Large-scale server-side infrastructure for e-learning: development, design and experience

Dr Neil Simpkins
Department of Communication and Systems
Faculty of Mathematics, Computing and Technology
Open University
Walton Hall
Milton Keynes,
MK7 6AA, UK
Email: n.k.simpkins@open.ac.uk

Neil Simpkins is currently a lecturer in web application development at The Open University and has developed a range of short courses for the university’s ‘Web Application Development’ certificate, as well as being a contributing author of new courses such as ‘E-business technologies: foundations and practice’, ‘My Digital Life’ and ‘Web Technologies’.

He has worked as a project leader for an international software house, initially developing enterprise level web applications and later working on application security for a range of leading financial institutions.

He has lectured at several other universities, primarily developing and delivering courses in programming, language processing and data mining. He worked as an engineer developing network and agent technology for a range of EU funded projects. He has also worked at the European Commission as a consultant, first in language processing technology and later in programme level technology project concertation. He has an honours degree in computing and a doctorate in natural language processing.
ABSTRACT

The rapid growth of E-business has greatly increased the demand for technology graduates with experience in server-side technology and has thus become an increasingly important area for educators.

Server-side skills are in increasing demand and recognised to be of relatively greater value than comparable client-side aspects (Ehie, 2002; e-skills 2011). In response to this many educational organisations have developed E-business courses but their approaches cannot generally be applied in the distance learning context.

Here the design, development and subsequent experiences of a scalable architecture for the provision of a set of server-side applications to a very large number of students are described. This infrastructure is intended to allow students to gain valuable experience of server side technology such as directory services, deployment and management of web services and other administrative applications.

Whilst students can be supported in installing the server software used in courses on their own machines, it is not possible to guarantee that this type of sophisticated software will function on such a wide range of platforms and in the context of other conflicting software, without very prolonged intervention which is not practical within the timescales of a course. To allow server side aspects to be included as a component of the course’s assessment with some fairness it is necessary to guarantee students access to such facilities even if this is not possible on a student’s own machine.

Providing server-side infrastructure is important so that students gain insight into central concepts behind server applications and management but also provides experience of other related aspects such as network operation, transmission delays and failures and ‘just in time’ collaboration, which basic characteristics of many distributed applications.

A key objective of this work is to establish a framework which can be applied in education and commerce to support very large-scale deployment of web applications and services for applications which themselves may have quite different purposes and properties.

KEYWORDS: e-Learning, distance learning, technology education, distance education, web technologies, online course, scalable web applications, server-side platforms, web application architecture.
INTRODUCTION
Many E-learning and E-business technology courses use some specialist software which, along with other course materials, such as Portable Document Format (PDF) documents and video, is distributed on Digital Versatile Disc (DVD) or by download.

Client-side software can be installed on student machines with the aid of a ‘wizard’ which guides the student, step by step, through the installation and which can also try to ensure that the platform is suitable for the software to function properly.

This type of software may typically include web browsers, programme development toolkits, libraries and Integrated Development Environments (IDEs), such as eclipse (http://www.eclipse.org/).

The range of student platforms and pre-existing software which can interact or conflict with the course software presents a significant source of potential problems. In practice, the occurrence of problems can be greatly reduced by careful engineering of the installation, rigorous cross platform testing and, for example, providing a fully independent installation.

An independent installation may duplicate supporting software which is already installed on the machine to ensure, for example, that appropriate software versions are used. A good example of this is the installation of Eclipse for which java may be installed, even if java has been installed on the machine previously, so that Eclipse runs on a known version of Java.

The rise of network computing and Internet based applications running on web and application servers has resulted in a greatly increased demand and importance for providing students with reliable access to server-side applications and middleware. Understanding, skills and experience of server-side aspects are of relatively greater value than comparable client-side aspects (Ehie, 2002; e-skills 2011) and in increasing demand.

In response to this a wide range of higher education establishments have developed E-business courses which include server-side components and there have been evaluations of a range of the mainstream alternative platforms that can be deployed to support server applications (Sandvig, 2007).

Providing server-side facilities for students on their own machines is far more problematic than installing simpler client-side applications. Server software is more sensitive to platform variations and requires substantial expertise for configuration, management and the diagnosis and resolution of any problems.

