will repeat sending response information until the user gets the message.

2. Test Scenarios

Test Scenario	Flow	
SA12 – Successfully locate the user and deliver the location information to the user	BF1	
SA13 – Unauthorized location request at step 3	BF1	OF1
SA14 - Invalid location information at step 5	BF1	OF2
SA15 - Weak wireless signal when user sends directory service request	BF1	OF3
SA16 - Weak wireless signal when user receives location response	BF1	OF4

3. Test Cases

Test Case	Test Scenario	User Service Request	User ID	Service Provider ID	Location Coordinate	Timestamp hh:mm:ss;MM/DD/YY	Packet Loss Rate	Expected Result

CA18	SA12	Where am I	1201	168	(100, 200)	13:00:00;07/03/2005	0	User receives one location information “11015 Jasper Ave., Edmonton, at 13:00:00, July 3, 2005
CA19	SA13	Where am I	1200	168	N/A	N/A	0	Unauthorized user 1200’s location request by service provider 168. Go back to step 2
CA20	SA13	Where am I	1201	167	N/A	N/A	0	Unauthorized user 1201’s location request by service provider 167. Go back to step 2
CA21	SA14	Where am I	1201	168	(null,200)	13:00:00;07/03/2005	0	Invalid location information. Go back to step 4
CA22	SA14	Where am I	1201	168	(200, null)	13:00:00;07/03/2005	0	Invalid location information. Go back to step 4
CA23	SA14	Where am I	1201	168	(null,null)	13:00:00;07/03/2005	0	Invalid location information. Go back to step 4
CA24	SA14	Where am I	1201	168	(100, 200)	00:00; 07/03/2005	0	Invalid timestamp. Go back to step 4
CA25	SA14	Where am I	1201	168	(100, 200)	13:00:00;/03/2005	0	Invalid timestamp. Go back to step 4

CA26	SA14	Where am I	1201	168	(100, 200)	13:00:00 ;07/03	0	Invalid timestamp. Go back to step 4
CA27	SA14	Where am I	1201	168	(100, 200)	25:00:00 ;07/03/2005	0	Invalid timestamp. Go back to step 4
CA28	SA14	Where am I	1201	168	(100, 200)	13:00:00 ;15/03/2005	0	Invalid timestamp. Go back to step 4
CA29	SA14	Where am I	1201	168	(100, 200)	null	0	Invalid timestamp. Go back to step 4
CA30	SA15	Where am I	N/A	N/A	N/A	N/A	1	Time out. Still have 2 chances of sending requests.
CA31	SA15	Where am I	N/A	N/A	N/A	N/A	1	Time out. Have one last chance of sending requests.
CA32	SA15	Where am I	N/A	N/A	N/A	N/A	1	Time out. The request is discarded after 3 times tries. The test case is finished
CA32	SA15	Where am I	N/A	N/A	N/A	N/A	0	The wireless signal changes to strong status at the3rd sending request. Goes back to step 2
CA33	SA16	Where am I	1201	168	(100, 200)	13:00:00 ;07/03/2005	1	Time out. Server will repeat step 8 to send location information to the user.

Appendix K: SAML Authentication Assertion Request / Response Messages

SAML Authentication Assertion Request Message

```
<saml:AuthnRequest RequestID=”128.14.234.20.12345678” MajorVersion="1"
 MinorVersion="0" xmlns:samlp="urn:oasis:names:tc:SAML:1.0:protocol">
   <saml:AuthenticationQuery >
      <saml:Subject>
        <saml:NameIdentifier
              SecurityDomain=”Service.com”
              Name=”joeuser” />
       </saml:Subject>
   </saml:AuthenticationQuery>
</saml:AuthnRequest>
```

