



Building Reliability and Performance into Web Services

Frank Cohen, CEO
fcohen@pushtotest.com
www.pushtotest.com
408/ 374-7426

September 24, 2002

This is a presentation that describes new test methodologies, techniques and tools to build reliability and performance into Web Services. This presentation was first made to the Web Services Special Interest Group (SIG) of the Software Development Forum (<http://www.sdforum.org>) on September 24, 2002.

© 2002 PushToTest. All Rights Reserved.

However, you have PushToTest's permission to share this document with interested parties. For more information contact PushToTest at <http://www.pushtotest.com/ptt> or by calling (408) 374-7426.

Agenda

- What are Web Services?
- Why is performance and reliability important?
- User-goal oriented test agents
- How did we go about the test
- What we found & what we suspect
- How to solve the problems for the long-term

PushT↓Test

Welcome to a presentation on building reliability and performance into Web Services. I put together an agenda would covers a new methodology for testing complex interoperable systems, especially Web Services and shows the results of a recent scalability test of a variety of Web Service application servers.

This should take about 45 minutes which gives lots of times for questions and answers.

Introduction

- Frank = the “go to” guy for reliability and performance
 - Open-source TestMaker tool
 - Deliver custom functions
 - Run scalability studies
 - Train team: Dev, QA, IT



3

PushToTest

This presentation was first given by Frank Cohen, founder of PushToTest. Frank can be reached at fcohen@pushtotest.com.

Frank Cohen is the "go to" guy when enterprises need to test and solve problems in complex interoperating information systems, especially Web Services. Frank is CEO of PushToTest, a test automation solutions business. PushToTest maintains TestMaker, a free open-source utility for building intelligent test agents to check Web Services for scalability, performance and functionality. PushToTest Global Services customizes TestMaker to an enterprise's specific needs, conducts scalability and performance tests, and trains enterprise developers, QA analysts and IT managers on how to use the test environment for themselves. This unique business approach delivers inexpensive solutions, expert insight and immediate answers.

Frank is also contributing author to Java Web Services Unleashed from SAMS Publishing, and Java P2P Unleashed also from SAMS. Frank's own book on Testing Web Services is due later this year.

PushToTest is partners with BEA, Sun, CapeClear and IBM.

Web Service Hyperbole

- The end of Enterprise Application Integration
- An entirely new architecture for distributed computing
- Another Microsoft conspiracy
- Dramatically reduces software development costs and timeframes

4

PushTest

A lot has been said and written about Web Services. These are my personal favorites. From my perspective each one is mistaken.

Web Services makes development, integration and architecture much easier. However, Web Services is not a huge revolution.

To understand what drives such hyperbole lets look at how two major efforts came together. The intersection of these trends powers the huge interest in Web Services.

Technology Intersection

- Enterprises already own Web infrastructures
 - Looking for productivity gains
- Developers like XML
 - Windows Registry / property files
 - Didn't know how to use XML in their code
 - Implementing with XML wins!
 - APIs move self-describing XML data
 - Everything looks like an interface

5

PushTest

When independent technology innovations intersect the world enjoys life changing products, services and techniques. For example, the light bulb required both electricity generation and filament fiber technology.

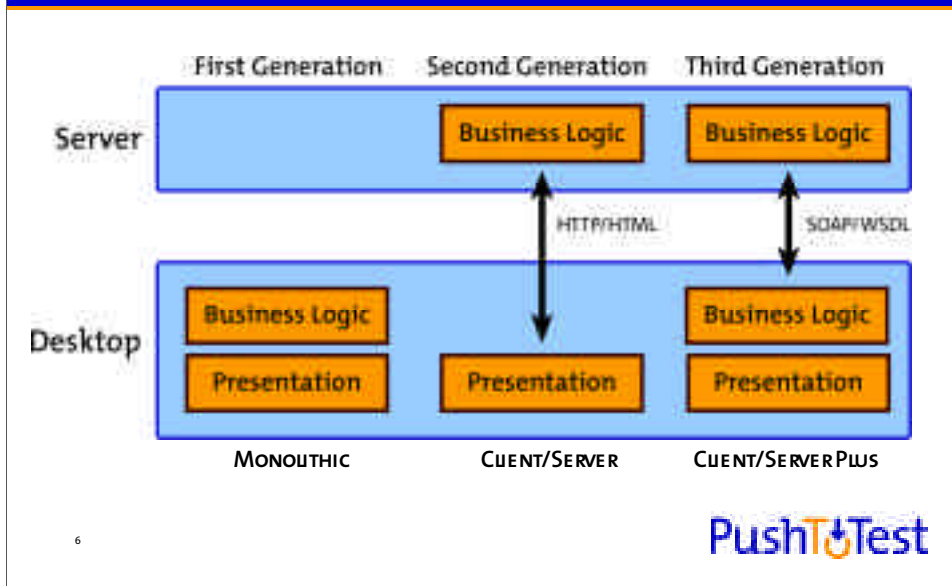
In the case of Web Services enterprise managers over the past 5 years invested in computers, servers, routers, and other Web infrastructure. Now they want to increase productivity of their teams through software implementations that would run on the existing equipment.