Students cannot in general be expected to come with the necessary skills to successfully install and manage server-side software unaided. In addition, in the context of a course aimed at information technology management, students cannot be expected to be equipped with accomplished programming or software configuration skills. The consequence of these factors is that any provision of experience in server-side areas in a purely ‘at a distance’ E-learning context is extremely unusual.

There is also a need to provide a ‘level playing field’ for students and to be able to assess their work on a server-side platform. In terms of a level field it should be the
case that a student should not be disadvantaged because they have some unforeseen problem installing the software against another student for whom the software installs without problem. For assessment purposes it is useful to be able to actually run, rather than rely on inspection alone, the server-side solutions which students have created and deployed. The only means by which student work can be assessed in this way, in a distance learning setting, is by providing a standard server platform onto which students can remotely deploy and test their own work.

In the following sections the development of an architecture is outlined which provides support for running a very large number of instances of a web application called ‘Axis2’. This web application has been chosen as an illustration as it provides the appropriate facilities for students to deploy and manage their own web services on a server, which is a non-trivial task in itself, and this is an interesting component used in a new course developed at the Open University (OU).

However, the architecture has far wider relevance in education and commerce for supporting very large-scale deployment of web applications and services. The same architecture and principles involved can be employed to support very large numbers of users accessing a single (multi-user) web application or service, or to support large numbers of instances of any other web application.

A CASE STUDY: Providing Web Service Architecture

A recently developed course at the OU ‘E-business technologies: foundations and practice’ (Open University, 2009) introduces the wide range of rapidly changing technologies and business aspects underlying modern E-business models. The course helps students to develop skills to enable them to understand, appraise, select and manage complex solutions based on the evolving set of protocols and E-business technology standards.

The decision to attempt to include practical server-side technology aspects into the course was generally viewed as high-risk because of the novelty of this component and the possibility that the solution might break down given large student numbers. To mitigate risks the course presentation was publicised as a ‘pilot’ and a fallback solution was designed whereby the practical server-side elements of the course could be omitted from the course’s assessment even during the course presentation.
The course is divided into four blocks; E-business, basic protocols, web services and service orchestration. The web services block aims to provide students with some practical experience of the elements that make up Service Orientated Architecture (SOA) (Figure 1) and the typical realisation of SOA as Web Service Architecture (WSA) which is illustrated here later.

The standard SOA model (Cerami, 2002; Papazoglou, 2007) introduces three key actors:

1) A web service ‘provider’ who makes the service itself available and who is also responsible for publishing a description of the service in one or more service directories. The service itself can also be hosted in a number of different locations if some redundancy is appropriate.

2) A web service ‘broker’, who maintains a directory of services which holds descriptions of web services. A web service description can also be published in more than one directory to provide some redundancy.

3) A service ‘requestor’ or client who wishes to find and use a service which provides a specific function that is required as part of some application. A ‘client’ may be a user interacting via an interface, an application, or another web service which is combining the service with others to provide a higher level service.

The elements of this block are illustrated in Figure 1. The web service provider first publishes service descriptions to a number of directories. Access to publish in a directory normally requires registration and authorisation. The service directory is then accessed by potential clients who can search using keywords for candidate services. A single search will return a list of potential services matching the search criteria.

If a client identifies a potentially useful service, the directory information can be used to construct a request which is sent to the service. The service may then, if some response is appropriate, return a response. For practical operation there are, of course, a whole set of additional considerations which are not included in the model. A service provider may well expect payment for a service and also provide security for a service. These issues are largely the concern of an emerging set of standards which
extend the model with transactions, security etc (Astor & Yendluri, 2004). These areas are outside of the discussion here.

Figure 2  Web service architecture operations

The actual operation of a web service may differ slightly from the earlier SOA model shown in Figure 1. Information in the UDDI is generally incomplete if a client wants to put a request to the service itself directly (Colgrave & Januszewski, 2004). A UDDI entry may consist of a simple text description of a service and little more information may be available except the location of the service known as the service’s ‘endpoint’ (a URL). Given the endpoint a client can obtain a full description of the service’s interface as a Web Services Description Language (WSDL) (Christensen, Curbera, Meredith & Weerawarana, 2001) document. This XML document specifies exactly what information is required to construct a request to the service. It describes an input data that the service expects as well as providing the information required to understand the service’s response message, if one is to be returned. The additional request for a full service description (WSDL) is shown in the extended model in Figure 2.

