ASSIS Requirements Documentation

Warwick Bailey, Icodeon Ltd, Cambridge, UK
<warwick@icodeon.com>

Copyright © 2004 Icodeon Ltd, Studio 471, 48 Regent Street, Cambridge CB2 1FD, England

2004

Abstract

This paper contrasts two standards based approaches to assembling applications from remote web services.

The first approach integrates web services that encapsulate business logic and uses the BPEL4WS (Business Process Execution Language for Web Services) standard.

The second approach integrates web services that provide fragments of content mark up and uses the WSRP (Web Services for Remote Portals) standard.

A major requirement for the ASSIS project is the choice of a standards based way of assembling applications from disparate web service components. This paper suggests that the BPEL4WS is a preferred approach.

Table of Contents

ASSIS 1
Introduction 1
Web Services in the JISC e-Learning Framework 1
Web Services and Applications in the ASSIS Project 2
Connecting Discrete Web Services in ASSIS Applications 2
Design Decisions for Connecting Web Services in ASSIS Applications 3
Scenarios for Connecting Web Services in ASSIS Applications 3
Scenario 1: Using BPEL4WS 4
Scenario 2: Using WSRP 6
Conclusions 12

ASSIS

Introduction

The ASSIS project will build applications for formative assessment in e-learning, by integrating run time engines for the IMS QTI specification (for managing assessment) and the IMS Simple Sequencing specification (for managing content ordering). These run time engines will be hosted as discrete web services. The major technical challenge for the ASSIS project is to investigate a standards based way of assembling applications from disparate web service components.

Web Services in the JISC e-Learning Framework

The JISC e-Learning Framework is a service-oriented factoring of the core services required to support e-Learning

applications, portals and other user agents. Each service defined by the framework is envisaged as being provided as a networked service within an organization, typically using web services. The ASSIS project is an initial investiga# tion into the procedure for assembling applications from these discrete services.

Web Services and Applications in the ASSIS Project

The ASSIS project will build two applications, each assembled from a selection of web services. The project pro# posal envisages each of these applications being made available, both as a standalone web application and as a port# let built to the JSR 168 specification. The applications and their component web services are:

- The *ASSIS Player Application* is a run time environment for IMS Content Packages that contains IMS QTI items, IMS Simple Sequencing behaviours and possibly static XHTML content. This application will be as# sembled using two web services:
  - The APIS *QTI Run* service: this web service encapsulates a QTI response processor that renders and pro# cesses QTI item XML.
  - The ISIS *SS Run* service: this web service encapsulates a sequencing engine that traverses an activity tree of content, evaluating sequencing rules and limit conditions.
- The *ASSIS QTI Select Application* is a run time environment for identifying the IMS QTI items in a store and as# sembling selected items into an IMS Content Package. that contains IMS QTI items and possibly static XHTML content. This application will be assembled using two web services:
  - The ASSIS *QTI Query* service: this web service abstracts a search mechanism over one or more item bank stores of IMS QTI items.
  - The APIS *QTI Run* service: this web service encapsulates a QTI response processor that renders and pro# cesses QTI item XML (as above).

The key challenge for the ASSIS project is to investigate a standard approach in which web applications and portlets may be easily built by connecting together the discrete web services exposed within the JISC e-Learning Frame# work.

Connecting Discrete Web Services in ASSIS Applications

Broadly speaking there are two approaches to connecting discrete services: the first is *Business Process Manage# ment* and the second is *Aggregation of Content*. The first approach operates by assembling a high level process from components that expose business logic. The latter approach operates by assembling a composite view from compon# ents that expose fragments of content mark up.

Business Process Management

In the context of web services, *Business Process Management* is the approach to connecting discrete web services together to create higher-level business processes. The primary function of each of the web services is to encapsulate some business logic.

With the introduction of web services, terms such as 'web services composition' and 'web services flow' were used to describe the composition of web services in a process flow to achieve some business goal. More recently, the terms *choreography* and *orchestration* have been used to describe this. Choreography is typically associated with the pub# lic message exchanges between multiple parties, rather than a specific business process that is executed by a single party. Orchestration describes how web services can interact with each other at the message level, including the

business logic and execution order of the interactions. These interactions may span applications and/or organiza# tions, and result in a long lived, transactional, multi-step process model.

The two applications required as ASSIS project deliverables, the *ASSIS Player Application* and the *ASSIS QTI Select Application*, are examples that require a standards-based approach for connecting web services together to create higher-level business processes. In other words, they are applications for which web services orchestration would be a potential requirement.