At the same time most software developers realized they really liked XML. It was much better than using the Microsoft Windows Registry or text-based property files to store and describe application data. They were looking for a way to use XML in their applications.

Using XML to implement an application's interfaces is a clear win to developers. Plus, these XML described interfaces could work across platforms and programming languages. With XML everything looks like an interface.

These intersecting technologies power the wide-spread enthusiasm for Web Services.

Architecture Trends



At the same time, software developers were again experimenting with software architectures, especially with the location of application business logic and presentation code. Presentation code handles windows, mice, keyboard and other user interactions. Business logic is the instructions that define the behavior and operation of an application.

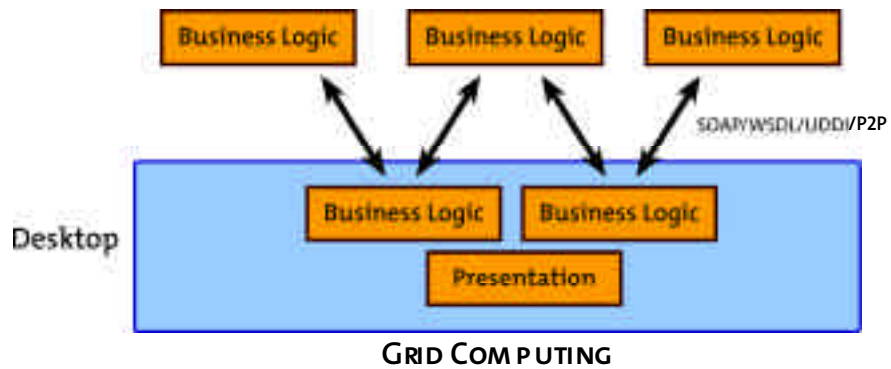
The first generation software architecture built the presentation and business logic on a single system. But that didn't last for long...

In the second generation, Client/Server architecture brought back the large centrally controlled datacenter so familiar in the 1960s when mainframe's ruled the information world. In client/server architecture the desktop system is a "dumb" terminal that only needs to display the data provided by the server.

The early Internet was modeled after client/server architecture where the browser made simple request to a server.

As browser's improved in functionality - applets, Javascript, ActiveX, DHMTL were introduced - some systems included business logic on the desktop side. However, the majority of function remained on the server.

Architecture Trends



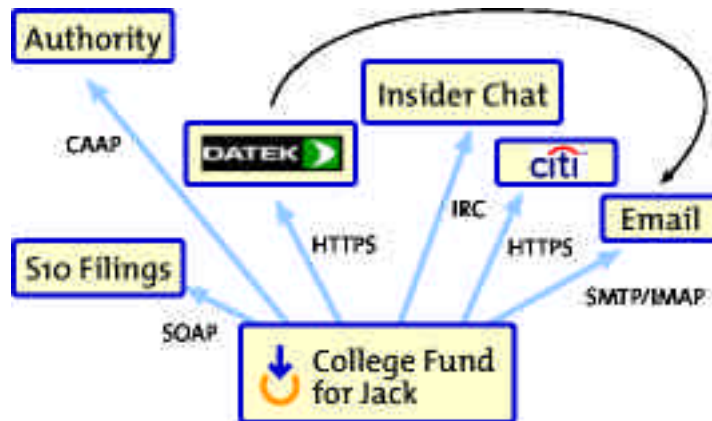
7

PushTest

The next stage enables an application to host business logic modules on the desktop or server. The modules discover each other using UDDI and P2P technologies. Also multiple copies of the business logic modules may run in a grid of datacenters to allow failover, dynamic routing and functional specialization.

