A Survey on Web Services Testing

Huiyong Xiao
hxiao@cs.uic.edu
Department of Computer Science
University of Illinois at Chicago

Abstract

This paper aims to give the reader a general idea of Web Service testing. Web Services are a collection of standards and protocols that allow us to make processing requests to remote systems by speaking a common, non-proprietary language and using common transport protocols such as HTTP or SMTP. Although web services are becoming more and more popular as an emerging technology, few papers and stuffs are published about its testing. In this paper, we discussed in detail the architecture and features of Web Services, and classes of testing technologies on web services, and we gave a typical toolkit for each sort of testing.

1. Overview

1.1 Web services

- Definition

Different vendor, standards organization, or marketing research firm defines Web Services in a different way. Gartner, for instance, defines Web Services as "loosely coupled software components that interact with one another dynamically via standard Internet technologies." Forrester Research takes a more open approach to Web Services as "automated connections between people, systems and applications that expose elements of business functionality as a software service and create new business value."[1]

Basically, Web Services can be considered a universal client-server architecture that allows disparate systems to communicate with each other without using proprietary client libraries. In essence, Web Services are a collection of standards and protocols that allow

us to make processing requests to remote systems by speaking a common, non-proprietary language and using common transport protocols (HTTP, SMTP). As an emerging technology driven by the will to securely expose business logic beyond the firewall, web services will enable application-to-application e-marketplace interaction, removing the inefficiencies of human intervention. Through Web Services companies can encapsulate existing business processes, publish them as services, search for and subscribe to other services, and exchange information throughout and beyond the enterprise [2].

- An example (travel agent)

Normally without web services, if you want to book a holiday over the Internet you might browse a travel agent's web site to select a holiday and book it. Of course, you may also hire a car and check on the weather. However, behind the scenes, the web server might actually be accessing data in a proprietary format in some kind of database, or communicating, again in a proprietary fashion, to back-office systems to get information about your holiday.

Now, let us consider a web service for the holiday booking, which is respond to requests for flight information from travel agents' web sites. What is more, these web services will be able to publicize their existence and communicate with web sites, and other web services, in an open, public format. When you book a holiday, you'll still browse to your travel agent's web site but behind the scenes something different will happen. To find holiday details, rather than access proprietary data in a database, the web server will access different web services for flight, accommodation and weather information. Figure 1 describes the layout of such a web service [3].