Recently, a number of specifications have been proposed by major software vendors for web service orchestration. Predominant amongst there are the *Business Process Execution Language for Web Services* (BPEL4WS) and the *Business Process Management Language / Web Services Choreography Interface* (BPML/WSCI). It is still very un# clear as to which ones will emerge as industry standards for web services orchestration and choreography, but con# sensus does appear to be building around BPEL4WS with support from IBM, Microsoft and BEA. Sun support the WSCI standard.

From the practical perspective of the ASSIS project, there is currently more technical documentation and developer tools currently available for BPEL4WS. The developer tools include a Java runtime engine that can be integrated with the popular Jakarta Tomcat servlet container and a visual editor that plugs into the Eclipse development plat# form for composing business processes from the exposed WSDL interface descriptors of web services. Currently these tools are available as alpha release from IBM with a 90 day evaluation limit. This limit does not seems to be enforced programatically in the software. Rather it is currently part of the legal preamble to the EULA.

Aggregation of Content

In the context of web services, aggregation of content is the approach to connecting discrete web services together to create a composite view from isolated pieces of content. The primary function of each of the web services is to provide some fragment of XML or XHTML markup that may be incorporated into a single web page.

Broadly speaking there are two technologies for creating the composite view: *Web Publishing Frameworks* and *Portals*.

The dominant open source web publishing framework is the Cocoon Project from the Apache Software Foundation. From the perspective of the ASSIS project, Cocoon provides a robust framework for incorporating fragments of XML and XHTML into a single view and may easily be packaged as web application hosted in the Tomcat servlet container.

The dominant open source portal applications/portlet containers that support the JSR 168 standard are the Pluto Portlet Container, again from the Apache Software Foundation, and the uPortal portal. Pluto provides a framework for incorporating portlets of XHTML markup into a single view and may easily be packaged as web application hosted in the Tomcat servlet container. Further to this, the portlets of XHTML markup may now be hosted remotely and accessed through the *Web Services for Remote Portlets* (WSRP). An implementation of WSRP in Java (WSRP4J) has recently been donated to the Apache Software Foundation by IBM and may also be deployed to Tomcat as a web application.

Design Decisions for Connecting Web Services in ASSIS Ap# plications

A major requirement for the ASSIS project is the ability to connect discrete web services and a design decision has to be made as to the best technical way to achieve this. The web services may be connected at the level of business process (using web services orchestration) or at the level of view (using aggregation of content). The following sec# tion explores these two scenarios for the ASSIS project. The web services orchestration scenario is investigated us# ing BPEL4WS. The aggregation of content scenario is investigated using WSRP.

Scenarios for Connecting Web Services in ASSIS Applications

This section works through two scenarios for connecting web services in ASSIS applications in order to evaluate the strengths and weaknesses of each approach. The first scenario explores the use of BPEL4WS to connect services at the level of business process. The second scenario explores the use of WSRP to connect services at the level of con# tent fragment aggregation via portlets.

Scenario 1: Using BPEL4WS

Business Process Execution Language for Web Services (BPEL4WS) is an XML-based flow language that defines how business logic encapsulated in a number of remote web services may be combined to achieve some higher level process. In the case of the *ASSIS Player Application*, for example, a BPEL xml would be written to describe the in# vocations to the *QTI Run* web service (that encapsulates the logic of a QTI response processor) and *SS Run* web ser# vice (that encapsulates the logic of a sequencing engine).

The description in the BPEL xml document can involve processes contained within or between institutions and en# terprises, may be synchronous or asynchronous, and can describe robust fault handling. BPEL4WS builds on exist# ing standards (it is a layer on top of WSDL) and a Java run time environment, known as Business Processes for web Services for Java (BPWS4J), is available that installs as a Tomcat web application. This Java technology builds on familiar components developed by the Apache Software Foundation: the binaries of WSIF, Xerces, Xalan, and the Axis and SOAP engines. The scenario described below, called the "Loan Approval" scenario is one of the sample processes bundled with the BPWS4J distribution. The scenario example has been successfully edited, built from source, installed and run on Icodeon servers.