All of these architectures run in a Web environment and can host Web Services. So lets look at how a Web Service operates in this environment.

Find The Web Service



PushTest

Let's look at an experience I am going through now. And it will illustrate a definition of Web Services with a practical example. It will also show how Test Agent technology is used to solve Web Service scalability and reliability problems.

My son is now 11 years old. That means in 7 years he will be ready for college. I am driven to be ready for his financial needs by the time he is ready to start college.

I manage Jack's college fund by using the Datek service to access the stock market, Citibank's Web site to manage savings, an information retrieval service to learn the SEC filings from companies I invested in, and a chat service to keep up with rumors around the market and companies.

I access these services using multiple applications (browser, email client, chat client.) These applications use multiple protocols (HTTP, HTTPS, SOAP, IRC, SMTP, POP3, etc.) My behavior using these applications is very proactive, risk averse, and yet sometimes opportunistic. Plus I know the criteria for success, namely having enough money to put Jack into the college of his choice.

Now imagine the box labeled "College Fund for Jack" in this chart is a "Web Service." This service proactively monitors market data, handles emails, and makes investments using funds from a set of bank accounts.

Wouldn't that be great!?

“Services” need to be proactive



- College Fund Service benefits
 - Time-saver for me
 - More accurate with more data
- So where are services today?
 - “Chron” jobs
 - Search engines

PushToTest

Having a “College Fund for Jack” Web Service would be a great time-saver for me and the service would be able to retrieve more market data so its management of the investments would result in a bigger fund for Jack.

So why isn’t this kind of Web Service already here? This is 2002 and we’re 6 or 7 years into the Internet age. Why are intelligent, proactive, multi-protocol services not yet a reality?

There are a few services already. Most production servers include *chron* jobs where a program fires up at a certain time and conducts some business activity such as monitoring stock funds. There are also Web searching engines that regularly look for the best price on products and build an index to Web pages. Unfortunately these are programmed to only conduct one specific function.

Where is the platform for developing a wide variety of services?

Issues To Build Services



- GUI programming responds to events
- Java and C offer no semantics for accomplishing tasks
- Few libraries offer multiple protocols
- Dev, QA, IT are separated by old Mainframe roles

PushToTest

10

Imagine the “College Fund for Jack” service being coded in a Java application. As a developer myself, this looks like a complex development project:

1) Developers today seem to cleave into two camps: Desktop/GUI application developers and server-side developers. The desktop developers use graphical development tools to lay-out screens filled with buttons, text boxes and more. These are event-driven programs that wait for a user to click a button or press a key. These are rarely proactive in nature.

On the server-side most developers write object-oriented server code that responds to requests from a browser or application. Again, these are event-driven programs that are rarely proactive in nature. Desktop/GUI and server-side developers need to learn how to build proactive services. And that’s not easy.

2) Development platforms (Java, C#, C, C++, etc) do not define the semantics for accomplishing tasks. While a language does a good job defining a primitive String object its really up to me to define an object that indicates it is on-track to succeed with Jack’s college fund.

3) The College Fund service needs to access services using multiple protocols (HTTP, HTTPS, SOAP, IRC, SMTP, POP3, etc.) Very few libraries exist that offer the developer an easy API to use multiple protocols.

4) And lastly, to build a highly reliable and well scaling Web Service requires the cooperation of developers, QA analysts and IT managers. Most enterprises today still have these team members separated out into groups. And those groups sometimes snipe at each other.

User Goal-Oriented Test Agents



- Encapsulate behavior
- User Archetype: Frank
- “Go to” Guy
- 7 Years To Go
- Put kid in college, risk averse, but little time
- Obsessive, looks at stock 3 times a day