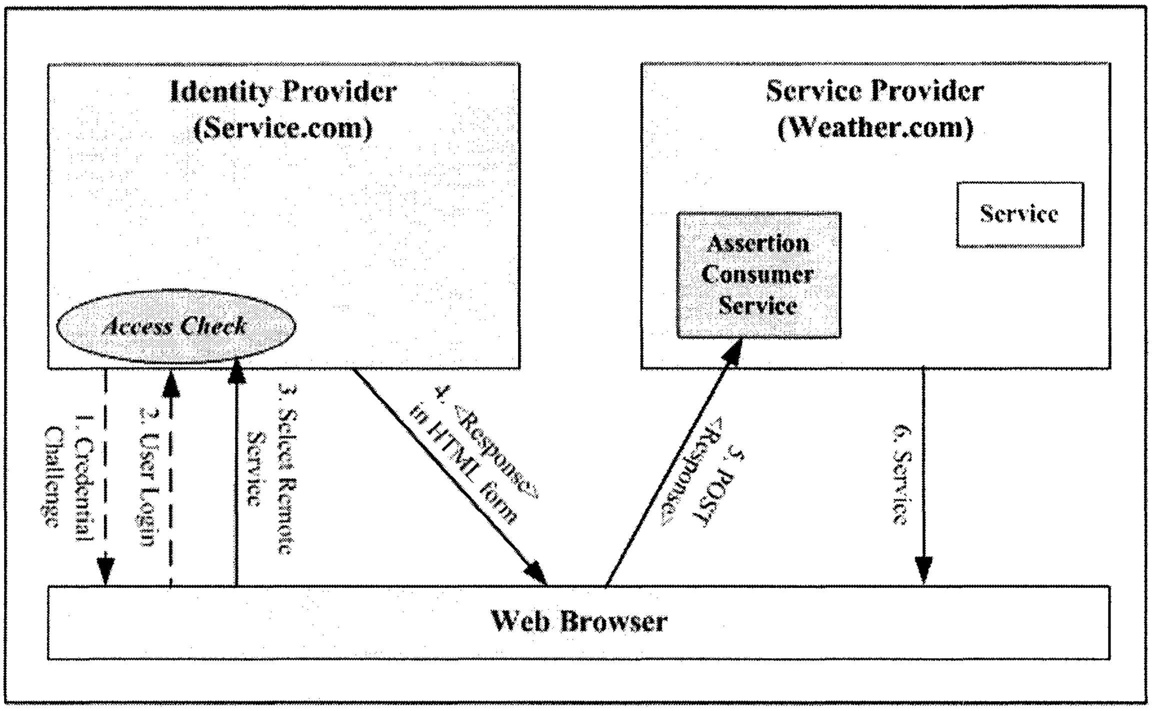
A SAML assertion is being requested pertaining to the supplied subject “joeuser”.

SAML Authentication Assertion Response Message

```
<saml:AuthnResponse xmlns:samlp="urn:oasis:names:tc:SAML:1.0:protocol">
 <saml:Status>
     <StatusCode Value =”samlp:Success”/>
 </saml:Status>
 <saml:Assertion AssertionID=”128.9.167.32.12345678” MajorVersion="1"
    MinorVersion="0" Issuer=”Service.com” IssueInstant”2005-07-20T10:02:00Z”>
    <saml:Conditions
          NotBefore="2005-07-20T10:02:00Z"
          NotOnOrAfter="2005-07-20T10:07:00Z"/>
    </saml:Conditions>
    <saml:AuthenticationStatement
          AuthenticationMethod=”password”
          AuthenticationInstant=”2005-07-20T10:02:00Z”>
          <saml:Subject>
                SecurityDomain=”Service.com”
                Name=joeuser” />
          </saml:Subject>
    </saml:AuthenticationStatement>
  </saml:Assertion>
</saml:AuthnResponse>
```

The SAML response provides details as to the version of SAML being used and what request it is responding to. Within the response is the SAML assertion and authentication statement.

Appendix L: Web Browser Single Sign-On Profile



The process is as follows:

1. The user will have been challenged to supply their credentials to the site Service.com.

2. The user successfully provides their credentials and has a security context with the Identity Provider.

3. The user selects a menu option (or function) on the displayed screen that means the user wants to access a service or application on another site Weather.com.

4. The SP sends a HTML form back to the browser. The HTML form contains a SAML response along with a SAML assertion. Typically the HTML form will contain an input or submit action that will result in a HTTP POST.

5. The browser, either due to a user action or via an “auto-submit” , issues a HTTP POST containing the SAML response to be sent to the Service provider's Assertion Consumer service.

6. The SP's Assertion Consumer service validates the assertion on the SAML Response. If assertion is correct, it sends a HTTP redirect to the browser causing it to access the required service.