Modelling Workflow Using Web Services

K. Porteous

A dissertation submitted for the partial fulfilment of the requirements for the degree of Bachelor of Arts (Honours) at the University of Otago, Dunedin, New Zealand

11 November 2005

Acknowledgements

I would like to thank Dr Stephen Cranefield and Professor Martin Purvis for all their help and feedback throughout the year. Especially to Stephen Cranefield for helping me figure out how to use the tools I needed for my research.
I would also like to thank both of you for taking the time to proof read the documents I had to submit as part of INFO 480; in particular the dissertation and literature review.

I would also like to thank the ChemSearch laboratory for answering all my questions and providing me with the necessary information to model the workflow of your laboratory. This workflow was needed to test my project.

PART A

RESEARCH REPORT

Using Web Service Technology to Model the Business Processes of a Chemical Analysis Laboratory

Student Name: Katrina Porteous

Supervisors: Dr Stephen Cranefield, Professor Martin Purvis

November 11, 2005

Abstract

The use of web services technology is becoming more widespread with many businesses wishing to provide services to their clients over the Internet. Although web services technology is currently the best method for providing services remotely over the Internet it does not provide any concept of state or any way to model workflow. This has resulted in new specifications to deal with these issues, namely WSRF and BPEL4WS.

In order to effectively model certain types of workflow, such as that of the ChemSearch laboratory, there needs to be a method for integrating BPEL4WS and WSRF, so that stateful workflow can be modelled.

This research proposed a method of integration using a proxy service to enable BPEL4WS to support WSRF that was evaluated by modelling the workflow of the laboratory.

The results showed that the method enabled BPEL4WS to work with WSRF and provided a mechanism that ensured a client always had a valid endpoint reference to the WSRF web service.

This research also raised some problems with the specifications and tools used to implement this method that would need to be resolved to ensure the widespread adoption and standardisation of these specifications.

Contents

List of Figures

List of Tables

1 Introduction

There is a growing interest in the scientific community to be able to share resources, data and research. This has introduced the concept of e-science [1], which involves using grid service technology [Hunter et al., 2004] to solve these issues. Grid services are based on web services standards that have been extended for use in e-science applications. Grid services provide middleware that handles security, the ability to locate and invoke the services made available, resource management, etc. Web services is a form of middleware that provides a way of describing, finding and invoking services remotely over the Internet using XML-based standards such as SOAP (Simple Object Access Protocol), WSDL (Web Services Description Language) and UDDI (Universal Description, Discovery and Integration) [Johnson, 2005]. There have been a number of ideas raised in the grid services community that have been scaled-down to a number of specifications that have been proposed as standards in web services.

This research focuses on using one of these standards, namely WSRF (WS-Resource Framework) [Czajkowski et al., 2004], in conjunction with a business process modelling language, such as BPEL4WS (Business Process Execution Language for Web Services) [Andrews et al., 2003], to enable stateful workflow to be modelled. The integration of two such specifications is needed to model certain types of workflow like that of the ChemSearch laboratory, where their clients need to be able to see the status of their report as different tests have been carried out on their samples. WSRF will be used to model the state of the report, while BPEL4WS will enable the workflow of the laboratory to be modelled.

The details of this research, the method of integration and subsequent evaluation from modelling the workflow of the ChemSearch laboratory are discussed in subsequent sections of this document.

[1]E-Science - A term that applies to collaboration amongst scientists, where large amounts of data, resources, etc. are shared through the use of Internet-based technologies [Hunter et al., 2004]

2 Key Terms

This section provides some key terms that are mentioned throughout this document.

A *web service* as defined by the W3C [W3C, 2004a] is "...a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards".

SOAP (Simple Object Access Protocol) is a protocol for distributed environments that uses XML technologies as the message format, for the information that is passed between two nodes in the connection [W3C, 2003].

WSDL (Web Services Description Language) defines the interface of the web service and the operations the service will provide.

State is basically data that is persistent and can be changed via interaction with web services.

A *stateful resource* is a resource used by a web service that has associated state and is implemented as an XML document. It allows a web service to perform operations on data from a previous invocation of the web service. Examples of a stateful resource include a row of data in a database or a single file in a file system [Foster et al., 2004].

A *WS-Resource* is the combination of a web service and a stateful resource.

An *endpoint reference* as defined by WS-Addressing [W3C, 2004b] uniquely identifies a web service on the network. It specifies an address as a URI to the service and optional reference properties that are used by WSRF to uniquely identify a particular WS-Resource.

The *singleton pattern* describes in this context the creation of a WSRF web service when there is only ever one instance of the web service created. This means the service does not require a unique identifier as there is only ever one copy of this web service in existence.

The *factory pattern* defines the mechanism by which new instances, which in this case are WS-Resources, are created. This means all instances are created using the factory instead of creating an instance by using the "new" keyword. By default the class constructor of a resource is made private to force the creation of new instances to occur through the factory. This provides explicit control over the creation of new instances.

A *correlation set* is a group properties that uniquely identify an instance of a BPEL4WS workflow [Andrews et al., 2003].

3 Review of the Related Literature

This section provides a critical analysis of the current literature related to this topic. In particular it relates the literature to the problem this research is hoping to solve.

3.1 Introduction

The introduction of the Internet and its subsequent growth has led to the development of Internet-based solutions to the distributed computing problem. One such solution is web services, which provides a means of describing, finding and invoking services remotely over the Internet [Johnson, 2005]. However, web services technology is not suited for resource sharing within collaborative environments because web services describes services that are available and how they can be invoked. This has led to the introduction of grid technology. The popularity and interoperability of web services technology has influenced the uptake of web services technology for use in grid services. This is to capitalise on the service-oriented architecture prevalent in web services. The requirement of workflow to be modelled to fully support collaborative environments within the scientific community has led to the uptake of BPEL4WS [Andrews et al., 2003]. Since my research focuses on the use of WSRF and BPEL4WS to model stateful workflow, i.e. workflow that relies on the preservation of data between invocations of services, this literature review will firstly discuss BPEL4WS and WSRF and then focus on the use of BPEL4WS in a grid environment combined with WSRF, which allows for stateful resources, to enable stateful workflow to be modelled.

3.2 Web Services

Web services represent a form of middleware that uses XML-based standards such as SOAP [W3C, 2003], WSDL [W3C, 2001] (Web Services Description Language) and UDDI [UDDI, 2004] (Universal Description, Discovery and Integration) to provide services remotely over the Internet [Johnson, 2005]. SOAP is a protocol for distributed environments that uses XML technologies as the message format for the information that is passed between two nodes in the connection [W3C, 2003]. WSDL defines the services that will be offered over the network. Within a WSDL document a number of messages and operations can be defined. A message describes the data that will be transmitted and an operation describes the messages that will be sent and received upon invocation [W3C, 2001]. UDDI defines a structure that describes the services that are available, who makes them available, and the interfaces that can be used to access the provided services [UDDI, 2004]. These standards provide the basis for web services technology that enables the construction of loosely-coupled distributed systems built on a service-oriented architecture [Atkinson et al., 2005]. According to the W3C ([W3C, 2004a]) a service-oriented architecture is "a set of components, which can be invoked, and whose interface descriptions can be published and discovered". A service-oriented architecture separates the interface definitions from their implementations, which means a service only needs to know the definition of the interface not how it is implemented. The use of a service-oriented architecture offers numerous benefits including well-defined interfaces [Pasley, 2005], platform independence and hides service implementation details from the developer [W3C, 2004c].

The adoption of web services technology is increasing, due in part to many businesses wishing to provide services to their clients over the Internet. While web services is suitable for these types of situations, it is not suitable for collaborative environments, such as those in the scientific community. This is because web services describes services that are available and how they can be invoked. They do not enable resource-sharing, which is a vital component of a collaborative environment [Foster et al., 2001]. This has led to the concept of grid computing.

3.3 Grid Services

Grid computing places emphasis on "large-scale resource sharing and innovative applications" while maintaining high performance [Foster et al., 2001, p. 1]. Grid computing revolves around the concept of the grid, which is defined as "a collection of resources owned by multiple organizations that is coordinated to allow them to solve a common problem" [Gartner Group, cited by [Oracle, 2005], p. 3]. The key concept of a grid is to allow collaboration of resources and typically involves large amounts of data.
This concept of a grid can be further broken down into three different types of grids [Gartner Group, ibid]. The first type is a computing grid, which obtains spare processing cycles from different locations to execute large applications. The second is a data grid,

which uses resources obtained from different locations to store the data used by the grid, distributed across the combined resources. The third and final type is a collaborative grid, which "ties together multiple collaboration systems from several owners to allow collaboration on a common issue" [Gartner Group, ibid, p. 3].

This growing interest within the scientific community in being able to share their resources, data, etc. in a collaborative environment, a concept termed e-science [Hunter et al., 2005], has led to the development of grid technology [Foster et al., 2002], which is based around this idea of grid computing. E-science is defined as being "large scale science that will increasingly be carried out through distributed global collaborations enabled by the Internet" [Hunter et al., 2005, p. 2]. The problems with this kind of environment are that it will involve large amounts of data, large numbers of computing resources and high performance visualisation software to display the results in a meaningful way to the user [Hunter et al., 2005].

Grid technologies attempt to solve this problem through "sharing and coordinated use of diverse resources in dynamic virtual organisations" [Foster et al., 2002, p. 5]. A virtual organisation refers to a group of people or organisations participating in a collaborative environment, where it is clearly defined what is being shared, who is able to share it and the conditions under which it is being shared [Foster et al., 2001]. The argument for the use of grid technology is that it will lead to distributed systems that are more reliable, scalable and secure [Foster et al., 2002]. Although grid technology addresses the issues in e-science and may lead to better distributed systems, it has been suggested that grid technology could be combined with web service technology to take advantage of the benefits of web services. These include the use of WSDL for interface definitions that are independent of the transport protocol and data format and the availability of tools to transform the interface definitions into code [Foster et al., 2002]. This combination of grid technology and web services is referred to as an Open Grid Services Architecture (OGSA), which defines this combination as a grid service [Foster et al., 2002].

3.4 Workflow

Workflow is defined as "a set of rules that define the interactions between a set of services in order to be composed into a meta-service" [Krishnan et al., 2005, p. 2]. The concept of workflow is important [Yang et al., 2004] in the grid environment because scientists follow a structured process when conducting their experiments and analysing their results. This has led to a number of tools and specifications being developed to model this idea of workflow, including Triana [Triana, 2005] and GridAnt [GridAnt, 2005].

Triana is "a distributed problem-solving environment that makes use of the Grid to enable a user to compose applications from a set of components" [Taylor et al., 2005, p. 1]. Triana was proposed as a middleware-independent solution in response to the numerous proposed standards offered by grid and web services. It hides the middleware implementation details from the developer; so that the developer does not need to know which middleware technology is being used. This is in contrast to GridAnt, which aims to provide a tool that a scientist could use to model workflow [Amin et al., 2004]. GridAnt provides "a simple, extensible, platform-independent, and client-controllable workflow mechanism" [Amin et al., 2004, p. 2]. While Triana has a number of uses including provision of a workflow engine for grid applications, managing the workflow between grid components and provides a graphical script editor that supports BPEL4WS (Business Process Execution Language for Web Services) [Taylor et al., 2005], GridAnt makes use of an open source tool from Apache [Apache, 2005] called Ant [Ant, 2005]. Ant is a Java build tool that is based on Java classes. It defines tasks that group a collection of commands together to be executed [Ant, 2005]. Additional workflow vocabulary is defined on top of that provided by Ant in order for GridAnt to enable workflow composition.

Although GridAnt is primarily a command-line tool, it does provide a graphical visualisation tool as well. Triana also provides a visualisation tool that is easy to use. This is because it has a smaller subset of functionality than other workflow languages such as BPEL4WS, which makes BPEL4WS more suitable for modelling complex workflow. For example, Triana does not explicitly support control constructs; loops and branching are handled by specific components instead. The argument is that this is simpler and more flexible as it allows more control over the control constructs. However, it is also noted that explicit support of control constructs, such as while loops, is often required by scientific processes, but inclusion of these constructs in the Triana workflow language would detract from its ease of use [Churches et al., 2001].

Systems were constructed to test both tools [Amin et al., 2004], [Churches et al., 2001]. Initially the Triana system underwent a simulation to determine whether Triana was robust at discovering local and distributed services and whether it could be executed on a variety of resources [Taylor et al., 2005]. At this stage no scientific performance tests were done since the focus of the paper is on the Triana system itself and not at this stage for developing workflow. The simulation used four different computers with varying specifications, operating systems and locations. Three of the computers were set up as servers with one acting as both a client, to run the Triana user interface, and a server. The data was distributed to the servers and then sent back to the client and re-ordered. The time was taken to determine how long it took for the data to get back to the client.

It is not clear what analysis was done on the data, but presumably it would be some sort of averaging. It would have been clearer if the results had been graphed or at least presented in a table rather than just discussed in the text.

The results showed that local machines did more processing than remote ones, although some data did get processed by remote machines. This fulfills the requirement of Triana to be able to discover both local and remote services. In order to fully test the robustness of the service discovery three services were hosted on a single server. This returned the data in approximately the same time as hosting each service on a separate server. This demonstrates that the service discovery is robust, firstly, because it was able to locate the different resources even when they are hosted on the same machine and secondly, because it did not take a longer period of time to locate the services. The varying nature of the servers indicates that Triana can be executed on a variety of resources.

Secondly, Triana was applied to the problem of an inspiral search. This involved a discussion on how Triana could be applied to this problem and the steps necessary to undertake the execution of the workflow. This discussion alone is not enough to prove the conclusion that Triana is for the "scientist not interested in computer science implementations" [Churches et al., 2004, p. 9]. In order to prove this a number of scientists would have had to use the system and evaluate it. If the results of such an evaluation showed that scientists with no knowledge of workflow could use Triana then the conclusion that Triana is easy to use would be valid.

In the case of GridAnt a typical scientific application was applied to the system to determine if the system achieved its goals of being a "simple, extensible, platform-independent and client-controllable workflow mechanism" [Amin et al., 2004, p. 2]. This was achieved by identifying the key aspects of the application, what the issue was with the aspect and then relating it to a particular part of the GridAnt functionality. For example, one of the issues with the application was that running an experiment requires extensive user interaction to deal with issues that may arise during the conducting of the experiment. This means that a tool to monitor this kind of event must be flexible to allow the experiment conditions to be modified while keeping this task simple. GridAnt achieves this by enabling the reuse of graphical components. A discussion then ensued of the issues raised from using the system with a real-world application.

It is unclear that the goal of GridAnt to be simple has been achieved. Although inclusion of some screen shots of the workflow do seem to be fairly easy to follow; this is not proof enough that a system is simple.

A key aspect of the simplicity of GridAnt is to enable a scientist to model the workflow themselves. In practise this would mean a new workflow for every experiment or at least adapting the existing workflow for each new experiment. A scientist might not want to spend valuable time changing the workflow for each experiment that has to be done. As workflow modelling can be a complicated task as all the variables have to be taken into account when modelling, a scientist might prefer to run the experiment without the use of workflow modelling.

One problem that was noted from conducting this experiment was that the client would have to be connected to the GridAnt system until the completion of the workflow.

It was suggested that this problem could be solved by providing a proxy service that the client could connect to instead. This involves modifying the GridAnt system to have a service-oriented architecture, enabling the GridAnt system to act as the proxy service [Amin et al., 2004].

This means that the client can start the workflow by using the services provided by the proxy service, alleviating the need for the client to stay connected to the system while the workflow is completed. This seems like a solution similar to that provided by web services workflow tools, where a message is sent to start the workflow and then the workflow sends a message back to the client when it is finished. This is a much better model for running workflow-based systems especially if there is a lot of processing involved. A large experiment could take days to process a large volume of data and it would be unacceptable for a client to stay connected to the workflow for that length of time.

It was interesting to note that the developers of the Triana system believed that Triana could be used by scientists, but the developers of GridAnt thought that Triana provided back-end functionality for workflow designed by experts and would be unsuitable for use by scientists. In the end neither parties proved that their implementation was easy to use or that it would be suitable for a scientist to use. Although, Triana did prove some important goals of the system through initial testing.

While Triana shields the developer from having to know anything about the middleware technology used, it is primarily for grid-based applications. It seems that the current research is leading towards the integration of web and grid services [Chao et al., 2004], [Krishnan et al., 2005], [Hey et al., 2005].

Currently there is work being done on compiling specifications that will be appropriate for use in both web and grid services [Atkinson et al., 2005], [Hey et al., 2005]. The reasons for choosing a particular specification include whether it has been standardised, its widespread use and adoption and the appropriateness of the specification for use in both environments. These specifications, collectively called WS-I+, include WSDL, SOAP, UDDI, WS-Addressing and BPEL4WS.

The advantage of integrating web and grid service technologies not only enables existing work to be merged, i.e. from the business and scientific aspects, but it incorporates the advantages of both technologies. Triana and GridAnt would be inappropriate for use in web services because they focus on solving a particular problem within grid services. GridAnt would be inappropriate as a workflow tool since the workflow it produces is not a web service, it would be difficult to use it with WSRF, which is a web service. Triana, on the other hand, provides support for BPEL4WS, which is a web services specification and the workflow it produces can act as a web service, so in this respect it could work with WSRF. However, this tool is simplistic in nature, so would not be appropriate for modelling complex workflow. Hence if a method could be found to enable it to work with WSRF, it would not generalise for all types of workflow. Fortunately there are a number of specifications and tools for modelling workflow in existence for use in web services that enable complex workflow to be modelled.

One of these is BPEL4WS, which supports the service-oriented architecture lacking in GridAnt [Amin et al., 2004]. BPEL4WS supports the modelling of complex business processes and is the most widely supported workflow specification [Atkinson et al., 2005], [Pasley, 2005], [Amin et al., 2004], [Yang et al., 2004]. As a result there are a number of commercial tools available based on BPEL4WS, some of which could be appropriate for a scientist to use and would support the requirement for ISO accreditation for a laboratory.

3.5 BPEL4WS

There have been a number of new specifications proposed in web services that describe the composition and orchestration of web services. One of these specifications BPEL4WS provides the means to model business processes and interactions [Andrews et al., 2003]. It allows workflows to be modelled that invoke web services, which are executed using a workflow engine. It is based on a number of XML specifications, namely WSDL 1.1, XML Schema 1.0, XPath 1.0 and WS-Addressing; the most important one being WSDL 1.1 [Andrews et al., 2003]. Since BPEL4WS is based on web services, a business process can either use services defined in an existing WSDL document or create new services as required. In addition a business process describes the behaviour of and the partners involved in the business process. BPEL4WS refers to services using the port type defined in the WSDL document rather than the actual service itself so that the business process can be reused in "multiple deployments of compatible services" [Andrews et al., 2003, p. 15].

The idea of integrating web and grid services has led to the investigation of using BPEL4WS in a grid services environment [Yang et al., 2004], [Chao et al., 2004]. This offers the advantage of capitalising on the benefits of web services, such as its interoperability and service-oriented architecture [Chao et al., 2004]. However, these benefits are reduced when the specification is adapted to better fit the grid environment [Yang et al., 2004].

In a grid services environment BPEL4WS is used to describe the workflow between grid services. This is not a complete integration as there are a number of issues associated with using BPEL4WS in a grid services environment [Yang et al., 2004], [Chao et al., 2004]. These are that grid services defines the concepts of GSH (Grid Service Handles) and GSR (Grid Service Reference) that basically allow a client to access a grid service, and which are not supported by BPEL4WS. Also grid services makes use of concepts defined in WSDL 1.2, which are also not supported by BPEL4WS as it is based on WSDL 1.1. To get around these problems, a wrapper is used to make a grid service client look like a web service, which is referred to as a proxy web service [Chao et al., 2004]. To execute the process, the client is invoked, which calls operations on the proxy web service, according to the workflow defined by BPEL4WS and then the grid services are invoked by the proxy web service. A similar solution was suggested in the GridAnt system [Amin et al., 2004], where the GridAnt system itself could act as a proxy service for the client to connect to.

The advantage of this architecture is the minimal impact on the workflow when the service is deployed to a different location. In order to test this architecture, a system was implemented based on the model-view-control pattern, which used three grid services, one for each of the view, model and control. The partners in the process are implemented as web services, one for each grid service. Although the paper states that experiments were done on this implementation, no specific details are mentioned. Instead there is a discussion of the issues raised from this implementation and the similarities and differences between BPEL4WS and grid services. This on its own is not enough to validate the conclusion that the architecture enables grid services to use BPEL4WS. Since grid services is based on WSDL 1.2 and BPEL4WS is based on WSDL 1.1, it seems that using grid services in BPEL4WS will be limited to the common functionality provided by WSDL 1.1 and 1.2, until such time as BPEL4WS supports WSDL 1.2. Therefore, BPEL4WS may not be the best choice for a workflow language in grid services, as it may seriously impede the functionality of the grid service used in the BPEL4WS process.

The key difference in the two approaches of using BPEL4WS in a grid services environment is modifying the specification [Yang et al., 2004] or modifying the implementation [Chao et al., 2004]. The first approach reduces the benefits of web services and will mean constant adaptation of the "new" specification to stay in-line with updated versions of BPEL4WS. This is not an appropriate method for this research, as it will mean the solution could not be generalised, especially since BPEL4WS has yet to be standardised. In general it is best to avoid methods that will involve altering the specification, rather it is better to build on the specification making necessary adaptations at the level above the specification. The other issue with adapting the specification is that it may make the solution incompatible with other specifications, reducing the value of your solution in general.

The use of BPEL4WS in a grid services environment has been extended to incorporate the use of WSRF [Slomiski, 2005], [Leymann, 2005]. WSRF [Czajkowski et al., 2004] (WS-Resource Framework) is a specification derived from OGSI (Open Grid Services Infrastructure) that has been re-engineered and returned to the web services development community as a proposed standard.

3.6 WSRF

WSRF introduces the concept of a stateful resource, see section 2, that allows state to be preserved between executions of a web service's operations.

The introduction of WSRF means that a web service can have state associated with it. Up until now a web service had no notion of state, but it was implied through the definitions of its interfaces [Foster et al., 2004]. This is because "WSDL is essentially stateless because the language is unaware of states between operations" [Staab et al., 2003, p. 74]. It is claimed that explicit definition of state leads to improved interoperability, simplification of service definitions and improved discovery, management and development tools [Foster et al., 2004]. It also alleviates the developer from coming up with their own implementation of state, which may be incompatible with other developers implementations. By standardising the way in which state is accessed it means web services that use state can access it without having to know how it was implemented. This is an important point especially to get two different specifications to work together.

Basically WSRF describes the association of a web service with a stateful resource, which is called a WS-Resource. A web service can locate a stateful resource through its unique identifier, which is encapsulated within an endpoint reference, defined as part of the WS-Addressing [W3C, 2004b] specification. This endpoint reference contains the address of the service, whose operations are to be invoked. If a stateful resource is involved in the implementation of the service, the resource identifier is included within the reference properties defined as part of the endpoint reference. This information is then encoded as part of the SOAP header within the message request that is sent to the service. The service can then extract this information to gain access to the required stateful resource. This provides a more specific way of referencing a resource from the web service, in a similar way to obtaining references to an object in object-oriented programming.

The use of WSRF in either grid or web services is not widespread [Wasson et al., 2005]. There has been some work done on using WSRF with .NET web services [Wasson et al., 2005], [Humphrey et al., 2004] in order to evaluate WSRF. This was done by implementing a simple scenario that required the modelling of state, hence the use of WSRF. This enabled WSRF to be evaluated by looking at the issues that were raised during the implementation process. Any aspect of the implementation that raised questions about WSRF was discussed, which showed some of the difficulties in using WSRF. In particular these were developing the client and server-side code [Humphrey et al., 2004]. The most important step in developing a WSRF web service is determining what is to be modelled as state. This is not something that a specification can easily specify as it is more of a design decision, in that how the service is defined reflects what should be implemented as state. Therefore, it does not seem relevant to include this as one of the issues of WSRF.

