



Load Testing SOAs which Utilize Web Services

How to Leverage Existing Tools when Testing
Service-Oriented Architectures Based on Web
Services

Last Updated: 7th May, 2007



Introduction

Service-Oriented Architectures (SOA), despite being a hot topic, are far from new. In the recent past, Microsoft's Distributed Component Object Model (DCOM) and the Object Management Group's Common Object Request Broker Architecture (CORBA) represented the state-of-the-art in SOA for intranet implementations, but their inherent complexities have hampered their adoption on the internet. What is new is the migration of SOA onto the web.

Today's internet is dominated by the use of web browsers; therefore the logical path for SOA on the web is to adopt the technologies used by browsers. This is the fundamental basis of web services. The web services implementation that is almost ubiquitous is the Simple Object Access Protocol (SOAP). It is a combination of eXtensible Markup Language (XML) standards and the web's underlying network protocol, the HyperText Transport Protocol (HTTP).

So, we've peeled away some of the layers of the SOA onion and found HTTP sitting in the middle. This is good news for load testing. Most load testing tools use a capture/replay paradigm for recording specific protocols into a trace file. After completing the capture, the tool will generate an executable script that replays the captured protocol and produces network traffic to simulate the client/server interaction described in the trace file. Multiple copies of this script can be executed simultaneously to simulate multiple users, or connections, and thus generate a load on the system under test. In the case of web services delivered over the internet, we know that the network protocol used is HTTP. This is the same protocol used by browsers such as Internet Explorer to deliver content to end users, hence the 'web' in web services. From a load testing perspective, what this also means is that if your tool currently has the capability to record and replay end-user, browser-based loads then it also has the capability to generate web services loads as well.

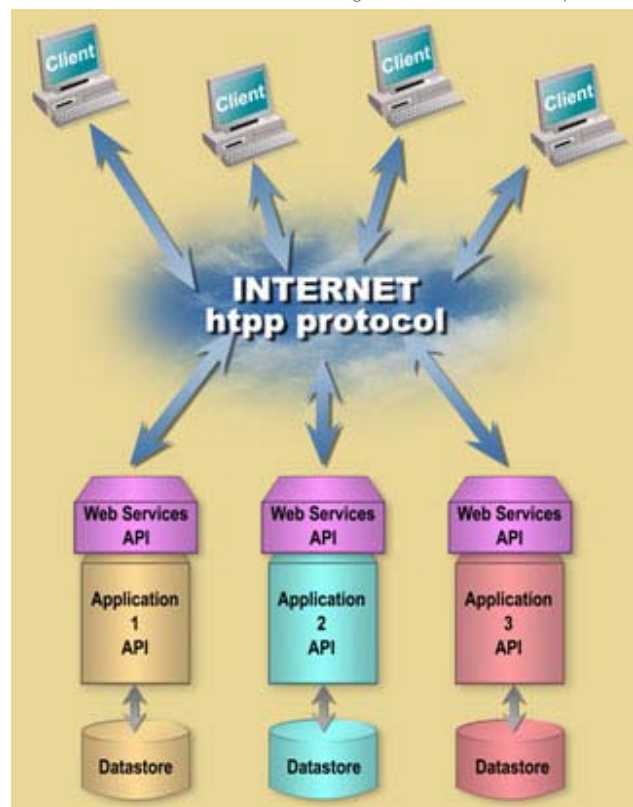
Potentially, your load test tool of choice will have web services specific functionality as well, although this may require additional licensing (i.e. more money) to use. This white paper will describe the steps required to create scripts at the HTTP protocol level that can be used to load test SOAP Web Services. This will provide you with the information to determine if purchasing additional web services functionality for your load test tool can be justified over the additional effort required to create scripts using the HTTP capabilities available in the tool.

It is not practical to show how all and any available load test tools could be used for web services load testing, but undoubtedly a worked example is the clearest way to illustrate the process. To do this requires a scripting environment freely available and a web service commonly accessible on the web.

This white paper provides a practical, working example of constructing a SOAP testing script for the Google Web Services API (<http://www.google.com/apis/>) using the scripting language Python (<http://www.python.org>). It is hoped that everyone who uses the web will be familiar with search engines in general and can work their way around Google searches.