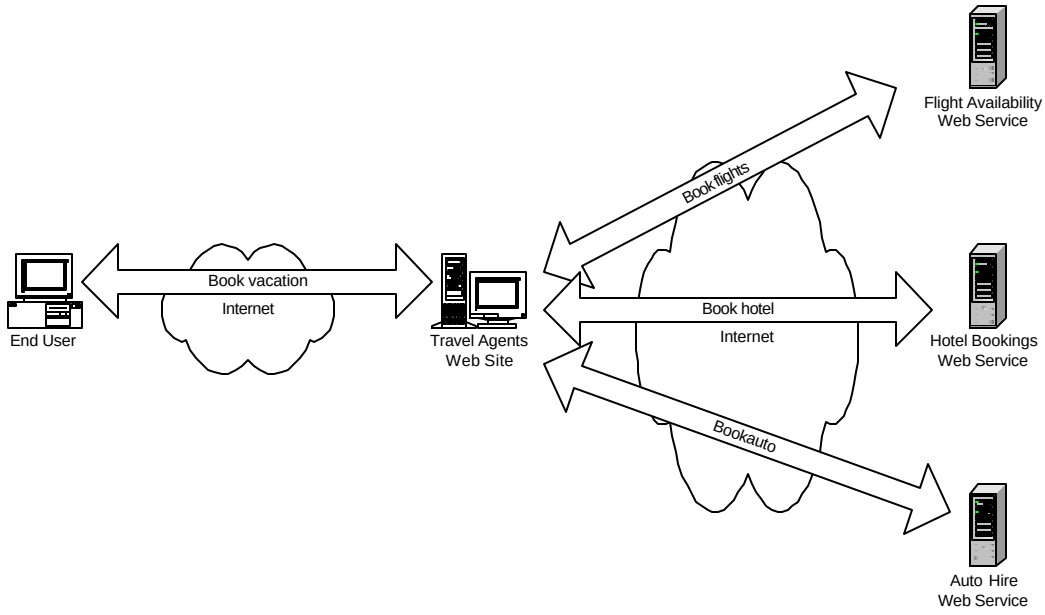

Figure 1 Layout of the web service for holiday booking example

- Web service architectures

Due to the variety of the definition of web services, the architecture of a Web Services stack varies from one organization to another. The number and complexity of layers for the stack depend on the organization. Each stack requires Web Services interfaces to get a Web Services client to speak to an application server or middleware, such as Common Object Request Broker Architecture (CORBA), Java 2 Enterprise Edition (J2EE), and .NET. To enable the interface, you need Simple Object Access Protocol (SOAP), SOAP with Attachments (SwA), and Java Remote Method Invocation (RMI) among other Internet protocols. Here we give out the web services stack from WebServices.Org as shown in Table 1. One can refer to paper **[3]** for other web services stacks and the comparison between them.

Table 1 Web Services stack from WebServices.Org

Layer	**Example**
Service Negotiation	Trading Partner Agreement
Workflow, Discovery, Registries	UDDI, ebXML registries, IBM WSFL, MS XLANG

Service Description Language	WSDL/WSCL
Messaging	SOAP/XML Protocol
Transport Protocols	HTTP, HTTPS, FTP, SMTP
Business Issues	Management, Quality of Service, Security, Open Standards

Service Negotiation

The business logic process starts at the Services Negotiation layer (the top) with, say, two trading partners negotiating and agreeing on the protocols used to aggregate Web Services. This layer is also referred to as the Process Definition Layer covering document, workflow, transactions, and process flow.

Workflow, Discovery, Registries

The stack then moves to the next layer to establish workflow processes using Web Services Flow Language (WSFL) and MS XLANG, an XML-based language to describe workflow processes and spawn them. The W3C web site has not indicated whether it has received the WSFL proposal for consideration. If it has, the proposal has not yet been posted on the web site.

WSFL specifies how a Web Service is interfaced with another. With it, you can determine whether the Web Services should be treated as an activity in one workflow or as a series of activities. While WSFL complements WSDL (Web Services Definition Language) and is transition-based, XLANG is an extension of WSDL and block structured-based. WSFL supports two model types: flow and global models. The flow model describes business processes that a collection of Web Services needs to achieve. The global model describes how a set of Web Services interacts with one another. XLANG, on the other hand, allows orchestration of Web Services into business processes and composite Web Services. WSFL is strong on model presentation while XLANG does well with long-running interaction of Web Services.

Among the software supporting WSFL is IBM MQ Series Workflow (now known as WebSphere Process Manager) that automates business process flows, optimizes Enterprise Application Integration (EAI) with people workflow, provides scalability, and

complies with the Workflow Coalition and multi-platform capabilities. MSXLANG is the language implemented in BizTalk, the XML integration server from Microsoft.

Web Services that can be exposed may, for example, get information on credit validation activities from a public directory or registry, such as Universal Description, Discovery and Integration (UDDI). The ebXML, E-Services Village, BizTalk.org, and xml.org registries, and Bowstreet's (a stock service brokerage) Java-based UDDI (jUDDI) are other directories that could be used with UDDI in conjunction with Web Services for business-to-business (B2B) transactions in a complex EAI infrastructure under certain conditions.

Hewlett Packard Company, IBM, Microsoft, and SAP launched beta implementations of their UDDI sites that have conformed to the latest specification (UDDI v2), including enhanced support for deploying public and private Web Service registries, and the interface (SOAP/HTTP API) that the client could use to interact with the registry server. In addition to the public UDDI Business Registry sites, enterprises can also deploy private registries on their intranet to manage internal Web Services using the UDDI specification. Access to internal Web Service information may also be extended to a private network of business partners.

Service Description Language

As you move down the stack, you need WSDL to specify how to connect to a Web Service. This language is an XML format for describing network services. With it, service requesters can search for and find the information on services via UDDI, which, in turn, returns the WSDL reference that can be used to bind to the Web Service.

Web Service Conversational Language (WSCL) helps developers use the XML Schema to better describe the structure of data in a common format (say, with new data types) the customers, Web browsers, or indeed any XML enabled software programs can recognize. This protocol can be used to specify a Web Service interface and to describe service interactions.

Messaging

Now, we get to the Messaging Layer in the stack where SOAP acts as the envelope for XML-based messages, covering message packaging, routing, guaranteed delivery and security. Messages are sent back and forth regarding the status of various Web Services

as the work progresses (say, from customer order to shipping product out of the warehouse).