The "Loan Approval" scenario bundled with the BPWS4J distribution envisages a client application and two remote web services. This is functionally equivalent to *ASSIS Player Application* or the *ASSIS QTI Select Application*, in that each employ a consumer application and two provider web services. In the case of the BPWS4J "Loan Approv# al" scenario, three java files are provided, one consumer application and two remote web services. The comparison is detailed in the table below:

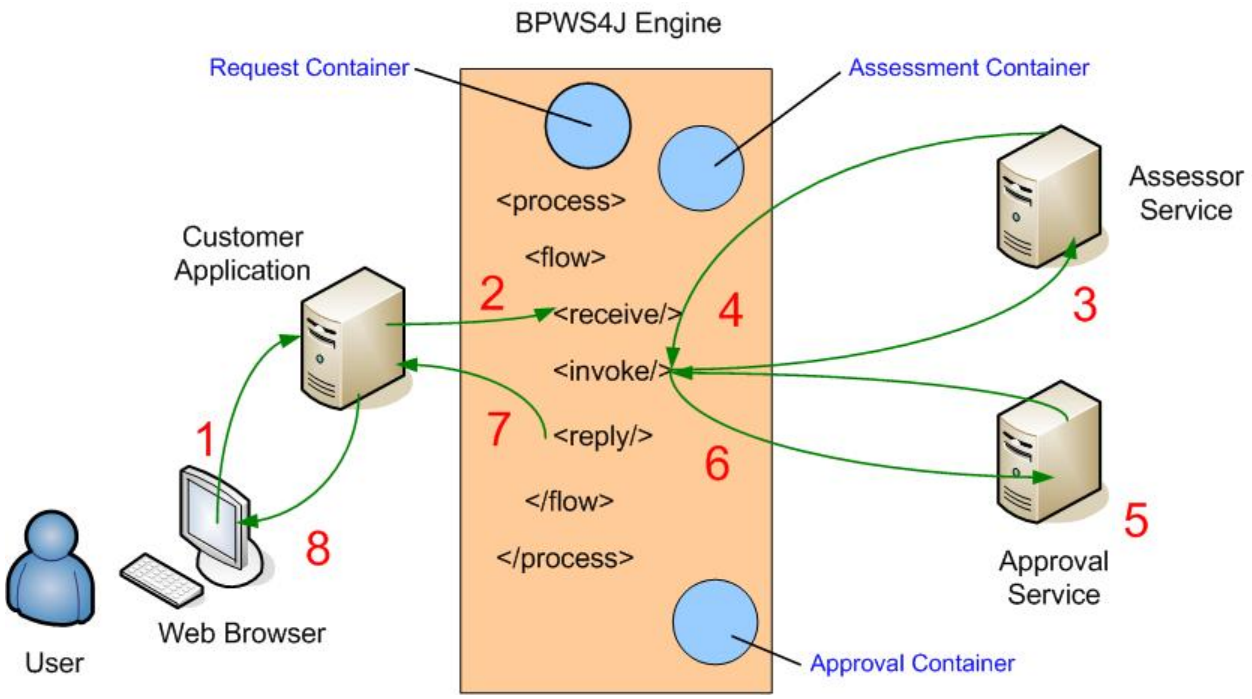
Table 1. Comparison of BPWS4J and ASSIS Scenarios

Scenario Name	**BPEL4WS Loan Approv# al**	**ASSIS Player**	**ASSIS QTI Select**
Consumer Application	Customer.java	Player	QTI Select
Web Service 1	LoanAssessor.java	QTI Run	QTI Query
Web Service 2	LoanApprover.java	SS Run	QTI Run

The "Loan Approval" scenario is a high level business process, built by defining a composition between the *Cus# tomer* application and the two discrete component web services. The customer submits a loan application, which is first reviewed first by the *Assessor* web service, and then by the *Approver* web service. The result of the review is then returned to the *Customer*.

The sequence of web service calls is defined in a single BPEL xml file. The orchestration of the web service calls is managed at run time by the BPWS4J engine that is triggered by a SOAP message sent from the client *Customer* consumer application. The sequence of method calls and web service invocations by the BPWS4J engine is illus# trated in the figure below:

Figure 1. BPWS4J Orchestration of Two Web Services and a Consumer Application



1. The user accesses a web application to initiate the overall business process using the *Customer* application.

2. The *Customer* application sends a SOAP message to the BPWS4J engine, hosted as a Tomcat web application. The message is received and placed in the *Request Container*. The container definition is described by elements of the BPEL xml document deployed to the BPWS4J engine. For example, three message containers, the *Re# quest Container*, *Assessment Container* and the *Approval Container* are described in the BPEL xml document.

3. The BPWS4J engine then invokes a call to the first remote web service that maintains the business logic for the loan assessment.