Some of the other issues that were raised included that the client and service are more tightly-coupled than a standard client and web service. This problem is because the client has to keep track of endpoint references for the WSRF services it wants to invoke and the loosely-coupled nature is one of the benefits of using web services. If there was a standard mechanism to store and retrieve state, such as delegating it to another web service, this would not be as much of a problem. This will be one key area to look at when developing the method for combining WSRF and BPEL4WS.

Another key issue that was raised is the complexity of the server-side code. A port type can be imported into a WSDL file that has associated state defined in the separate WSDL file where the port type was defined. This means a port type can be imported without knowing about any state associated with it. In particular this is the case when a WSDL file is to declare a state variable as this involves using the *get resource property* port type. Although this is a problem, it can be resolved by finding the pre-existing WSDL files and finding out what state is associated with them. For example, when declaring a web service to be a WSRF service, the *get resource property* port type has to be used in the WSRF WSDL file. This requires importing the WSDL file, where the *get resource property* port type is defined, in the WSRF WSDL file.
i.e. by placing the piece of code, below, in the WSRF WSDL file.

```
<wsdl:import
  location="http://docs.oasis-open.org/wsrf/2004/06/
              wsrf-WS-ResourceProperties-1.2-draft-01.wsdl"
  namespace="http://docs.oasis-open.org/wsrf/2004/06/
               wsrf-WS-ResourceProperties-1.2-draft-01.wsdl"/>
```

The state defined in the imported WSDL file can be determined by locating the WSDL for the service. From the example above, this would be done by using the URL of the

service as specified by the location.
Although it is important to understand exactly what operations the service will define, this could be construed as wasting valuable development time. In saying that, it is not good practice to include code, files or anything else into a system that is being developed, which is not clearly understood.

The other issue associated with the complexity of the server-side code is the lack of an intuitive interface. This occurs because WSRF does not include package names in the operations that are referencing a stateful resource. This is because the package name is part of the stateful resource and is located within the SOAP header using an endpoint reference [Humphrey et al., 2004]. This issue is the same as the previous one in that it can be resolved by completely understanding the WSDL file that defines the stateful resources and operations for the service.

Overall, it would seem that most of these issues relate to your understanding of what is to be modelled as state. These should not really be issues as you shouldn't have defined operations or imported other operations into your WSDL without understanding exactly what the functionality is that they provide. It would seem that the WSDL would not be a well-defined web service if it is not clearly understood what services the operations provide and it would be difficult to determine what state the web service would be using. These issues occur at the design level and are similar to the issue of what to model as state, something which the specification should not have to define.

The results indicated that improvement is needed to address the usability [Humphrey et al., 2004] and tight coupling [Wasson et al., 2005], [Humphrey et al., 2004] problems with WSRF. The usability problems with WSRF could be reduced by the developer having a greater understanding of WSRF before choosing to use it to implement a web service.
The tight coupling issue could be solved by providing a standard mechanism for storing and retrieving endpoint references. This stems from the client needing a valid endpoint reference to the resource. Specific issues that need to be addressed in regard to this are how much state the client needs to keep track of, the life-time of client-side information

and gaining endpoint references to resources, should the existing ones be lost or become invalid [Wasson et al., 2005].

The key issue for my research will be gaining endpoint references to services, which I will solve by using the solution outlined in the next paragraph. The other issue in regard to the life-time of client-side information also seems irrelevant and does not seem to me to be a relevant issue for WSRF. WSRF shouldn't need to care about the life-time of client-side information, namely endpoint references, only that there should be a defined mechanism to retrieve an existing endpoint reference to a WS-Resource should an existing one be lost or invalid.

This problem of gaining endpoint references to services can be achieved to some extent by using a factory pattern, which is responsible for the creation and destruction of resources. By using a factory to create the WS-Resource the developer has more control over the creation and destruction of resources and reduces the chances of the client obtaining an invalid reference to a resource.

The issue of maintaining valid endpoint references is a big problem, because if the client goes down for any reason and loses the endpoint references, there is no way to get them back. However, there is a specification that defines how to obtain a new endpoint reference to the WS-Resource in the event that the existing one becomes invalid. This specification called WS-RenewableReferences will go a long way to try and resolve some of these issues [Czajkowski et al., 2004].
This will be an issue that the method for integrating WSRF and BPEL4WS will have to try and resolve.

The second paper [Wasson et al., 2005] that implements a more complex system using WSRF does not really contribute anything further to either resolving some of these issues or providing any more detail on any of these issues. The conclusions that their tooling and programming abstractions have made some of the concerns more manageable does not seem to apply to WSRF but rather to their implementation in .NET.

The increased interest in adopting a language such as BPEL4WS and either adapting it for use in grid service environments or adapting grid services to make use of BPEL4WS is to take advantage of the concepts arising in both fields. The adoption of BPEL4WS in grid services has its advantages, such as "it can easily and seamlessly interact with standard Web services (that may not be part of Grids) and grid services such as provided by OGSI and WSRF" [Slomiski, 2005, p. 3].

3.7 BPEL4WS and WSRF

The reasons behind using BPEL4WS over other languages are that its workflow can act as a web service, it can be composed of other web services and there is good tool support for BPEL4WS. However, the key point is that BPEL4WS supports WS-Addressing endpoint references, which is the same specification used by WSRF to locate its stateful resources [Slomiski, 2005], [Leymann, 2005]. The other key point is the widespread adoption of BPEL4WS over other languages [Yang et al., 2004], [Leymann, 2005].

There are two main options when integrating BPEL4WS and WSRF in a grid environment: modelling a WSRF web service as a partner or modelling the workflow as a WS-Resource.

The use of BPEL4WS did not involve adapting the language [Leymann, 2005], as it was needed to be compatible with future releases of BPEL4WS, unlike other proposed solutions [Yang et al., 2004]. This is important so the integration of the two specifications does not need to be refactored to be compatible with updated versions of BPEL4WS; it also preserves the benefits of a web services based specification. This will be the approach taken in my research, as it is important for the generalisation and interoperability of the method that it be compatible with future versions of BPEL4WS.

It is suggested that a web service that implements WSRF can be implemented as a partner in BPEL4WS, as a partner is defined to be a service that receives or sends messages as defined in WSDL [Slomiski, 2005]. A WSRF web service can be implemented as a partner because BPEL4WS does not specify deployment information. WS-Resource instances get created through a factory, which is basically any web service that can create an instance of a WS-Resource. This factory then sends a message that contains a reference to the newly created WS-Resource. In BPEL4WS references are assigned to services, so that these can be invoked when required. These references get assigned automatically either when the service is invoked or when the process is executed by a workflow engine [Slomiski, 2005]. This basically says that instances are created implicitly, whereas a WS-Resource is created explicitly through the use of a factory [Leymann, 2005]. This means if BPEL4WS creates a WS-Resource it will automatically assign it a reference as specified by WS-Addressing. However, a BPEL4WS process would need to assign the reference returned from the invocation to the factory service and add the reference properties to the SOAP header for this to work properly. Since to BPEL4WS, any web service is a partner and it doesn't specify deployment information, the process has no way of telling if the partner it is communicating with is a standard web service or a WSRF web service. Since standard web services do not require reference properties in the endpoint reference, it seems unlikely that BPEL4WS would assign reference properties to the SOAP header. This means that integrating BPEL4WS and WSRF will be more complicated than the suggestions made here. Hence, merely assigning an endpoint reference returned from an invocation to the factory service to the WS-Resource will not be a solution to my research problem, see section 5.

Another approach [Leymann, 2005] suggests treating the workflow as a WS-Resource. This means that when the first message is sent, the workflow engine must create an instance of the workflow and return a response that contains the reference to the newly created workflow. Using this model of integration, BPEL4WS will need to support stateful resources, since the workflow itself is now acting as a WS-Resource. This can be supported by defining a variable that has the same type as the stateful resource. This solution does not seem viable for integrating WSRF and BPEL4WS, since the workflow itself invokes other web services. If the workflow was invoking WS-Resource services the workflow would still have to assign the endpoint reference to the WS-Resource partner. Therefore, this will not be an approach that will be investigated as a possible solution to the integration problem.

It is also suggested that correlation sets, see section 2, could be used to identify a message sent to a particular process instance [Leymann, 2005]. However the author notes that this would not work in practise because correlation sets are not always required by the BPEL4WS process. In the case where there are no correlation sets, reference properties would have to be defined. The argument is that correlation sets could be used instead of defining reference properties. The problem with this argument is that nowhere in the specification does BPEL4WS specify where correlation set data gets passed in the message, although it did seem to hint that it might appear within the body of the message. Either way it cannot be guaranteed that this information will get added to the SOAP header as required by a WS-Resource. The other problem with this argument is that reference properties are not a required part of an endpoint reference so the entire reason for correlation sets versus reference properties is invalid.

It is unclear at this stage if the process is supposed to be acting as a WS-Resource, or whether this method would be used to identify a particular instance of a process and hence a particular instance of a WS-Resource. Although, this argument is for the purpose of illustrating that BPEL4WS process instances are created and accessed in a similar way to a WS-Resource, there is still the fundamental problem of the process invoking WS-Resource operations and assigning the reference properties to the message SOAP header. This fundamental issue with WSRF is not mentioned at all and since this issue is not mentioned it makes any subsequent argument for the integration of BPEL4WS and WSRF rather weak.

The one problem arising from integrating WSRF and BPEL4WS is that a proxy service will be required to add a SOAP header to any messages sent from a WSRF web service, as BPEL4WS does not need to use WS-Addressing when interacting with the workflow engine [Slomiski, 2005]. This would seem the most likely approach to solve the integration problem. This was just an investigation looking at how and to what extent BPEL4WS could work with WSRF. No specific implementation has been developed or tested, so it can only be assumed based on the evidence provided that this solution would work. However, from arguments already mentioned this would be the most likely solution.

The integration of a modified BPEL4WS specification [Yang et al., 2004] with OGSI has the advantage of enabling workflow in a grid environment to be better modelled and while it is based on an OGSI grid it could easily be modified to work with a WSRF grid. However, this is not an appropriate approach for integration in a web services environment as changing the underlying specification could remove compatibility with other standards.

Both approaches to integrating BPEL4WS with WSRF show that the specifications overlap to some extent; however the key issue is the requirement of WSRF endpoint references to contain the resource identifier, which is defined within the reference properties element of the endpoint reference. This is a problem because it is not required for BPEL4WS to include reference properties as part of its endpoint references. It would seem that the approach to model the workflow as a WS-resource [Leymann, 2005] is not really a viable option as it removes the purpose of using BPEL4WS to begin with.

3.8 Conclusion

The current research seems to be leading towards the integration of web and grid services to receive the benefits of both technologies, in particular the service-oriented architecture of web services which is thought to be an integral foundation of e-science [Foster et al., 2002]. In order for grid services to provide a complete solution to the e-science problem there needs to be a way to model workflow. Although there have been a number of specifications and tools proposed in both web and grid services, it would seem that BPEL4WS is the most promising candidate for modelling complex workflow. The use of a web services specification that supports the service-oriented architecture will be beneficial to grid services in the event of the integration of web and grid services. This has led to numerous proposals suggesting enhancements to BPEL4WS for compatibility in the grid environment. This integration idea has also led to the possibility of combining grid services specifications, such as OGSI and WSRF, with web services specifications, such as BPEL4WS for use in a grid environment. Although the integration is feasible, BPEL4WS restricts some functionality of grid services, but making changes to the specification will remove the advantages of using it in the first place.

Nearly all the papers looked at, suggested some form of intermediate mechanism for BPEL4WS to communicate with WSRF. This seems to be necessitated by the requirement that reference properties are added to the SOAP header of the message sent to the WS-Resource. Therefore, the implementation of a proxy service that performs this task will be the most likely candidate to integrate WSRF and BPEL4WS and will be the approach to solve the integration problem that is the basis of this research. The second problem will be addressing how the client maintains endpoint references and how it can get new endpoint references should the existing ones become lost or invalid. None of the work mentioned seems to have any concrete ideas on how to address this problem, although hopefully the future implementation of WS-RenewableReferences will achieve this goal.

There does not seem to be any concrete criteria to evaluate an implementation. However, from looking at the issues raised from discussion of other tools/technologies, a criteria can be derived from identifying key aspects of WSRF/BPEL4WS that have been mentioned as needing improvement. Already a number of criteria can be identified such as:

The method will need to assign reference properties to the SOAP header. Necessitated by the lack of proof that BPEL4WS will perform this task.

The method will need to provide a means of storing and retrieving endpoint references.

The method will need to provide a method to enable retrievement of endpoint references in the event one is lost or becomes invalid.

4 ChemSearch Laboratory

ChemSearch is a small analytical laboratory that analyses samples sent to them by their clients. A client requests certain tests to be done on their samples by filling out an order form that is sent to the laboratory along with the samples to be tested. On receiving the order, the laboratory checks the order form against the samples received, to ensure that the wrong order form has not been put in with the samples, and also checks over the tests ordered. Frequently, clients request the same tests to be done for their samples. If there is a test requested that is not normally ordered, the laboratory will ring the client to confirm the tests to be done. Once the tests have been approved, the laboratory prioritises the tests based on the number of samples requiring the same test, the urgency of the test or sample and the lifetime of the samples. When a test has been completed the initial results are recorded, which can be used to determine the final result for a test. On completion of all tests ordered by the client, a copy of the report detailing all the results of the tests is sent to the client.

Typically, the process involves the client ringing or emailing to find out the results of a particular test or the status of their report. Ideally, the client should be able to access their results and the status of their reports without contacting the laboratory.

5 Research Problem

This section outlines the main problem this research plans to solve and additional sub-problems.

5.1 Main Problem

The focus of this research was to find a method that would enable WSRF to be integrated with BPEL4WS. This method would then be evaluated to determine whether the specifications as they stand can be integrated or whether further refinement of the specifications may be needed to fully support this integration.

5.2 Sub Problem 1

The first sub problem is the modelling of the workflow of the ChemSearch laboratory and the development of an on-line system for their clients so that it is suitably validated for its purpose as needed by the ISO accreditation of the laboratory. This will provide a means to test and evaluate the proposed method of integration.

5.3 Sub Problem 2

The second sub problem is providing a mechanism for storing and retrieving endpoint references that are required to locate a particular instance of a WS-Resource web service.

6 Scope

This research is only focused on finding one method of combining WSRF with BPEL4WS. This method will then be evaluated by modelling the workflow of the ChemSearch laboratory, which will enable any issues with the method to be brought to light. As a result, BPEL4WS will only be used to model the processes of the ChemSearch laboratory that involve the processing of samples and production of reports.

Since the focus of this research is on the method to integrate WS-Resource with BPEL4WS, the prototype system will be focused on an implementation of this method. As a result, only the business processes that can be implemented as web services or stateful web services will be considered. This project will involve building a prototype system, which will not necessarily be ready for production use in the ChemSearch laboratory. This is because firstly, the system will use new technologies and tools that may not be ready for use in a production environment. Secondly, since the focus is on the method used and not implementing the system, the system will not be thoroughly tested, and thirdly the system may not be considered as "suitably validated" as required by the standard ISO/IEC 5.4.7.2 for the competence of testing and calibration laboratories.

7 Importance of the Research

The use of web services technology is increasing because web services is based on open Internet-based standards, which provide interoperability, while not being limited to a specific implementation. BPEL4WS provides a way to model workflow as a composition of web services and as such gains the benefits associated with web services. However, BPEL4WS does not provide any concept of state, i.e. a business process cannot be modelled using some value from a previous invocation of the process. BPEL4WS does have a number of advantages including being able to model the workflow using a graphical notation, also a developer can focus on modelling the processes without worrying about execution details as this is all done for you.

The introduction of WSRF means that a web service can have a stateful resource; this provides a standard way to access and model state. This project will provide a method of combining BPEL4WS with WSRF to enable processes that require state to be modelled. As a result, this method will inherit the advantages offered by both web services and BPEL4WS. The need to combine these two technologies is illustrated by the problem that the processes of the ChemSearch laboratory cannot be adequately represented in WSDL and although BPEL4WS allows business processes to be modelled, some of the ChemSearch processes cannot be adequately modelled using this either. However, a combination of the two technologies would allow for stateful processes to be modelled. As a result, any stateful process could be modelled using this method. The outcome of this project will be an evaluation of the method, which will bring to light possible limitations of the technologies or tools. Any limitation revealed about the tools or technologies used will benefit the field by increasing the adoption of web services technology as the specifications and standards it is built on will be of a higher quality.

On a more specific level this method will enable the workflow of the ChemSearch laboratory to be modelled taking into account their ISO certification. As a result, a prototype system will be implemented that will minimise the time ChemSearch has to spend updating clients on the status of their reports, as their clients will be able to access this information directly themselves through the on-line system. It will also provide a standards-based solution that will enable the laboratory to work in cooperation with another laboratory if need be.

8 Research Methods

Firstly, sample data will be obtained from the ChemSearch laboratory to enable the business processes of the laboratory to be modelled using UML (Unified Modelling Language). This will involve setting up meetings with the laboratory, to make sure the processes are being modelled correctly. Then a method will be used to integrate WS-Resource with BPEL4WS in a web services environment, contrary to the current literature in this area which seems to focus on the integration of the two specifications in a grid services environment. Since this project involves using recent specifications and technology, the method used cannot be clearly defined. As a result it is not known how easy or difficult it is going to be to model the requirements of the laboratory using these specifications. So this project may involve looking at another method for integrating the two specifications, based on how easy or difficult it is to implement the first method, perhaps using a tool such as GridAnt [GridAnt, 2003].

It will be necessary to explain the reasons for using a second method, especially if it is because BPEL4WS could not be integrated with WSRF. The first method will probably use a BPEL4WS engine to compile and run the BPEL4WS; however WSRF requires the use of a tool, such as Apache WSRF to compile and deploy stateful resources. This means it will probably be necessary to find a BPEL4WS engine that can either compile and deploy the stateful resources itself or be used in conjunction with Apache WSRF. Once a method has been found, the processes can be modelled using BPEL4WS, with the stateful components modelled using WSRF. The web services can then be deployed and a system can be implemented that uses these services to provide ChemSearch clients access to their reports. This will involve an investigation into security issues so that proper authorisation and authentication occurs. If time allows some aspects of security could be implemented in the system. There is a specification that has been proposed to address the security issues that exist in web services, called WS-Security. This could be used to implement some security in the system.

This system will then be validated to check that it fulfils the needs of the laboratory and then the method used to combine BPEL4WS and WS-Resource will be validated to determine how good the solution is. This is because in theory this solution could be one of an infinite number of solutions. This can then lead on to investigating the use of single-sign on, if the ChemSearch laboratory wished (in the future) to share information with another laboratory, then the clients of ChemSearch would not want to have to log into two separate systems to gain access to their results. Single-sign on provides a solution to this problem by allowing the user to sign in once for both systems.

9 Methodology

This project involves the development of a solution to allow the modelling of the workflow of the ChemSearch laboratory, which includes stateful web services. This will take into account the ISO certification of the laboratory. This solution will then be evaluated to assess its advantages and disadvantages and its success in fulfilling the requirements of the ChemSearch laboratory. This will also involve evaluating the technology and tools used to implement this solution. This will not be hypothesis-driven research as this research will involve analysis of the problem and the development of a solution and then the evaluation of that solution as to how well it solves the problem and whether it meets the necessary requirements of that problem. Hypothesis-driven research involves proposing a hypothesis, then conducting "research" to test the hypothesis to find out whether it was valid or not. The point of this research is to find a method of integrating WS-Resource with BPEL4WS, so in this case a hypothesis could be that a method can be found to integrate WS-Resource with BPEL4WS. However, if it turned out that a method could not be found, you could not say that the hypothesis was incorrect, because although the methods you found did not provide a solution to the problem these only account for a small subset of the possible solutions. Hence, a method of integrating WS-Resource with BPEL4WS could exist and so you could never truly say whether the hypothesis was correct or not. Therefore, it would be better not to use hypothesis-driven research in this case.

10 Tools

The tools used to implement this method and the subsequent on-line system were Cape Clear Orchestration Studio, [CapeClear, 2005], to model the BPEL4WS process and the WSRF project from Apache, [Apache, 2005], using the Tomcat server, [Tomcat, 2005], and Ant, [Ant, 2005], to compile and deploy the web services to Tomcat. The BPEL4WS tool was a plug-in for the Eclipse Integrated Development Environment, [eclipse, 2005].

11 The Method of Integration

The key driving point behind the integration problem was that WSRF uses endpoint references to uniquely identify a stateful resource. These endpoint references are defined by the WS-Addressing [W3C, 2004b] specification that optionally defines reference properties for an endpoint reference. It is these reference properties that contain the unique identifier to the newly created resource. In order for the server to locate the requested resource, it needs to be able to access the unique identifier for the service. As a result the reference properties are sent in the SOAP header of the message. Therefore, in order for BPEL4WS and WSRF to work together, BPEL4WS needs to copy the reference properties into the SOAP header of the message. It could not be determined from the BPEL4WS specification whether BPEL4WS would copy the reference properties into the SOAP header of the message. As a result, an intermediate mechanism is needed to provide this service. This was verified by looking at related literature, see section 3 for further details.

Even though the related literature specified the use of an intermediate mechanism, there was no concrete evidence that BPEL4WS could not work directly with WSRF. Hence it was determined first whether BPEL4WS could interact directly with WSRF, details of this are given in section 14.1. It was determined that at this time BPEL4WS cannot be combined directly with WSRF.

As a result the solution described here uses an intermediate WSRF web service to copy the reference properties into the SOAP header of the message. This is similar to a solution proposed in the literature [Slomiski, 2005], where it was suggested that a proxy web service could be used to copy the reference properties into the SOAP header. This results in three web services. The first is a factory web service that creates the stateful resource and returns the endpoint reference containing a unique identifier to the newly created resource. This unique identifier is the value passed in the SOAP header of the message request to the WS-Resource. The second is the proxy web service, which keeps a reference to the endpoint reference returned by the factory service. This is then used for all subsequent calls to the WS-Resource. The third web service is the WS-Resource, which contains the stateful resources for the system.

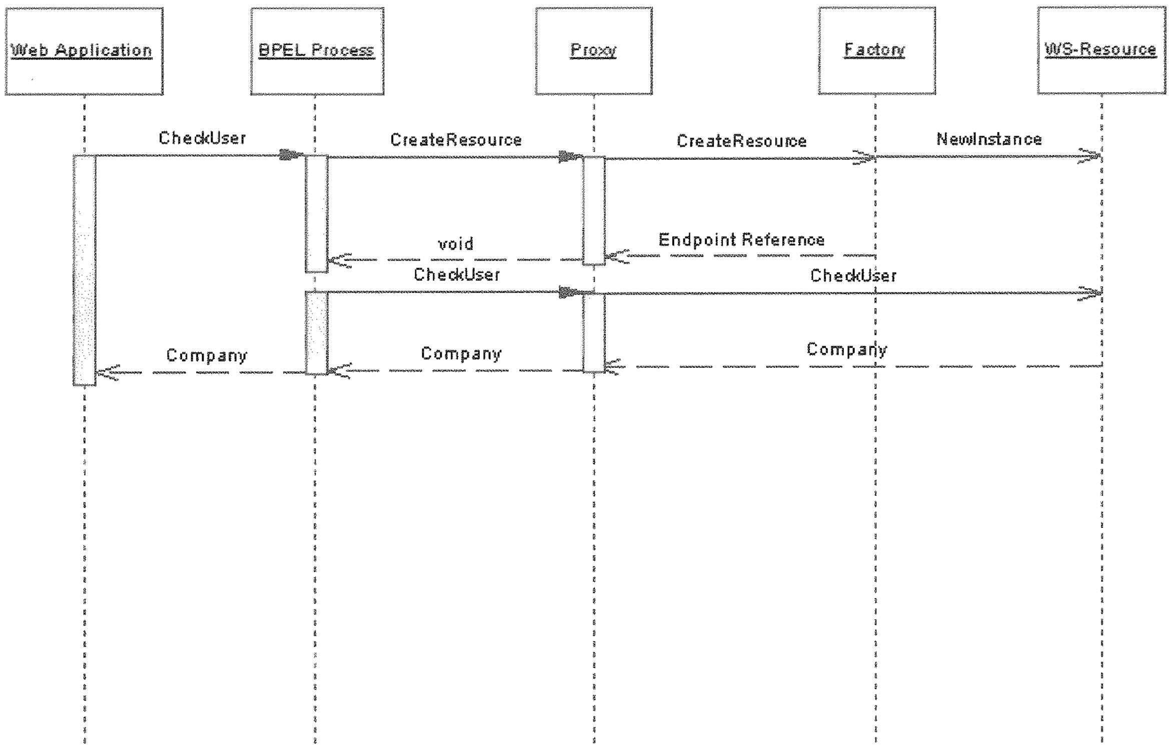


Figure 1: Sequence diagram showing the interaction of the three web services and the BPEL4WS process

A sequence diagram showing the interaction of these three web services with the process and an external web application is shown in figure 1.