The approach adopted to provide this model as a practical element in the course has to acknowledge certain constraints:

- Students have email and online access to a tutor but that tutor has no access to the student’s machine at home and may not possess the skills required to diagnose and fix any problems the student would have with software
- Students are not expected to be ‘programmers’. They are interested in and able to deal with technology but not necessarily at the level of coding or configuration
- The ‘client platform’ varies in hardware specification and operating system and can have a wide range of other co-resident software
• Student coursework will include an assignment involving the use of server-side software. This work will be assessed by tutors and to some extent may be assessed by testing its functionality. This will however require that the tutors can examine students’ work running on the platform where it was deployed by the student.

To provide software to support the full web service model it would be difficult to host all the software on a student’s own machine. If a student had any problems then these would be exceedingly difficult and time consuming to diagnose and rectify.

To provide a stable server platform for students to work with and to facilitate assessment the server-side elements of web service architecture are hosted remotely from the student at the university campus.

COMPONENTS AND PROTOCOLS

There are a range of protocols and components that might be used to realise the web services architecture. The path taken was to choose open source components, a choice which importantly eliminates licensing issues, and also to prefer those that can provide a platform at a level above that requiring a deep level of understanding of lower level programming issues.

![Figure 3 Web service operation: component and protocol instances](image)

The web services architecture (Figure 2) is realised by a range of components and protocols shown in Figure 3.

The software used for each of the components is described in the following sections.
**UDDI Registry**

A directory component is most usually realised as a Universal Description, Discovery and Integration (UDDI) ([http://uddi.xml.org/](http://uddi.xml.org/)) registry which is a platform-independent, XML-based registry standard.

There are several freely available implementations of UDDI (International Business Machines Corporation, 2001; Microsoft Corporation, 2008). jUDDI ([Apache Software Foundation, 2011](http://juddi.apache.org/)) is an excellent open source option. This implementation can be most easily run together with Tomcat ([http://tomcat.apache.org/](http://tomcat.apache.org/)) as the application server and with a database, such as MySQL ([http://www.mysql.com/](http://www.mysql.com/)) to store web service descriptions and other information such as user logins and passwords.

The jUDDI software is packaged as a standard web application which has only to be placed in the ‘webapps’ directory of a Tomcat installation to start working. Installation of the database system and its configuration to work with jUDDI is also straightforward (Resler 2005).

![jUDDI Console page](image)

**Figure 4** jUDDI Console page

jUDDI provides a ‘console’ which can be used to search the registry or publish entries for businesses and services (Figure 4). Each function of the jUDDI API is supported by a web page which provides an XML ‘skeleton’ of the call to be made to the API. As an example, the ‘find_business’ link in Figure 4 takes a user to the XML skeleton shown in Figure 5. A user has then only to fill in appropriate parts of the XML, which are indicated by asterisks in the skeleton, and click a button to perform a search or publish an entry. The skeleton need only be partially completed in certain cases. So, for
example when searching, a partial skeleton can be used to retrieve a set of matching descriptions. Wild cards may also be used in searching. In other cases elements in the XML are mandatory, for example when publishing a web service description it must be given a name and owner.

The registry can be searched by keyword using words from one of the registry classification systems (International Organization for Standardization, 2007; United nations development programme, 1989; U.S. Census Bureau, 2007) or by publisher name, for example.

The registry can be searched by keyword using words from one of the registry classification systems (International Organization for Standardization, 2007; United nations development programme, 1989; U.S. Census Bureau, 2007) or by publisher name, for example.

Figure 5 jUDDI ‘find business’ API call web page

A web service registry entry returned by a search will include a plain text description of the service as well as the service ‘endpoint’. The endpoint is simply a URL which can be used to access the service itself usually located elsewhere on a network. Typically a client will use the endpoint to first request a WSDL document, which includes sufficient information for an actual request to be formulated, sent to the service and for the response from the service to be understood and processed, if any response is generated.