If you are unfamiliar with Python, please do not let that put you off. Python is a very readable language and any language-specific elements have been kept to a minimum in order to aid understanding and increase portability. Obviously, the script presented will probably not work directly in your load test tool, but the construction of the

Service-Oriented Architecture Delivering Web Services 2 Pre-requisites



script should be sufficiently generic to enable you to re-code the example using your load tool's scripting languages and Web/HTTP support libraries.

Pre-requisites

As a learning exercise, we will be constructing a script to test the publicly available Google Web Services using the open-source scripting language Python. Despite being publicly available, the Google Search API requires that you supply a license key to operate correctly. The license key is free and allows you to make up to 1000 queries via the API per day, but you will need to have, or to open, a Google Account to receive your license key. Just follow the instructions on the Google Web Services API homepage (<http://www.google.com/apis/>) and you should be fine. At the end of the process you will have:

- ▶ The Google Web API developer's kit;
- ▶ A license key;
- ▶ A Google Account.

If you do not need the Google Account, you do not have to use it again after you have obtained the license key.

There are a couple of options for getting Python; it is available from the Python website at <http://www.python.org>, but a somewhat friendlier package for Windows users is available from ActiveState (<http://www.activestate.com/python>). Click on the 'Download Now' link under the ActivePython heading. You will be asked for some personal details - but they are not required and you do not need to fill them in - click on the 'Next' button and then download the appropriate package.

The Windows packages use the standard Windows Installer, so installation of the package should be very familiar and quite simple. Please note; you will need administrator privileges to install Python on your computer. Once Python has been installed, you should invoke the Pythonwin IDE from the Start menu in the usual way (see Figure 1).

Once the Pythonwin IDE appears, open a new file by pressing the first button on the toolbar and choosing the 'Python Script' option. At this point, even though there is nothing in the new file, you should save the file in your preferred directory using a suitable name. You do not need to specify a suffix for the file, Python scripts are saved as .py files by default.

I prefer to layout the file window and the interactive window so that I can see both the script and its output together (see Figure 2). Figure 2 also shows a script with a simple for loop. Three useful things can be seen in this example; the first line shows the declaration of a list object that contains three string objects within it. The next line of code shows that the for loop is ended with a ':'. Python uses the colon character to indicate the start of a code block. The block itself is identified by the indentation of the block rather than explicit block markers such as begin, end or '{', '}' pairs. The final print statement has returned to the previous indentation level so is not considered to be part of the code block associated with the for loop, hence it is executed after the completion of the loop.