PushTest

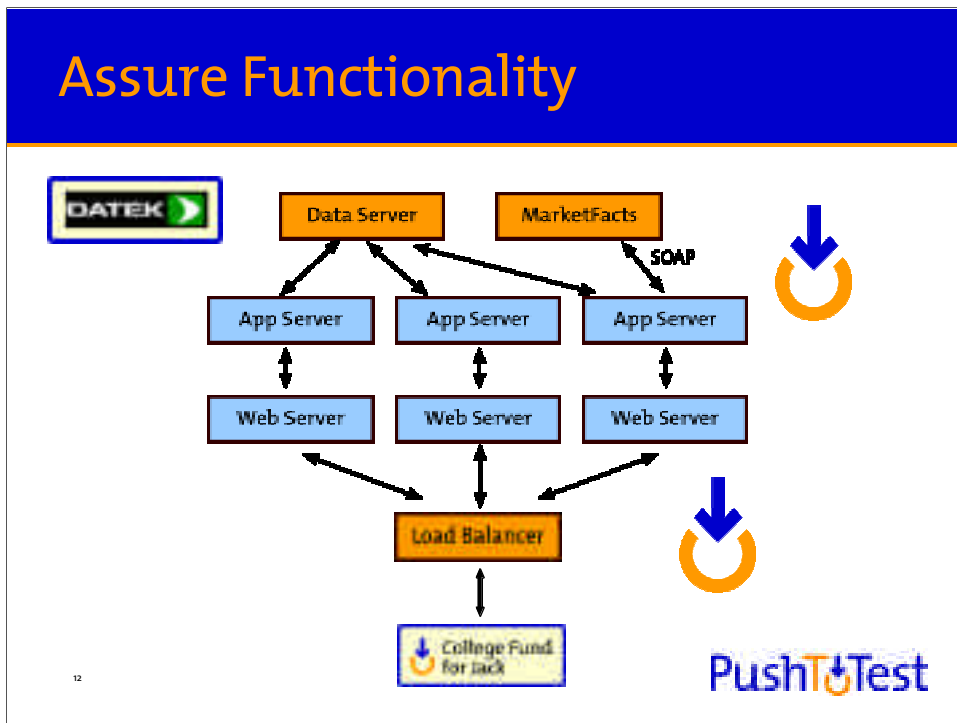
I'd like to introduce you to a new methodology, new technique and set of tools that make building reliable, scalable and well-performing Web Services very, very easy.

Let's look at how we would implement the “College Fund for Jack” as a User Goal-Oriented Test Agent, I will call it a *test agent* for short. We begin by looking at the prototypical users of the system. Since we already talked about how I manage the college fund we might as well define the first user archetype around my goals and behavior.

User archetypes were taught to me by Alan Cooper (alan@cooper.com,) one of the leading usability gurus in the world. User archetypes describe the behavior of prototypical users of an information system. I find user archetypes very beneficial: They make it easy to describe system usage to others within an organization, they reduce the risk of missing an important bug in a forest of bug reports, and they focus development and deployment efforts towards meeting the user's goals instead of simply shipping a piece of software.

Some even go so far as to describe user archetypes background, likes and dislikes and common quirks. For example, the *Frank* user archetype is very proactive, he has only has a limited time to successfully manage the college fund, he is risk averse, but the time pressure to succeed allows him to make opportunistic decisions occasionally, and he is obsessive about managing the account. Some times he looks at his stocks 3 times a day, even though he is in the market for the long-term.

Assure Functionality



When we look into Datek's datacenter we find a Web environment built for scalability. I've coined the term Flapjacks to describe this infrastructure. A row of Web Servers and Application servers connect to a Data server, and in this case to an external MarketFacts server. Everything sits behind a load balancer. When more hungry patrons - Web browser users - show up Datek can throw some more Flapjacks - more servers - onto the grill to serve the additional load.

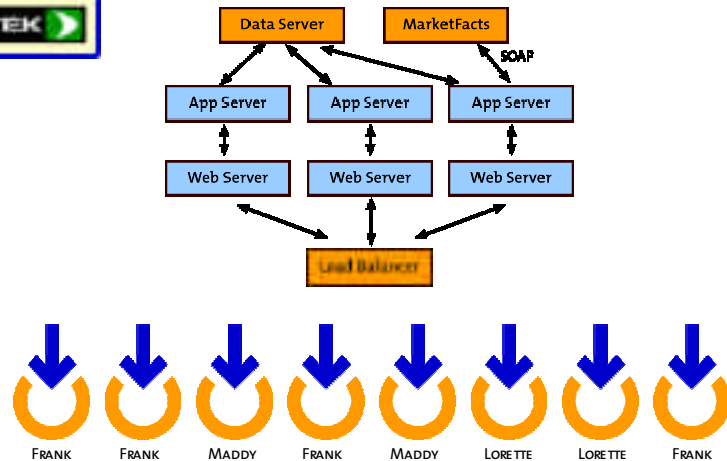
Now imagine the "College Fund for Jack" as a piece of Java code that is modeled after the *Frank* user archetype. The Java code speaks multiple protocols and can make requests to the Datek system just like a real user behind a keyboard and browser would. Instead of being a simple Web Service, this piece of Java code is a test agent that can consume and drive services.