Transport Protocols

When a series of messages completes its rounds, the stack goes to its last layer: the transport layer, using Hypertext Transfer Protocol (HTTP), Secure HTTP (HTTPS), Reliable HTTP (HTTPR), File Transfer Protocol (FTP), or Standard Mail Transfer Protocol (SMTP). Then, each Web Service takes a ride over the Internet to provide a service requester with services or give a status to a service provider or broker.

Business Issues

Finally, the Business Issues row in the table lists other key areas of importance to the use and growth of Web Services. Without consideration to these points, Web Services could quickly become objects of ridicule.

- Two types of web services (intranet/internet web services)

There are two broad types of web services – web services used in an intranet and web services used on the Internet. Intranet web services are web services used internally by organizations but not exposed to the general public. For example, an intranet web service might be responsible for handling vacations requests from employees. The company's intranet web site would then access this web service and employees could request vacations. Managers could authorize the vacations and colleagues could check when other employees were on holiday. Human resources could then write a simple application in Visual Basic to make sure that helpdesk staffs don't take all their vacation at the same time. All this is possible without having detailed knowledge of how or where this information is kept.

1.2 Challenges in testing web services

Testing intranet and Internet web services provides subtly different problems. With an intranet web service, you, as an organization, are likely to have control over who has access your web service. Since it is on an internal network, only internal users can have access to it, so you have a theoretical maximum. Similarly, you can make certain

assumptions about security. With an Internet web service, anybody can access it. This means that there are additional scalability and security considerations.

Another challenge in testing web services is that they do not inherently display a user interface that can be tested. Although some development tools will build a web page around a web service, this is not part of the web service itself. For example, Visual Studio .NET will generate a page which allows you to invoke methods of web services and view the XML returned but this is not very efficient for anything but the simplest of web services. This lack of user interface means that web services are hard to test manually, but are an ideal candidate for automated testing. A consequence of this is that some programming skills are almost certainly needed for testers who need to test web services. A web service is not the sort of application you can test by key-bashing.

1.3 Different types of testing

In general, there are five different sorts of testing as follows that you can carry out on web services [3, 4, 5]:

Proof of concept testing

There are many different choices to be made to build a web service– which tool vendor to use, which programming language and which database backend, for instance. So we will have to understand if the architecture we have chosen for the web service is the correct one. If we can clarify and resolve these issues early in the development lifecycle, then we will save a lot of time and money further on down the line.

Functional testing

Functional testing is to ensure the conformance of the functionality of the web service with the expected. The functionality includes if the web services implement security/ authentication, or if the web services support all the expectative communications protocols, or how the web services deal with the unexpected access requests from clients. Especially to say, bounds testing and error checking is most important.

Regression testing

A regression test is normally a cut-down version of a functional test. Its aim is to guarantee that the web service is still working between builds or releases. For instance, is

the performance still acceptable after the latest builds? Since regression testing is a repetitive task in its nature, it will usually be automated.

Load / stress testing

The aim of load / stress testing is to find the scalability of the web service with the possible increase of the number of accessing clients. While functional and regression testing are facilitating to make sure the correctness of the web service with a single user, what we need to know now is if it is capable of coping with 10, 100 or 1000 users, or how many users it will cope with. In load testing, response time is an important referential criterion to ensuring the normal load of the web services.

Monitoring

Once the web service is started and being used by real clients, it will be essential to monitor the web service, ensuring if the response time of the web service is adequate etc.

2. Proof of concept testing

Proof of concept testing is usually designed and applied as early as possible in the development life cycle, so that we are able to make sure the architecture right very early in the development process (illustrated in Figure 2). To some extent, a proof of concept test is normally a cut-down load test. But here, there's no need to run it on powerful hardware, or get exact answers. The aim of proof of concept testing is to answer the question like "Are we going in the right direction?"