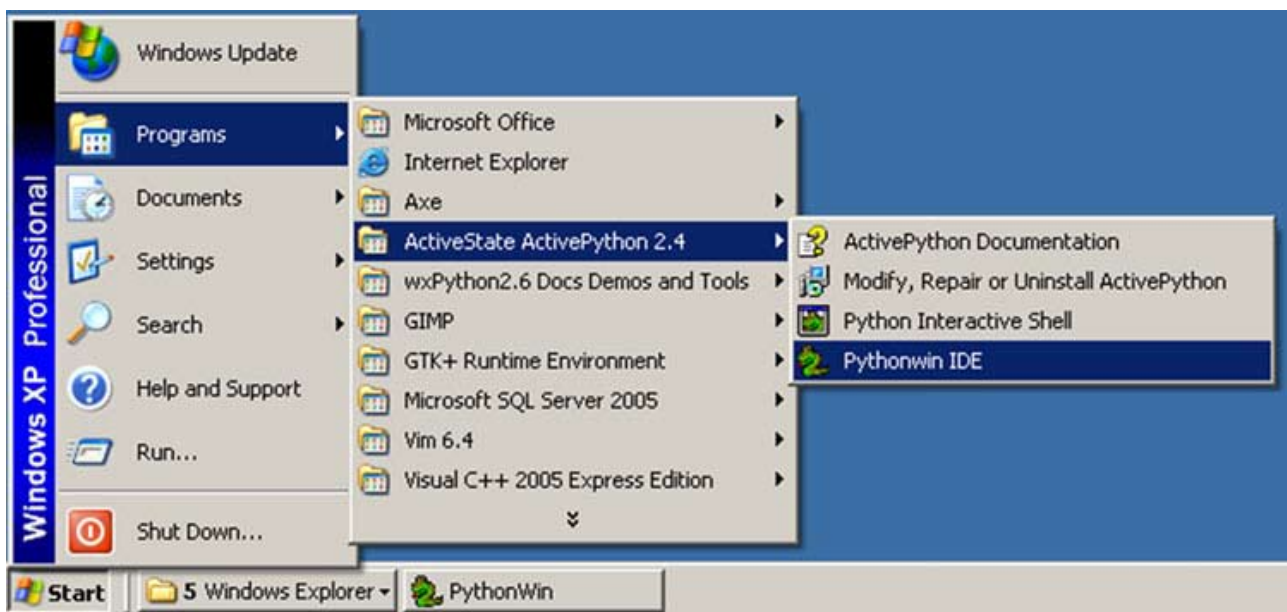
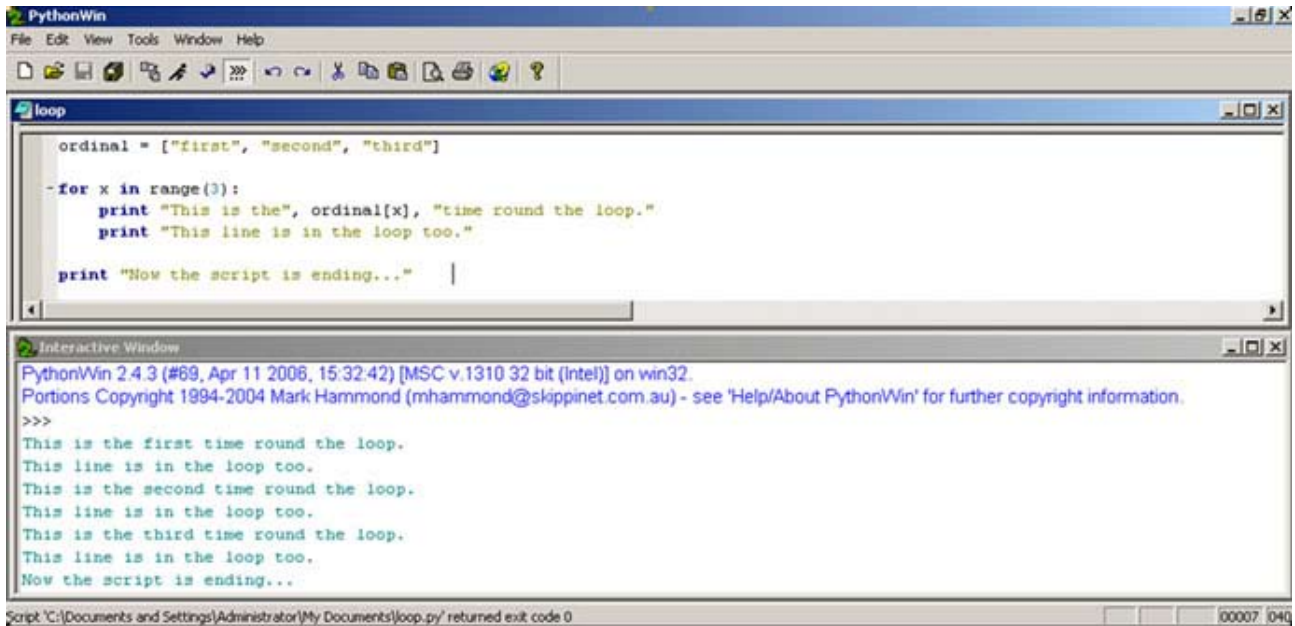


Figure 1: Selecting the Pythonwin IDE



```
PythonWin
File Edit View Tools Window Help
loop
ordinal = ["first", "second", "third"]
for x in range(3):
    print "This is the", ordinal[x], "time round the loop."
    print "This line is in the loop too."
print "Now the script is ending..."

Interactive Window
PythonWin 2.4.3 (#69, Apr 11 2006, 15:32:42) [MSC v.1310 32 bit (Intel)] on win32.
Portions Copyright 1994-2004 Mark Hammond (mhammond@skippinet.com.au) - see 'Help/About PythonWin' for further copyright information.
>>>
This is the first time round the loop.
This line is in the loop too.
This is the second time round the loop.
This line is in the loop too.
This is the third time round the loop.
This line is in the loop too.
Now the script is ending...
Script 'C:\Documents and Settings\Administrator\My Documents\loop.py' returned exit code 0
```

Figure 2: Selecting the Pythonwin IDE

Calling the Web Service

What happens when a client sends a Web Service request?

