

# DISTRIBUTED TOOL FOR PERFORMANCE TESTING

Nenad Stankovic

*University of Aizu  
Tsuruga, Ikki-machi, Aizu-Wakamatsu  
Fukushima 965-8580, Japan*

Email: nenado@msn.com

**Abstract:** *Voice and data networks require infrastructure components of highest quality and that is achieved through testing. However, software performance and performance testing have been less studied and reported on. This paper presents the tool developed and used in the performance and stability testing of a networks integration infrastructure.*

*We also report on the experience with our tool and performance problems related to Java. Our findings are based on the work and feedback by the programmers with a solid object-oriented background. Their experience with Java was mixed, but the tool and concepts of distributed programming were new.*

**Keywords:** Internet, Wireless, Wired telephony, Tool, Software Performance

## 1. INTRODUCTION

Over the course of the last decade or so the Internet has become a regular source of news and information, messaging, entertainment, and many business services for individual users and companies around the world. Although omnipresent, the Internet has an important drawback as it is restricted to a location. Like the Public Switched Telephone Network (PSTN) call, the lack of mobility restricts where those services can be accessed and content consumed. The proliferation of wireless telephony has created a widespread need for content to go wireless and become accessible via mobile devices. The drive towards convergence of mobile and static devices to accommodate anytime and anywhere access to multimedia communications is resulting in

new services for the user and new business opportunities for the operator.

Each of the three mentioned networks has its own characteristics and technologies as historically and physically their creation was independent, and their design and evolution was driven by different requirements. In the world of modern and integrated communications it is essential to deliver targeted and timely information and services, development of which is driven by customer needs. For seamless and reliable end-to-end services it is required to secure a transparent, performing and high quality interconnect among these networks. In turn, the interoperability between the networks and the communication between end users across boundaries of different domains require elegant technical solutions. Overall it should appear as if only one unified network exists.

The separate domains of computer, wireless, and wired telephony networks still remain with their

individual characteristics. Recently, though, they became integrated together. To enable the seamless interoperability between key applications and network domains, resolution of user identities and addressing (Figure 1), a multi-domain integration infrastructure has been developed. It is based on open standards and technologies (e.g. WAP [16], HTTP), and relevant initiatives (e.g. MTA [10], 3GPP [1]). As a result, the user is not concerned with the underlying technologies and directly enjoys the richness of the supplied services and content, irrespective of the point of access as much as possible. For example, the image captured on a multimedia enabled mobile phone can be accessed for viewing in the browser on a PC, but not on the legacy mobile phone that is only SMS enabled. However, legacy phones can receive a SMS notice about the multimedia message. In the same time, the convergence between similar services regardless of service providers has been progressing by addressing the harmonization requirements for service technologies. Also, these services and their implementation should not be tied to any proprietary or domain specific technology.

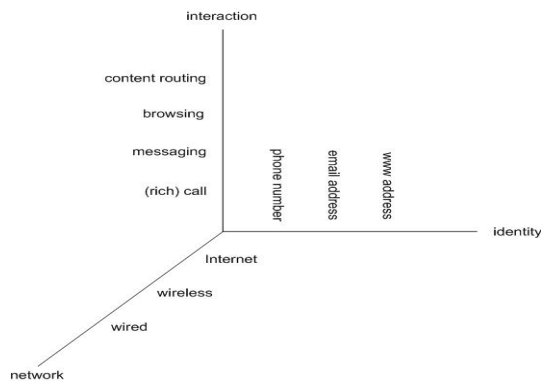


Figure 1 Networks Interoperability

The internetworking infrastructure, as dictated by the domains it integrates, is characterized by high availability, scalability, and reliability, while its pattern of usage can change rapidly and dramatically in time. It must be extremely dynamic, reactive, robust, and capable of handling thousands of different types of requests concurrently. Its performance must be predictable, stable under a wide range of loads, and ensure quality of service, all in the same time. To build this infrastructure is a formidable task that requires highly developed skills, and rapid development techniques through composability and reuse. Subsequently, functional and high volume stability and performance testing is conducted. The testing is labour and resource

intensive. It had been reported that on large software projects testing often could amount to 50 or 60 percent of the total development cost [11], even without factoring in the cost of equipment for testing. The risk of a project failing to meet its nonfunctional requirements is greatly increased without ongoing and comprehensive quality testing throughout the whole software lifecycle.