In addition to the jUDDI console the registry can also be accessed by students using Eclipse, an IDE which is mainly used in the course to automatically create web service clients but which also has a UDDI browser. Some of the features of Eclipse for web service development are outlined later.
**Web Service Hosting**

An application server, such as Tomcat, provides very good support for running and managing standard Java web applications but this does not include facilities for supporting web service protocols and standards themselves directly. Axis2 (Apache Software Foundation, 2004) provides a simple means to raise the level of the server and provide better support for web services.

Axis is itself a web application which ‘hosts’ other web service applications and handles all the interactions between the hosted application and the outside network so that the application appears to be a standard web service even though it consists only of core functionality for the service and has none of the apparatus required to handle SOAP (Mitra & Lafon, 2007; Snell, Tidwell and Kulchenko, 2001) messages for example.

Axis provides a facility for uploading a web service in the form of an ‘archive’ to the application server using a simple web page. An archive can be based on Java code, for example, which is compiled and then wrapped up with an XML service description file. The XML file simply describes the web service and names the Axis Java class code which handles the exchange of SOAP messages. An archive can be generated using a simple client-side tool, also provided by the Axis project (Apache Software Foundation, 2008).

Axis can also automatically generate ‘on the fly’ a full WSDL document describing a web service based on the content of the service archive. When a client makes a request to a web service using a SOAP message Axis will parse the message and formulate an appropriate call to the code implementing the actual service. Axis will also process any information returned by the application and package this up as the SOAP message which forms the service’s response. So Axis automatically handles client interactions with the actual web service (Figure 3).

Whilst Axis is primarily focused on SOAP it also provides limited but usable access to a web service using REST (Apache Software Foundation, 2006; Richardson and Ruby, 2007).

**Client**

The Eclipse IDE ([http://www.eclipse.org/](http://www.eclipse.org/)) provides a powerful client which can be enhanced with any number of a large range of extensions or ‘plugins’ ([http://www.eclipseplugincentral.com/](http://www.eclipseplugincentral.com/), [http://www.eclipse-plugins.info/eclipse/index.jsp](http://www.eclipse-plugins.info/eclipse/index.jsp)) which are also freely available. To create a rich client with facilities for developing, deploying, testing and using web services the following extensions have been employed:

- The Web Tools Platform (WTP) ([http://www.eclipse.org/webtools/](http://www.eclipse.org/webtools/)) which provides a set of tools for developing and testing web services. WTP provides easy to use means to develop and test a web service locally as well as to access UDDI and use remote service.

- An Axis2 plugin (Apache Software Foundation, 2004) which is used to create a web service ‘archive’ file. This tool allows a web service project developed in Eclipse to be easily wrapped up as a single file which can be deployed to a server running the Axis2 server-side software.
• The soapUI plugin ([http://www.soapui.org/eclipse/index.html](http://www.soapui.org/eclipse/index.html)), which provides similar development facilities to the WTP but which also can be used to ensure that a web service conforms to the WSI ([http://www.ws-i.org/](http://www.ws-i.org/)) ‘basic profile’ which specifies a set of properties that a service must have to have a good potential for interoperability, one of the most important aspects of WSA.

To produce the client for a specific web service with Eclipse is extremely simple. The IDE provides a wizard which, given the URL of a WSDL document, will download and parse the WSDL and then generate a set of web pages which can be used as the client to the web service. The client actually consists of a frameset of web pages which includes a form for entering any data the web service requires (Figure 6). The web pages are automatically generated based on the service’s WSDL document and are then deployed and run on a proxy application server (Tomcat) which takes the data entered into the web form and produces a SOAP message which is forwarded via the proxy to the web service.

![Figure 6 Example web services test client generated by eclipse](image)

The WTP allows students to create a web service very rapidly as it automatics the process of wrapping up a Java class as a web service and will manage a local application server, such as Tomcat ([http://tomcat.apache.org/](http://tomcat.apache.org/)), to which the service can very simply be deployed and then tested.

All that is needed to create a web service is the code that performs the function that the service offers. Eclipse will then automatically:

• Create a WSDL description of the web service
• Create Java code to handle the SOAP messages that are sent to and from the web service
• Start a local application server and deploy the web service to the server
• Create a client to test the service locally.