In order to interact with the web service on the server, we must send it a correctly formatted HTTP request that contains appropriate HTTP headers and the required XML within the body of the request. In SOAP terminology, the HTTP request body contains the SOAP envelope that the web service 'opens' after delivery to unpack the parameters of the SOAP web service request. Next, the web service request is actioned by the server and an HTTP response is returned to the client. If the request is OK, the response should contain some results for further processing at the client end. If the request was wrong in some way, an HTTP error code will be generated and the body of the response may provide some additional information as to the nature of the error in the SOAP request.

What does a Web Service call look like?

How do we find out what to send to the server? One way is to look at the Web Service Description Language (WSDL) file for the web service in question. The WSDL file is the formal XML definition of the web service. It can be used to generate language bindings for developers who have to implement the functionality of a new web service. In theory it can be used to construct the SOAP envelope for the HTTP body, but this is not usually a productive approach for testing.

Another way to find out the required XML is to capture the traffic between an existing client and the server, using your current load test tool. Typically, developers can provide a test client that can be used to verify the functionality of the web service. Capturing a trace from this program will provide the basis of the HTTP request.

Finally, it is not uncommon to be able to find the XML request/response pair provided to you directly. This is the case with the Google Web API developer's kit. In Figure 3 we see the contents of the file doGoogleSearch.xml, which is supplied in the kit.

Of most interest to us are the highlighted fields for using your license key ('key') and the search terms ('q'). There are other parameters that can be changed and these are documented fully in the Google API developer's guide, although we will not be using them in this simple example.

We need to paste the example into our script file and, using the facilities of our chosen tool, enable the script to modify our preferred parameters. In Python we can do this using a template string that allows parameters to be substituted at run-time. Python allows you to try code samples in the Interactive Window directly, as with the example shown in Figure 4. See how the '%s' placeholders are substituted with the actual strings provided at the time the print statements are executed.

As stated earlier, you will find that your own load test tool provides support for the HTTP protocol via a module or library of functions that you can use. Python also provides an additional module for handling HTTP functionality called

'httplib'. The complete script for creating a Google search SOAP request is shown in Appendix A. If all goes well, when you run the script you should get something like the output shown in Appendix B. This output from the web service is XML data that corresponds to the results that would have been returned to a web browser if you were using the typical Google browser user interface. You can check this by taking the search terms from the script and inputting them directly, then comparing the results from a browser search with the results from the web services search.

You should now be able to map the construction of the simple Python script presented here into the scripting

Figure 3: doGoogleSearch.xml

Figure 4: Using a string template in Python

```
>>> aboutMe = ""My %s is %s""
>>> print aboutMe%("name", "Joe")
My name is Joe
>>> print aboutMe%("hair", "black")
My hair is black
>>>
```

language of your load testing tool. You can validate its operation against the Google Web Service by comparing its output to the output of the Python script. Once you are confident that the load testing tool's script is working in the same manner as the Python script, then you can repeat this exercise against the web services you actually want to test. Finally, using the facilities of your load testing tool, run multiple copies of your own web services scripts concurrently to generate the required level of load.

Conclusion

SOAP based web services are directly implemented using the web's HTTP protocol. This open standard allows different implementation technologies to interoperate with SOAP Web Services seamlessly, irrespective of the underlying technologies at either end of the transaction. For example, J2EE systems can interact with .NET systems and vice versa. In this example, we used Python to interact with the Google Web Services. We have no knowledge of the implementation details - and from an acceptance testing perspective we really do not care. We are only interested in web services functioning according to their specifications and/or Service Level Agreements (SLA). The SLA is then performance tested against the agreed Key Performance Indicators (KPI) which should be defined within the SLA. As web services operate across HTTP connections, any protocol-level load testing tool that you already have available should be capable of load testing web services as well as end-user web sites.

AppLabs has a wealth of experience in all aspects of automated load testing and the knowledge transfer that could increase the success of your test programme.