This paper is primarily concerned with the performance testing of the hardware and software required by the internetworking infrastructure. In principle, performance testing is conducted as end-to-end black box application testing where test cases are derived from the use cases and functional specification. We augment the functional tests by generating input load that is a function of time, content, and expected user behaviour. A problem with testing of such a heterogeneous collection of applications is that we need many different tools in the process. Commercial tools are not only expensive, but different tools apply different techniques in setting up test cases. They often lack portability and generate results that may not be easy to correlate and reproduce with other tools. More importantly, it may be impossible to customize and optimize their performance. For example, the capture and replay tool always picks up an attachment file from disk, rather than from an in memory cache. The disk-to-memory copy overhead has a significant negative impact on the generated load by slowing a test down. On the other hand, most test programs are structured similarly and are rather easy to code even when sophisticated.

Based on these premises we wanted to design a generic and extensible test-bed framework that facilitates the development of test programs and setting up of test cases, and resolves mentioned problems. By reusing proven software artefacts a new project or testing task does not have to reinvent and rediscover what got accomplished before. The mentioned characteristics and requirements were successfully implemented and evaluated by building a large-scale distributed tool. Built as a scalable distributed testing tool it has proven easy to adopt and suitable for rapid prototyping of test cases, with a small runtime image and very good efficiency. The framework is extensible to support new applications and programming models that require those services and abstractions.

In the section that follows we first introduce the networks integration infrastructure. In Section 3 we motivate the tool and describe the main patterns that are found in performance and stability testing. We briefly describe the framework and the components that were used to accomplish this task. Section 4 presents a complex test case that involves the main infrastructure components, as well as much of the

functionality built into the tool as to execute those performance and stability tests. Section 5 correlates our and similar research work and experience. Section 6 concludes the paper.

## 2. NETWORKS INTEGRATION INFRASTRUCTURE

Figure 2 shows the infrastructure that integrates the Internet with wireless and wired telephony networks. It consists of a number of gateways (-GW) and service centers (-SC). The reason is that historically these three domains have evolved independently, and they are not based on compatible standards or modes of operation. The basic

requirements for this infrastructure are familiar (e.g. billing, transient and persistent message storage, notifications), and they can use any available or proprietary technology deemed appropriate in the implementation. The more difficult part of this program is to provide the same content and quality of service irrespective of the domain, of course, to the extent possible due to missing support for features or physical constraints of those devices. The constraints primarily affect the multimedia content. A large number of specialized services is required to convert messages for instant access on the receiving device. Also, many communication protocols must be supported, such as SMTP, SS7 [12], T1 [13], WAP, etc. Thus, the services were separated and grouped into multiple applications that may be enabled or disabled as necessary. Therefore:

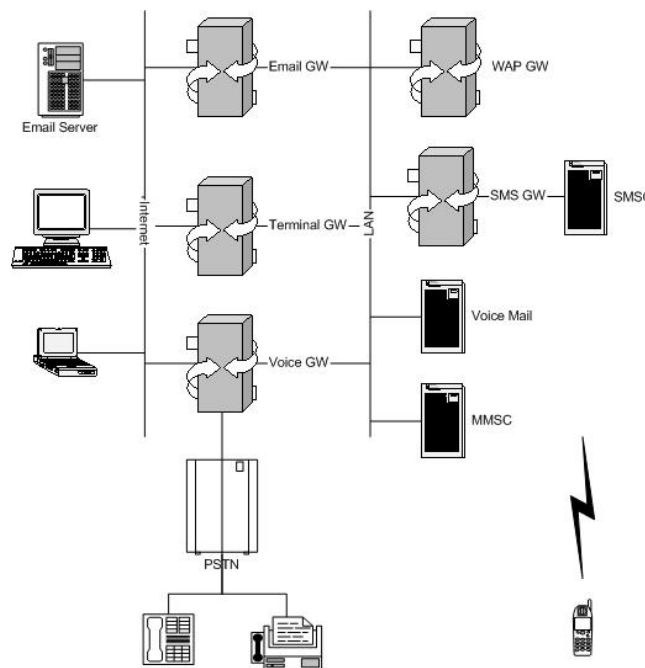


Figure 2 Networks Integration

- The Multi-Media Service Center (MMSC) is the point of connectivity into and from the wireless domain. Its main role is to receive multimedia messages, provide temporary storage and attempt to forward the message to its recipient
- The Multimedia Message Service (MMS) delivery to email address requires that the MMSC communicates with the email server. The Email GW provides the connectivity between these end points, together with routing of messages based on content and identity
- The Terminal Gateway delivers a notice of the MMS message to the legacy terminal. It stores the MMS content in its own local storage, and sends an SMS to the recipient, informing the user of a Web address where the content can be viewed in a Web browser. The Terminal GW also provides an album for messages and images. These images can be used to create a new MMS either on the mobile phone or in the browser
- The Voice GW supports unified messaging and routing of phone calls in all three domains, such

as Voice over Internet Protocol (VoIP), PSTN voice, and advance speech recognition. Instead of receiving a text notice, the voice message can be encapsulated as aMMS and sent directly to the phone

- Content conversion is important when the message is in a format that is not supported by the recipient. The MMSC routs the MMS to a content converting application, after which a new message is sent out
- The WAP Gateway receives MMS from wireless devices via a WAP Post, and transmits the MMS from the MMSC to the wireless devices via a WAP Get

The Email, Terminal and Voice GWs are known as external applications to the MMSC. They have been developed in Java and J2EE [5], while the other applications are based on C and C++ with a significant embedding into the hardware platform for better performance.

### 3. TOOL

The development work on these applications has been conducted in parallel, as has been the testing performed both by development and QA teams. This program organization requires dedicated computing resources for development and testing, specialized tools for testing, but unrestricted access could not be granted to everyone. Initially, therefore, most of the code could not be tested for performance before handed over to QA.

While coding, developers also produce programs to test their classes for correctness, and performance. Initially, there was no standardized approach to design, write, execute, and manage test artifacts that could also facilitate reuse and controlled experimentation. Although the developers spent a good portion of their time on writing test programs, generally that part of their work was not reused by their colleagues, or shared. The test programs have been developed and executed locally on the PC, because a UNIX workstation was not available for individual testing.

#### 3.1 Requirements

To resolve those shortcomings, a number of possibilities were explored and evaluated in meeting the following requirements:

- A simple model of testing that can be adopted quickly, with a framework that supports process control and configuration for rapid definition of

test cases and test profiles for performance and stability testing

- A model that does not require additional programming skills. Test scripts should be defined as text files that are interpreted by the tool. These test scripts should provide macros and configuration parameters that could be shared among multiple scripts
- Easy and safe sharing of computational resources between programmers. Easier customization and definition of new classes and libraries for new applications and services
- Dynamic configurability and transparent reassignment of resources. Test machines are a scarce resource, and they are often reassigned and reconfigured. It is not acceptable to constantly update and rebuild the test program after a change in the setup
- Scalability that allows executing thousands of user profiles and multitude of test cases concurrently, without programming intervention
- An ability to replay the trace file when reproducing a problem experienced in the field

The framework was selected over JMeter [7] and similar products also because of a simpler API, transparent support for distribution, and better performance, as explained below. Initially, the programmers were introduced to the tool in a 1 day get started course in which they were taught about the concepts and components, and how to prepare and execute a test script in a distributed environment.