4. The BPWS4J engine receives the response from the *Assessor Service* and places this message in the *Assessment Container*.

5. The BPWS4J engine then invokes a call to the second remote web service that maintains the business logic for the loan approval.

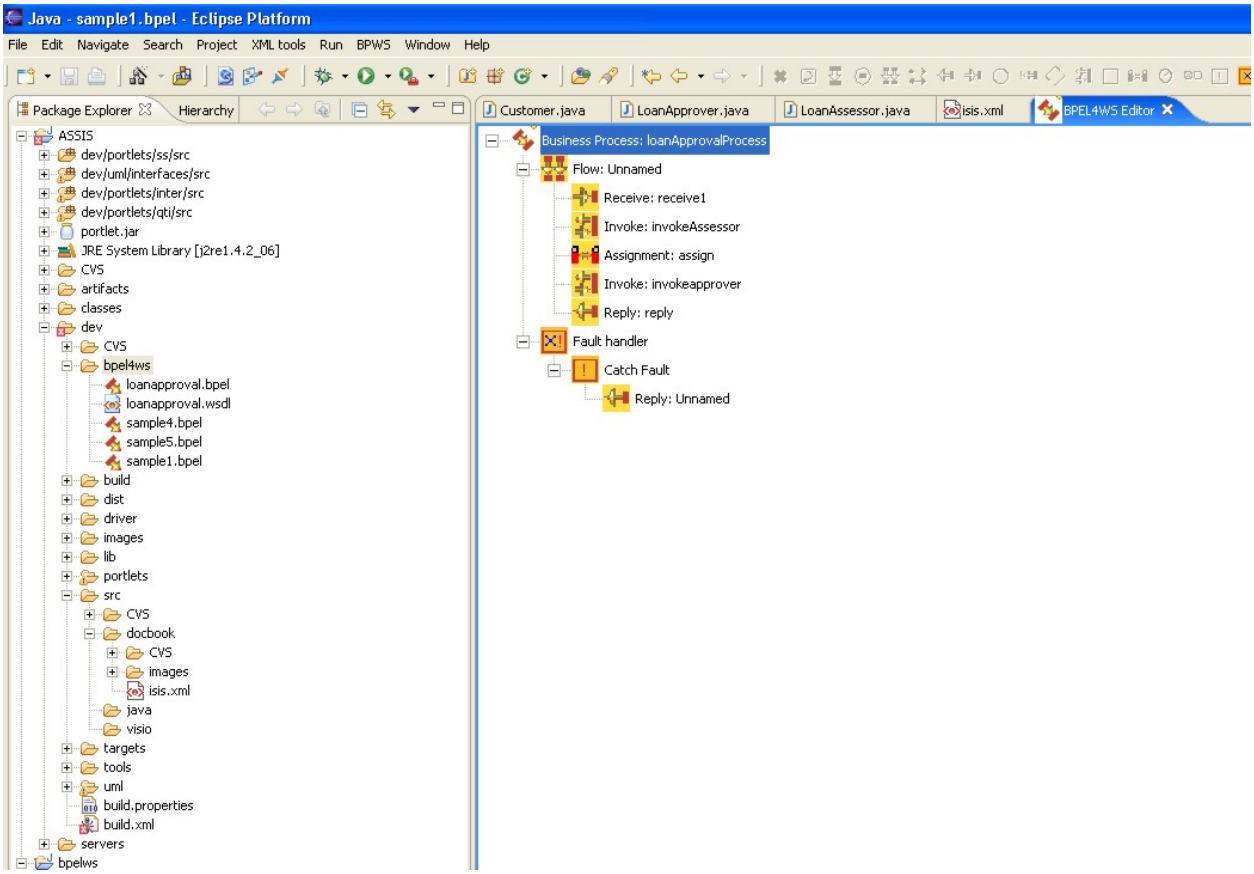
6. The BPWS4J engine receives the response from the *Approval Service* and places this message in the *Approval Container*.

7. A SOAP message is then sent as a reply to the *Customer* application.

8. The *Customer* application renders the contents of the SOAP reply message to the user.

In summary, the orchestration of two remote web services and a single consumer application by BPEL4WS has been simplified to a 4K BPEL xml document interpreted by the BPWS4J engine hosted on Tomcat. The BPEL xml docu# ment is shown below in the visual editor that is available as an Eclipse IDE plug-in.

Figure 2. Visual Editing of the BPEL xml Document in the Eclipse IDE



The "Loan Approval" scenario is a relatively modest example of how discrete services may be wired together to achieve some higher level business process, but much more elaborate compositions are possible.