The *Frank* test agent not only drives the Datek service but it also measures the responses. It identifies fast response times from slow response times. It also identifies functional problems by validating the responses it gets from the Datek service.

As an added advantage, the "Frank" test agent could be run from behind the load balancer. Measuring the response time before and after the Load balancer shows the overhead of using a Load balancer.

In fact test agents can show which components of the system add the greatest overhead, and which components should be optimized for reliability and scalability.

Better Scalability Too!



13

PushToTest

Why stop at one user archetype? Every system has multiple prototypical users, each with their own goal. By running multiple test agents, each based on a user archetype's behavior and goals, the test environment approaches the real production environment where many types of users make many types of requests concurrently.

Running multiple concurrent test agents produces meaningful knowledge of a system's reliability, scalability and functionality.

Why is this Important?

- A way out of Quality, Reliability, Performance stagnation
 - Automation saves money and increases customer satisfaction
 - Developers write agents too
- Businesses need a way to know:
 - Data center capacity planning
 - Knowing when your system is crap
 - Find the service that is causing the problem
 - Regression



PushToTest

You may be wondering *Why is this important?* To a business hosting a Web Service the reliability and scalability of their systems is the key to higher profits, increased opportunity realization, and increased customer satisfaction.

I regularly speak with many enterprise information managers on a weekly basis. Overall, I am sad to report that quality, reliability and performance are usually treated as job number 2. It's hard enough delivering the basic functionality.

I put it to these information managers that their software developers have probably already heard of Extreme Programming (XP), unit testing and XML. And that they are *turned-on* by what they see. Giving them a framework for building test agents is a win for the company.

What's more test agents are useful for developers needing functionality tests, QA analysts wanting scalability and regression tests, and IT managers wanting to check new software before putting it on the production servers. Test agents are a win for everyone.

Building Agents

- Script language
- Library of test objects
- Execution environment
 - Local and remote
 - Batch and service
- Results analysis
- Procedural and Event-Driven
- Self documenting



PushTest

15

What does it take to build a test agent? In my experience all the tools and technology is already out on the Internet, and it's all free! But it still requires some effort to assemble the right tools and technology.

Test agents may be written in an object-oriented language like Java, C#, and C, but these compiled languages often do not provide the kind of run-and-tweak-and-run-and-tweak speed usually needed. I highly recommend finding a scripting language that supports objects. I found Jython, the popular Python language implemented in Java, to be ideal. Jython gives you all the scripting of Python plus the ability to use any Java object directly in the script.

A library of test objects that implement protocol handlers is needed. My popular TestMaker utility includes the Test Object Oriented Library (TOOL) that provides HTTP, HTTPS, XML-RPC, and SOAP protocol handlers.

Test agents then need a way to run on a local machine, and also from a remote machine. That might be as simple as a command-shell script for the local machine, and using a terminal emulator (telnet or ssh) for the remote machine.

The agents also need to run either in "batch" mode, where the agent runs once and then quits, or runs as a server daemon that keeps going.

Additional needs include a way to analyze the results, provide event-driven and scripted operations and a way to document the test agent code for others to maintain.

Of course you will find all of these in TestMaker at <http://www.pushtotest.com>. And it's all free!

Case Study: Elsevier Science



- Content publishing with SOAP APIs
- Which SOAP implementations are most scalable?
- Impact of encoding styles: SOAP RPC Literal vs. Document
- WebSphere, WebLogic, Apache Axis, CapeClear, SunONE App Server

16

PushToTest

Next I would like to present a case study of a project I recently completed for Elsevier Science. This will show the benefits of test agents in action and show how your choice of Web Service tools will impact the performance and reliability of a Web Service system.

Elsevier Science is the leading research content publisher to the medical industry. Their next generation platform uses SOAP to build application-programming interfaces. They needed to know if their choices of SOAP messaging styles would scale and perform to handle millions of transactions.