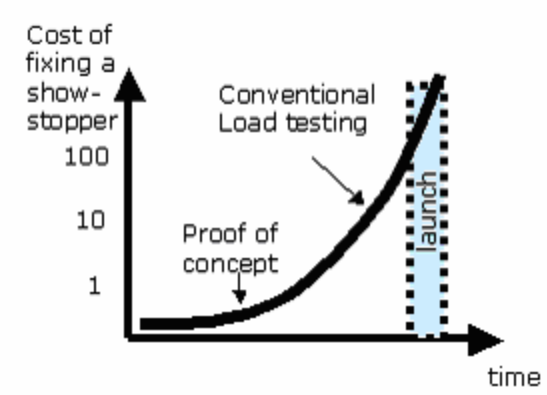

Figure 2 Role of Proof of concept in web services testing

Presently, there exist few tools concentrating on Proof of concept testing. The Advanced .NET Testing System (ANTS) released by Red Gate is used for the complete

life-cycle testing of XML web services, .NET Web Applications, and conventional Web sites. Among many toolkits in the ANTS suite, ANTS <concept> is used for proof-of-concept testing for companies experimenting with XML Web Services, Web applications, and sites written in Visual Studio .NET.

ANTS <concept> features the ability to understand the capacity and response times of your new .NET Web services so you can load test up to 100 virtual users for testing architecture and designs, and recordable scripts for automated testing. ANTS is written entirely in Visual Studio .NET; Visual Basic .NET is the scripting language. Figure 3 shows the user interface of ANTS <concept>.

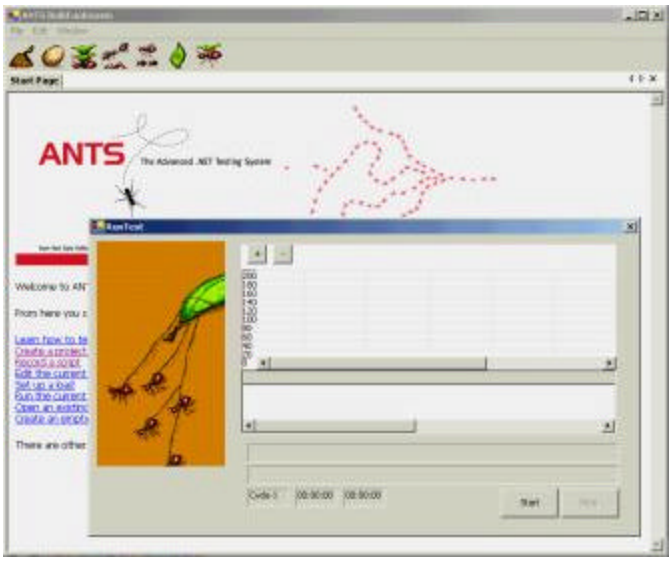

Figure 3 user interface snapshot of ANTS <concept>

Some key point about ANTS <concept>:

1. The ability to understand the capacity and response times of your prototype .NET web applications and services. Load test with up to 100 virtual users for concept testing architecture and designs. If you require more virtual users than this then ANTS <load> may be more appropriate for your needs.

2. Web Services Methods are treated as objects within a Visual Basic .NET allowing powerful scripts to be constructed. Details can be inserted from a database into your web service to mimic realistic load situations. Scripting is in the familiar environment of

Visual Studio using Visual Basic - which means you have a short and permanently useful learning curve.

3. Recordable scripts for automated testing of browser based web applications.

4. ANTS is written entirely in C# using Visual Studio .NET. It features Visual Studio for Applications and was the first product, globally, to do so.

3. Functional and Regression Testing

Functional testing, simply stated, verifies that an application or a service does what it is supposed to do and doesn't do what it shouldn't do [6]. For example, if you were functionally testing a word processing application, a partial list of checks you would perform includes creating, saving, editing, spell checking and printing documents. (Again, this list is quite incomplete!)

Positive functional testing entails exercising the application's functions with valid input and verifying the outputs are correct. Continuing with the word processing example, a positive test for the printing function might be to print a document containing both text and graphics to a printer that is online, filled with paper and for which the correct drivers are installed.

Negative functional testing involves exercising application functionality using a combination of invalid inputs, unexpected operating conditions and other "out-of-bounds" scenarios. Continuing the word processing example, a negative test for the printing function might be to disconnect the printer from the computer while a document is printing. What probably should happen in this scenario is a plain-English error message appears, informing the user what happened and instructing him/her on how to remedy the problem. What might happen, instead, is the word processing software simply hangs up or crashes because the "abnormal" loss of communications with the printer isn't handled properly.

Regression testing is the process of testing changes to computer programs to make sure that the older programming still works with the new changes. Regression testing is a normal part of the program development process and, in larger companies, is done by code testing specialists. Test department coders develop code test scenarios and exercises that will test new units of code after they have been written. These test cases form what

becomes the test bucket. Before a new version of a software product is released, the old test cases are run against the new version to make sure that all the old capabilities still work. The reason they might not work is because changing or adding new code to a program can easily introduce errors into code that is not intended to be changed.