Appendix A: Complete SOAP Example Script

```

from httplib import *
SOAPTemplate = """<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:
SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/1999/XMLSchema">
<SOAP-ENV:Body>
<ns1:doGoogleSearch xmlns:ns1="urn:GoogleSearch"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<key xsi:type="xsd:string">%s</key>
<q xsi:type="xsd:string">%s</q>
<start xsi:type="xsd:int">0</start>
<maxResults xsi:type="xsd:int">10</maxResults>
<filter xsi:type="xsd:boolean">true</filter>
<restrict xsi:type="xsd:string"></restrict>
<safeSearch xsi:type="xsd:boolean">false</safeSearch>
<lr xsi:type="xsd:string"></lr>
<ie xsi:type="xsd:string">latin1</ie>
<oe xsi:type="xsd:string">latin1</oe>
</ns1:doGoogleSearch>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>"""
# Google search options - for populating the SOAPTemplate
key = 'PUT YOUR OWN LICENCE KEY HERE'
searchTerms = 'Magical Trevor dugongs manatee'

envelope = SOAPTemplate%( key, searchTerms ) # populate the outbound SOAP

# create additional HTTP headers
headers = {'Content-Type': 'text/xml; charset="utf-8"', 'SOAPAction': ''}

search = HTTPConnection('api.google.com')
# prepare an HTTP connection
search.request('POST', '/search/beta2', envelope, headers) # 'POST' the envelope

# fetch the HTTP response codes, then the answer to the SOAP request
response = search.getresponse()
answer = response.read()
# show the request and the response
print "----Sending Envelope-----"
print envelope
print "----Getting Response-----"
print response.status
print response.reason
print
print "----Getting Results-----"
print answer

```

Appendix B: SOAP Script Output

```

----Getting Response-----
200
OK
----Getting Results-----
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/1999/XMLSchema">
<SOAP-ENV:Body>
<ns1:doGoogleSearchResponse xmlns:ns1="urn:GoogleSearch" SOAP-ENV:
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<return xsi:type="ns1:GoogleSearchResult">
<directoryCategories xmlns:ns2="http://schemas.xmlsoap.org/soap/encoding/"
xsi:type="ns2:Array" ns2:arrayType="ns1:DirectoryCategory[0]">
</directoryCategories>
<documentFiltering xsi:type="xsd:boolean">true</documentFiltering>
<endIndex xsi:type="xsd:int">10</endIndex>
<estimateExact xsi:type="xsd:boolean">false</estimateExact>
<estimatedTotalResultsCount xsi:type="xsd:int">149</estimatedTotalResultsCount>
<resultElements xmlns:ns3="http://schemas.xmlsoap.org/soap/encoding/"
xsi:type="ns3:Array" ns3:arrayType="ns1:ResultElement[10]">
<item xsi:type="ns1:ResultElement">
<URL
xsi:type="xsd:string">http://en.wikipedia.org/wiki/Weebl&apos;s_cartoons</URL>
<cachedSize xsi:type="xsd:string">47k</cachedSize>
<directoryCategory xsi:type="ns1:DirectoryCategory">
<fullViewableName xsi:type="xsd:string"></fullViewableName>
<specialEncoding xsi:type="xsd:string"></specialEncoding>
</directoryCategory>
<directoryTitle xsi:type="xsd:string"></directoryTitle>
<hostName xsi:type="xsd:string"></hostName>
<relatedInformationPresent xsi:type="xsd:boolean">true</relatedInformationPresent>
<snippet xsi:type="xsd:string">There is a banner that says &quot;&lt;b>Magical
Trevor&lt;/b>&quot; above the stage.
&lt;b>...&lt;/b> it&#39;s quite&lt;br>ugly&quot;</snippet>
<summary xsi:type="xsd:string"></summary>
<title xsi:type="xsd:string">Weebl&#39;s
cartoons - Wikipedia, the free encyclopedia</title>
</item>
</resultElements>
<searchComments xsi:type="xsd:string"></searchComments>
<searchQuery xsi:type="xsd:string">Magical Trevor dugongs manatee</searchQuery>
<searchTime xsi:type="xsd:double">0.276183</searchTime>
<searchTips xsi:type="xsd:string"></searchTips>
<startIndex xsi:type="xsd:int">1</startIndex>
</return>
</ns1:doGoogleSearchResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```