This method also needed to fulfill the requirement of storing and retrieving endpoint references that could, if possible, provide a means for requiring an endpoint reference in the event that the one currently held is lost or becomes invalid.

Due to the nature of the method, the proxy service will need to store and retrieve the endpoint references as it is the only entity that directly interacts with the factory service, which is shown in figure 1. This could be achieved by using a hash map to map a unique identifier [2], which in this case was the site, to the endpoint reference. However, to provide a solution that fulfilled the optional requirement for obtaining lost or invalid endpoint references, a different method was implemented. Instead of the proxy service matching incoming requests with endpoint references, this was handled by the factory service.

A sequence diagram depicting the interaction between the factory service and WS-Resource is shown in figure 2.

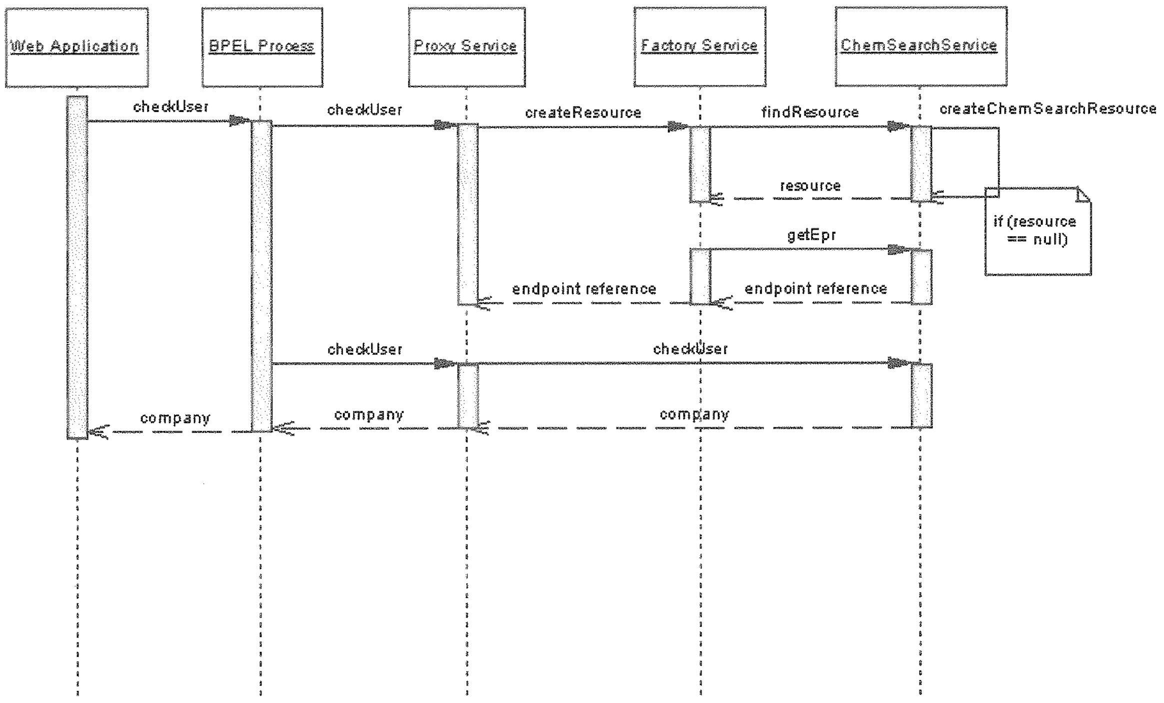


Figure 2: Sequence diagram showing the interaction, creation and retrieval of a WS-Resource

[2]This is a unique identifier to find a given endpoint reference. Not to be confused with the unique identifier encapsulated within the endpoint reference that uniquely identifies the WS-Resource

Basically, a method called *find resource* was added to the WS-Resource home class, which in this case was called *ChemSearchHome*, see appendix D. This method takes in the resource context, which enables the class to find a resource using the value of the reference properties passed in the request message header, and a string. This string uniquely identifies an endpoint reference, since in the case of the laboratory a client can have many reports, therefore there needs to be a unique identifier to distinguish between the different reports the client might have. This method obtains a reference to the map maintained by the cache that holds all the WS-Resource instances. Each resource class, in this case it is *ChemSearchResource*, contains a reference to the endpoint reference for the resource and to the site (the string). As a result the method can loop through the map comparing the string parameter to the resource's site value. If it finds a resource in the map that contains that site, it returns that resource. If not, a new resource exception is thrown and subsequently caught by the same method. At this point it calls the *createChemSearchResource*, which also takes in as parameters the string and resource context. This creates a new instance of the *ChemSearchResource* and an endpoint reference for that resource. This newly created resource is then returned to the factory. The factory then uses the resource to find the endpoint reference, see appendix C, which it returns to the proxy service to store for subsequent calls to that resource.

This method of storing and retrieving endpoint references is discussed in sections 14.2.2 and 14.2.3 respectively.

12 Development

In order to test this method of integration, it was applied to modelling the workflow of the ChemSearch laboratory, see section 4.

From communications with the laboratory it was determined that the main steps in the workflow were:

- Get new orders
- Check orders

- Prioritise tests
- Perform tests
- Record the results
- Produce a report
- Send the report to the client

Currently the ChemSearch laboratory records all the data for their tests in spreadsheets. While this may be an appropriate form for them, it was not appropriate for a web service to access and update. Hence an alternative storage mechanism was needed. Due to the ISO accreditation of the laboratory the alternative needed to be suitably validated for its use, but the cost needed to be low in order for it to be a worthwhile option for the laboratory to use. For these reasons the chosen database was MySQL [MySQL, 2005].

In order to model the workflow of the laboratory there needed to be a system to properly test this. Although the original proposal only covered building an on-line system for the client, ChemSearch, this only modelled a small part of the workflow of the laboratory, which would have been very much one-sided. The client would be able to access the state of their reports, but there would be no mechanism to update this state. This required building a system to model the workflow of the laboratory. By using WSRF web services to enable state to be modelled; this introduced a major issue in the design and construction of the workflow. Ideally the on-line system would access the workflow of the laboratory to gain access to their test results and reports. In practise this was not the case. It became apparent that there would need to be two separate and distinct workflows, one for the laboratory and one for the client. The reason for this was the nature of WSRF, which enables a factory service to only create one instance of a WS-Resource or to create multiple instances of the WS-Resource. To secure the data for the laboratory's clients it was necessary to have a new instance of the WS-Resource for each client, while the laboratory would only require one instance of the WS-Resource as each staff member needs to update the state of the same WS-Resource.

It became apparent when modelling the workflow of the laboratory that although it appeared to be simple on the surface, this was not the case. In order to take advantage of the usefulness of WSRF, it would be necessary to create a new WS-Resource for each order placed by a client. In practise this is difficult to implement. The creation of a new WS-Resource is relatively straight-forward, the workflow would call the create resource method provided by the proxy web service each time a new order is placed. The difficult part is determining the unique identifier to return when the client requests the state of their report since a client can have multiple reports and hence multiple WS-Resources. There is also the problem that because of the sensitive nature of the data, there should be proper user authentication. As this is modelled as part of the workflow, the creation of the WS-Resource needs to take place at this point, since all functionality is implemented in the WS-Resource web service, see figure 1.

In order for the laboratory to update the state of the clients' reports, they both needed to be part of the same WSRF web service. In order to fulfill this requirement, the laboratory and the clients shared part of the same workflow. This means the laboratory can use the site to find the endpoint reference for the client they wish to enter results for.

The heart of the problem was the amount of data obtained from one order. It was apparent from order forms obtained from the ChemSearch laboratory that each client could have multiple areas that each had multiple sites. A site could have multiple tests carried out on the samples sent to the laboratory. This equates to a lot of data, which is particularly difficult to model given that the system presented to the client to enter this data was from an on-line system. In a particular case one area for a particular client had 18 sites, and according to the order form provided by the laboratory there were a total of 31 possible tests that could be requested by the client. This equates to a lot of data in the worst case scenario that all 18 sites request all 31 tests to be done. Hence some simplification in this area was needed, since the number of sites could vary significantly between clients. The design decision was to enable a client to choose an area first, which would then open up a new form from which they could select a site and the subsequent tests to be performed, rather than entering all the data for all tests at once.

This design had one major drawback. It meant that a new order was created for each site rather than for each area. This was not due to WSRF, but rather that it would be difficult to determine when one order began and the next ended, since you could not make the assumption that all sites could have tests ordered for them.

As a result, since a report is based on the combined orders for an area, the laboratory could update the status for a particular site, which was stored in the database. When the report status was requested the web service retrieved the status of all the sites and worked out whether the report was ready and if not how many sites still had tests pending. This is not an ideal solution, but in order to model the workflow properly, there had to be some simplification.

Even if the workflow had been modelled such that there was a new WS-Resource for each order, there would still have been a need for persistent storage of the stateful resources along with the unique identifiers of each resource, in the event of network failure or failure of the server itself, as the state of each WS-Resource is only persistent for the lifetime of the server.

The use of two workflows was also based on the differing stateful resources required by the clients and the laboratory. While some of them overlapped, the inherent way in which the workflow was implemented affected the stateful resources that were required. However, since the focus is on testing the method and not the implementation of the workflow, this is not such a big issue.

The subsequent sections will focus on discussion of the particular implementations of the workflow for both the client and the laboratory.

12.1 Client-Side Workflow

The client-side workflow is given below:

- Order Tests
- Get previous orders

- Confirm the order
- Get results
- Get report

A graphical version of the workflow is given in figure 3 as modelled using BPEL4WS.

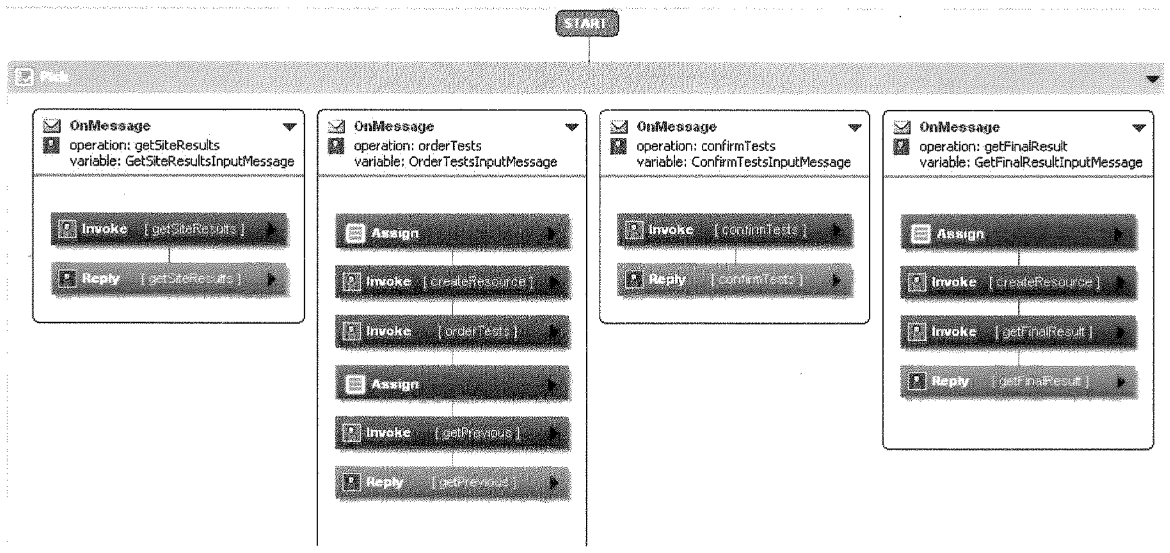

START
Pick
OnMessage
operation: getSiteResults
variable: GetSiteResultsInputMessage
Invoke [getSiteResults]
Reply [getSiteResults]
OnMessage
operation: orderTests
variable: OrderTestsInputMessage
Assign
Invoke [createResource]
Invoke [orderTests]
Assign
Invoke [getPrevious]
Reply [getPrevious]
OnMessage
operation: confirmTests
variable: ConfirmTestsInputMessage
Invoke [confirmTests]
Reply [confirmTests]
OnMessage
operation: getFinalResult
variable: GetFinalResultInputMessage
Assign
Invoke [createResource]
Invoke [getFinalResult]
Reply [getFinalResult]

Figure 3: A model of part of the client-side workflow

The client-side workflow consists of two separate parts: when a client places an order with ChemSearch and when the client retrieves the results for their tests and the status of their report. These are shown in the flowcharts below.

Figure 4 shows the process of ordering tests. The tests that are ordered are compared against the last order placed by the client, for a particular area. This is because a client usually orders tests for a specific area. If there are tests ordered by the client that were not ordered in the previous order for that area, the client is asked to confirm the order. Normally a client will order the same tests each time an order is placed for a particular area. Any tests that are included that are not normally ordered could be because of human error. As a result, ChemSearch always checks before committing to performing these tests. Once the client has confirmed the order, the tests to be done are stored in the database.

The workflow must keep track of the last order made by a client for a given area and the tests just ordered. Therefore, the workflow will use a WSRF web service to perform this task.

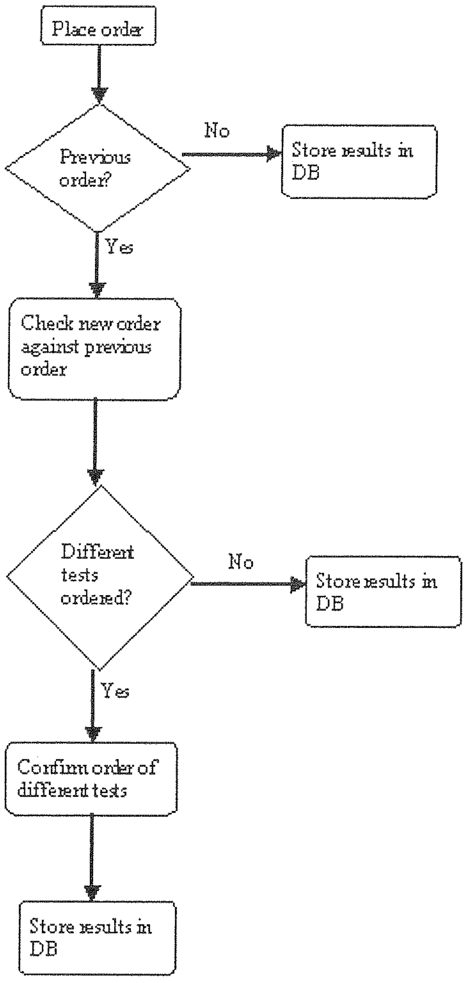


Figure 4: A flowchart of the process of ordering tests

Figure 5 shows the process of retrieving the results of the tests and the status of the report. If tests have been completed the client can view the results, the current status of their report and the number of tests still to be completed. If the status of the report is completed, i.e. all tests have been done for a particular area, the client can view their report. The report shows all the tests that were ordered and their results.

The workflow must keep track of the results for the tests and the status of the client's report.

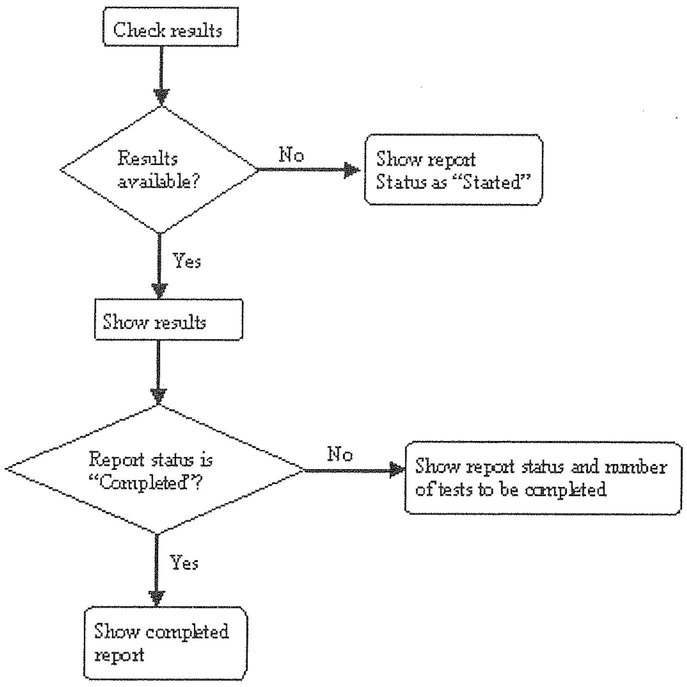

Check results
Results available?
No
Show report Status as "Started"
Yes
Show results
Report status is "Completed"?
No
Show report status and number of tests to be completed
Yes
Show completed report

Figure 5: A flowchart of the process of retrieving results

One of the issues that arose from this implementation was how to determine which report to return, since a client could potentially place many orders and hence have many reports for the same area. There are a number of possible solutions to this problem the first is to have the database purge old data from the database regularly, or have a variable set when the client requests a new order for the same area. The issue with this is when to purge the old data, since it is hard to determine when the client will no longer wish to see an old report. Due to the time constraints of this project, these types of solutions have been scoped out of the implementation.

Perhaps a possible future extension to this project would be to enable the workflow to automate the sending of an email to the client containing a PDF version of their report. Once this report has been sent the workflow could wait for a response from the client as acknowledgment that they received the report. The workflow could then update the database rendering that order completed or purging the data out as required.

12.2 Laboratory-side Workflow

The laboratory-side workflow is given below:

- Get new orders
- Check the order against the samples received
- Prioritise the tests
- Perform the tests
- Record the results
- Produce the report
- Send the report to the client

A graphical version of the workflow is given in figure 6 as modelled using BPEL4WS.

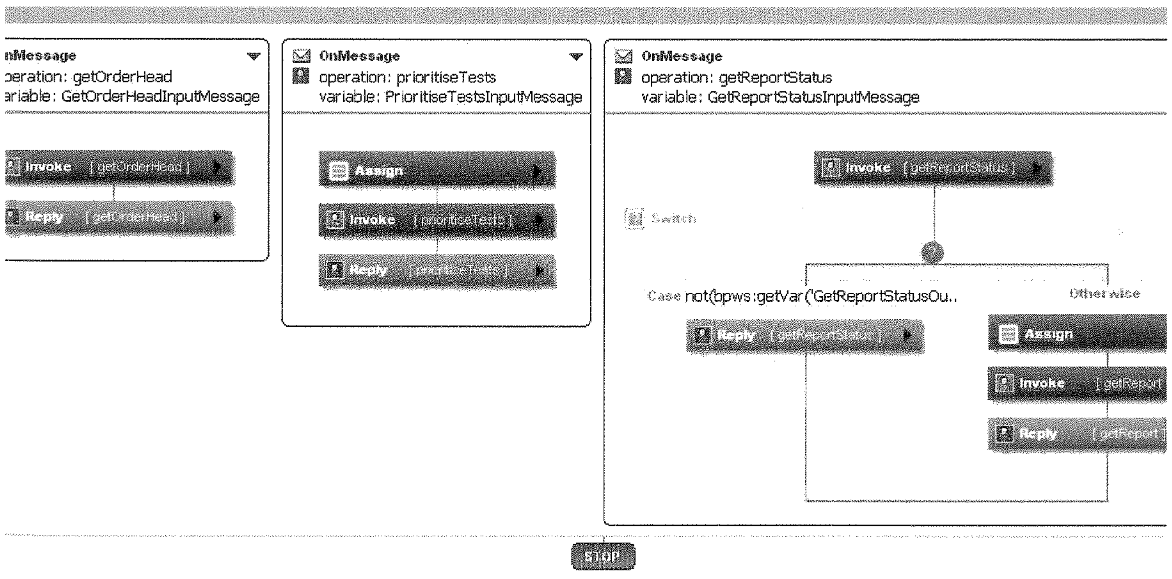

nMessage
peration: getOrderHead
ariable: GetOrderHeadInputMessage
Invoke [getOrderHead]
Reply [getOrderHead]
OnMessage
operation: prioritiseTests
variable: PrioritiseTestsInputMessage
Assign
Invoke [prioritiseTests]
Reply [prioritiseTests]
OnMessage
operation: getReportStatus
variable: GetReportStatusInputMessage
Invoke [getReportStatus]
Switch
Case not(bpws:getVar('GetReportStatusOu..
Otherwise
Reply [getReportStatus]
Assign
Invoke [getReport
Reply [getReport]
STOP

Figure 6: A model of part of the laboratory-side workflow

The laboratory-side workflow consists of two separate parts: when a new order is received by ChemSearch and when tests are completed for a client. These are shown in the flowcharts below.

Figure 7 shows the process of receiving a new order from a client. On the order form there is a box to enter the number of samples to be tested for a given site. This number is then checked against the actual samples received from the client. If these are not the same the client is notified, so that new samples can be sent. If they are the same, the tests are prioritised based on a number of factors mentioned below,. and stored in the database for future reference.

As a result, the workflow must keep track of the tests that have been ordered.

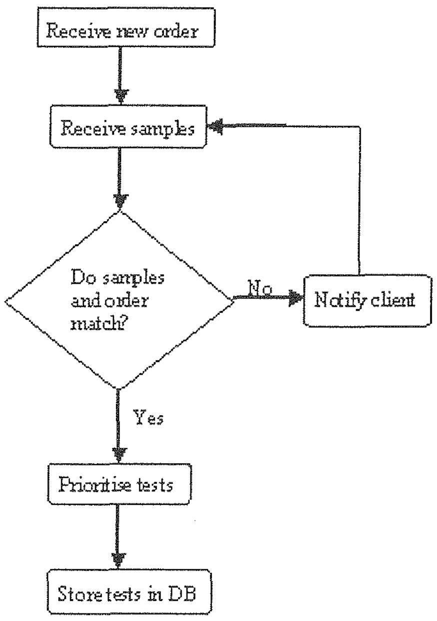


Figure 7: A flowchart of the process of receiving new orders

Figure 8 shows the process of producing the client's report. Once a test has been completed, the initial result is recorded. Often a test needs to be repeated, if this is the case the test is done again and the final result is calculated and stored in the database. If all the tests have been completed for the client, the report is produced containing the results for all the requested tests. This is then sent to the client.

The workflow must keep track of which tests have been completed, their results and the current status of the client's report. The main purpose for this is so that the same WSRF web service can be used by both the laboratory and their clients. In this way the client can access their results and the status of their reports as the tests are completed by the laboratory. This is because they are using the same WS-Resource, which keeps track of the results and the status of the report.