Once a service has been tested locally the Axis2 plugin can be used to produce an ‘Axis archive file’ which can be uploaded to the Axis2 server. This procedure is also very simple as the Axis plugin provides a wizard and the Axis2 server has a simple facility to upload services. A WSDL does not have to be included in the archive as the Axis2
server software will generate a WSDL based on the archive file which will have the appropriate endpoint specification for the service’s new location.

The WTP also has facilities for searching and publishing entries to UDDI registries. The WTP is simpler and less error prone to use than the jUDDI console as it provides more graphical support and does not require XML coding by the user (Figure 7).

![Figure 7 Eclipse UDDI explorer](image)

**SCALABILITY**

All of the software components used (Tomcat, Java, jUDDI, MySQL, Axis2) are quite mature and have been used in large scale applications. The exception to this is perhaps jUDDI which has a quite basic set of ‘account management’ web pages, for example.

The Axis2 web services platform provides very simple and usable management facilities for web service delivery. The web application is designed to be for use by a single user because, within an organisation, it is common practice that specific staff are charged with deploying web services. Students each need a private copy of the Axis2 platform on which to deploy their own services. The jUDDI application in contrast is multiuser so that it can support any number of users each with their own username and password which authorises them to manage their own UDDI entries. The directory can course be searched and entries retrieved without any credentials.
To allow students to manage their own web services it is necessary to run as many instances of the Axis2 web application as there are students on the course. The course for which Axis2 was required was expected to attract around five hundred students.

There is a very wide range of potential configurations of the server software components that could be deployed and from the starting point it was by no means certain which arrangement, if any, would be sufficiently reliable and scalable to support complex web applications for the target of at least five hundred students.

Each web application running on a single Tomcat server must have a unique name for its deployment directory on the server. This name will also be part of the URL used to access the application on the server. At the Open University it is quite simple to support this by adopting a convention that each copy of Axis2 is named after the unique student identifier given to a student when joining the Open University.

When a Tomcat application server is started up it automatically loads all the web applications that have been placed in the ‘webapps’ directory of the installation. As the number of web applications increases Tomcat takes longer to start up and also uses increased memory. To provide circa five hundred students with Axis2 was thus not straightforward.

Testing Tomcat with increasing numbers of Axis2 deployments it was found that Tomcat ceased to fully start at around seventy eight instances. In addition, just below this limit, Tomcat took many minutes to start up which would mean a substantial loss of service if Tomcat or the physical server itself ever needed to be restarted. At the level of around fifty Axis2 instances Tomcat would reliably start up in an acceptable time for our purposes although the process can require two to three minutes when fifty instances of Axis2 are deployed.

To improve performance and support up to five hundred Axis2 instances it is necessary to employ multiple Tomcat servers connected to a single Apache web server (Figure 8).

![Figure 8 Configuration of Apache and multiple Tomcat instances](image)

The Apache web server can be configured to load one or more ‘modules’ which extend its functionality. A wide range of modules are freely available [http://modules.apache.org/](http://modules.apache.org/) which include ‘mod_jk’ (Shachor 1999) a module which handles communications between Tomcat and Apache. This module is available as a pre-compiled binary object which is ready to be loaded into Apache. To load the
module and specify which HTTP requests are passed to Tomcat requires some straightforward configuration. In addition to supporting a single Tomcat connected to Apache mod_jk allows a set of Tomcats to be connected to Apache. Each Tomcat is configured to listen on a unique port for requests which are passed on by mod_jk. For each Tomcat it is also necessary to specify one or more URL ‘patterns’ which mod_jk matches against the URL of each incoming user request to determine which specific Tomcat should receive the request. The configuration of Apache and Tomcat to use the mod_jk module is described by Shachor (1999).

At the Open University we have written a simple PHP script which, when a student first logs onto the server running Apache, deploys an Axis2 web application instance to one of the Tomcat application servers. The script also records the students unique ID and Tomcat allocation for future use. Students are allocated to an Apache Tomcat on a round-robin basis and remain with the same specific application server for the duration of the course. This approach to allocating students to a Tomcat results in an even distribution of students across the Tomcats so that they are also evenly loaded.