The BPEL4WS process itself is basically a flow-chart like expression of an algorithm. Each step in the process is called an *activity*.

There are a collection of primitive activities, including the following: invoking an operation on some Web service (*invoke*), waiting for a message to operation of the service's interface to be invoked by someone externally (*receive*), generating the response of an input/output operation (*reply*), waiting for some time (*wait*), copying data from one place to another (*assign*), indicating that something went wrong (*throw*), terminating the entire service instance (*ter# minate*), or doing nothing (*empty*).

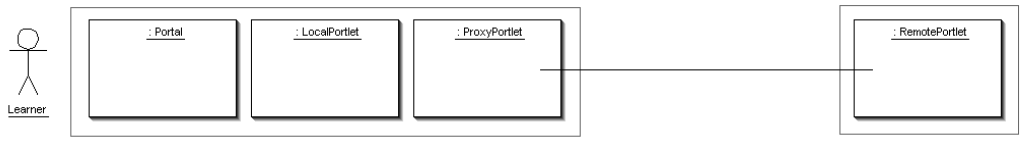
These primitive activities can be combined into more complex algorithms using any of the structure activities provided in the language. These are the ability to define an ordered sequence of steps, either sequentially or in paral# lel (*flow*), the ability to have branching using case-statement approach (*switch*), the ability to define a loop (*while*), or the ability to execute one of several alternative paths (*pick*).

Scenario 2: Using WSRP

Web Services for Remote Portlets (WSRP) is a standard that defines how one portlet will communicate with a re# mote server to display a remotely hosted portlet. WSRP builds on existing standards, such as SOAP and WSDL, and all the technology for hosting WSRP based portals is now available as open source software through the Apache Pluto Portlet Container and the Apache WSRP4J incubator project. A key point to note is that WSRP portlets are *presentation orientated*, and *not* data or business logic orientated, like many web services. This immediately raises an issue for ASSIS applications that will need to introduce some business logic for interportlet communication. For example, a portlet managing the interaction of a user with a QTI item will need to inform a different portlet man# aging the sequencing state of the boolean or float result of the QTI response processor.

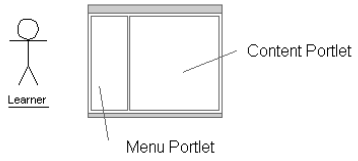
The figure below shows the typical application architecture for WSRP deployment. The consumer portal portlet container can host a blend of local JSR 168 portlets and remote portlets provided by a WSRP producer. The remote portlets are contained within a proxy portlet shell in the portlet container.

Figure 3. Typical WSRP Architecture



In the case of the *ASSIS Player Application* for example, the portal portlet container would host two proxy portlets. The first of the proxy portlets would be a *Menu Portlet*, displaying a view of the activity tree. The second proxy portlet would be a *Content Portlet*, displaying a view of a QTI item. A sketch of the expected composite view dis# played by the portal portlet container to the learner is shown in the figure below:

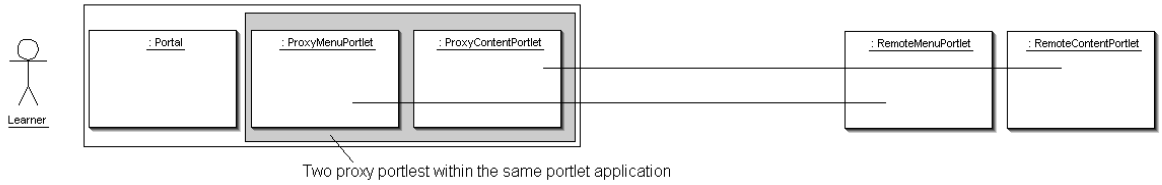
Figure 4. Menu Portlet and Content Portlet Aggregated in ASSIS Player Application



In the *ASSIS Player Application*, these two portlets will need to influence the state of the other. The *Menu Portlet* will need to inform the *Content Portlet* of the correct QTI item to display, and the *Content Portlet* will need to in# form the *Menu Portlet* of the result of the QTI response processing.