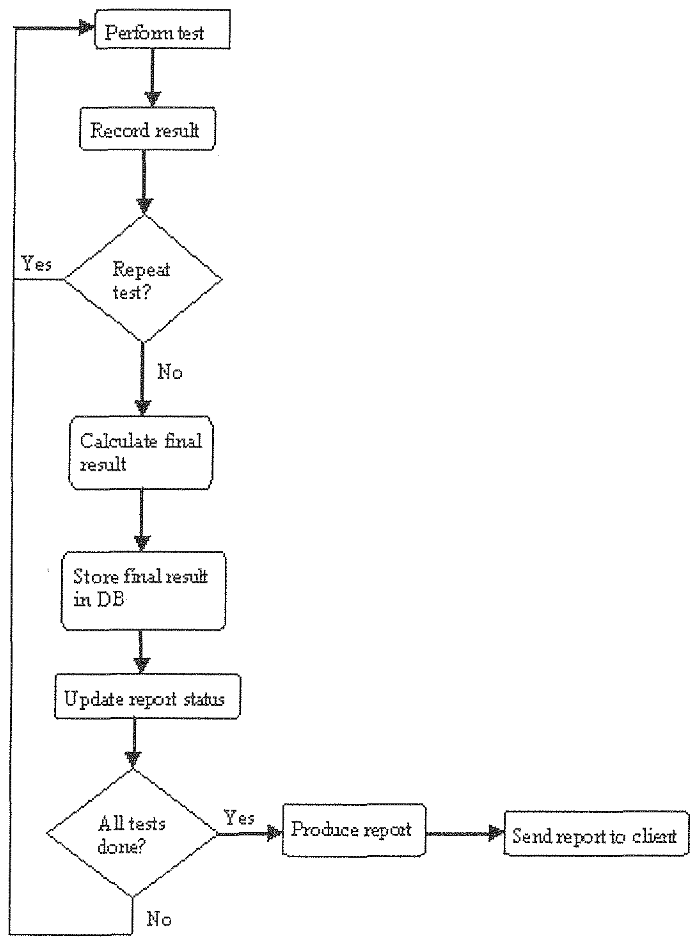


Figure 8: A flowchart of the process of producing the client's report

This side of the workflow was rather complicated, as the prioritisation of the tests depended on a number of factors including the urgency, lifetime of the sample, length of time to perform the test and the number of samples that needed the same test to be done. The majority of the factors were sent to the client in the form of comments on the order form. Retrieving this data would require a change to the order form so that a client could select the tests that were urgent and give them a priority. Since there are 31 possible tests this would make the order form rather cluttered and not very user-friendly. To this end the comments box was kept and the comments were displayed with the prioritisation of the tests. To keep things simple the workflow prioritised the tests based on the number of samples for each test and returned this to the system along with the comments. At the system end, the laboratory could make changes to the suggested prioritisation based on the comments or other factors such as staff illness, which may affect the tests to be done and cannot be monitored by the workflow.

Overall, there were too many factors that could not be modelled by the workflow. These include checking incoming orders against the samples received, performing the tests and analysing the results. Ideally the workflow should automate as many of these factors as possible, so that the benefits from modelling workflow are maximised. A large amount of human input impedes the effectiveness of the workflow and removes the motivation for using it. The reason for this is that sequential web service invocations, i.e. invocations that need to send a response back to the client, end up being no different than a standard class that invokes each service one after the other. Modelling the workflow of the laboratory in this fashion would require complex programs that could monitor the tests, incoming samples and analyse the results, such as current tools that have been developed to model workflow [Amin et al., 2004], [Taylor et al., 2005]. Due to the time constraints and focus of this research, automating the workflow in this manner is clearly outside the scope of this research.

In any event the modelling of the workflow raised some serious issues with BPEL4WS in particular, these are discussed in detail in section 14.3.1.

13 Results

This section outlines the results obtained from applying the method of integration, see section 11, to model the workflow of the ChemSearch laboratory. As a result this method can be evaluated by using a set of criteria defined in section 13.0.1. This evaluation raised some key points about the web services specifications and the tools used. Therefore, a list of these points will be given in subsequent sections, 13.1 and 13.2 with further details described in section 14.

13.0.1 Method of Integration

Reviewing the related literature revealed that criteria were not applied to evaluating proposed systems, tools or solutions. Rather it looked at problems that arose and discussed how the solution achieved it and any issues that still remained. To this end the difficulties with the web services specifications and tools will be listed here with discussion to follow in section 14. In order to truly evaluate the method of integration criteria has been established based on the major requirements that I have determined from reviewing the related literature and WSRF specification. These are listed below.

- The method of integration enables BPEL4WS to be combined with WSRF - Y/N
- The method of integration can be generalised for use in any workflow that needs WSRF - Y/N
- Does the method of integration provide a means of storing endpoint references - Y/N
- Does the method of integration cater for easy retrieval of endpoint references - Y/N
- Can the method of integration be applied to other WSRF and BPEL4WS tools - Y/N
- Does the method of integration scale well, if applied to large, complex workflows - Y/N
  (i.e. workflows that may contain a number of WS-Resources)

The results from the application of this criteria to the method of integration outlined in section 11 are shown in table 1.

Note: Scalability refers to how much extra work is required to adapt the method of integration to incorporate multiple WS-Resources. It does not pertain to the creation of

Criterion	Result
The method enables BPEL4WS to be combined with WSRF (See section 14.2 for further details)	Y
The method can be generalised for use in any workflow that needs WSRF (See section 14.2.1 for further details)	Y
Does the method provide a means of storing endpoint references (See section 14.2.2 for further details)	Y
Does the method cater for easy retrieval of endpoint references (See section 14.2.3 for further details)	Y
Can the method be applied to other WSRF and BPEL4WS tools (See section 14.2.4 for further details)	Y
Does the method scale well, if applied to large, complex workflows (See section 14.2.5 for further details)	N

Table 1: The evaluation of the method for combining WSRF with BPEL4WS

WS-Resources or work that would normally need to be done in the process of creating the workflow. It only refers to the work that has to be done over and above normal workflow creation in order to implement the method of integration

Although no concrete data was obtained from modelling the workflow, there were a number of issues raised that logically fall into the category of results. No actual discussion of the findings will appear in this section, as this section is solely for the purpose of outlining the issues with the method of integration and tools in general.

13.1 Web Services Specifications

This section outlines firstly the key points raised with the web services specifications, namely WSRF, section 13.1.1, and BPEL4WS 13.1.2 and then the key points with the tools used, see section 10 for further details, firstly Cape Clear, section 13.2.1 and then Apache WSRF section 14.4.2.

13.1.1 WSRF

This section lists the key points raised from using WSRF, see section 14.3.2 for more details.

- The necessity of making the state persistent in an external data source questions the need for WSRF.
- There is some difficulty in determining what needs to be a stateful resource and how this should be managed.
- In particular there is no standard mechanism defined in the WSRF specification for storing and retrieving endpoint references.

13.1.2 BPEL4WS

This section lists the key points raised from using BPEL4WS, see section 14.3.1 for more details.

- The general usability of BPEL4WS is a problem, in particular its flexibility and usefulness in the modelling of this type of workflow.
- The un-testability of the workflow to try and find errors.
- Assignment of variables is inflexible.

13.2 Tools

The use of these particular tools, see section 10, impacted on the method implemented. The key points from using these tools are listed below.

In general the lack of good documentation was a serious issue along with the severe incompatibilities of the tools.

13.2.1 Cape Clear Orchestration Studio

The key points raised about this tool are given below, see section 14.4.1 for more details.

- The poor quality of the documentation.

- The SOAP messages were incompatible with the messages sent from the WSRF client-side stubs.
- Cape Clear was unable to invoke a WSRF web service.
- The assign statement copied the message structure instead of the value and did not allow for copying multiple, individual values.
- Modification to the WSDL meant re-engineering the entire workflow.

13.2.2 WSRF Apache Project

A list of the key points raised from the use of this tool are given below, see section 14.4.2 for more details.

- Lack of client-side stubs
- Forced to use WS-Addressing stubs which didn't add reference properties to the SOAP header.

14 Discussion

In section 11, I outlined the method that was implemented to solve the integration problem. I will now outline firstly some other options I explored and why these were not feasible in practise. I will then move on to discuss in detail the results I obtained about this method of integration. In subsequent sections I will discuss the results obtained on each of the specifications and tools in this order. I will conclude this section with a brief summary of the results in general that will focus on improvements that I think will be necessary for these two technologies to work together. Finally, I will end with a section on the system that was implemented and the issues and future work that could be done.

14.1 Alternatives

The purpose of this section is to describe alternatives that were investigated to determine if BPEL4WS would work directly with WSRF.

The BPEL4WS specification states that an endpoint reference as defined by WS-Addressing [W3C, 2004b] can be assigned dynamically to a partner link, which in BPEL4WS terms is basically a web service. While the current Cape Clear documentation stipulated the support for assigning endpoint references this was not the case in practise. I discovered that in fact the current version of Cape Clear Orchestration Studio did not support the dynamic assignment of end point references. Neither the BPEL4WS specification nor Cape Clear documentation specifically mentions whether reference properties included as part of the endpoint reference would be added to the SOAP header of the message. The specification only goes as far as acknowledging that data can be placed in the header or in the body of a SOAP message. it does not precisely state the placement of reference properties. Based on my experience with WS-Addressing stubs, it is very unlikely that reference properties would be copied into the SOAP header, since they are an optional part of an endpoint reference. This is backed up by research into this area, which also suggested that it was unlikely to include reference properties in SOAP message headers [Slomiski, 2005].

The WS-Addressing stubs were used to enable the system developed in Eclipse to send and receive XML messages and did not by default add reference properties to the SOAP header. It does extract the reference properties from the endpoint reference passed to it, but these are never set when the call object is created, so never get added to the SOAP header. Perhaps this is because no checking is done to make sure the reference properties are not null. I am not sure what effect it would have on the message if a null value was added to the SOAP header, but I do know that having null values in the body of the message causes a null pointer exception when the message is deserialised - the process of extracting application data from the XML message.

Since WSRF only checks for the existence of reference properties when there is an entry in the jndi-config.xml file for the service, see figure 9, then requiring a value to exist in the SOAP header would not cause any problems if the value did not uniquely identify a WS-Resource and the call was not invoking a WS-Resource instance.

Therefore I think it would go a long way to aid the integration process if there were some underlying code that checked for the existence of reference properties in the endpoint

```
<parameter>
        <name>resourceKeyName</name>
        <value>chemsearch/ChemSearchServiceResourceIdentifier</value>
</parameter>
```

Figure 9: An extract from the jndi-config.xml file for the WS-Resource service

reference. If there is an entry for the reference properties then these should automatically be added to the SOAP header. From my experience in trying to integrate WSRF and BPEL4WS, the BPEL4WS process needs to do some form of checking for the existence of reference properties for BPEL4WS to truly support WSRF web services. If by default these reference properties were added to the SOAP header it would alleviate the need for BPEL4WS to know when the web service it is accessing is a WSRF one or not (assuming of course that BPEL4WS is communicating directly with the WS-Resource and not some intermediary mechanism).

Even if the Cape Clear Orchestration Studio had supported the dynamic assignment of endpoint references and these were included in the SOAP header by default, this solution would not have worked. This is because the Cape Clear tool threw an exception every time it tried to invoke a WS-Resource web service. After some investigation and looking at the error messages produced by the tool, I discovered that it was the imported WSDL definitions needed to declare the web service as being WSRF that were causing the problem, see appendix B. Further investigation would be needed to determine whether other BPEL4WS tools had the same problem, or whether this was just an issue with the Cape Clear tool. Since a specification, in this case BPEL4WS, is usually just a blue print for how workflow could be designed and implemented it seems unlikely that this is an issue stemming from the specification itself. It seems more likely that it is the way in which the Cape Clear tool validates the BPEL4WS workflow.

Therefore the only option that seemed to remain was the method outlined in section 11. A discussion of this method is given in the following section.

14.2 Method of Integration

The method found to integrate BPEL4WS and WSRF was similar to solutions proposed in the research [Slomiski, 2004], [Amin et al., 2004]. The solution proposed within this report was outlined in section 11 and so a description of that method will not reappear here.

The results shown in table 1 indicate that the method for combining WSRF and BPEL4WS can be applicable to any workflow needing both BPEL4WS and WSRF. However, there is some doubt as to its scalability. Each criterion in the table will be discussed further in the appropriate sections below.

14.2.1 Generalisation

The solution to the integration problem uses a third WSRF web service, referred to as the proxy service (see figure 1). The BPEL4WS workflow only interacts with this web service and so does not need to know about the two web services common to WSRF, i.e. the factory service and WS-Resource. Since the actual web service containing the WSRF specific implementations is abstracted out of the workflow, this solution should generalise to any implementation of a workflow containing WSRF and BPEL4WS. This is because a WSRF web service that uses the singleton pattern, as defined in section 2, does not require any of the implementations necessary to define a WSRF web service. This means that as far as BPEL4WS is concerned the WSRF web service is like any other normal web service and since a BPEL4WS workflow, according to the specification, is itself a web service this method should work with any BPEL4WS workflow that adheres to the basic principles of the BPEL4WS specification version 1.1.

14.2.2 Storing Endpoint References

In order for BPEL4WS to be combined with WSRF there needs to be a mechanism that stores the endpoint references for any newly created WS-Resources. Figure 10 demonstrates the need for storing endpoint references.

The WSRF specification does not currently provide a blueprint for how to implement this solution; however a description of how this was implemented is given in section 11.

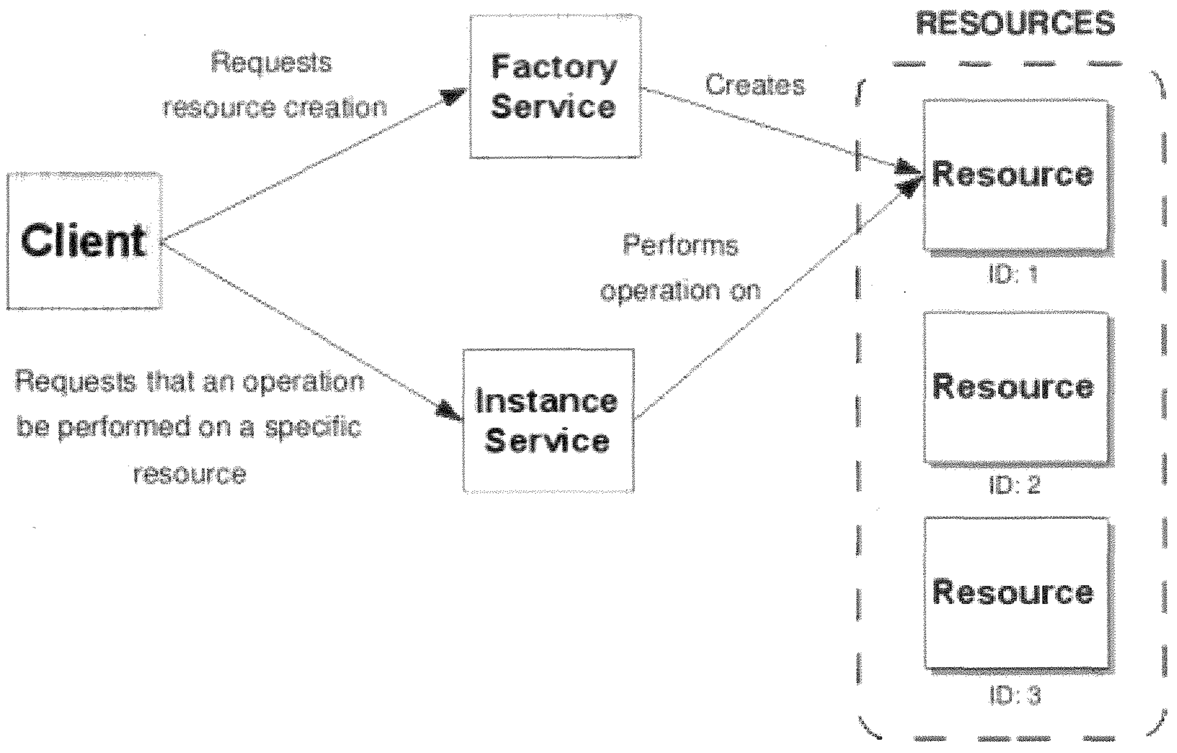


Figure 10: The WS-Resource Factory Pattern [Sotomayor, 2005]

This method of storing endpoint references means the WS-Resource keeps a reference to the endpoint reference as well as it being added to the underlying cache. This endpoint reference is then returned to the proxy service, which maintains the reference until the next time a web service is invoked. Currently this has to be done by making the endpoint reference static, as a new call to a web service by default creates a new instance of the service class.

The endpoint reference could be made persistent in the event of server failure by writing it to an external data source such as a file or database.

The need to make the endpoint reference static is the main disadvantage of this method. By the definition of WSRF the endpoint reference should really be modelled as state, since it needs to be maintained across multiple invocations of a service. In practise it cannot be modelled this way, because it would end up in an infinite number of WSRF services. As the main benefit of using the proxy service in the first instance is that it does not require being a WSRF service. By making it a WSRF service, it would be incompatible with BPEL4WS and so a new service would be needed, which would need endpoint references to call the proxy service and this would go on and on.

This is why it is important for there to be a standard way to store endpoint references to avoid using implementations that are less than ideal.

For the purpose of enabling BPEL4WS to work with WSRF this solution is acceptable.

14.2.3 Retrieving Endpoint References

This problem is closely related to the previous issue, but retrieving an endpoint reference requires a unique identifier to match the WS-resource with the incoming call to the web service.

This is implemented by using the factory service to check if a resource exists using the site as a unique identifier for the resource, see section 11 and figure 2 for further details.

The advantage of this method is that it provides a solution to the lost or invalid endpoint reference problem, since the factory always returns a valid reference to the resource.

This method could be generalised to any implementation of WSRF by defining a pattern of use. One such pattern could define a method that takes in a unique identifier and locates all the resources stored by the service. If a resource exists that matches the unique identifier passed to this method, then a reference to this resource is returned. Otherwise an exception is thrown that when caught invokes another method that creates a new instance of the resource, passing into the constructor the unique identifier and a new endpoint reference. This resource is then returned to the caller of the method.

By defining a pattern, retrieving valid endpoint references will always work regardless of the WSRF implementation, so long as it adheres to the basic specification, or the programming language used.

However, there is currently a specification called WS-RenewableReferences being defined that will hopefully solve the problem of lost or invalid endpoint references. When this becomes available it will alleviate the developer from dealing with this issue.

Until such time the implementation described within this document fulfills the purpose of retrieving endpoint references.

14.2.4 Application to Other Tools

This method should be applicable to other tools for the same reasons as section 14.2.1. Since the method removes the need for the BPEL4WS workflow to know anything about WSRF this method should work with any BPEL4WS tool regardless of its support for WSRF. In the event that a tool is used that enables the assignment of endpoint references that copies the contents of the reference properties into the SOAP header, the proxy layer can be removed so that the workflow works directly with the WSRF web service, without the need for any additional implementation.

14.2.5 Scalability

One of the disadvantages of this method is that it doesn't scale well, see figure 11. This is because there is now effectively two web services instead of one. In theory if another WS-Resource was needed the operations defined in its WSDL file could be added to the existing proxy service, instead of creating a new one thus limiting the scalability problem. However, this would only be appropriate for very small web services as the proxy service would soon become very large and unmanageable. Therefore, for large complex services it would be better to define a separate proxy service for each WS-Resource needed. In implementing it this way, it also alleviates the need for the proxy service to keep track of two different sets of endpoint references — one for each defined WS-Resource.

The key issue with this method in terms of scalability is that two sets of WSDL need to be updated instead of one. If a change is made in one file and not the other, the method signatures will be different. This means one method will not be able to call the other.

However it does mean the API for the proxy service can be made simpler to deal

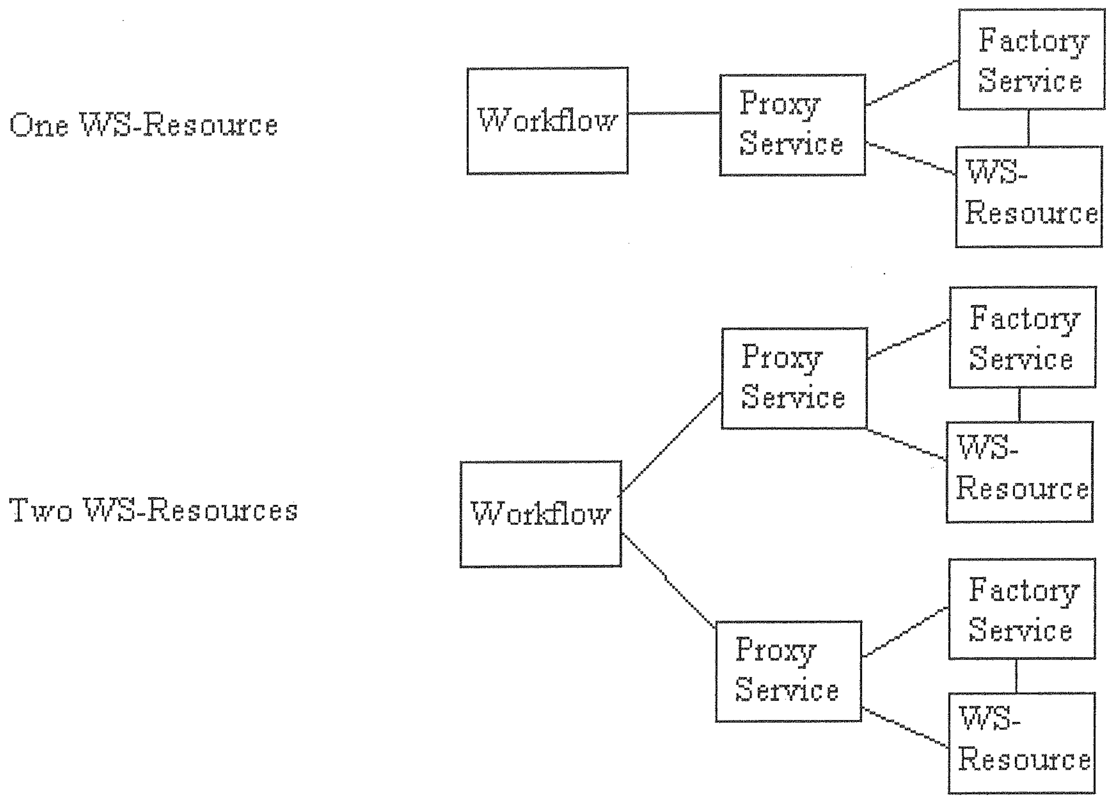


Figure 11: The Scalability Problem

with issues with tool incompatibilities or general limitations of the workflow. In my case the incompatibilities, to be discussed in detail in later sections, did not affect the WS-Resource WSDL as the proxy web service just implemented additional processing to deal with the issues with the tools.

14.2.6 Summary of the method

Overall, the method successfully allows WSRF to be combined with BPEL4WS. The key issue with this method is the scalability as for each WS-Resource web service that is added there needs to be an accompanying proxy service. This also introduces issues with duplication of data as the method signatures defined in both the proxy and WS-Resource WSDL files have to be the same. The main advantage of this method is that the BPEL4WS workflow can treat the proxy service as a normal web service as it does not contain any of the signatures that define a WSRF web service. As WSRF usually has two web services, a factory service that creates the WS-Resource and the WS-Resource itself, the use of a proxy service alleviates the workflow from knowing about two web services, instead it only has to interact with one.

In implementing it this way, the method of integration should work with any tool no matter the level of support for WSRF and can easily be modified should a future version of BPEL4WS fully support WSRF; just remove the proxy web service.

The other key advantage of this method is that it alleviates the need for the workflow to keep track of endpoint references; instead this is delegated to the proxy and factory services. If the workflow had to keep track of endpoint references, the BPEL4WS specification would have to provide support for matching incoming requests to the right WS-Resource instance. Currently the BPEL4WS specification supports the use of correlation sets, which enables incoming message signatures to be matched with existing process instances. It is unclear from the specification if there is an option to create a new process instance or use an existing one. In delegating this responsibility to a web service, the implementation of that service can deal with how to achieve this. This provides greater flexibility as the web service will be implemented in a specific programming language that will enable the developer to choose the most appropriate way to implement this. However, there should really be a pattern defining an implementation. Currently the WSRF specification does not provide support for this.