The mod_jk configuration options include the specification of patterns which are matched against requests for pages and other resources. Each pattern is associated with a destination ‘worker’ which in this case is one of the Tomcat instances. Whenever the URL in a request to Apache matches a pattern then it is passed to the corresponding Tomcat associated with that pattern. The patterns we specify, using mod_jk’s ‘JkMount’ instruction, are very simple, so for our first two workers ‘Tomcat a’ and ‘Tomcat b’ (Figure 8) the configuration is:

```
JkMount /axisa* tomcat_a
JkMount /axisb* tomcat_b
```

A student’s homepage is hosted on Apache but all their other pages are hosted on a specific Tomcat instance and the links between the homepage and the student’s applications are adjusted by the PHP script so that the URLs contain the appropriate pattern for the correct Tomcat. A student with their Axis2 deployed on ‘Tomcat b’ might have a URL for this of:

```
http://t320webservices.open.ac.uk/axisb_nks34/
```

This URL matches the ‘/axisb*’ pattern associated with Tomcat b in the JkMount instruction.

As might be expected all the links within the applications, such as Axis2, are either relative or constructed in a relative fashion based on the current URL so no links within an application have to be changed to meet the convention required.

The PHP script also sets up jUDDI credentials in a MySQL database so that the student has the appropriate permissions required to publish UDDI entries. The MySQL tables are automatically created by a script which is provided with jUDDI. The script is provided in forms suitable for a range of database systems including Oracle and Postgres.

A student’s homepage lists the student’s URL for their private Axis2 application, gives their UDDI credentials and has links to the jUDDI and Axis2 main pages (Figure 9). The PHP homepage is the only web content that we have created ourselves. Links from this
page to jUDDI and Axis2 take students directly to the default homepage of the two web applications.

Figure 9 Student server account homepage

A ‘Delete all services’ link is employed on the homepage (Figure 9) so that students have a very simple means to reset their Axis2 account. This function simply deletes all the web service archives that a student has uploaded. This was required as the Axis2 service administration functions fail to delete malfunctioning services in some cases.

Using multiple Tomcats in this way allows five hundred accounts to be supported by ten Tomcat instances for Axis2 instances. Another dedicated Tomcat instance is used to support all the students’ jUDDI accounts and another dedicated Tomcat, again running Axis2, is used to support some demonstration web services that students can access and use.

The entire configuration of Apache and twelve Tomcat servers is run using VMware (http://www.vmware.com/products/vi/esx/) on a single virtual machine which allows the available resources to be increased if necessary without re-installing software. The final configuration used four gigabytes of memory to run all twelve Tomcats together with Apache on Red Hat Enterprise Linux (version 5). This is a surprisingly small use of memory for such a large number of server instances and web applications.

Whenever a student visits their server homepage the PHP script uses their unique identity, which is part of their login to the university web sites, to retrieve the name of the Tomcat on which their account is actually hosted and then creates a set of appropriate URL links dynamically in the homepage. The specification of links in the student’s homepage means that they are consistently directed to the appropriate Tomcat instance. This approach allows a specific Tomcat to be allocated to an application or a user using a URL pattern with persistence even over different sessions.

The mod_jk module is not only able to route requests, according to how the request’s URL matches a pattern but it can itself maintain a single session between a client and a single selected Tomcat server. This functionality is known as ‘sticky session’ support.
This allows a user or application to be dynamically allocated to a specific Tomcat but for the requests and responses to be properly routed between the client and the same Tomcat for the duration of a session. This configuration of the architecture is suitable for an application where, for example, the user logs in then performs some actions and then ends their session by logging out. It makes no difference which Tomcat handles the user’s session in this instance.

It is also possible to switch sticky sessions off which would mean that each request from a client is randomly allocated to a Tomcat. This is a configuration suited to, for example, a simple request and single response application, such as searching a public database.

An additional module ‘lb’ (Shachor, 1999) can be used to provide load balancing across the Tomcat instances in the architecture. This is not a feature that can be used in the configuration for Axis2 as students must be consistently returned to the same Tomcat whatever the load on that instance. In addition to load balancing the ‘lb’ module will identify any Tomcat that is failing to respond to requests and redirect requests to another Tomcat, thus providing ‘failover’ functionality.