#### 3.2 Patterns

Often, test programs are not complex to code but still require a considerable effort to prepare when sophisticated. This does not address refinements, distribution, and scalability issues. Without guidance by an existing body of work a test case implementer must engineer a solution from scratch, and on a case by case basis. Architecture plays a central role in software engineering by developing high level views of the system, and defining models, components and guidelines for composing systems. Further, software engineering has recognized patterns as a means to capture common and proven design concepts, to protect and reuse them [2]. A pattern can be adapted before being used, either by wrapping a new class around it, or through inheritance. When designing test programs we can identify the following common patterns:

- Configurator for reading textual configuration files and scripts that allow easy test setup with no need for recompilation
- Driver that enables execution of a test case

- Logger that provides the methods to create a standardized log file
- Profile that characterizes a test case by defining its behavior and content
- Scheduler that provides typical modes of test case invocation
- Testbed is the factory that instantiates a test case

An important pattern developed for the tool is the Scheduler pattern. It provides three modes of task invocation: uninterruptible, timer driven, and counter driven. The uninterruptible mode simply lets a task run forever. The timer driven mode stops a task after a predefined time period. In both cases the invocation of a task can be time sliced, either randomly or within a fixed time interval. The counter mode stops a task after a given number of invocations. Generally, a Scheduler is used as is, and requires no additional programming. A Scheduler becomes distributed by joining a scheduling group. Each distributed Scheduler belongs to only one group in which it coordinates activities with other members and each group behaves as a single entity.

The Profile pattern defines the virtual user. Profile is an abstract class that is closely tied to a Driver and test case. A Profile has the attributes and methods to set up a test case. Each setup script can define any number of profiles. Each Profile is assigned a thread to run on by the scheduler, and a driver by the Testbed. Different profiles may use different scheduling policy and drivers. However, this is more a matter of convenience when testing multiple applications in the same time, rather than a necessity when measuring the performance of a single application in a single end-to-end test case.

The Driver pattern defines the standard interface to a target application. A Profile requires the communication channel when simulating an external entity or user, or simply invoking a method for white or black box testing. For example, when testing the Email GW, after composing an email message with or without attachments in a Profile, an SMTP channel is established, and the message is sent to the Email GW. A library of standard Drivers and Profiles was implemented for testing of Java classes, relational databases, J2EE beans, etc. In terms of programming work, drivers are the most demanding due to their domain specific and case-by-case nature, and the number of different protocols and systems engaged by the infrastructure. However, most drivers are simple to write, as they can reuse existing code or call a standard Java APIs (e.g. JavaMail [6]). This has, nevertheless, caused problems initially because some programmers had to code the same specific drivers, before a comprehensive database of drivers got established.

The Configurator pattern provides the methods to read in and parse configuration files and scripts. These are text files with attribute value pairs, one pair per line. The same attribute can be redefined nondestructively multiple times, so that multiple profiles can be defined in each file and they can use the same attribute name. Some test cases require a more complicated mechanism than only sending messages. For example, an interaction with a sequence of actions is required between the user and service provider, such as browsing an album where the user must login and choose between the albums and messages.

Also, a man-to-machine dialogue can take place, where the dialogue is based on a grammar. For example, when leaving a message on the phone a hierarchy of menus that lead the user to a goal must be followed. Voice GW uses VoiceXML [14] to define the grammar. Rather than programming multiple profiles for such a test, the Configurator class provides the methods to parse and prepare a dialogue script for execution. Test scripts, message content (e.g. body, attachment), and configuration files can be accessed from multiple locations in the network via a specialized file server.

The Logger class defines the interface to create standardized log records suitable for online viewing or analysis in a spreadsheet. When a test case executes it generates the timestamps that are used in performance analysis. Those data can be saved to file or redirected to a console for online monitoring.

### 3.3 Distributed Framework

This tool is based on a novel object-oriented framework implemented in Java that identifies and enhances common services, and implements a generic set of classes applicable to multiple programming tasks in a distributed environment. It emphasizes separation of concerns in the design and hierarchy of layers in the implementation. Reuse of components facilitates easy customizability and adaptability to different environments and application needs. It also features grouping of distributed collaborating objects, and a session layer that enables seamless and safe sharing of resources among a number of developers.

The tool does not support graphical features in the UI. Instead all results are presented as text to the user. The reason is that graphics, although it enhances the user experience, does not bring a substantial value to the outcome of a test that would compensate for the computational cost it introduces. We believe that the CPU time spent on graphics could be better used to generate more load and thus a more realistic and reliable test result.