Perhaps this will be developed once the specification has been standardised and received widespread support. It is common in these types of situations that a best practise is developed saving developers from having to reinvent the wheel - finding a solution to a common problem.

I will now discuss the specifications and the tools used to implement this method in the context of modelling the workflow of the ChemSearch laboratory.

14.3 Specifications

The two main specifications used to model the workflow of the laboratory were BPEL4WS [Andrews et al., 2003] and WSRF [Czajkowski et al., 2004].

In section 13 a list of key points raised from the implementation of the method will now be discussed.

It should be noted that the main issue with these specifications is lack of standardisation.

14.3.1 BPEL4WS

This section will discuss in detail the key points mentioned in section 13.1.2.

The first problem mentioned was the general usability of BPEL4WS.

The workflow of the ChemSearch laboratory dealt with large amounts of data and at times was quite complicated. I found that BPEL4WS seemed to be unequipped firstly to deal with the volume of data and secondly was rather inflexible, but that could have been due to the fact that a BPEL4WS workflow is itself a web service. One of the problems of modelling this type of workflow, where there requires a significant amount of human input, is returning results. Dynamically changing the return type of a web service invocation would alleviate this problem, however this is not feasible in practise. In particular, it is not a problem that could be solved by refining the BPEL4WS specification; it is more of a problem with web services in general. This is because web services does not enable dynamically modifying the WSDL file. The WSDL file would need to be changed to enable the return type of a service to be changed. Since this process involves recompiling the WSDL file and modifying the service implementation, this approach is not feasible in practise. As a result this functionality is not provided by current web services technology. Improving the BPEL4WS specification would not solve this problem because it is still based on web services technology.

If a BPEL4WS process defines the workflow so that multiple web services are required to return results, all these invocations must have the same return type. To illustrate this point I will use an example from the workflow. Logically checking the status of the report should involve returning the report if all the tests have been completed. In order to achieve this, the workflow waits for a message requesting the report status, which returns a string representing the status of the report. In order to return the report if all the tests have been completed the report status must have the same return type as the report. The report method returns an array of sites, tests and results, while the report status returns a string. Hence for the process to return the report, the report status must return a string plus an array of sites, tests and results.

The second mentioned problem was the untestability of the workflow.

Since a BPEL4WS process is based on WSDL files provided to it, it is more of a modelling tool than a programming language such as Java. One of the benefits of using a language such as Java is that you can print data out to the console to figure out the source of a problem when something goes wrong. It would be beneficial if BPEL4WS had some support for this. Currently it is a trial and error process to determine the source of the problem. This is because the error message that is printed out to the console does not specify the part of the process that caused the error. As a result the process is modified and then re-compiled and deployed to determine if the change fixes the error in the workflow. This can be a time-consuming process, especially if there are a number of different errors in the workflow. Since fixing an error requires modifying the workflow, re-compiling and deploying the workflow before it can be tested again to see if the change fixes the error.

The final problem was assigning values to variables.

Currently the assignment of values to variables is rather inflexible. There would be greater flexibility if the specification provided a means of assigning multiple values to the same variable, when there are multiple values defined in the variable itself, as illustrated in figure 12.

Currently BPEL4WS only allows the assignment from one variable to another. If two variables do not have the same message structure assigning can make the variable invalid and produces an error when the service is invoked. This is because the message

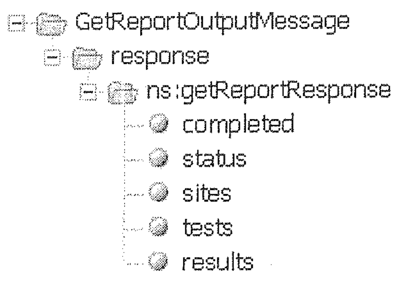


Figure 12: The structure of a message variable defined in BPEL4WS

sent does not match the one expected. For example, often it was the case that one invocation of a service followed naturally from another. The issue is when both services return different values. It would be easier on development if both values could be assigned into a single variable that could be returned. Currently the only option is to have the WSDL for both services return the exact same variable, which has a duplication issue in that you have to update the WSDL in two separate places instead of one. Unfortunately these types of problems can only be determined once you are ready to test the workflow and at this stage any changes to the WSDL require re-engineering the workflow.

Overall, I think the specification needs some work. It seems to be better suited to modelling workflow when there is little need for human input.

14.3.2 WSRF

This section will discuss the key points outlined in section 13.1.1.

The first problem raised the issue of whether WSRF was needed or not.

This issue arises because any web service can have state by making its data persistent, for example, in a database. However, this does not provide a template for how to do this, which is one of the benefits of WSRF. Therefore, there are significant advantages, such as a defined pattern for implementing state, widespread support in the event of standardisation and the availability of tools to support implementation, in providing a specification that deals exclusively with the manipulation of state. One further argument for a specification that deals with state is that it removes multiple, different, incompatible implementations.

The second problem was determining what should be modelled as state.

This was raised as a potential problem [Humphrey et al., 2004], [Wasson et al., 2005] but in my experience it really has nothing to do with the specification; it is more of a design decision. Personally, I think using WSRF was very useful in helping model the workflow of the laboratory. It provides a standard way to model and access state, although it is only persistent for the lifetime of the server. This is where an external data source can come in handy.

The third and final problem was that there is no standard mechanism to store and retrieve endpoint references.

This is a problem because every WSRF web service has to store and retrieve endpoint references in order to invoke WS-Resources. If there is no standard mechanism to do this, developers have to implement this themselves. This wastes valuable development time, as in general a problem that exists for multiple people should have a pattern defined to solve it, to stop the development of a solution that already exists. Therefore, it is in the best interests of the WSRF specification to provide a pattern for how to implement this problem. In doing so it will improve the chances of the specification being widely adopted and will make it more robust.

14.4 Tools

Although there were a couple of issues that needed addressing at the specification level, most of the issues encountered stemmed from the tools used. Therefore this section will focus on possible improvements to the mentioned tools, in light of the implementation of the method.

14.4.1 Cape Clear Orchestration Studio

The main problems with the Cape Clear tool were identified in section 13.2.1.

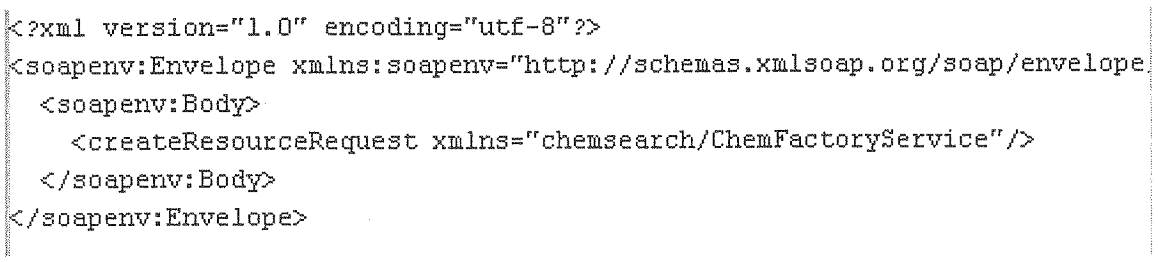
The first problem with Cape Clear was the quality of the documentation.

The documentation included with the Cape Clear tool was of poor quality. It mentioned specific details of how to implement, for example assignment of endpoint references to partner links, which the current version of the Cape Clear tool did not support. This is totally unacceptable for a commercially available product. One of the main reasons for choosing this tool in the first place was that it would produce workflow that could be suitably validated for its use. No matter how good the tool is, if it has poor documentation the overall quality of the product is reduced.

The obvious solution to this – don't include material in the documentation that is not yet supported by the tool.

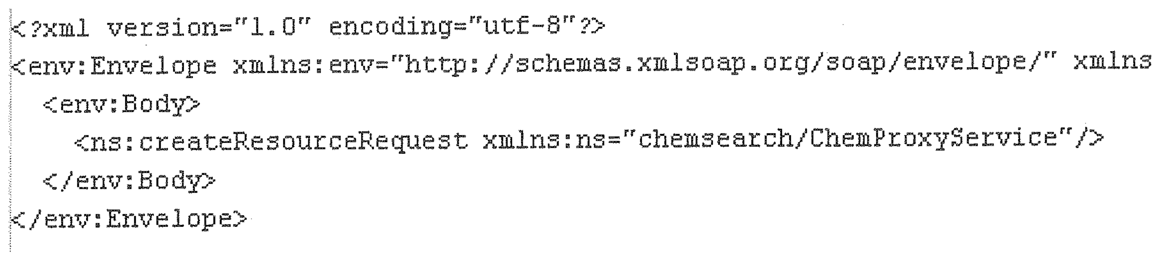
The second problem with Cape Clear was the SOAP messages.

There was a problem with the incompatibility of the SOAP messages sent from the WS-Addressing stubs to the Cape Clear Server. From looking at the messages sent and received, the WS-Addressing messages used the part name of the message from the WSDL file to wrap the data in, whereas the Cape Clear Server used the actual schema name. This is shown in figures 13 and 14, respectively.



```
<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope
  <soapenv:Body>
    <createResourceRequest xmlns="chemsearch/ChemFactoryService"/>
  </soapenv:Body>
</soapenv:Envelope>
```

Figure 13: The SOAP message sent from the WS-Addressing stubs



```
<?xml version="1.0" encoding="utf-8"?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/" xmlns
  <env:Body>
    <ns:createResourceRequest xmlns:ns="chemsearch/ChemProxyService"/>
  </env:Body>
</env:Envelope>
```

Figure 14: The SOAP message sent from the Cape Clear Server

This was rather inconvenient as you could only find this out when running the workflow from within Eclipse.

As a result the only messages that were acceptable to the Cape Clear server from the client-side stubs were messages with no input parameters, as an assign could be done server-side to format the messages into a form that was acceptable to the process. Alternatively messages with simple types for parameters, such as strings and ints, were acceptable. This meant that the laboratory had to use strings to send the data to the web service in order for the messages to be acceptable to the process. This is where the implemented solution has a key benefit. The proxy web service can extract the data out from the string and pass it to the WS-Resource web service in a more appropriate format. In the event that a BPEL4WS tool supports WSRF the WSDL for the WS-Resource will not have to change and the system will only need minor changes to work. Compare this to the changes that would have to be made to the WS-Resource WSDL if the process interacted with it directly.

However, problems with the incompatibilities of the tools cannot be determined until it comes time to test the workflow. Any changes that have to be made to the WSDL at this point require the WSDL to be re-compiled and deployed and the workflow has to be completely recreated from scratch.

A standard mechanism to define the way SOAP messages are created would alleviate this problem

The second mentioned problem was the inability of Cape Clear to invoke a WSRF web service.

This problem arose from the imported WSRF definitions that define a WSRF web service. For some reason Cape Clear threw an exception when trying to invoke these services. From the error message it seemed that Cape Clear did not support the way in which the imported WSDL file was defined. Since both BPEL4WS and WSRF support WSDL 1.1, this cannot be a specification problem rather it must be the way in which Cape Clear validates the BPEL4WS process and since by default the BPEL4WS process also imports whatever definitions are contained in the underlying WSDL files, these will affect the validation of the process.

Whether this is a bug in the tool has yet to be determined. However, this problem needs to be addressed in case it applies to more than just the WSRF definitions. There is no point in defining workflow if it doesn't recognise imported WSDL definitions.

The final problem mentioned was modifying the WSDL involved recreating the workflow from scratch.

If the underlying WSDL files change the workflow has to be completely re-engineered. I suppose the argument for this is that the WSDL should be properly defined before the workflow is developed. The problem with this argument is that some of these problems that I have outlined do not become apparent until you come to test the workflow. Perhaps some of these problems will be resolved once BPEL4WS has been standardised and more widely adopted.

Alternatively, defining a mechanism where the workflow could be refreshed so that any changes made to the underlying WSDL would be reflected in the workflow automatically, would avoid this problem.

14.4.2 Apache WSRF project

This section will discuss the key points raised from the use of this tool to implement and test the method of integration.

A list of these points can be found in section 13.2.2.

The first problem was the lack of client-side stubs.

In order for elipse to invoke the BPEL4WS process deployed at the CapeClear server, there needs to be client-side stubs. These stubs are responsible for creating the SOAP message that contains the parameters for the service that is being invoked, and extracting the response from the SOAP message that is returned by the CapeClear server.
These were not included in the Apache WSRF project.

The lack of provision of client-side stubs meant an alternative had to be found for SOAP messages to be sent from within the Eclipse Integrated Environment. This was needed in order to fully model the workflow, see section 12 for a detailed discussion of this reason.

Fortunately the Apache project supplied WS-Addressing stubs that could be used to generate the client-side code in Java necessary to send and receive SOAP messages. However, although it allowed the SOAP messages to be sent and received, the stubs by default did not add the reference properties found in a WS-Addressing endpoint reference to the SOAP header.

This leads on to the final problem with Apache WSRF, namely that the WS-Addressing stubs did not add reference properties to the SOAP header.

This meant the generated code had to be hacked to enable the reference properties to be added to the SOAP header, while this wasn't a difficult task to do it meant that every time a WSDL file was created this code had to be added to the stubs.

Given that endpoint references are part of the fundamental framework of WSRF it would benefit WSRF if its implementations could provide a mechanism for automatically adding reference properties to SOAP headers.

This would require checking if the reference properties were not null before adding them in case sending null values in the header would cause problems. Since the stubs from the Apache project automatically create all the code for you to create and send messages, surely it wouldn't be too difficult to add another section of code to deal with endpoint references.

Adding this kind of support would go a long way to increasing the adoption of WSRF technologies, as it is quite off-putting to use a tool where you have to hack the generated code, as there is an underlying assumption that if there is something wrong with your project it is unlikely to be the generated code, especially if the project in question is not in a beta release.

Since the Apache WSRF project has only just been released, maybe there will be support for endpoint references in a future release once the majority of the bugs have been removed.

Overall, the method of integration enabled WSRF and BPEL4WS to work together, but it did raise the problems stated in table 2. Most of these problems could be resolved, such as SOAP messages, adding reference properties to the message header, etc., by providing a pattern of use that defines a generic solution to these problems, rather than each developer coming up with their own solution. This leads to re-inventing the wheel, which is highly undesirable. Overall, any limitations with the WSRF tool could be due to the fact that the version of the tool used in this research was only a beta release. Therefore, the problems mentioned in table 2 could be resolved in a future release of the tool, once most of the bugs have been removed from it.

The Cape Clear tool on the other hand was very inflexible, due to the nature of BPEL4WS, and had problems with imported WSDL definitions. It would be interesting

Problem	Specification/Tool
Reference properties do not get copied into the SOAP header of a message	BPEL4WS specification, WSRF tool
Endpoint references cannot be dynamically assigned to a web service	Cape Clear tool
Lack of testability of the workflow	BPEL4WS specification
No client-side stubs	Apache WSRF tool
No standard mechanism for storing and retrieving endpoint references	WSRF specification
Usability of BPEL4WS	BPEL4WS specification
Lack of compatibility with WSRF	BPEL4WS specification, Cape Clear tool
Assignment of variables	BPEL4WS specification
Problems with SOAP messages	Cape Clear tool
Changes to WSDL require workflow to be recreated	Cape Clear tool
Poor documentation	Cape Clear tool, WSRF tool

Table 2: Summary of the problems encountered during the integration of WSRF and BPEL4WS

to know whether it is just WSRF imports that it could not resolve or whether the tool cannot deal with imported WSDL definitions that it does not already support. Cape Clear was able to resolve WS-Addressing imports defined in the factory service, but this could be because Cape Clear supports WS-Addressing.

The poor quality of the documentation was unacceptable for a commercially available tool and it would seem from this analysis that the Cape Clear tool needs a lot of work, before it could be recommended as being suitable for modelling BPEL4WS workflow.

In terms of the specifications, WSRF seems a worthwhile contribution to the web services community and provides a standard means of accessing and modelling state that proved very useful in modelling the workflow of the ChemSearch laboratory. The BPEL4WS specification on the other hand needs some work to make it more flexible and user-friendly. A mechanism to test the workflow would be very useful for error detection.

On a more general note, web services technology has a lot to offer over other conventional distributed technologies, such as CORBA (Common Object Request Broker Architecture), with their service-oriented architecture and interoperability. Unlike CORBA, web services uses open Internet-based standards, which means web services can be deployed on any platform. The service-oriented architecture prevalent in web services technology means systems using web services are loosely coupled, which promotes modularity. The use of web services technology would greatly benefit the ChemSearch laboratory or any laboratory for that matter due to the open-source, Internet-based standards. The addition of workflow modelling specifications, such as BPEL4WS, mean the workflow of the laboratory can also be modelled using web services technology. The number of open-source tools available for developing web-services also makes building web services a semi-trivial task and for laboratories, such as ChemSearch, there should be plenty of tools available that could be "suitably validated" as required by the standard ISO/IEC 5.4.7.2 for the competence of testing and calibration laboratories.

15 Future Work

This section discusses future work that could be done.

It might be interesting in possible future work to look at the extension of BPEL4WS and hence this method for use in a grid services environment.

16 Conclusion

In conclusion, this project highlighted some problems with the specifications that would need to be resolved for their integration to be truly successful. The method used to integrate BPEL4WS and WSRF fulfilled all the major requirements but would not scale well. Although scalability is an issue it also meant that BPEL4WS didn't need to know that it was interacting with a WSRF web service and meant that tool incompatibilities could be handled in the middle layer rather than at the WS-Resource end. This means that if a BPEL4WS tool was more compatible with WSRF, it would only require small changes. However, it would be better for the BPEL4WS process to interact with WSRF through the proxy service, rather than directly with WSRF. This prevents BPEL4WS having to store and retrieve endpoint references and communicate with two web services, i.e. the proxy and the factory. It would seem that the benefits of this method far out weigh the cost of scalability.

Overall, the two major problems with these specifications is the lack of good documentation and standardisation. In particular the optionality of reference properties in endpoint references is a problem that needs to be resolved. As long as it is optional for these to be added to the SOAP header, there can be no elegant solution to the integration process. As there needs to be some code at some point that determines whether a reference property exists for a particular endpoint and if it does to copy it into the SOAP header. Since the WS-Addressing client-side stubs used with WSRF did not perform this functionality it is unlikely to expect BPEL4WS or any other web service technology to support this in the near future.

17 References

[Amin et al., 2004]	Amin K., von Laszewski G., Hategan M., Zaluzec N. J., Hampton S., Rossi A., GridAnt: A Client-Controllable Grid Workflow System, Proceedings of the 37th Annual Hawaii International Conference on System Sciences, Volume 07, Number 7, pp 70210c, 2004.
[Andrews et al., 2003]	Andrews T., Curbera F., Dholakia H., Goland Y., Klein J., Leymann F., Liu K., Roller D., Smith D., Thatte S., Trickovic I., Weerawarana S., Business Process Execution Language for Web Services Version 1.1, 2003. Available at: ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf.
[Ant, 2005]	Ant, `http://ant.apache.org/`, accessed 11 July 2005.
[Apache, 2005]	Apache, `http://www.apache.org/`, accessed 11 July 2005.
[Atkinson et al., 2005]	Atkinson M., DeRoure D., Dunlop A., Fox G., Henderson P., Hey T., Paton N., Newhouse S., Parastatidis S., Trefethen A., Watson P., Webber J., Web Services Grids: An Evolutionary Approach, Concurrency and Computation: Practice and Experience, Volume 17, pp 377-389, 2005.
[CapeClear, 2005]	CapeClear, `http://www.capeclear.com/`, accessed 4 October 2005.
[Chao et al., 2004]	Chao K., Younas M., Griffiths N., Awan I., Anane R., Tsai C., Analysis of Grid Service Composition with BPEL4WS, 18th International Conference on Advanced Information Networking and Applications, Volume 01, Number 1, pp 284, 2004.

[Churches et al., 2005] Churches D., Gombas G., Harrison A., Maassen J., Robinson C., Shields M., Taylor I., Wang I.,
Programming Scientific and Distributed Workflow with Triana Services.
Available at:
http://www.extreme.indiana.edu/groc/ggf10-ww/programming_scientific_and_distributed_workflow_with_triana_services/TrianaWorkflow.pdf.

[Czajkowski et al., 2004] Czajkowski K., Ferguson D., Foster I., Frey J., Graham S., Sedukhin I., Snelling D., Tuecke S., Vambenepe W.,
The WS-Resource Framework, Version 1.0, 2004.
Available at:
http://www-128.ibm.com/developerworks/library/ws-resource/ws-wsrf.pdf, accessed 11 July 2005.

[eclipse, 2005] eclipse, `http://www.eclipse.org/`, accessed 4 October 2005.

[Foster et al., 2001] Foster I., Kesselman C., Tuecke S.,
The Anatomy of the Grid: Enabling Scalable Virtual Organizations, First International Symposium on Cluster Computing and the Grid, Volume 00, pp 6, 2001.

[Foster et al., 2002] Foster, I., Kesselman, C., Nick, J., Tuecke, S.,
The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, Globus Project, 2002.
Available at: `http://www.globus.org/research/papers/ogsa.pdf`.

[Foster et al., 2004] Foster I., Frey J., Graham S., Tuecke S., Czajkowski K., Ferguson D., Leymann F., Nally M., Sedukhin I., Snelling D., Storey T., Vambenepe W., Weerawarana S.,

Modelling Stateful Resources with Web Services, Version 1.1, 2004.
Available at:
http://www-128.ibm.com/developerworks/library/
ws-resource/ws-modelingresources.pdf, accessed 11 July 2005.

[GridAnt, 2005] GridAnt, `http://www-unix.globus.org/cog/projects/gridant/`, accessed 11 July 2005.

[Hey et al., 2005] Hey T., Fox, G.,
Special Issue: Grids and Web Services for e-Science,
Concurrency and Computation: Practice and Experience,
Volume 17, Number 2-4, 2005, pp 317-322.

[Humphrey et al., 2004] Humphrey M., Wasson G., Morgan M., Beekwilder N.,
An Early Evaluation of WSRF and WS-Notification via WSRF.NET,
The Fifth IEEE/ACM International Workshop on Grid Computing,
Volume 00, pp 172-181, 2004.

[Hunter et al., 2005] Hunter J., Cook R., Pope S.,
E-Research Middleware: The Missing Link in Australia's
e-Research Agenda, 2004.
Available at:
http://www.dstc.edu.au/Publications/eReseachMiddleware.pdf,
accessed 18 July 2005.

[Johnson, 2005] Johnson B., Building a Web Service–
The Beginning–What is a Web Service?,
http://www.developerfusion.com/show/3245/, accessed 11 July 2005.

[Krishnan et al., 2005] Krishnan S., Wagstrom P., von Laszewskil G.,
GSFL: A Workflow Framework for Grid Services.
Available at: http://www.cs.indiana.edu/ srikrish/

publications/gsfl.pdf, accessed 18 July 2005.

[Leymann, 2005] Leymann F.,
Choreography for the Grid: Towards Fitting BPEL to the Resource Framework.
Available at: http://www.cc-pe.net/CCPEwebresource/c8545to872workflow/c854leymann/c854Leymann.pdf,
accessed 11 July 2005.

[MySQL, 2005] MySQL, `http://www.mysql.com/`, accessed 4 October 2005.

[Oracle, 2005] Executive Briefing: Grid Computing.
Available at: http://reg.itworld.com/servlet/Frs.frs?Context=LOGENTRY&Source=cwstrip&Source_BC=0&Script=/LP/10003705/reg.

[Pasley, 2005] Pasley J.,
How BPEL and SOA Are Changing Web Services Development,
IEEE Internet Computing, Volume 09, Number 3, pp 60-67, 2005.