PushToTest built a new test environment for Elsevier Science. The new environment includes a Test Web Service that handled RPC, RPC/Literal, and document-style SOAP messages, and intelligent test agents to interact with the Web Service.

We installed the test environment on IBM WebSphere, BEA WebLogic, SunONE Application Server, Apache SOAP and Apache Axis. The results showed performance differences of as much as 176% between the application servers and huge scalability problems between the different types of SOAP message styles.

With Sun Microsystems support I ran the tests on advanced Sun Solaris servers. I then trained the Elsevier team (developers, QA, IT) to use the test environment themselves.

Elsevier Science will use the test environment when new application server software and new builds of the Elsevier Science software become available.

Methodology

- Measure transactions per second
 - Measure at client side only
 - Round-trip times
 - Successful operations only
- Increasing levels of load and payload size
- Test Web Service (TWS)
 - SOAP RPC, SOAP RPC Literal, Document
- TestMaker Agent using Apache SOAP

17

PushToTest

Elsevier's goal was to understand the performance impact of different SOAP encoding styles. We built the test environment by customizing TestMaker to support SOAP RPC and SOAP document-style requests, and by implementing a Test Web Service (TWS) that responds to both encoding styles.

The request to TWS contained two parameters, one to define the size of the response and the second to introduce a delay before responding. TWS responded by creating a response document containing random jibberish words that appeared in 5 response elements, each element had one child element.

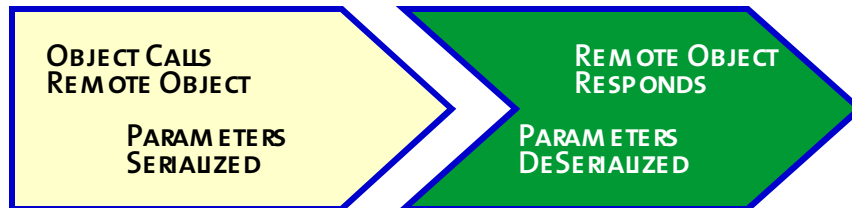
A TestMaker test agent uses the Apache SOAP library to make requests to TWS. TWS is implemented on all of the application servers, each one required a different implementation.

The test agent varied the number of concurrent requests to TWS and the payload size of the response. The test agent logged the results to a delimited log file, which is then summarized by a tally script.

The tally script determined the Transactions Per Second (TPS) value by counting the timing values for the successful transactions. Success was determined by the absence of transport or SOAP faults.

SOAP Encoding Styles

- SOAP RPC



- Developer ignores transports, envelopes
- Automatic encoding
- Complex data types
- Everything is serialized

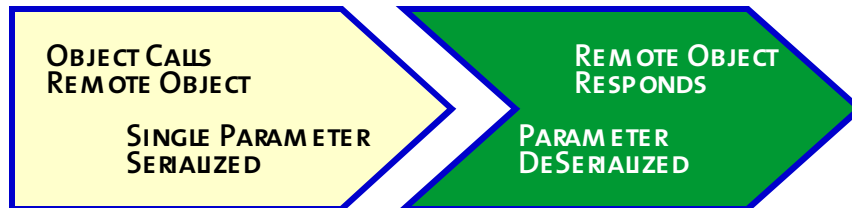
Before I show you the results, let me make sure you understand the differences between the three styles of SOAP encoding.

SOAP RPC is an encoding style that offers the most simplicity to developers. The developer makes a call to a remote object, passing along parameters. The SOAP stack must serialize the parameters, deal with transports such as HTTP, receive the response, deserialize the response parameters and return the results to the calling method. Whew!

SOAP RPC handles all the encoding and decoding, even for very complex data types. Each parameter is serialized and deserialized. And the call is bound automatically to the remote object.

SOAP Encoding Styles

- SOAP RPC Literal



- Developer ignores transports, marshalling
- XML data already encoded
- Only one element to serialize

19

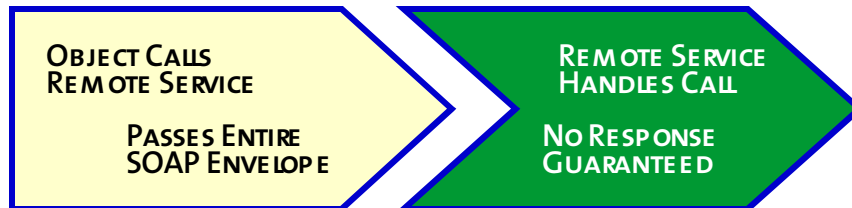
Now imagine you are a developer with some data already in XML format. SOAP RPC Literal encoding lets you pass a single parameter to a remote object. The single object might be the root element in an XML tree.