Obviously, Regression testing can be considered as some kind of cut-down functional testing to some degree. Actually that's why existing testing productions for the functional testing and regression testing are commonly integrated into a single program.

In order to provide the fastest, most accurate means of conducting automated function and regression testing for Web applications, we have to find appropriate testing tools [7]. Developers and QA (quality assurance) professionals responsible for Web development will have to use this automated software testing tool throughout the entire application life cycle from early in development through application deployment. The following list the main requirements from the functional and regression testing tools.

Stay Ahead of Changing Web Applications

As the business-critical Web applications proliferate and the technology underlying them becomes more complex, automated testing becomes essential. To provide real value, an automated testing solution must help you keep pace with applications that can change rapidly. The scripts you create for these tests might also be used as the basis for conducting load and scalability tests, and for monitoring after you have deployed your application.

Focus on Testing, Not Programming Test Scripts

Testers should be allowed to write comprehensive tests without programming in a proprietary scripting language. A powerful, intuitive Visual Script technology, an automated test case generator, sophisticated content matching, and the ability to execute data-driven tests etc. all may be considered into the features of the testing tool.

Create Sophisticated Test Scripts Quickly

Such testing tool should maximize productivity by virtually eliminating the need for programming and provides multiple ways to create tests.

Fix Web Applications Efficiently

Users may be allowed to drill into problems quickly and focus on implementing solutions. For example, e-Tester's Visual Script graphically displays the contents of each Web page and test cases using an intuitive tree format, shown as Figure 4. Any errors or differences identified by e-Tester are highlighted using an intuitive system of color-coded flags, so that users can quickly and easily create scripts that test all the objects on each page and indicate errors with color-coded flags.

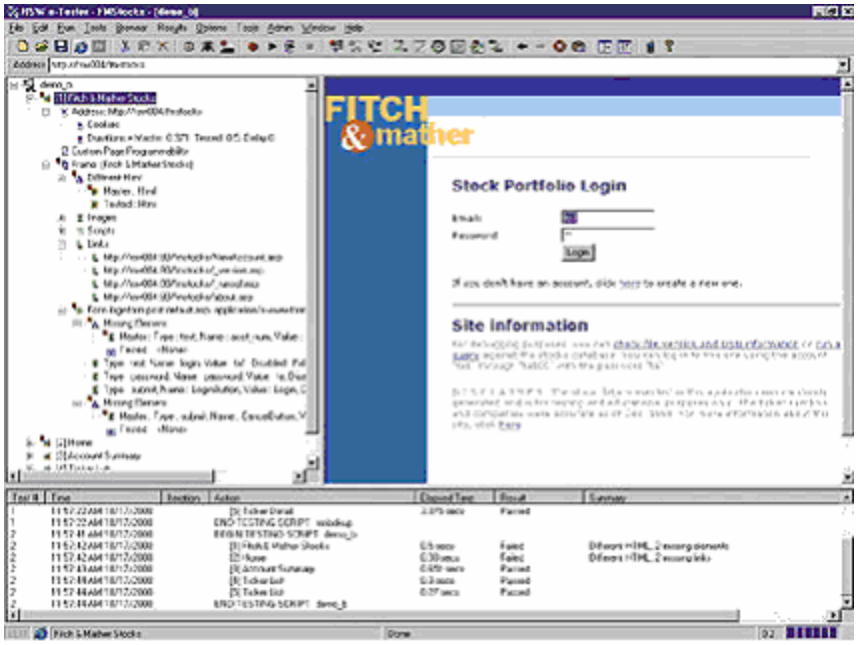

Figure 4 Graphic interface of e-Tester for users to solve errors

e-Tester from Empirix is an Enterprise-caliber, easy-to-use solution for automated functional testing and regression testing of Web applications. Its key features can be summarized as:

1. Visual Script technology provides the fastest and easiest way to create test scripts, with no programming required.
2. Extending Visual Scripts with Visual Basic for Applications and other standard programming languages allows you to handle a wide array of testing challenges.
3. The Data Bank Wizard simplifies creation of data-driven tests.

4. Visual test results provide an ideal way for Development, QA, and Operations groups to communicate and share data.
5. Test management and integration with defect tracking make e-Tester the centerpiece of a total quality solution.
6. e-Tester scripts integrate seamlessly with Empirix scalability, monitoring, test management, and defect tracking components to provide a comprehensive enterprise application performance solution.