[Slomiski, 2005] Slomiski A.,
On Using BPEL Extensibility to Implement OGSI and WSRF Grid Workflows, March 2005.
Available at:
http://www.extreme.indiana.edu/groc/ggf10-ww/on_using_bpel_extensibility_to_implements_ogsi_and_wsrf_grids/C871_GridWorkflow2004_Mar05_On_Using_BPEL_extensibility_to_implements_OGSI_and_WSRF_Grids_5.doc.

[Sotomayor, 2005] Sotomayor B.,
The Globus Toolkit 4 Programmer's Tutorial,
http://gdp.globus.org/gt4-tutorial/singlehtml/progtutorial_0.1.1.html,

accessed 4 October 2005.

[Staab et al., 2003] Staab S., van der Aalst W., Benjamins V. R., Sheth A., Miller J. A., Bussler C., Maedche A., Fensel D., Gannon D., Web Services: Been There, Done That?, IEEE Intelligent Systems, Volume 18, Number 1, pp 72-85, 2003.

[Taylor et al., 2005] Taylor I., Wang I., Shields M., Majithia S., Distributed Computing with Triana on the Grid, Concurrency and Computation: Practice and Experience, Volume 17, 2005, pp 1-18.

[Tomcat, 2005] Apache Tomcat, `http://tomcat.apache.org/`, accessed 4 October 2005.

[Triana, 2005] Triana, `http://www.trianacode.org/`, accessed 11 July 2005.

[Tuecke et al., 2003] Tuecke S., Czajkowski K., Foster I., Frey J., Graham S., Kesselman C., Maquire T., Sandholm T.,Snelling D., Vanderbilt P., Open Grid Services Infrastructure (OGSI), Version 1.0. Available at: http://www-unix.globus.org/toolkit/ draft-ggf-ogsi-gridservice-33_2003-06-27.pdf, accessed 18 October 2005.

[UDDI, 2004] UDDI Spec Technical Committee, UDDI Version 3.0.2, 2004, http://uddi.org/pubs/uddi_v3.htm, accessed 11 July 2005.

[W3C, 2001] W3C, Web Services Description Language (WSDL) 1.1, 2001, http://www.w3.org/TR/wsdl, accessed 11 July 2005.

[W3C, 2003] W3C, SOAP Version 1.2 Part 1: Messaging Framework, 2003, http://www.w3.org/TR/soap12-part1/, accessed 11 July 2005.

[W3C, 2004a] W3C, Web Services Glossary, 2004a, http://www.w3.org/TR/ws-gloss/, accessed 18 July 2005.

[W3C, 2004b] W3C, Web Services Addressing (WS-Addressing), 2004b.
Available at:
http://www.w3.org/Submission/2004/SUBM-ws-addressing-20040810/.

[W3C, 2004c] W3C, Web Services Architecture, 2004c,
http://www.w3.org/TR/ws-arch/, accessed 12 July 2005.

[Wasson et al., 2005] Wasson G., Humphrey M.,
Exploiting WSRF and WSRF.NET for Remote Job Execution
in Grid Environments,
Proceedings of the 19th IEEE International Parallel and
Distributed Processing Symposium, Volume 01, Number 1, pp 12, 2005.

[Yang et al., 2004] Yang Y., Tang S., Zhang W., Fang L.,
A Workflow Language for Grid Services in OGSI-based Grids,
Lecture Notes in Computer Science, Volume 3251, pp 65-72, 2004.

Appendices

A FactoryService WSDL

The WSDL pertaining to the creation of client-side WS-Resources is given below.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
    name="ChemSearchFactoryService"
    targetNamespace="chemsearch/ChemSearchFactoryService"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tns="chemsearch/ChemSearchFactoryService"
    xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <wsdl:types>
        <xsd:schema
            targetNamespace="chemsearch/ChemSearchFactoryService"
            xmlns="http://www.w3.org/2001/XMLSchema"
            xmlns:tns="chemsearch/ChemSearchFactoryService"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema">
            <xsd:import
                namespace="http://schemas.xmlsoap.org/ws/2003/03/addressing"
                schemaLocation="http://schemas.xmlsoap.org/ws/2003/03/addressing"/>
            <xsd:element name="createResourceRequest" type="xsd:string"/>

            <xsd:element name="createResourceResponse">
               <xsd:complexType>
                  <xsd:sequence>
                     <xsd:element ref="wsa:EndpointReference"/>
                  </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
     </xsd:schema>
</wsdl:types>

<wsdl:message name="CreateResourceRequest">
   <wsdl:part name="request" element="tns:createResourceRequest"/>
</wsdl:message>
<wsdl:message name="CreateResourceResponse">
    <wsdl:part name="response" element="tns:createResourceResponse"/>
</wsdl:message>

<wsdl:portType name="ChemSearchFactoryServicePortType">
    <wsdl:operation name="createResource">
        <wsdl:input message="tns:CreateResourceRequest"/>
        <wsdl:output message="tns:CreateResourceResponse"/>
    </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="ChemSearchFactoryServiceBinding"
      type="tns:ChemSearchFactoryServicePortType">
    <soap:binding style="document"
          transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="createResource">
        <soap:operation style="document"/>
        <wsdl:input>
            <soap:body use="literal"/>
        </wsdl:input>
        <wsdl:output>
            <soap:body use="literal"/>
        </wsdl:output>
    </wsdl:operation>
</wsdl:binding>
```

```
    <wsdl:service name="ChemSearchFactoryService">
        <wsdl:port binding="tns:ChemSearchFactoryServiceBinding"
              name="ChemSearchFactory">
            <soap:address
                  location="http://localhost:8080/wsrf/services/ChemSearchFactory"/>
        </wsdl:port>
    </wsdl:service>
  <plnk:partnerLinkType name="ChemSearchFactoryService">
     <plnk:role name="server">
       <plnk:portType name="tns:ChemSearchFactoryServicePortType"/>
     </plnk:role>
   </plnk:partnerLinkType>
</wsdl:definitions>
```

B ChemSearchService WSDL

The WSDL pertaining to the modelling of client-side state is given below. since the proxy service is almost identical to the WSDL for this service, it has been ommitted from the appendix.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
    name="ChemSearchService"
    targetNamespace="chemsearch/ChemSearchService"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tns="chemsearch/ChemSearchService"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsrp="http://docs.oasis-open.org/wsrf/2004/06/wsrf-
          WS-ResourceProperties-1.2-draft-01.xsd"
    xmlns:wsrpw="http://docs.oasis-open.org/wsrf/2004/06/wsrf-
          WS-ResourceProperties-1.2-draft-01.wsdl"
```

```
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<wsdl:import
    location="http://docs.oasis-open.org/wsrf/2004/06/wsrf-
              WS-ResourceProperties-1.2-draft-01.wsdl"
    namespace="http://docs.oasis-open.org/wsrf/2004/06/wsrf-
               WS-ResourceProperties-1.2-draft-01.wsdl"/>
 <wsdl:types>
   <xsd:schema
        targetNamespace="chemsearch/ChemSearchService"
        xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:tns="chemsearch/ChemSearchService"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema">

        <xsd:element name="checkUserRequest">
           <xsd:complexType>
              <xsd:sequence>
                 <xsd:element name="username" minOccurs="1"
                      maxOccurs="1" type="xsd:string"/>
                 <xsd:element name="password" minOccurs="1"
                      maxOccurs="1" type="xsd:string"/>
               </xsd:sequence>
           </xsd:complexType>
        </xsd:element>

     <xsd:element name="checkUserResponse" type="xsd:string"/>

     <xsd:element name="getSitesRequest" type="xsd:string"/>

     <xsd:element name="getSitesResponse">
        <xsd:complexType>
           <xsd:sequence>
              <xsd:element name="sites" minOccurs="1"
```

```
                maxOccurs="unbounded" type="xsd:string"/>
        </xsd:sequence>
     </xsd:complexType>
  </xsd:element>

<xsd:element name="resultsSitesRequest">
    <xsd:complexType/>
</xsd:element>

<xsd:element name="resultsSitesResponse">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="sites" minOccurs="1"
              maxOccurs="unbounded" type="xsd:string"/>
       </xsd:sequence>
    </xsd:complexType>
</xsd:element>

<xsd:element name="getFinalResultRequest">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="test" minOccurs="1"
              maxOccurs="1" type="xsd:string"/>
         <xsd:element name="result" minOccurs="1"
              maxOccurs="1" type="xsd:double"/>
         <xsd:element name="site" minOccurs="1"
              maxOccurs="1" type="xsd:string"/>
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>

<xsd:element name="getFinalResultResponse">
```

```
    <xsd:complexType/>
</xsd:element>

<xsd:element name="resultsTestsRequest">
    <xsd:complexType/>
</xsd:element>

<xsd:element name="resultsTestsResponse">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="tests" minOccurs="1"
                 maxOccurs="unbounded" type="xsd:string"/>
            </xsd:sequence>
    </xsd:complexType>
</xsd:element>

<xsd:element name="getTestsRequest">
    <xsd:complexType/>
</xsd:element>

<xsd:element name="getTestsResponse">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="tests" minOccurs="1"
                 maxOccurs="unbounded" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>

<xsd:element name="getAreasRequest">
    <xsd:complexType/>
</xsd:element>
```

```
  <xsd:element name="getAreasResponse">
     <xsd:complexType>
        <xsd:sequence>
           <xsd:element name="areas" minOccurs="1"
                maxOccurs="unbounded" type="xsd:string"/>
          </xsd:sequence>
      </xsd:complexType>
  </xsd:element>

  <xsd:element name="resultsAreasRequest">
      <xsd:complexType/>
  </xsd:element>

  <xsd:element name="resultsAreasResponse">
     <xsd:complexType>
        <xsd:sequence>
           <xsd:element name="areas" minOccurs="1"
                   maxOccurs="unbounded" type="xsd:string"/>
          </xsd:sequence>
      </xsd:complexType>
 </xsd:element>

<xsd:element name="getSiteResultsRequest">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="site" minOccurs="1"
              maxOccurs="1" type="xsd:string"/>
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>
```

```
<xsd:element name="getSiteResultsResponse">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="tests" minOccurs="0"
              maxOccurs="unbounded" type="xsd:string"/>
         <xsd:element name="results" minOccurs="0"
              maxOccurs="unbounded" type="xsd:double"/>
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>

<xsd:element name="getTestResultsRequest">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="test" minOccurs="1"
              maxOccurs="1" type="xsd:string"/>
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>

<xsd:element name="getTestResultsResponse">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="sites" minOccurs="0"
              maxOccurs="unbounded" type="xsd:string"/>
         <xsd:element name="results" minOccurs="0"
              maxOccurs="unbounded" type="xsd:double"/>
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>

<xsd:element name="getReportStatusRequest" type="xsd:string"/>
```

```
<xsd:element name="getReportStatusResponse">
   <xsd:complexType>
      <xsd:sequence>
          <xsd:element name="completed" minOccurs="1"
               maxOccurs="1" type="xsd:boolean"/>
          <xsd:element name="status" minOccurs="1"
               maxOccurs="1" type="xsd:string"/>
          <xsd:element name="sites" minOccurs="0"
               maxOccurs="unbounded" type="xsd:string"/>
          <xsd:element name="tests" minOccurs="0"
               maxOccurs="unbounded" type="xsd:string"/>
          <xsd:element name="results" minOccurs="0"
               maxOccurs="unbounded" type="xsd:double"/>
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>

<xsd:element name="orderTestsRequest">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="site" minOccurs="1"
              maxOccurs="1" type="xsd:string"/>
         <xsd:element name="tests" minOccurs="1"
              maxOccurs="31" type="xsd:string"/>
         <xsd:element name="comments" minOccurs="1"
              maxOccurs="1" type="xsd:string"/>
         <xsd:element name="samples" minOccurs="1"
              maxOccurs="1" type="xsd:int"/>
      </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

```
<xsd:element name="orderTestsResponse">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="oldTests" minOccurs="0"
              maxOccurs="31" type="xsd:string"/>
         <xsd:element name="newTests" minOccurs="0"
              maxOccurs="31" type="xsd:string"/>
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>

<xsd:element name="getReportRequest">
   <xsd:complexType/>
</xsd:element>

<xsd:element name="getReportResponse">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="completed" minOccurs="1"
              maxOccurs="1" type="xsd:boolean"/>
         <xsd:element name="status" minOccurs="1"
              maxOccurs="1" type="xsd:string"/>
         <xsd:element name="sites" minOccurs="0"
              maxOccurs="unbounded" type="xsd:string"/>
         <xsd:element name="tests" minOccurs="0"
              maxOccurs="unbounded" type="xsd:string"/>
         <xsd:element name="results" minOccurs="0"
              maxOccurs="unbounded" type="xsd:double"/>
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>
```

```
<xsd:element name="getPreviousRequest">
   <xsd:complexType/>
</xsd:element>

<xsd:element name="getPreviousResponse">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="oldTests" minOccurs="0"
              maxOccurs="31" type="xsd:string"/>
         <xsd:element name="newTests" minOccurs="0"
              maxOccurs="31" type="xsd:string"/>
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>

<xsd:element name="confirmTestsRequest" type="xsd:boolean"/>

<xsd:element name="confirmTestsResponse">
   <xsd:complexType/>
</xsd:element>

  <xsd:element name="Site" type="xsd:int"/>
  <xsd:element name="Tests" type="xsd:int"/>
  <xsd:element name="Client" type="xsd:int"/>
  <xsd:element name="Area" type="xsd:int"/>
  <xsd:element name="ReportStatus" type="xsd:string"/>
  <xsd:element name="Comments" type="xsd:string"/>
  <xsd:element name="Samples" type="xsd:int"/>
  <xsd:element name="ChemSearchResourceProperties">
    <xsd:complexType>
        <xsd:sequence>
```

```
                <xsd:element minOccurs="1" maxOccurs="1" ref="tns:Client"/>
                <xsd:element minOccurs="1" maxOccurs="1" ref="tns:Area"/>
                <xsd:element minOccurs="1" maxOccurs="1" ref="tns:Site"/>
                <xsd:element minOccurs="1" maxOccurs="31" ref="tns:Tests"/>
                <xsd:element minOccurs="1" maxOccurs="1" ref="tns:ReportStatus"/>
                <xsd:element minOccurs="1" maxOccurs="1" ref="tns:Comments"/>
                <xsd:element minOccurs="1" maxOccurs="1" ref="tns:Samples"/>
             </xsd:sequence>
          </xsd:complexType>
       </xsd:element>
     </xsd:schema>
   </wsdl:types>
   <wsdl:message name="CheckUserInputMessage">
      <wsdl:part element="tns:checkUserRequest"
           name="checkUserRequest"/>
   </wsdl:message>
   <wsdl:message name="CheckUserOutputMessage">
<wsdl:part element="tns:checkUserResponse"
          name="checkUserResponse"/>
   </wsdl:message>
   <wsdl:message name="GetAreasInputMessage">
     <wsdl:part element="tns:getAreasRequest"
          name="getAreasRequest"/>
   </wsdl:message>
   <wsdl:message name="GetAreasOutputMessage">
     <wsdl:part element="tns:getAreasResponse"
          name="getAreasResponse"/>
   </wsdl:message>
   <wsdl:message name="GetSitesInputMessage">
      <wsdl:part element="tns:getSitesRequest"
           name="getSitesRequest"/>
   </wsdl:message>
```

```
    <wsdl:message name="GetSitesOutputMessage">
       <wsdl:part element="tns:getSitesResponse"
             name="getSitesResponse"/>
    </wsdl:message>
    <wsdl:message name="GetTestsInputMessage">
       <wsdl:part element="tns:getTestsRequest"
             name="getTestsRequest"/>
    </wsdl:message>
    <wsdl:message name="GetTestsOutputMessage">
       <wsdl:part element="tns:getTestsResponse"
             name="getTestsResponse"/>
    </wsdl:message>
<wsdl:message name="ResultsTestsInputMessage">
       <wsdl:part element="tns:resultsTestsRequest"
             name="resultsTestsRequest"/>
    </wsdl:message>
    <wsdl:message name="ResultsTestsOutputMessage">
       <wsdl:part element="tns:resultsTestsResponse"
             name="resultsTestsResponse"/>
    </wsdl:message>
<wsdl:message name="ResultsSitesInputMessage">
       <wsdl:part element="tns:resultsSitesRequest"
             name="resultsSitesRequest"/>
    </wsdl:message>
<wsdl:message name="GetFinalResultInputMessage">
 <wsdl:part element="tns:getFinalResultRequest"
          name="getFinalResultRequest"/>
    </wsdl:message>
    <wsdl:message name="GetFinalResultOutputMessage">
       <wsdl:part element="tns:getFinalResultResponse"
             name="getFinalResultResponse"/>
    </wsdl:message>
    <wsdl:message name="ResultsSitesOutputMessage">
       <wsdl:part element="tns:resultsSitesResponse"
             name="resultsSitesResponse"/>
    </wsdl:message>
<wsdl:message name="ResultsAreasInputMessage">
 <wsdl:part element="tns:resultsAreasRequest"
           name="resultsAreasRequest"/>
    </wsdl:message>
    <wsdl:message name="ResultsAreasOutputMessage">
       <wsdl:part element="tns:resultsAreasResponse"
             name="resultsAreasResponse"/>
    </wsdl:message>
    <wsdl:message name="GetSiteResultsInputMessage">
       <wsdl:part element="tns:getSiteResultsRequest"
             name="getSiteResultsRequest"/>
    </wsdl:message>
    <wsdl:message name="GetSiteResultsOutputMessage">
       <wsdl:part element="tns:getSiteResultsResponse"
             name="getSiteResultsResponse"/>
    </wsdl:message>
    <wsdl:message name="GetTestResultsInputMessage">
       <wsdl:part element="tns:getTestResultsRequest"
             name="getTestResultsRequest"/>
    </wsdl:message>
    <wsdl:message name="GetTestResultsOutputMessage">
 <wsdl:part element="tns:getTestResultsResponse"
           name="getTestResultsResponse"/>
    </wsdl:message>
    <wsdl:message name="GetReportStatusInputMessage">
       <wsdl:part element="tns:getReportStatusRequest"
             name="getReportStatusRequest"/>
    </wsdl:message>
    <wsdl:message name="GetReportStatusOutputMessage">
       <wsdl:part element="tns:getReportStatusResponse"
             name="getReportStatusResponse"/>
    </wsdl:message>
<wsdl:message name="OrderTestsInputMessage">
       <wsdl:part element="tns:orderTestsRequest"
             name="orderTestsRequest"/>
    </wsdl:message>
    <wsdl:message name="OrderTestsOutputMessage">
       <wsdl:part element="tns:orderTestsResponse"
             name="response"/>
    </wsdl:message>
    <wsdl:message name="GetPreviousInputMessage">
       <wsdl:part element="tns:getPreviousRequest"
             name="getPreviousRequest"/>
    </wsdl:message>
    <wsdl:message name="GetPreviousOutputMessage">
       <wsdl:part element="tns:getPreviousResponse"
             name="response"/>
    </wsdl:message>
    <wsdl:message name="ConfirmTestsInputMessage">
       <wsdl:part element="tns:confirmTestsRequest"
             name="confirmTestsRequest"/>
    </wsdl:message>
    <wsdl:message name="ConfirmTestsOutputMessage">
       <wsdl:part element="tns:confirmTestsResponse"
             name="response"/>
    </wsdl:message>
<wsdl:message name="GetReportInputMessage">
       <wsdl:part element="tns:getReportRequest"
             name="getReportRequest"/>
    </wsdl:message>
```

```
   <wsdl:message name="GetReportOutputMessage">
      <wsdl:part element="tns:getReportResponse"
            name="response"/>
   </wsdl:message>
   <wsdl:portType name="ChemSearchServicePortType"
         wsrp:ResourceProperties="tns:ChemSearchResourceProperties">
      <wsdl:operation name="GetResourceProperty">
         <wsdl:input name="GetResourcePropertyRequest"
             message="wsrpw:GetResourcePropertyRequest"/>
          <wsdl:output name="GetResourcePropertyResponse"
             message="wsrpw:GetResourcePropertyResponse"/>
          <wsdl:fault name="ResourceUnknownFault"
             message="wsrpw:ResourceUnknownFault"/>
          <wsdl:fault name="InvalidResourcePropertyQNameFault"
             message="wsrpw:InvalidResourcePropertyQNameFault"/>
      </wsdl:operation>
      <wsdl:operation name="checkUser">
         <wsdl:input message="tns:CheckUserInputMessage"/>
         <wsdl:output message="tns:CheckUserOutputMessage"/>
      </wsdl:operation>
      <wsdl:operation name="getAreas">
         <wsdl:input message="tns:GetAreasInputMessage"/>
         <wsdl:output message="tns:GetAreasOutputMessage"/>
</wsdl:operation>
<wsdl:operation name="getSites">
      <wsdl:input message="tns:GetSitesInputMessage"/>
      <wsdl:output message="tns:GetSitesOutputMessage"/>
</wsdl:operation>
<wsdl:operation name="getTests">
      <wsdl:input message="tns:GetTestsInputMessage"/>
      <wsdl:output message="tns:GetTestsOutputMessage"/>
</wsdl:operation>
```

```
<wsdl:operation name="getTestResults">
      <wsdl:input message="tns:GetTestResultsInputMessage"/>
      <wsdl:output message="tns:GetTestResultsOutputMessage"/>
</wsdl:operation>
<wsdl:operation name="getSiteResults">
      <wsdl:input message="tns:GetSiteResultsInputMessage"/>
      <wsdl:output message="tns:GetSiteResultsOutputMessage"/>
</wsdl:operation>
<wsdl:operation name="getReportStatus">
      <wsdl:input message="tns:GetReportStatusInputMessage"/>
      <wsdl:output message="tns:GetReportStatusOutputMessage"/>
</wsdl:operation>
<wsdl:operation name="orderTests">
      <wsdl:input message="tns:OrderTestsInputMessage"/>
      <wsdl:output message="tns:OrderTestsOutputMessage"/>
</wsdl:operation>
<wsdl:operation name="getPrevious">
      <wsdl:input message="tns:GetPreviousInputMessage"/>
      <wsdl:output message="tns:GetPreviousOutputMessage"/>
</wsdl:operation>
<wsdl:operation name="confirmTests">
      <wsdl:input message="tns:ConfirmTestsInputMessage"/>
      <wsdl:output message="tns:ConfirmTestsOutputMessage"/>
</wsdl:operation>
    <wsdl:operation name="getReport">
      <wsdl:input message="tns:GetReportInputMessage"/>
      <wsdl:output message="tns:GetReportOutputMessage"/>
    </wsdl:operation>
<wsdl:operation name="resultsSites">
      <wsdl:input message="tns:ResultsSitesInputMessage"/>
      <wsdl:output message="tns:ResultsSitesOutputMessage"/>
    </wsdl:operation>
```

```
<wsdl:operation name="resultsTests">
      <wsdl:input message="tns:ResultsTestsInputMessage"/>
      <wsdl:output message="tns:ResultsTestsOutputMessage"/>
    </wsdl:operation>
<wsdl:operation name="getFinalResult">
      <wsdl:input message="tns:GetFinalResultInputMessage"/>
      <wsdl:output message="tns:GetFinalResultOutputMessage"/>
</wsdl:operation>
<wsdl:operation name="resultsAreas">
      <wsdl:input message="tns:ResultsAreasInputMessage"/>
      <wsdl:output message="tns:ResultsAreasOutputMessage"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="ChemSearchServiceBinding"
        type="tns:ChemSearchServicePortType">
      <soap:binding style="document"
            transport="http://schemas.xmlsoap.org/soap/http"/>
      <wsdl:operation name="GetResourceProperty">
         <wsdl:input name="GetResourcePropertyRequest">
            <soap:body use="literal"/>
        </wsdl:input>
        <wsdl:output name="GetResourcePropertyResponse">
          <soap:body use="literal"/>
        </wsdl:output>
        <wsdl:fault name="InvalidResourcePropertyQNameFault">
          <soap:fault name=
                "InvalidResourcePropertyQNameFault" use="literal"/>
        </wsdl:fault>
        <wsdl:fault name="ResourceUnknownFault">
           <soap:fault name="ResourceUnknownFault" use="literal"/>
        </wsdl:fault>
      </wsdl:operation>
```