## 4. EXAMPLE

Performance testing is based on a number of repetitive tasks, with a multitude of user profiles. With application testing, the number of test cases is defined by the number of use cases which is not very large. However, for any realistic performance testing the number of user profiles per test case should be large to simulate closely the real world. Rather than repeating a small number of positive test cases over and over, it is necessary to engage a large number of profiles with a different content and type of user-to-system interaction as to produce a result of value.

Thus, errors are also of interest but they must be applied to the extent that would not exaggerate their impact. For example, it is possible to mistype the email address or telephone number, or to create a voice message that got aborted or that exceeded the maximum allowed length. Also, the user may pick the wrong menu option, and must repeat the process starting from a previous correct option. Modern service systems provide enough flexibility to minimize the occurrence and therefore impact of such user errors on system performance and user experience.

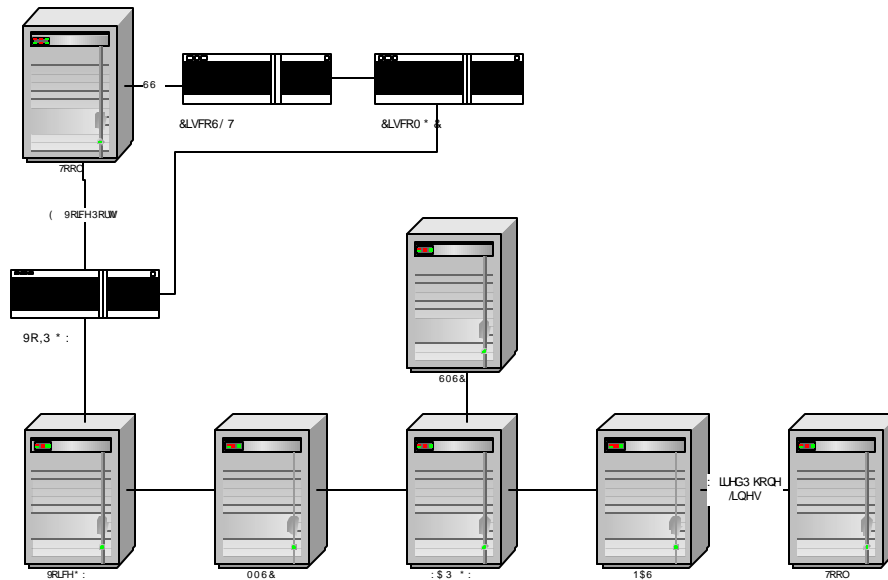


Figure 3 Voice GW Testing Setup

The configuration in Figure 3 has been used in the testing of the Voice GW for performance and stability. This configuration is rather complex, since it involves many distributed software systems and hardware components. It has multiple points of input for load generation, such as the wired telephony, VoIP, and wireless messaging. In terms of development work, this was the most demanding task taking almost 2 man months to complete the VoIP and PSTN drivers. The complexity was due to the usage of native libraries for SS7 and to simulate incoming PSTN lines that had to be linked into the Java Virtual Machine [8]. Also, the software for voice activity detection had to be incorporated.

The tool sends MMS messages to the MMSC, simulates callers who are dialing Voice GW subscribers, reroutes the calls to the Voice GW, and plays the voice messages for recording. The Voice

GW plays back the voice, text and fax messages to subscribers. It sends MMS messages to multimedia terminals via the WAP GW.

As mentioned before, the standard handling of configuration and script files had to be improved to support dialogues with speech and DTMF grammars. For example, the legacy user dials a mobile phone number, and the call gets rerouted to a call center that starts playing a menu for leaving a voice message. The test script must be able to select the menu option, detect when there is no speech activity on the channel, wait for tone, record the message, save it or even modify it before a save option is selected. Also, an option to verify that the correct menu prompt was played by the Voice GW could be used in the dialogue.

## 5. RELATED WORKS

Weyuker and Vokolos [17] report that not many case studies from industry have been published in domain of software performance testing and tools. Interestingly, their work is also rooted in telephony, but their contribution is mainly in the domain of performance testing per se rather than testing tool development. While they made recommendation on how to approach software performance problems, the reported findings are not based on a whole software lifecycle. They experimented with an existing gateway system that was being redeployed on a new platform. Nevertheless, we share their conclusion that *performance testing differs in many ways from functional testing*.