4. Load / Stress testing

In order to know how the web service responds as more and more users are simulated, we must keep all other factors (hardware and networking, for example) constant to carry out the testing in a controlled environment. This is important to produce objective results.

The ultimate goal of load testing is to reassure you, and confirm if the web service will respond acceptably for up to x clients making y requests a second. Unpredictable user behavior and system variables expose the web service to the risk of highly visible failure. Load and stress testing offers the following advantages [8]:

- Predictable and budgeted costs
- Avoidance of costly retroactive fixes
- More accurate scalability projections
- Optimal performance leading to an increased return on investment

There are several aspects of load and stress testing. To conduct load testing, we apply stress to the web services by simulating real users and real activity.

We can perform **capacity testing** to determine the maximum load the web service can handle before failing. Capacity testing reveals the web services' ultimate limit.

We may also perform **scalability testing** to determine how effectively the web service will expand to accommodate an increasing load. Scalability testing allows us to plan the web service capacity improvements as the business grows, and to anticipate problems that could cost the revenue down the line. Scalability testing also reveals when

your site cannot maintain good performance at higher usage levels, even with increased capacity.

While carrying out the load testing, the tool being used should be able to identify how the following values change as we increase the number of clients. These are all measurements of how the client is experiencing the web service:

1. **Time to connect**: This is the time it takes to make a connection from the client to the web service. This should be as low as possible.
2. **Time to first byte**: This is the time it takes for the client to start receiving data back from the web service. If the web service needs to do a lot of thinking for each request, then this time could be significant.
3. **Time to last byte**: This is the time it takes for the client to receive the last byte of information back from the service. If the service needs to return a large amount of data (if it is returning maps, or images, for example), then this could be significant.

Notice that the way these measurements change as the load on the web service is increased. Ideally we want these metrics to remain constant. It's very likely that we'll find that the web service scales (i.e. the response time for requests remains constant) until a certain number of virtual users, and then it stops scaling. To troubleshoot this, we have to analyze the performance on the server the service is running on. Keep in mind that the saturated CPU, the thrashing disk, or the network traffic all possibly causes such kind of performance problems.

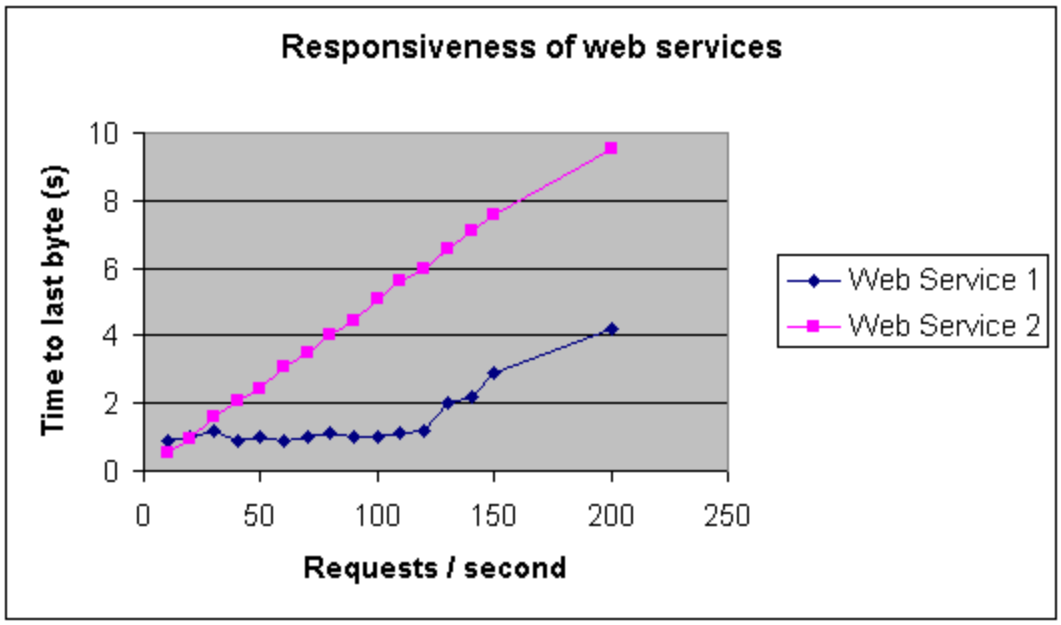

Figure 5 Web services with different scalability properties

In the example in Figure 5, Web Service 1 scales well until about 120 requests / second (the time to last byte is constant), but then stops scaling. Web service 2 scales poorly - as the number of requests / second increases, the responsiveness decreases linearly.