The architecture can be configured according to the type of application being used. Three different types can be identified which allow different degrees of load balancing:

- Customisable web applications. If the web application is configured or extended by the user then the application is changed and becomes distinct from other instances of the application. This is the case with Axis2, where web services are deployed inside of the Axis2 web application so that they can be run later. The same application server instance has to be made available to a user every time they use Axis2. To achieve this, a specific URL has to be used, conforming to a pattern which corresponds to a specific server instance. Neither load balancing nor failover can be used with this type of application within the architecture.

- Session based applications. If the web application embodies any degree of the concept of ‘state’, such as a login page with authorisation, then the same application server has to be used for an entire session. In this case it is possible to balance loading at the session level and to use failover, although if a failure occurs the session will usually have to be restarted.

- Simple ‘request-response’ based applications. If the application delivers a simple response without employing any concept of ‘state’, such as ‘authenticated user’ in a secure application, then sessions are not required and load balancing across Tomcats can be performed at the level of individual requests.

**EXPERIENCE OF DEPLOYMENT**

At the end of the pilot run of the course in 2008 the number of deployed Axis2 accounts had reached three hundred and seventy six. These accounts were supported for over two months with very few problems. On a small number of occasions one of the Tomcats would be restarted when it was found not to be serving pages correctly.
In all cases restarting the Tomcat was sufficient to rectify any problem and rapidly return service.

Limited feedback from students has indicated that the server facilities performed extremely well. The university student help desk for reporting problems has also reported very few communications from students concerning problems with their server accounts, which is a strong indicator of a very reliable delivery of services.

From discussions in the course’s online forums it became clear that the server side facilities caused far fewer problems than the other two main components of the applications being used:

1. The client side tools (Eclipse, Tomcat and plugins) have been found to pose problems in a range of ways, some of which can be very difficult to diagnose and correct at a distance. However, after the first run of the course we have experience in the causes of common problems and have compiled appropriate ‘Frequently Asked Documents’ (FAQs). Common sources of problems include conflicts with other software in the use of ports for Eclipse’s local application server, incorrect setting of the environment variables and insufficient access privileges.

2. Network connectivity. Quite commonly we have found that connectivity is blocked by a student’s own firewall or antivirus software. In a much smaller number of cases there are problems, typically signalled by timeouts, due to slow network traffic over the Internet. This last aspect is an inherent characteristic of the service architecture and its operation and, although it might be a problem for students, is a useful ‘real world’ practical experience.

The server side applications we are currently providing for students to use are components making up a part of the SOA model. The operation of this model over a typical network or over the Internet will involve handling delays in messaging, failures and errors. Students are exposed to these problems when they use the services they have created and host under Axis2. This experience is useful as an introduction to the last taught part of the course where students go on to study approaches to handling failure and exceptions which occur within processes which are composed of calls to a number of web services.

A more problematic situation arose when a new version of Java was offered as an automated download and installation to students from Oracle who provide Java. The Eclipse IDE used by students to create web services was also switched by this update to use the newly installed Java. This meant that, for those students that accepted the update, their web services were compiled by Eclipse using a new release of Java. The executable web services produced were thus incompatible with the Java being used to run Axis2 on the server. As the Eclipse installation provided with the course comes complete with the required version of Java it was simply a matter of altering a setting in Eclipse to use this Java. Because Eclipse is quite complicated in having many settings this did cause a few students minor problems.

The altering of installed software in this way was not something foreseen and circumvented even the careful engineering of the course’s software installation which
is ‘complete’ in including all the software required to run the tools on a basic Windows platform.

**MONITORING AND RELIABILITY**

Since 2008 the course has been run each year with between four and five hundred students per presentation. During these three additional presentations we have seen generally similar problems to those in the initial presentation. It is usual that one Tomcat per presentation needs to be restarted but because of the relatively rapid start-up of this component students do not suffer any significant loss of service.