Menasce [9] reports on mission critical Web-based applications and defines the metrics and reports on the results that are relevant in assessing those systems. While his recommendations are basic but generally relevant, the applications that are reported on are from E-commerce, and some noncritical Web applications. On the other hand, it would be also interesting to extend the work to asynchronous applications because they require a different approach in end-to-end testing.

Recently, Gorton and Liu [4] investigated different J2EE compliant middleware and found that their adoption and use is not straightforward, which complies with our findings. The BEA, Borland, IBM middleware yielded similar performance results. The more interesting question remains, as to how much of that raw performance can be exploited and retained when business specific code is added. The problem is not only how computationally expensive is each use case, what is the background noise, but also how to identify and mitigate the bad design and implementation decisions.

## 6. CONCLUSION

In this paper we have focused on presenting and evaluating the distributed test-bed framework in a production environment, where it served as a basis for building a comprehensive set of performance testing tools and test programs. The tool evolved together with this complex program of software and hardware applications, and although it proved a difficult task at times, it fulfilled its original objectives. It managed to stay ahead in development as compared to the other projects, and did not cause any delays in the overall program schedule.

The easy configurability and the simple model of programming and setting up of test cases have proven valuable in an environment where change is

constant and activities evolve quickly. Even though the tool was new to the programmers, it was generally quickly adopted and used without major problems or dissatisfaction. The tool gave the programmers a chance to constantly interact with the application as it was being developed, and to understand and resolve problems as they appeared. As a result, quality was significantly improved, and the number of problems discovered and reported by QA was proportionally reduced.

## REFERENCES

- [1] 3G. [www.3gpp.org](http://www.3gpp.org)
- [2] Bruegge, B., and Dutoit, A.H., 2004. *Object-Oriented Software Engineering*, Pearson Education, Inc. Upper Saddle River, NJ 07458.
- [3] Campione, M., Walrath, K., and Huml, A., 2000. *The Java Tutorial: A Short Course on the Basics*, Addison-Wesley, Reading, MA, 3<sup>rd</sup> edition.
- [4] Gorton, I., and Liu, A., 2002. Software Component Quality Assessment in Practice: Success and Practical Impediments, ICSE, pp. 555-558.
- [5] J2EE, 2003. [java.sun.com/j2ee/1.4/docs/index.html](http://java.sun.com/j2ee/1.4/docs/index.html)
- [6] JavaMail. [java.sun.com/products/javamail/index.jsp](http://java.sun.com/products/javamail/index.jsp)
- [7] JMeter, [www.apache.org](http://www.apache.org)
- [8] Lindholm, T., and Yellin, F., 1999. *The Java Virtual Machine Specification*, Addison-Wesley, Reading, MA, 2<sup>nd</sup> edition.
- [9] Menasce, D.A. Load Testing, Benchmarking, and Application Performance Management for the Web, CMG Conference, Reno, NV, 2002
- [10] MITA – Mobile Internet Technical Architecture, IT Press, PL 760, 00043 Edita, Finland, 2001
- [11] Pressman, R.S. *Software Engineering: A Practitioner's Approach*, 5<sup>th</sup> ed., McGraw-Hill, 2003
- [12] SS7. [www.iec.org/online/tutorials/ss7](http://www.iec.org/online/tutorials/ss7)
- [13] T1. [www.t1-t3-dsl-line.com](http://www.t1-t3-dsl-line.com)
- [14] VoiceXML. [www.w3.org/Voice/Guide](http://www.w3.org/Voice/Guide)
- [15] VoIP. [www.fcc.gov/voip](http://www.fcc.gov/voip)
- [16] WAP. [www.wapforum.org](http://www.wapforum.org)
- [17] Weyuker, E.J., and Vokolos, F.I., 2000. Experience with Performance Testing of Software Systems: Issues, an Approach, and Case Study, *IEEE TSE*, 26(12), pp. 1147-1156.