In order to solve the problem when the system hits a bottleneck, we may add another processor to the server, or even add another server to see if it doubles its capacity. If the web service will scale in this way, then we will be able to cope with extra demand by adding more and more hardware at the problem. If the web service doesn't scale in this way, it means the web service won't perform no matter how much money is spent on expensive hardware.

e-Load [6] is a robust Web load testing solution that enables you to easily and accurately test the scalability and performance of your Web applications. Companies use this automated software load testing solution to predict how well their Web applications will handle user load. It can be used during application development and post-deployment to conduct stress testing. Generally speaking, e-Load mainly focuses on the following aspects in the load testing.

Ensure the Scalability of Your Enterprise Web Applications

The Internet affords you the opportunity to reach millions of new customers and business partners and to interact with existing ones in entirely new ways. In order to fully exploit that opportunity, you must ensure that your mission-critical Web applications effectively handle all levels of user traffic.

Whether they are serving tens or thousands of users, your Web applications must always be prepared to provide swift, seamless service. Under very heavy loads, the many technologies that support your Web applications can interact in unpredictable ways, jeopardizing your users' Quality of Experience. For that reason, it is necessary to thoroughly load and stress test those applications or risk alienating potential customers as well as loyal Web constituents.

Create Accurate Load Tests Quickly and Easily

Using e-Load, a scalability testing solution that was designed specifically for the Web, you can deploy new Web applications with confidence and ensure that existing ones can handle growing, volatile demand. e-Load, part of the award-winning e-Test suite, is the only load testing tool that requires no programming and can reuse regression tests without modification. An intuitive graphical user interface and our powerful Visual Script technology reduce testing time and enable you to create representative load scenarios with ease.

e-Load's intelligent architecture allows you to generate extremely large loads without purchasing additional testing hardware or sacrificing accuracy.

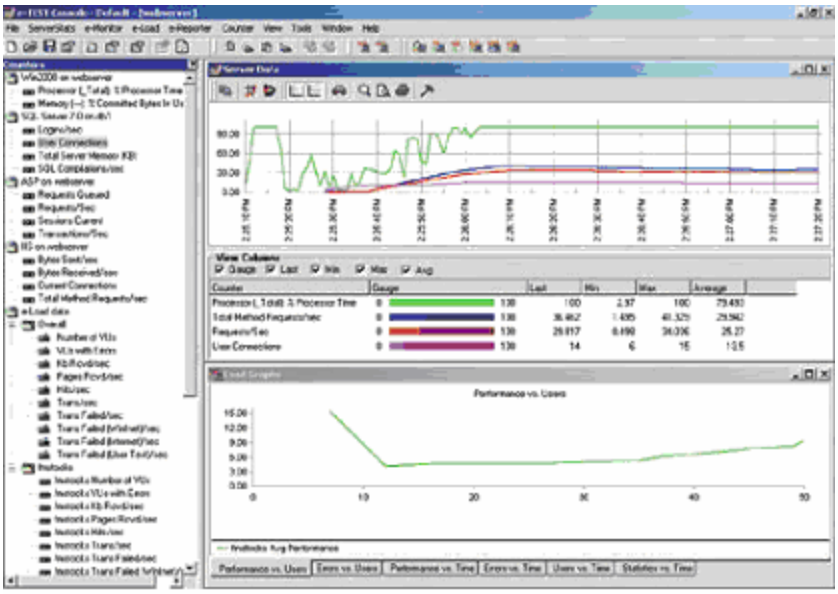

Figure 6 a snapshot of the graphic interface of e-Load

Pinpoint Bottlenecks to Improve Performance

e-Load comes with enterprise-caliber, integrated data collection and analysis tools that enable you to precisely and effectively eliminate performance problems. ServerStats provides a real-time, detailed view of the performance of each tier of the system under test. e-Reporter leverages this data to generate customized reports that allow you to analyze the application's performance under load and pinpoint bottlenecks. e-Reporter, available as a Windows-based application or through a Web portal, allows you to create reports and share critical information anywhere in your enterprise.

e-Load is a flexible, robust load testing solution that can be configured to meet the unique needs of your organization. Seamless integration with functional and regression testing and monitoring solutions from Empirix, combined with an extensive set of options, make e-Load a comprehensive Web application performance solution that can grow with you as your needs evolve. e-Load produces real-time graphs and reports to show you results while a test is in progress (See Figure 6).