The SOAP stack still deals with the transport issues to get the request to the remote object. The stack binds the request to the remote object and handles getting the response.

Since there is only a single parameter - the XML tree - the SOAP stack only needs to serialize one value.

SOAP Encoding Styles

- Document-style SOAP (Messaging)



- Developer handles transports, marshalling, XML data manipulation, everything

20

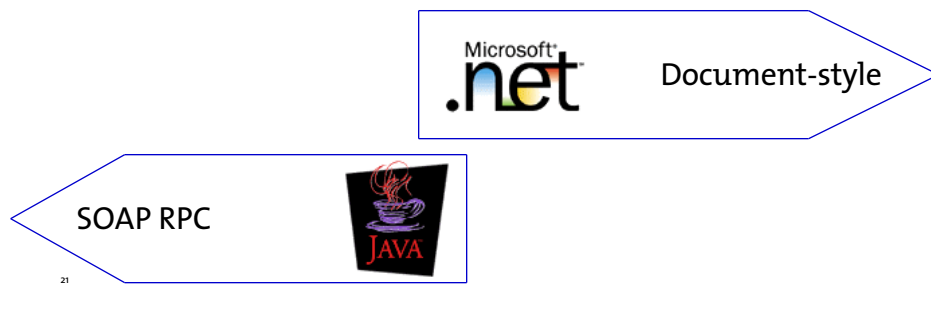
SOAP can also be used to create message-style services. A SOAP envelope is sent to a server without requiring anything be returned to the requestor. The message can contain any sort of XML payload that is appropriate to the remote service. This is the reverse of SOAP RPC calls where the return value is a necessary part of the method invocation process.

With document-style SOAP requests the developer handles everything, including determining the transport (HTTP, MQ, SMTP,) marshalling and unmarshalling the body of the SOAP envelope, and parsing the XML in the request and response to find the needed data.

The developer has to put up with all of the overhead of making the request, but at the end of the day document-style SOAP calls may be more flexible over time and during maintenance.

SOAP Encoding Styles

- SOAP RPC
- SOAP RPC Literal
- Document-style SOAP



While the three types of SOAP encoding methods provide a good range of power and flexibility they also introduce interoperability problems.

Most of the SOAP tools on the Java platform default to SOAP RPC encoding styles. For example, when using IBM WebSphere Application Developer the default encoding style is set to SOAP RPC. And yet when making a call to a service that expects document-style SOAP calls one could instead receive a SOAP exception.

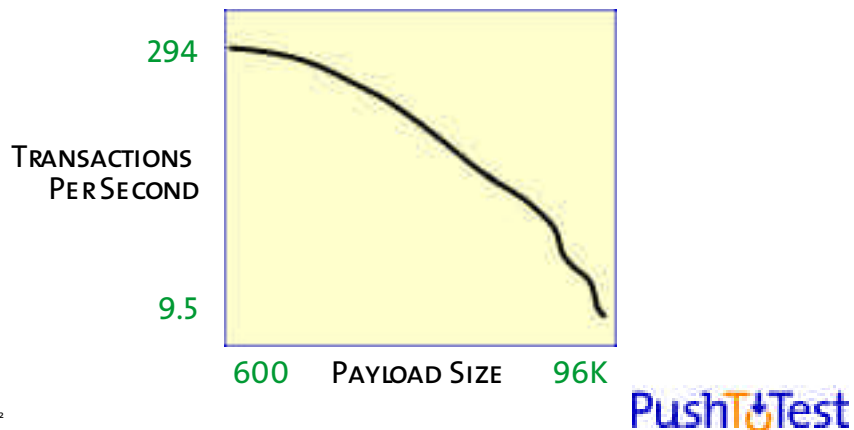
Conversely, .NET development tools implement document-style SOAP calls by default.

It's like watching two boats pass in the night. Both can interoperate but developers need to be wise to the different encoding styles to avoid problems.

Additionally, the choice of encoding style greatly determines the scalability and performance of a Web Service.