![Axis2 Happiness Page](http://1020webservices.open.ac.uk/ax1020/Axis2/page-happy.jsp)

**Figure 10** Axis2 Happiness Page

The requirement we do have is to be able to monitor each of the many Tomcats and Apache so that as soon as any cease to respond they can be restarted. Each Axis2 installation has a special Java Server Page (JSP) called ‘HappyAxis.jsp’, known as a ‘happyness’ page, which checks that Axis is installed correctly (Figure 10). If the Tomcat is able to respond to a request for this page then it is a good indication that Tomcat is working and if the page contains the string ‘The core axis2 libraries are present’ then this is a good indication that Axis2 is also working well. Experimenting with Axis has shown that this is the key string, displayed in green font, on the page which is replaced by a red coloured error message if any one of several potential problems are detected.

To monitor all the Tomcats we have added a special single deployment of Axis2 to each Tomcat server. This means that we have a fixed location for an Axis2 ‘happyness’ page on each Tomcat. One such page is then read from each Tomcat by a PHP script on the Apache server which searches for the key Axis2 string in the pages to determine the status of each Tomcat (Figure 11).
The Apache ‘happyness’ page is in turn monitored twenty four hours a day by the university’s IT operations staff using a monitoring tool (Nagios, 2011) which constantly requests the page every thirty seconds.

Because of the potential network problems, mentioned earlier, the main Apache ‘happyness checking’ page is allowed to fail twice in consecutive calls before an alarm is raised and human intervention takes place. This arrangement has proven extremely reliable in detecting the rare problem of a Tomcat ceasing to serve pages. Apache itself has, as might be expected, proven to be extremely robust and has yet to suffer any down time.

In addition to monitoring the servers we also have a need to manage student accounts on the server. To facilitate this we have added a PHP based account management page which provides a simple facility to perform the operations we require. When the deadline for completing coursework arrives this page allows all student accounts to be made read only so that the state of students work at that time is preserved for subsequent marking by tutors. The tool also allows accounts to be deleted, allowing the account to be redeployed when the student next logins in, or for an account to be made writable again if a student has, for example, been granted an extension in the time they have to complete the coursework.

CONCLUSION AND FUTURE WORK

This chapter has outlined the desirability for and development of a server configuration which allows the delivery of server accounts and web applications for large numbers of students.

Using this approach the Open University has been able to provide sophisticated server side educational facilities on the scale required to support an important degree level course. Identifying an appropriate architecture from amongst the wide range of
possibilities has allowed the required web service infrastructure to be deployed and has made it possible to accommodate a wide range of server side software applications.

Students are able to gain practical experience with server applications in an environment that is highly stable and reliable. They also gain experience of ‘over a network’ operation of applications. The operation of web services, for example, is difficult to orchestrate when there are delays or failures in responses or errors. Experience of these aspects is fundamental in gaining a real insight into modern distributed technologies and central to an understanding of web service architecture and use of clouds.

Students taking the course have recognised that aspects of the course are novel and challenging and have understood the distinction between server and client-side aspects.

In the future it will be beneficial to extend this work by exploring two additions.

Firstly, as students are currently allowed to upload their own packaged web services to the server the configuration remains susceptible to badly or maliciously coded services. This has not, so far, caused any problems. The situation can be improved in this area by implementing some additional configuration of Tomcat and associated monitoring tools (modules) so that students’ code is limited in its use of resources and we can identify which account is the source of such problems.

Secondly, the provision of a complete remote environment using one virtual machine per student. This environment might include an operating system, desktop, all the server-side software preinstalled and configured, as well as software to provide a local client for the server facilities. Students would then connect to a machine at the University using a simple remote desktop client. This approach has advantages in alleviating software problems for students but presents some problems of access to student accounts for updates, provision of support and assessment. Provision of a suitable and familiar desktop for students within the environment is not a simple matter as open source products are not commonly used and any leading proprietary product would have to be licensed.

This ‘virtualisation’ approach might be seen as more suitable in cases where students are required to actually install and configure the server-side software themselves. This approach also neglects the networking elements (time delays, failure, redundancy etc) and experience that the current approach embodies, which may be an unacceptable loss in terms of learning outcomes.

Having succeeded in delivering novel technology facilities into an E-learning course it will be possible in the future to explore the student experience more fully and to investigate the consequences in terms of the professional recognition and value of these skills in industry. The overall architecture is also currently being applied to other ICT courses which include server-side aspects.
REFERENCES


Richardson, L. & Ruby, S., (2007), *RESTful Web Services*, O'Reilly Media