The key features of e-Load can be summarized as:

- Maximize the efficiency and cost effectiveness of your Web infrastructure

- Easily and accurately ensure that mission-critical applications reliably handle peak loads
- Improve load capacity of Web applications throughout their entire life cycle
- Pinpoint performance bottlenecks with ease using powerful analysis and reporting tools

5. Monitoring

Despite copious testing, unforeseen problems can still occur once a system is placed into the production environment and made available to increasing thousands of users and customers. In most situations, these problems are typically related to performance. To stem these problems, some companies have implemented 'early warning' systems to continuously monitor key transactions and business functions [9].

The monitoring systems provided by software vendors are typically extensions of existing load-testing tools. They contain the ability to identify specific components within the application, and allow QA analysts or system administrators to establish acceptable thresholds for response time or transaction usage. In addition, these thresholds can be established to exclusively monitor transactions or business functions during specific times of the day when load is expected to be heavy.

But although usage of these tools can prove beneficial, they must be balanced with any known or perceived resource constraints within the environment. We've got to watch out for overhead and make sure the monitoring tool doesn't impact response time.

6. Automation

The most important question to ask in planning for test automation is whether to automate at all. There are certain aspects of testing—such as performance and stress testing—that require automation. You must decide the extent of test automation from a management standpoint. Nearly every book on software testing offers advice on determining the utility of automation efforts. For the most part, they can be summarized by a set of basic guidelines. Take the effort to develop automated testing when:

- There is a set of API functions with a significant number of potential test cases.
- Responses to inputs can be categorized as "pass" or "fail" and do not require human interpretation.

- A significant number of bug regression test cases exist with simple pass/fail evaluation criteria.
- It's less expensive in terms of time, resources, and funding to pay a software design engineer in Test to write, maintain, and run test automations than to employ multiple software test engineers to perform manual testing.

After making the decision to automate, start automation efforts early in the project. The APIs will change, as will the user interface elements of the ASP files. Still, it is effective to have the automation tests evolve along with the service software. Test developers should be involved in the code review process so that a sound knowledge of the intended direction of the project code is established. The process of automation starts as soon as the API functions are published to the project team. Once the test developer has the functions available, they should start writing test cases. The process of writing test cases for automation should go something like what is described next.

For every API function, write down every conceivable permutation of the input variables. List all boundary conditions:

1. Write a proposed test case for each variation of the input parameters. Attempt to exceed the limits of any underlying database fields. Throw in attempts at security violations, such as buffer overflow attacks. Examine all use cases defined during the process of creating the functional specification. Many use cases will be suitable as automation candidates. Any that are not are important test cases for manual testing.
2. Cull all the duplicates. Challenge each parameter and move on.
3. Write your final list of cases. Give this list, along with an API reference, to a reviewer and have them see if you have missed anything.
4. Revise the list of cases and start writing code.

Automated testing of the Favorites Service consisted of both direct function calls to the API and sending SOAP messages through Internet Information Services (IIS) to the API. For more details on the automated test process used in the Favorites Service, see our article Favorites Service Test Tools and Scripts.

7. Summary

The purpose of this paper is to give the reader a general idea of Web Service testing. As an emerging technology driven by the will to securely expose business logic beyond the firewall, Web Services enable application-to-application e-marketplace interaction, removing the inefficiencies of human intervention. However, few materials and information are published with respect to the testing on web services so far. In this paper, we discussed in detail the architecture and features of Web Services, and classes of testing technologies on web services. For each sort of testing technology, we gave and discuss a typical toolkit.

References

[1] Judith M. Myerson, Web Service Architectures, *http://www.webservicesarchitect.com*

[2] Colin Adam, Why Web Services?, *http://www.webservices.org*

[3] Neil Davidson, Testing web services white paper, *http://www.red-gate.com*

[4] Web services testing: Beyond SOAP, *http://searchwebservices.techtarget.com*

[5] Testing Web Services Today and Tomorrow, *http://www.therationaledge.com*

[6] *http://www.testwareinc.com*

[7] *http://www.empirix.com*

[8] *http://www.codeproject.com*

[9] Testing e-commerce, *http://www.adtmag.com*