What We Found

- SOAP RPC is not scalable

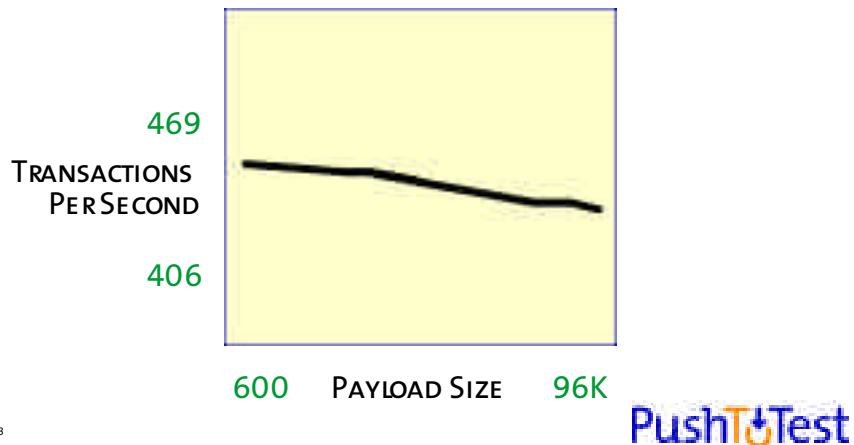


In the Elsevier scalability study we found that a developer's choice of encoding style greatly determines the scalability and performance of a Web Service.

The SOAP implementations universally showed scalability problems when using SOAP RPC encoding on large payloads. As this chart shows, the test agent recorded 294 transactions per second when making requests where the response SOAP envelope measured 600 bytes of SOAP RPC encoded data. As the test agent increased the response size the transactions per second plummeted. When making requests of 96,000 bytes of SOAP RPC encoded data the agent measured only 9.5 transactions per second.

What We Found

- Document-Style SOAP Scales Well



When the test environment uses SOAP Document-style encoding the scalability fared much better. Overall the transactions per second (TPS) values were higher. At the low response size the test agent measured 469 TPS. Remember that the SOAP RPC encoded requests give 294 TPS.

Additionally, when the test agent increased the response size the TPS values did not degrade significantly when using document-style encoded responses.

What We Also Found

- Every server implements a SOAP stack
 - WebLogic = JAX-RPC & JWS
 - WebSphere = SAX-based Axis, DOM-based Apache SOAP
 - .NET = SAX-based CLR objects
- Tools default to RPC or Document style
- WSDL helped but not entirely, network monitor helped most
- Moving targets as functionality and performance improve

24

PushTest

While building the test environment I noticed that each Web Service platform had its own implementation of a SOAP stack - some even shipped with more than one stack. For example, BEA WebLogic Server comes with a SOAP stack that implements the JAX-RPC API and one that implements the Java Web Services (JWS) APIs. IBM comes with the DOM-based Apache SOAP library, but also has the SAX-based Apache AXIS library. I would love the opportunity to test the differences between all of these in some future project.

I also noticed that while WSDL did a fair job at describing the interface to a SOAP service that many times WSDL was not complete. I found a network monitor was necessary to see actually what values were moved over the HTTP transport.

One thing that struck me while building the test environment for Elsevier Science is the nature of SOAP implementations. In the 4 month period from the start of the project to completion, BEA and IBM announced major new versions of their Web Service application servers.

Reliability and performance of SOAP application servers is a moving target. This further reinforced the lesson that a test environment is necessary to check reliability and performance now and as new implementations become available.

Elsevier Benefits

- A new test environment
- Test new SOAP implementations
- Test new software
- Monitor systems for reliability
- Understand performance



With the test environment complete Elsevier Science now has the means to understand reliability, scalability and functionality of its systems. The test agent technology will be employed as new builds of the Elsevier software and the Web Service application servers become available. The test environment is planned also to provide ongoing tests for regression, functional tests, and as a proof-of-service monitor.

Resources To Learn More

- fcohen@pushtotest.com
- www.pushtotest.com
- docs.pushtotest.com
- www-106.ibm.com/developerworks/
- www.theserverside.com
- www.gotdotnet.com

26

PushTest

Please feel free to contact me with questions. These resources will likely provide you with additional information on building reliable and well-performing Web Services.

Thank you for your kind attention during this speech.

-Frank Cohen, September 26, 2002