```
<wsdl:operation name="checkUser">
   <soap:operation style="document"/>
   <wsdl:input>
      <soap:body use="literal"/>
   </wsdl:input>
   <wsdl:output>
      <soap:body use="literal"/>
   </wsdl:output>
</wsdl:operation>
<wsdl:operation name="getAreas">
   <soap:operation style="document"/>
   <wsdl:input>
      <soap:body use="literal"/>
   </wsdl:input>
   <wsdl:output>
      <soap:body use="literal"/>
   </wsdl:output>
</wsdl:operation>
<wsdl:operation name="getSites">
   <soap:operation style="document"/>
   <wsdl:input>
      <soap:body use="literal"/>
   </wsdl:input>
   <wsdl:output>
      <soap:body use="literal"/>
   </wsdl:output>
</wsdl:operation>
<wsdl:operation name="getTests">
   <soap:operation style="document"/>
   <wsdl:input>
      <soap:body use="literal"/>
   </wsdl:input>
```

```
      <wsdl:output>
         <soap:body use="literal"/>
      </wsdl:output>
   </wsdl:operation>
   <wsdl:operation name="getSiteResults">
      <soap:operation style="document"/>
      <wsdl:input>
         <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
         <soap:body use="literal"/>
      </wsdl:output>
   </wsdl:operation>
   <wsdl:operation name="getTestResults">
      <soap:operation style="document"/>
      <wsdl:input>
         <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
         <soap:body use="literal"/>
      </wsdl:output>
   </wsdl:operation>
   <wsdl:operation name="orderTests">
      <soap:operation style="document"/>
      <wsdl:input>
         <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
         <soap:body use="literal"/>
      </wsdl:output>
   </wsdl:operation>
   <wsdl:operation name="getPrevious">
```

```
    <soap:operation style="document"/>
    <wsdl:input>
       <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
       <soap:body use="literal"/>
    </wsdl:output>
 </wsdl:operation>
 <wsdl:operation name="confirmTests">
    <soap:operation style="document"/>
    <wsdl:input>
       <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
       <soap:body use="literal"/>
    </wsdl:output>
 </wsdl:operation>
 <wsdl:operation name="getReportStatus">
    <soap:operation style="document"/>
    <wsdl:input>
       <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
       <soap:body use="literal"/>
    </wsdl:output>
 </wsdl:operation>
 <wsdl:operation name="getReport">
    <soap:operation style="document"/>
    <wsdl:input>
       <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
```

```
        <soap:body use="literal"/>
     </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="resultsAreas">
     <soap:operation style="document"/>
     <wsdl:input>
        <soap:body use="literal"/>
     </wsdl:input>
     <wsdl:output>
        <soap:body use="literal"/>
     </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="resultsSites">
     <soap:operation style="document"/>
     <wsdl:input>
        <soap:body use="literal"/>
     </wsdl:input>
     <wsdl:output>
        <soap:body use="literal"/>
     </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="resultsTests">
     <soap:operation style="document"/>
     <wsdl:input>
        <soap:body use="literal"/>
     </wsdl:input>
     <wsdl:output>
        <soap:body use="literal"/>
     </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="getFinalResult">
     <soap:operation style="document"/>
```

```
            <wsdl:input>
               <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
               <soap:body use="literal"/>
            </wsdl:output>
         </wsdl:operation>
     </wsdl:binding>
     <wsdl:service name="ChemSearchService">
         <wsdl:port binding="tns:ChemSearchServiceBinding" name="ChemSearch">
             <soap:address location=
                   "http://localhost:8080/wsrf/services/ChemSearch"/>
         </wsdl:port>
     </wsdl:service>
</wsdl:definitions>
```

C FactoryService Code

The functionality added to the FactoryService class is shown below.

```
public class ChemSearchFactoryService
   extends AbstractChemSearchFactoryService
   implements ChemSearchFactoryCustomOperationsPortType {

   public CreateResourceResponseDocument
      CreateResource(CreateResourceRequestDocument requestDoc ) {
      CreateResourceResponseDocument responseDocument =
          CreateResourceResponseDocument.Factory.newInstance();
      CreateResourceResponseDocument.CreateResourceResponse response =
          responseDocument.addNewCreateResourceResponse();
      try {
            Context initialContext = new InitialContext();
            ChemSearchHome home =
               (ChemSearchHome)initialContext.lookup(ChemSearchHome.HOME_LOCATION);
            ChemSearchResource resource =
               (ChemSearchResource)home.findResource(getResourceContext(),
                 requestDoc.getCreateResourceRequest());
            EndpointReferenceType endPointRefType = resource.getEpr();
            response.setEndpointReference(endPointRefType);
        } catch(Exception ex) {
              ex.printStackTrace();
        }
        return responseDocument;
    }
```

D ChemSearchHome Code

The functionality added to the ChemSearchHome class is shown below.

```
public class ChemSearchHome
        extends AbstractResourceHome
        implements Serializable {
    private int chemResourceID = 0;
    public static final QName SERVICE_NAME = javax.xml.namespace.QName.valueOf(
        "{chemsearch/ChemSearchService}ChemSearch");

    public Resource findResource(ResourceContext resourceContext, String site }
            throws ResourceException,
            ResourceContextException,
            ResourceUnknownException {
            ResourceKey key = resourceContext.getResourceKey();
            Resource resource = null;
            try{
```

```
            Map resources = super.m_resources;
            for (int i = 0; i < resources.size(); i++) {
                Iterator it = resources.values().iterator();
                while (it.hasNext()) {
                      ChemSearchResource r = (ChemSearchResource)it.next();
                      if (r.getSite().equals(site)) {
                        resource = r;
                      }
                }
            }
            if (resource == null) {
              throw new ResourceException();
            }
        } catch ( ResourceException re ) {
                resource = createChemSearchResource(resourceContext, site);
        }
        return resource;
}

publicChemSearchResource
        createChemSearchResource(ResourceContext resourceContext, String site) {
        ++chemResourceID;
        int chemID = chemResourceID;
        SimpleTypeResourceKey key = createResourceKey(chemID);
        ChemSearchResource chemResource =
              new ChemSearchResource(site,
                    getEprForResource(key, resourceContext.getBaseURL()));
        chemResource.init();
        add(key, chemResource);
        return chemResource;
}
publicEndpointReferenceType getEprForResource(ResourceKey key, String serviceUrl) {
        EndpointReferenceDocument eprDoc =
                EndpointReferenceDocument.Factory.newInstance();
        EndpointReferenceType epr = eprDoc.addNewEndpointReference();
        AttributedURI address = epr.addNewAddress();
        address.setStringValue(serviceUrl + "/" +
               SERVICE_NAME.getLocalPart());
        ReferencePropertiesType refProps =
               epr.addNewReferenceProperties();
        XmlObject xmlObject =
                XmlBeanUtils.addChildElement(refProps, key.getName());
        XmlBeanUtils.setValue(xmlObject,
                key.getValue().toString());
        return epr;
    }

    privateSimpleTypeResourceKey createResourceKey(int chemID) {
        SimpleTypeResourceKey key =
              new SimpleTypeResourceKey(QName.valueOf(
                  "{chemsearch/ChemSearchService}ResourceIdentifier" ),
                  "ChemSearch" + Integer.toString(chemID));
        return key;
    }
}
```

PART B

RESEARCH ARTICLE

Using Web Service Technology to Model the Business Processes of a Chemical Analysis Laboratory

Katrina Porteous, Stephen Cranefield, Martin Purvis

Department of Information Science
University of Otago,
PO Box 56, Dunedin, New Zealand

Abstract

The use of web services technology is becoming more widespread with many businesses wishing to provide services to their clients over the Internet. Although web services technology is currently the best method for providing services remotely over the Internet it does not provide any concept of state or any way to model workflow. This has resulted in new specifications to deal with these issues, namely WSRF and BPEL4WS. In order to effectively model certain types of workflow, such as that of the ChemSearch laboratory, there needs to be a method for integrating BPEL4WS and WSRF, so that stateful workflow can be modelled. This research proposed a method of integration using a proxy service to enable BPEL4WS to support WSRF that was evaluated by modelling the workflow of a chemical analysis laboratory. The results showed that the method enabled BPEL4WS to work with WSRF and provided a mechanism that ensured a client always had a valid endpoint reference to the WSRF web service. This research also raised some problems with the specifications and tools used to implement this method that would need to be resolved to ensure the widespread adoption and standardisation of these specifications.

Keywords: WSRF, BPEL4WS, workflow, state

1 Introduction

There is a growing interest in the scientific community to be able to share resources, data and research. This has introduced the concept of e-science [1], which involves using grid service technology, Hunter *et al.* (2004), to solve these issues. Grid services are based on web services standards that have been extended for use in e-science applications. Grid services provide middleware that handles security, the ability to locate and invoke the services made available, resource management, etc. Web services are a form of middleware that provide a way of describing, finding and invoking services remotely over the Internet using XML-based standards such as SOAP, WSDL (Web Services Description Language) and UDDI (Universal Description, Discovery and Integration), Johnson (2005).

Web services technology is not currently widely adopted, due to the lack of standardisation and deployed services available. Although, there are a number of useful specifications that have been proposed, they tend to be stand-alone and there is no mechanism available for combining the specifications so that they can be used together. There seems to be an underlying assumption that because they are all based on web services technology they should automatically be compatible.

Therefore, this research focuses on combining two web services specifications to determine whether or not they are compatible. Furthermore, the two particular specifications chosen, namely WSRF (WS-Resource Framework) and BPEL4WS (Business Process Execution Language for Web Services), will enable the modelling of stateful workflow such as that required by analytical laboratories. In this case the ChemSearch laboratory was chosen, so that the method of integration could be tested by modelling this laboratories workflow. In this particular case, the modelling of state is needed to enable the clients of the laboratory to obtain the results for a particular test or the status of their report. WSRF will be used to model the state of the report, while BPEL4WS will enable the workflow of the laboratory to be modelled. Although this particular laboratory was chosen, this method of integrating WSRF and BPEL4WS should generalise to any laboratory that requires stateful workflow to be modelled.

The details of this research, the method of integration and subsequent evaluation from modelling the workflow of the ChemSearch laboratory are discussed in subsequent sections of this article.

2 Background

2.1 Web Services

Web services are a form of middleware that uses XML-based standards such as SOAP, W3C (2003), WSDL (Web Services Description Language), W3C (2001), and UDDI (Universal Description, Discovery and Integration), UDDI (2004), to provide services remotely over the Internet, Johnson (2005). SOAP is a protocol for distributed environments that uses XML technologies as the message format for the information that is passed between two nodes in the connection W3C (2003). WSDL defines the services that will be offered over the network. Within a WSDL document a number of messages and operations can be defined. A message describes the data that will be transmitted and an operation describes the messages that will be sent and received upon invocation W3C (2001). UDDI defines a structure that describes the services that are available, who makes them available and the interfaces that can be used to access the provided services UDDI (2004). These standards provide the basis for web services technology that enables the construction of loosely-coupled distributed systems built on a service-oriented architecture Atkinson *et*

[1]E-Science - A term that applies to collaboration amongst scientists, where large amounts of data, resources, etc. are shared through the use of Internet-based technologies, Hunter *et al.* (2004)

al. (2005). According to the W3C (W3C (2004a)) a service-oriented architecture is "a set of components, which can be invoked, and whose interface descriptions can be published and discovered". A service-oriented architecture separates the interface definitions from their implementations, which means a service only needs to know the definition of the interface not how it is implemented. The use of a service-oriented architecture offers numerous benefits including well-defined interfaces Pasley (2005), platform independence and hides service implementation details from the developer W3C (2004c).

2.2 WSRF

WSRF introduces the concept of a stateful resource, which is a resource used by a web service that has associated state and is implemented as an XML document. It allows a web service to perform operations on data from a previous invocation of the web service. Examples of a stateful resource include a row of data in a database or a single file in a file system Foster *et al.* (2004), that allows state to be preserved between executions of a web service's operations.

The introduction of WSRF means that a web service can have state associated with it. Up until now a web service had no notion of state, but it was implied through the definitions of its interfaces Foster *et al.* (2004). This is because "WSDL is essentially stateless because the language is unaware of states between operations" Staab *et al.* (2003), p. 74. It is claimed that explicit definition of state leads to improved interoperability, simplification of service definitions and improved discovery, management and development tools Foster *et al.* (2004). It also alleviates the developer from coming up with their own implementation of state, which may be incompatible with other developers implementations. By standardising the way in which state is accessed it means web services that use state can access it without having to know how it was implemented. This is an important point especially to get two different specifications to work together.

Basically WSRF describes the association of a web service with a stateful resource, which is called a WS-Resource. A web service can locate a stateful resource through its unique identifier, which is encapsulated within an endpoint reference, defined as part of the WS-Addressing W3C (2004b) specification. This endpoint reference contains the address of the service, whose operations are to be invoked. If a stateful resource is involved in the implementation of the service, the resource identifier is included within the reference properties defined as part of the endpoint reference. This information is then encoded as part of the SOAP header within the message request that is sent to the service. The service can then extract this information to gain access to the required stateful resource. This provides a more specific way of referencing a resource from the web service, in a similar way to obtaining references to an object in object-oriented programming.

2.3 BPEL4WS

There have been a number of new specifications proposed in web services that describe the composition and orchestration of web services. One of these specifications BPEL4WS provides the means to model business processes and interactions Andrews *et al.* (2003). It allows workflows to be modelled that invoke web services, which are executed using a workflow engine. It is based on a number of XML specifications, namely WSDL 1.1, XML Schema 1.0, XPath 1.0 and WS-Addressing; the most important one being WSDL 1.1 Andrews *et al.* (2003). Since BPEL4WS is based on web services, a business process can either use services defined in an existing WSDL document or create new services as required. In addition a business process describes the behaviour of and the partners involved in the business process. BPEL4WS refers to services using the port type defined in the WSDL document rather than the actual service itself so that the business process can be reused in "multiple deployments of compatible services" Andrews *et al.* (2003), p. 15.

2.4 ChemSearch

ChemSearch is a small analytical laboratory that analyses samples sent to them by their clients. A client requests certain tests to be done on their samples by filling out an order form that is sent to the laboratory along with the samples to be tested. On receiving the order, the laboratory checks the order form against the samples received, to ensure that the wrong order form has not been put in with the samples, and also checks over the tests ordered. Frequently, clients request the same tests to be done for their samples. If there is a test requested that is not normally ordered, the laboratory will ring the client to confirm the tests to be done. Once the tests have been approved, the laboratory prioritises the tests based on the number of samples requiring the same test, the urgency of the test or sample and the lifetime of the samples. When a test has been completed the initial results are recorded, which can be used to determine the final result for a test. On completion of all tests ordered by the client, a copy of the report detailing all the results of the tests is sent to the client.

Typically, the process involves the client ringing or emailing to find out the results of a particular test or the status of their report. Ideally, the client should be able to access their results and the status of their reports without contacting the laboratory.

2.5 Integrating WSRF and BPEL4WS

The current research seems to be leading towards the integration of web and grid services to receive the benefits of both technologies, in particular the service-oriented architecture of web services which is thought to be an integral foundation of e-science Foster *et al.* (2002). In order for grid services to provide a complete solution to the e-science problem there needs to be a way to model workflow. Although there have been a number of specifications and tools proposed in both web and grid services, it would seem that BPEL4WS is the most promising candidate for modelling complex workflow. The use of a web services specification that supports the service-oriented architecture will be beneficial to grid services in the event of the integration of web and grid services. This has led to numerous proposals for suggesting enhancements to BPEL4WS for compatibility in the grid environment. This integration idea has also led to the possibility of combining grid services specifications, such as OGSI, Tuecke *et al.* (2003), and WSRF, with web services specifications, such as BPEL4WS for use in a grid environment. Although the integration is feasible, BPEL4WS restricts some functionality of grid services, but making changes to the specification will remove the advantages of using it in the first place.

The most common method for integrating WSRF and BPEL4WS suggested some form of intermediate mechanism for BPEL4WS to communicate with WSRF. This seems to be necessitated by the requirement that reference properties are added to the SOAP header of the message sent to the WS-Resource. Therefore, the implementation of a proxy service that performs this task will be the most likely candidate to integrate WSRF and BPEL4WS. The second problem will be addressing how the client maintains endpoint references and how it can get new endpoint references should the existing ones become lost or invalid. Currently, there seem to be no concrete ideas on how to address this problem, although hopefully the future implementation of WS-RenewableReferences will achieve this goal.

3 Tools

The tools used to implement this method and the subsequent on-line system were Cape Clear Orchestration Studio, CapeClear (2005), to model the BPEL4WS process and the WSRF project from Apache, Apache (2005), using the Tomcat server, Tomcat (2005), and Ant, Ant (2005), to compile and deploy the web services to Tomcat. The BPEL4WS tool was a plug-in for the Eclipse Integrated Development Environment, eclipse (2005).

4 Method

According to section 2 there is no proof that WSRF and BPEL4WS cannot be integrated together. Therefore, it was firstly determined whether BPEL4WS and WSRF could be combined together. If this proved to be unsuccessful, an intermediary mechanism would be used as suggested in section 2.

This method, see section 6.2, of integrating WSRF and BPEL4WS was then tested by modelling the workflow of the ChemSearch laboratory. The laboratory needed to provide reports to their clients listing the results for tests that had been requested for their samples. Frequently, their clients would either ring or email them asking the results for a particular test or report. Ideally, the lab would like to provide the results of the tests as they have been completed and an update on the status of the clients reports. In order to model this scenario, some part of the workflow needs to remember what tests have been completed and what the results are for those tests. This is where WSRF comes in, where it provides a mechanism by which the tests and results plus the status of the report can be modelled as state.

This brought to light a number of problems with the specifications and tools used. These problems are presented in section 5 and discussed in detail in section 6.

5 Results

This section outlines the results of integrating WSRF and BPEL4WS, for further details see section 6.

It was determined after some investigation, see section 6.1, that an intermediary mechanism would be needed for BPEL4WS to be combined with WSRF. This meant that a third web service was needed to add the reference properties to the header of the SOAP message to enable the services provided by the WS-Resource to be invoked. Due to the nature of WSRF, there are normally two WSRF web services: a factory service to create new instances of the WS-Resource and the WS-Resource itself. This is shown in figure 1.
The method for integrating WSRF and BPEL4WS was determined by the criteria given below.

- The method of integration enables BPEL4WS to be combined with WSRF - Y/N
- The method of integration can be generalised for use in any workflow that needs WSRF - Y/N
- Does the method of integration provide a means of storing endpoint references - Y/N
- Does the method of integration cater for easy retrieval of endpoint references - Y/N
- Can the method of integration be applied to other WSRF and BPEL4WS tools - Y/N
- Does the method of integration scale well, if applied to large, complex workflows - Y/N
(i.e. workflows that may contain a number of WS-Resources)

A summary of the results obtained from this integration are shown in tables 1 and 2.
Note: Scalability refers to how much extra work is required to adapt the method of integration to incorporate multiple WS-Resources. It does not pertain to the creation of WS-Resources or work that would normally need to be done in the process of creating the workflow. It only refers to the work that has to be done over and above normal workflow creation in order to implement the method of integration.

6 Discussion

6.1 Alternatives

The BPEL4WS specification states that an endpoint reference defined by WS-Addressing W3C (2004b) can be assigned dynamically to a partner link, which in BPEL4WS terms is basically a web service. While the current Cape Clear documentation stipulated the support for assigning endpoint references this was not the case in practise. We discovered that in fact the current version of Cape Clear Orchestration Studio did not support the dynamic assignment of end point references. Neither the BPEL4WS specification nor Cape Clear documentation specifically mentions whether reference properties included as part of the endpoint reference would be added to the SOAP header of the message. The specification only goes as far as acknowledging that data can be placed in the header or in the body of a SOAP message. it does not precisely state the placement of reference properties. Based on my experience with WS-Addressing stubs, it is very unlikely that reference properties would be copied into the SOAP header, since they are an optional part of an endpoint reference. This is backed up by research into this area, which also suggested that it was unlikely to include reference properties in SOAP message headers Slomiski (2005).

The WS-Addressing stubs were used to enable the system developed in eclipse to send and receive XML

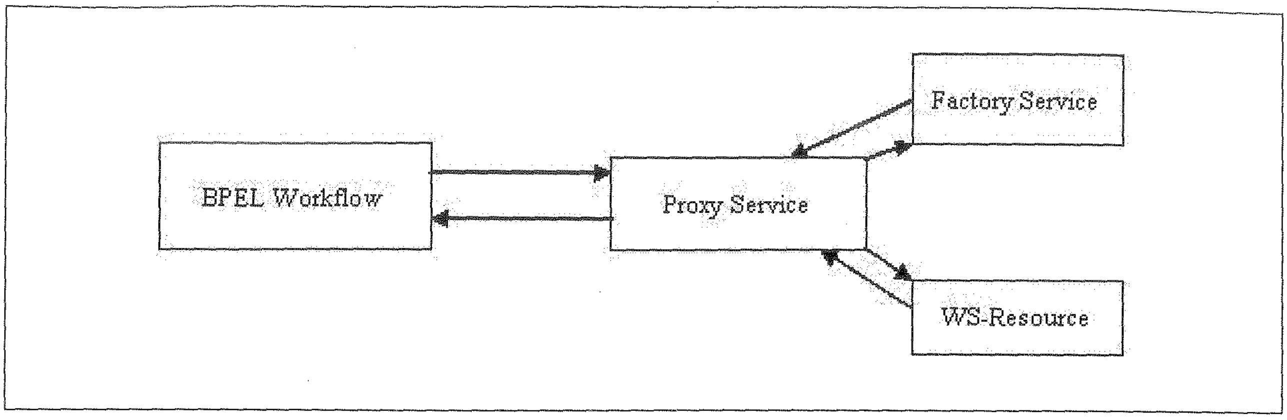


Figure 1: The Interaction of the Workflow with the Three WSRF Web Services

Table 1: The evaluation of the method for combining WSRF with BPEL4WS

Criterion	Result
The method enables BPEL4WS to be combined with WSRF	Y
The method can be generalised for use in any workflow that needs WSRF	Y
Does the method provide a means of storing endpoint references	Y
Does the method cater for easy retrieval of endpoint references	Y
Can the method be applied to other WSRF and BPEL4WS tools	Y
Does the method scale well, if applied to large, complex workflows	N

messages and did not by default add reference properties to the SOAP header. It does extract the reference properties from the endpoint reference passed to it, but these are never set when the call object is created, so never get added to the SOAP header. Perhaps this is because no checking is done to make sure the reference properties are not null. I am not sure what effect it would have on the message if a null value was added to the SOAP header, but I do know that having null values in the body of the message causes a null pointer exception when the message is deserialised - the process of extracting application data from the XML message.

Since WSRF only checks for the existence of reference properties when there is an entry in the jndi-config.xml file for the service then requiring a value to exist in the SOAP header would not cause any problems if the value did not uniquely identify a WS-Resource and the call was not invoking a WS-Resource instance.

Therefore, it would go a long way to aid the integration process if there were some underlying code that checked for the existence of reference properties in the endpoint reference. If there is an entry for the reference properties then these should automatically be added to the SOAP header. From my experience in trying to integrate WSRF and BPEL4WS, the BPEL4WS process needs to do some form of checking for the existence of reference properties for BPEL4WS to truly support WSRF web services. If by default these reference properties were added to the SOAP header it would alleviate the need for BPEL4WS to know when the web service it is accessing is a WSRF one or not (assuming of course that BPEL4WS is communicating directly with the WS-Resource and not some intermediary mechanism).

Even if the Cape Clear Orchestration Studio had supported the dynamic assignment of endpoint references and these were included in the SOAP header by default, this solution would not have worked. This is because the Cape Clear tool threw an exception every time it tried to invoke a WS-Resource web service. After some investigation and looking at the error messages produced by the tool, I discovered that it was the imported WSDL definitions needed to declare the web service as being WSRF that were causing the problem. Further investigation would be needed to determine whether other BPEL4WS tools had the same problem, or whether this was just an issue with the Cape Clear tool. Since a specification, in this case BPEL4WS, is usually just a blue print for how workflow could be designed and implemented, it seems unlikely that this is an issue stemming from the specification itself. It seems more likely that it is the way in which the Cape Clear tool validates the BPEL4WS workflow.

6.2 Method of Integration

The results given in table 1 show that the main problem with this method is scalability. This is because in order to model state there needs to be three web services. Firstly, a factory service to create a WSRF web service, a proxy web service that adds the reference properties to the SOAP header of the message and the WSRF web service that keeps track of the state. Since WSDL files that define the services provided by these web services are generally rather large, reusing the proxy and factory services to handle multiple stateful web services would be difficult to manage and maintain. Therefore, the only other option is to create a new factory and proxy service for each stateful web service. This is shown in figure 2. In the case of a large workflow that would require a considerable amount of state to be modelled, this method of integrating WSRF and BPEL4WS would cause extra work to be done to implement it since the proxy and WSRF WSDL files have to define the same input and output parameters.

The main advantage of this method is that the BPEL4WS workflow can treat the proxy service like a normal web service, since it does not contain any of the signatures that define a WSRF web service. Since WSRF usually has two web services, a factory service that creates the WS-Resource and the WS-Resource itself, the use of a proxy service

Table 2: Summary of the problems encountered during the integration of WSRF and BPEL4WS

Problem	Specification/Tool
Reference properties do not get copied into the SOAP header of a message	BPEL4WS specification, WSRF tool
Endpoint references cannot be dynamically assigned to a web service	Cape Clear tool
Lack of testability of the workflow	BPEL4WS specification
No client-side stubs	Apache WSRF tool
No standard mechanism for storing and retrieving endpoint references	WSRF specification
Usability of BPEL4WS	BPEL4WS specification
Lack of compatibility with WSRF	BPEL4WS specification, Cape Clear tool
Assignment of variables	BPEL4WS specification
Problems with SOAP messages	Cape Clear tool
Changes to WSDL require workflow to be recreated	Cape Clear tool
Poor documentation	Cape Clear tool, WSRF tool

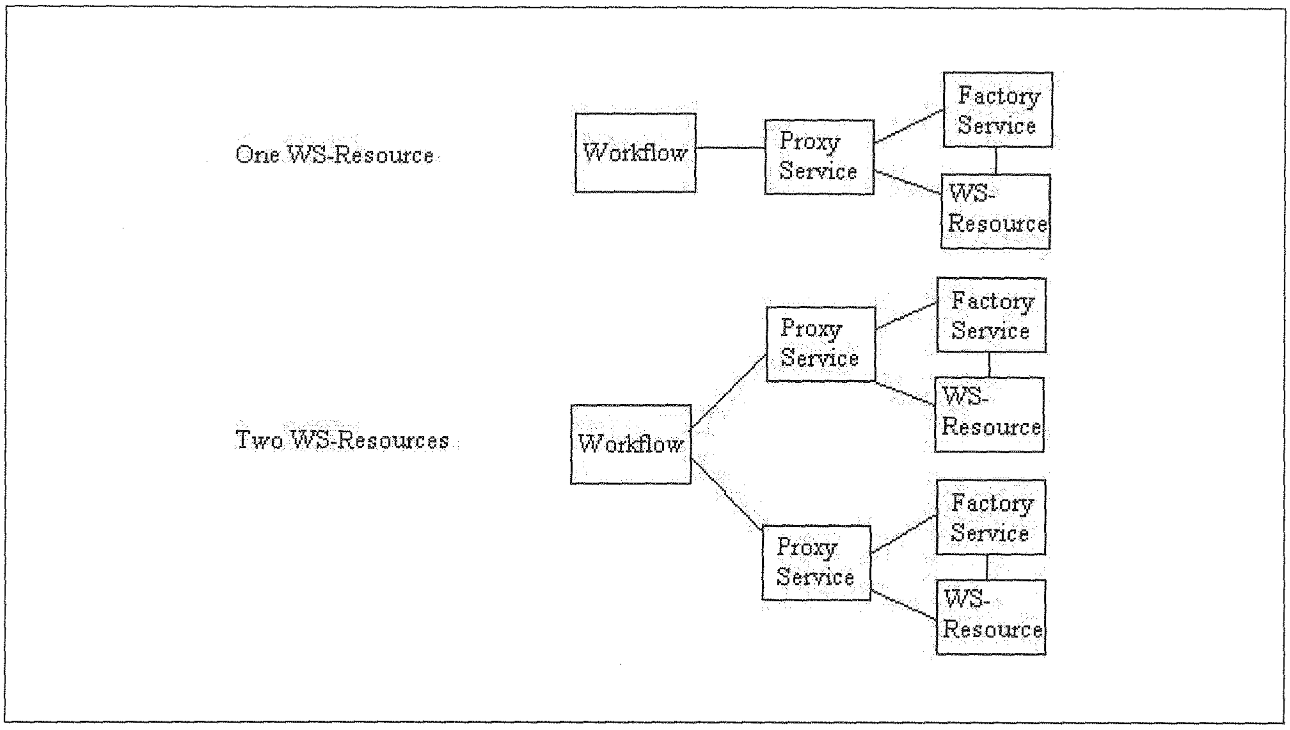


Figure 2: The Scalability Problem

alleviates the workflow from knowing about two web services; instead it only has to interact with one

This method also provided a solution to the problem of retrieving lost or invalid endpoint references. Basically, a method called *find resource* was added to the WS-Resource home class, which in this case was called *ChemSearchHome*. This method takes in the resource context, which enables the class to find a resource using the value of the reference properties passed in the request message header, and a string. This string uniquely identifies an endpoint reference, since in the case of the laboratory a client can have many reports, therefore there needs to be a unique identifier to distinguish between the different reports the client might have. This method obtains a reference to the map maintained by the cache that holds all the WS-Resource instances. Each resource class, in this case it is *ChemSearchResource*, contains a reference to the endpoint reference for the resource and to the site (the string). The method can then loop through the map comparing the string parameter to the resource's site value. If it finds a resource in the map that contains that site, it returns that resource. If not, a new resource exception is thrown and subsequently caught by the same method. At this point it calls the *createChemSearchResource*, which also takes in as parameters the string and resource context. This creates a new instance of the *ChemSearchResource* and an endpoint reference for that resource. This newly created resource is then returned to the factory. The factory then uses the resource to find the endpoint reference, which it returns to the proxy service to store for subsequent calls to that resource.

Figure 3, shows in more general terms the interaction of the client and the laboratory with the workflow and web services used by the workflow. The client and laboratory have separate workflows, however when a client places an order the factory service creates a new instance of the WS-Resource for that order (as part of the client's workflow). This means each order can be uniquely identified. When the laboratory has done some of the tests for that order, the laboratory accesses the client's workflow, which invokes the factory service. This uses the name of the client and the name of the site the test is being performed on, to check if an endpoint reference exists for that particular combination. If an endpoint reference does exist the factory service returns the endpoint reference to the proxy service, which stores it for a subsequent call to the WS-Resource. If not, the factory service creates a new WS-Resource and returns this to the proxy service to store. At this point the workflow invokes a method on the WS-Resource to update the status of the client's report and the tests that have been completed along with the results. This means the next time the client invokes the workflow to enquire about the status of their report, the workflow invokes the factory service to locate the endpoint reference for the client and the site for which the tests were performed. This endpoint reference is passed in the header of the SOAP message to the WS-Resource, which can then return the status of the report and the results for any tests that have been done.

Other than the problem of scalability, the method was successful in enabling the workflow of the laboratory to be modelled, however it did raise some problems with the specifications and tools used, to be discussed in section 6.3.

6.3 Tools and Specifications

The Cape Clear tool was very inflexible, due to the nature of BPEL4WS, and had problems with imported WSDL definitions. It would be interesting to know whether it is just WSRF imports that it could not resolve or whether the tool cannot deal with imported WSDL definitions that it does not already support. Cape Clear was able to resolve WS-Addressing imports defined in the factory service, but this could be because Cape Clear supports WS-Addressing. One of the other problems was that if the underlying WSDL files change, the workflow has to be completely re-engineered. An argument for this is that the WSDL should be properly defined before the workflow is developed. The problem with this argument is that some of these problems, such as the tool being unable to resolve the imported WSDL definitions, do not become apparent until you come to test the workflow. Perhaps some of these problems will be resolved once BPEL4WS has been standardised and more widely adopted.

Alternatively, defining a mechanism where the workflow could be refreshed so that any changes made to the underlying WSDL would be reflected in the workflow automatically, would avoid this problem.

The poor quality of the documentation was unacceptable for a commercially available tool and it would seem from this analysis that the Cape Clear tool requires additional work to be done, before it could be recommended as being suitable for modelling BPEL4WS workflow.

The WSRF Apache project needs to provide client-side stubs for WSRF and not just server-side to support applications that need to invoke WSRF web services via the workflow. This meant the generated code had to be modified to enable the reference properties to be added to the SOAP header, while this wasn't a difficult task to do it meant that every time a WSDL file was created, this code had to be added to the stubs.

Given that endpoint references are part of the fundamental framework of WSRF it would benefit WSRF if its implementations could provide a mechanism for automatically adding reference properties to SOAP headers.
This would require checking if the reference properties were not null before adding them in case sending null values in the header would cause problems. Since the stubs from the Apache project automatically create all the code for you to create and send messages, surely it wouldn't be too difficult to add another section of code to deal with endpoint references.
Adding this kind of support would go a long way to increasing the adoption of WSRF technologies, since it is quite off-putting to use a tool where you have to hack the generated code (and there is usually an underlying assumption that if there is something wrong with your project it is unlikely to be the generated code, especially if the project in question is not in a beta release).

Since the Apache WSRF project has only just been released, maybe there will be support for endpoint references in a future release once the majority of the bugs have been removed.

In terms of the specifications, WSRF seems a worthwhile contribution to the web services community and provides a standard means of accessing and modelling state that proved very useful in modelling the workflow of the ChemSearch laboratory. However, it does not provide as yet a standard mechanism to handle lost or invalid endpoint references and does not provide a standard mechanism by which reference properties can be added to the SOAP header for messages sent to WSRF web services. Providing such a mechanism would greatly benefit the uptake and use of WSRF.

The BPEL4WS specification on the other hand needs some work to make it more flexible and user-friendly. A mechanism to test the workflow would be very useful for error detection.
Currently BPEL4WS only allows the assignment from one variable to another. If two variables do not have the same message structure assigning can make the variable invalid and produces an error when the service is invoked. This is because the message sent does not match the one expected. For example, often it was the case that one invocation of a service followed naturally from another. The issue is when both services return different values. It would be easier on development if both values could be assigned into a single variable that could be returned. Currently the only option is to have the WSDL for both services return the exact same variable, which has a duplication issue in that you have to update the WSDL in two separate places instead of one. Unfortunately these types of problems can only be determined once you are ready to test the workflow and at this stage any changes to the WSDL require re-engineering the workflow.

Overall, the specification needs some work. It seems to be better suited to modelling workflow when there is little need for human input.

7 Conclusion

In conclusion, this project highlighted some problems with the specifications that would need to be resolved for their integration to be truly successful. The method used to integrate BPEL4WS and WSRF fulfilled all the major requirements but would not scale well. Although scalability is an issue it also meant that BPEL4WS didn't need to know that it was interacting with a WSRF web service and

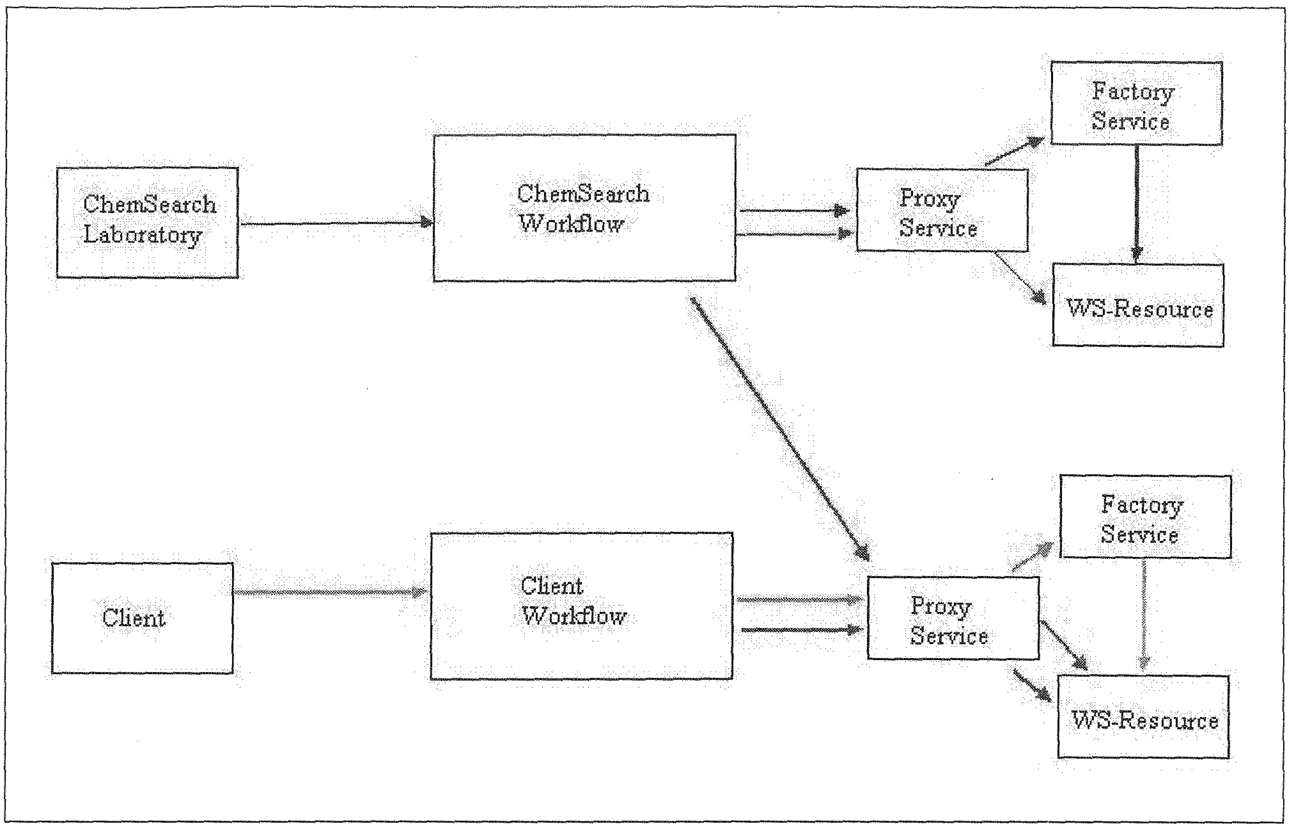


Figure 3: Architecture of the Workflow

meant that tool incompatibilities could be handled in the middle layer rather than at the WS-Resource end. This means that if a BPEL4WS tool was more compatible with WSRF, it would only require small changes. However, it would be better for the BPEL4WS process to interact with WSRF through the proxy service, rather than directly with WSRF. This prevents BPEL4WS having to store and retrieve endpoint references and communicate with two web services, i.e. the proxy and the factory. It would seem that the benefits of this method far out weigh the cost of scalability.

Overall, the two major problems with these specifications is the lack of good documentation and standardisation. In particular the optionality of reference properties in endpoint references is a problem that needs to be resolved. While it is optional for these to be added to the SOAP header, there can be no elegant solution to the integration process. There needs to be code at some point that determines whether a reference property exists for a particular endpoint and if it does to copy it into the SOAP header. Since the WS-Addressing client-side stubs used with WSRF did not perform this functionality it is unlikely to expect BPEL4WS or any other web service technology to support this in the near future.

8 References

References

Amin, K., von Laszewski, G., Hategan, M., Zaluzec, N. J., Hampton, S. & Rossi, A. (2004), GridAnt: A Client-Controllable Grid Workflow System, *in* 'Proceedings of the 37th Annual Hawaii International Conference on System Sciences', Vol. 07, Num. 7, IEEE, pp. 70210c.

Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana, S. (2003), 'Business Process Execution Language for Web Services Version 1.1'. Available at: `ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf`.

Ant (2005), `http://ant.apache.org/`, accessed 11 July 2005.

Apache (2005), `http://www.apache.org/`, accessed 11 July 2005.

Apache Tomcat (2005), `http://tomcat.apache.org/`, accessed 4 October 2005.

Atkinson, M., DeRoure, D., Dunlop, A., Fox, G., Henderson, P., Hey, T., Paton, N., Newhouse, S., Parastatidis, S., Trefethen, A., Watson, P., Webber, J. (2005), 'Web Services Grids: An Evolutionary Approach', *Concurrency and Computation: Practice and Experience* **17**, 377–389.

CapeClear (2005), `http://www.capeclear.com/`, accessed 4 October 2005.

Chao, K., Younas, M., Griffiths, N., Awan, I., Anane, R. & Tsai, C. (2004), Analysis of Grid Service Composition with BPEL4WS, *in* '18th International Conference on Advanced Information Networking and Applications', Vol. 01, Num. 1, pp. 284.

Churches, D., Gombas, G., Harrison, A., Maassen, J., Robinson, C., Shields, M., Taylor, I., Wang, I. (2005), 'Programming Scientific and Distributed Workflow with Triana Services'. Available at: `http://www.extreme.indiana.edu/groc/ggf10-ww/programming_scientific_and_distributed_workflow_with_triana_services/TrianaWorkflow.pdf`.

Czajkowski, K., Ferguson, D., Foster, I., Frey, J., Graham, S., Sedukhin, I., Snelling, D., Tuecke, S., Vambenepe, W. (2004), 'The WS-Resource Framework, Version 1.0'. Available at: http://www-128.ibm.com/developerworks/library/ws-resource/ws-wsrf.pdf, accessed 11 July 2005.

eclipse (2005), http://www.eclipse.org/, accessed 4 October 2005.

Foster, I., Kesselman, C. & Tuecke, S. (2001), The Anatomy of the Grid: Enabling Scalable Virtual Organizations, *in* 'First International Symposium on Cluster Computing and the Grid', Vol. 00, IEEE, pp. 6.

Foster, I., Kesselman, C., Nick, J., Tuecke, S. (2002), 'The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration', Globus Project. Available at: http://www.globus.org/research/papers/ogsa.pdf.

Foster, I., Frey, J., Graham, S., Tuecke, S., Czajkowski, K., Ferguson, D., Leymann, F., Nally, M., Sedukhin, I., Snelling, D., Storey, T., Vambenepe, W., Weerawarana, S. (2004), 'Modelling Stateful Resources with Web Services, Version 1.1'. Available at: http://www-128.ibm.com/developerworks/library/ws-resource/ws-modelingresources.pdf, accessed 11 July 2005.

GridAnt (2005), http://www-unix.globus.org/cog/projects/gridant/, accessed 11 July 2005.

Hey, T., Fox, G. (2005), 'Special Issue: Grids and Web Services for e-Science', *Concurrency and Computation: Practice and Experience* **17** (2-4), 317–322.

Humphrey, M., Wasson, G., Morgan, M. & Beekwilder N., An Early Evaluation of WSRF and WS-Notification via WSRF.NET, *in* 'The Fifth IEEE/ACM International Workshop on Grid Computing', Vol. 00, IEEE, pp. 172–181.

Hunter, J., Cook, R., Pope, S. (2004), 'E-Research Middleware: The Missing Link in Australia's e-Research Agenda'. Available at: http://www.dstc.edu.au/Publications/eReseachMiddleware.pdf, accessed 18 July 2005.

Johnson B. (2005), 'Building a Web Service-The Beginning-What is a Web Service?', http://www.developerfusion.com/show/3245/, accessed 11 July 2005.

Krishnan, S., Wagstrom, P., von Laszewski, G. (2005), 'GSFL: A Workflow Framework for Grid Services'. Available at: http://www.cs.indiana.edu/~srikrish/publications/gsfl.pdf, accessed 18 July 2005.

Leymann, F. (2005), 'Choreography for the Grid: Towards Fitting BPEL to the Resource Framework'. Available at: http://www.cc-pe.net/CCPEwebresource/c8545to872workflow/c854leymann/c854Leymann.pdf, accessed 11 July 2005.

MySQL (2005), http://www.mysql.com/, accessed 4 October 2005.

Oracle (2005), 'Executive Briefing: Grid Computing'. Available at: http://reg.itworld.com/servlet/Frs.frs?Context=LOGENTRY\&Source=cwstrip\&Source_BC=0\&Script=/LP/10003705/reg.

Pasley, J. (2005), 'How BPEL and SOA Are Changing Web Services Development', *IEEE Internet Computing* **09** (3), 60–67.

Slomiski, A. (2005), 'On Using BPEL Extensibility to Implement OGSI and WSRF Grid Workflows'. Available at: http://www.extreme.indiana.edu/groc/ggf10-ww/on_using_bpel_extensibility_to_implements_ogsi_and_wsrf_grids/C871_GridWorkflow2004_Mar05_On_Using_BPEL_extensibility_to_implements_OGSI_and_WSRF_Grids_5.doc.

Sotomayor, B. (2005), 'The Globus Toolkit 4 Programmer's Tutorial', http://gdp.globus.org/gt4-tutorial/singlehtml/progtutorial_0.1.1.html, accessed 4 October 2005.

Staab, S., van der Aalst, W., Benjamins, V. R., Sheth, A., Miller, J. A., Bussler, C., Maedche, A., Fensel, D., Gannon, D. (2003), 'Web Services: Been There, Done That?', *IEEE Intelligent Systems* **18** (1), 72–85.

Taylor, I., Wang, I., Shields, M., Majithia, S. (2005), 'Distributed Computing with Triana on the Grid', *Concurrency and Computation: Practice and Experience* **17**, 1–18.

Triana (2005), http://www.trianacode.org/, accessed 11 July 2005.

Tuecke, S., Czajkowski, K., Foster, I., Frey, J., Graham, S., Kesselman, C., Maquire, T., Sandholm, T.,Snelling, D., Vanderbilt, P. (2003), 'Open Grid Services Infrastructure (OGSI), Version 1.0'. Available at: http://www-unix.globus.org/toolkit/draft-ggf-ogsi-gridservice-33_2003-06-27.pdf, accessed 18 October 2005.

UDDI Spec Technical Committee (2004), UDDI Version 3.0.2, http://uddi.org/pubs/uddi_v3.htm, accessed 11 July 2005.

W3C (2001), 'Web Services Description Language (WSDL) 1.1', http://www.w3.org/TR/wsdl, accessed 11 July 2005.

W3C (2003), 'SOAP Version 1.2 Part 1: Messaging Framework', http://www.w3.org/TR/soap12-part1/, accessed 11 July 2005.

W3C (2004a), Web Services Glossary, http://www.w3.org/TR/ws-gloss/, accessed 18 July 2005.

W3C (2004b), Web Services Addressing (WS-Addressing). Available at: http://www.w3.org/Submission/2004/SUBM-ws-addressing-20040810/.

W3C (2004c), Web Services Architecture, http://www.w3.org/TR/ws-arch/, accessed 12 July 2005.

Wasson, G. & Humphrey M., Exploiting WSRF and WSRF.NET for Remote Job Execution in Grid Environments, *in* 'Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium', Vol. 01, Num. 1, IEEE, pp. 12.

Yang, Y., Tang, S., Zhang, W., Fang, L. (2004), 'A Workflow Language for Grid Services in OGSI-based Grids', *Lecture Notes in Computer Science* **3251**, 65–72.