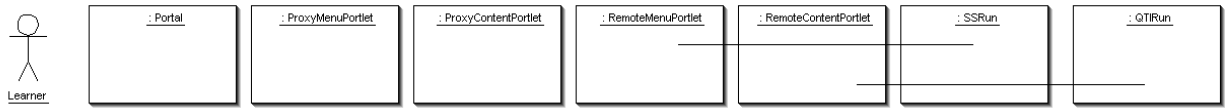
The JSR 168 specification provides only limited support for interportlet communication: namely shared session state for portlets that are part of the same *portlet application*. Session state modified by one portlet is accessible to the other portlet. The required architecture, shown in the figure below, is two proxy portlets sharing session state as part of a single portlet application and each proxy portlet receiving its mark up from a remotely hosted portlet:

Figure 5. Menu Proxy Portlet and Content Proxy Portlet as a Single Portlet Application



The distributed architecture is further extended as each remote portlet that supplies the mark up for the proxy port# lets is dependent on a e-Learning Framework web service. The *Remote Menu Portlet* will need to invoke calls to the ISIS sequencing service, *SS Run*, and the *Remote Content Portlet* will need to invoke calls to the APIS QTI service, *QTI Run*:

Figure 6. Distributed Architecture for the ASSIS Player Application



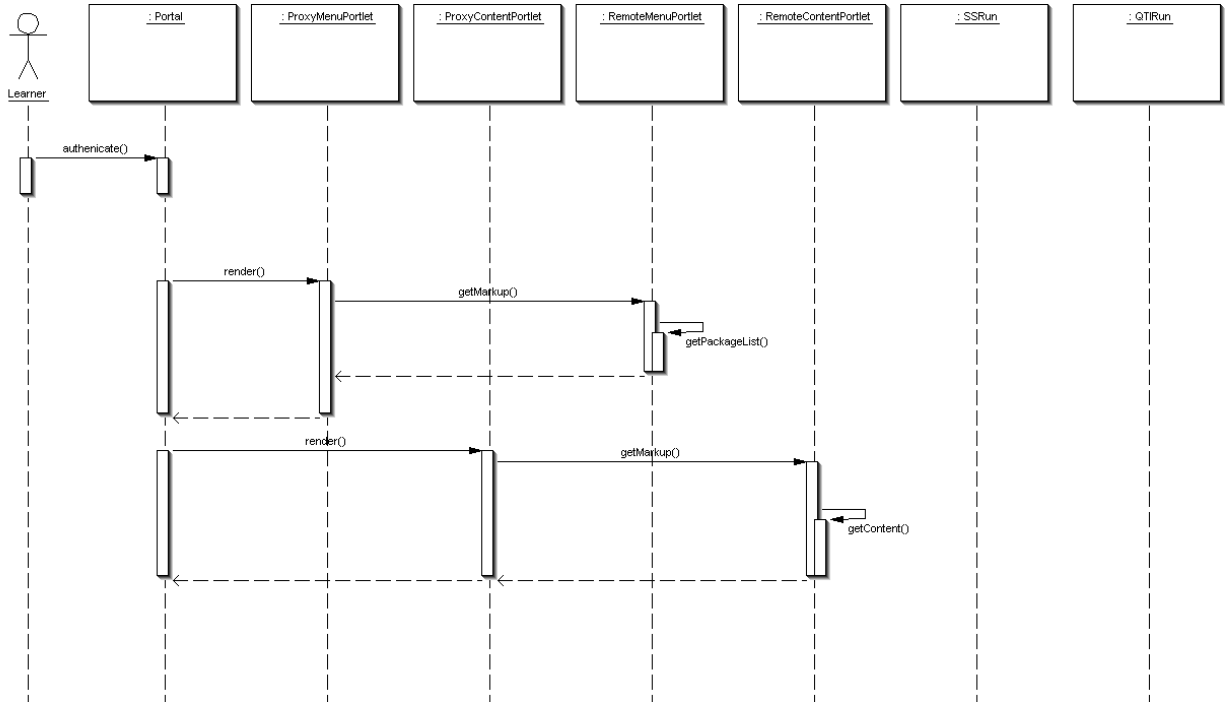
The following section outlines the method calls between the portlets and services within this distributed architecture. Only three steps are considered, but they are sufficient to illustrate the co-operation between the components.

1. Log on to Portal
2. Start Sequencing
3. Submit QTI Item

Step 1. Log on to Portal

In this step, a learner logs on to the portal to view an initial menu in the menu portlet and an initial display in the content portlet. Logging into the portal triggers the portlet container to render the two portlets according to their ini# tial state.

Figure 7. Log on to Portal



1. The portal page containing the *ProxyMenuPortlet* and the *ProxyContentPortlet* is accessed by the learner for the first time.
2. The portal calls the *render()* methods of both proxy portlets for their initial view.

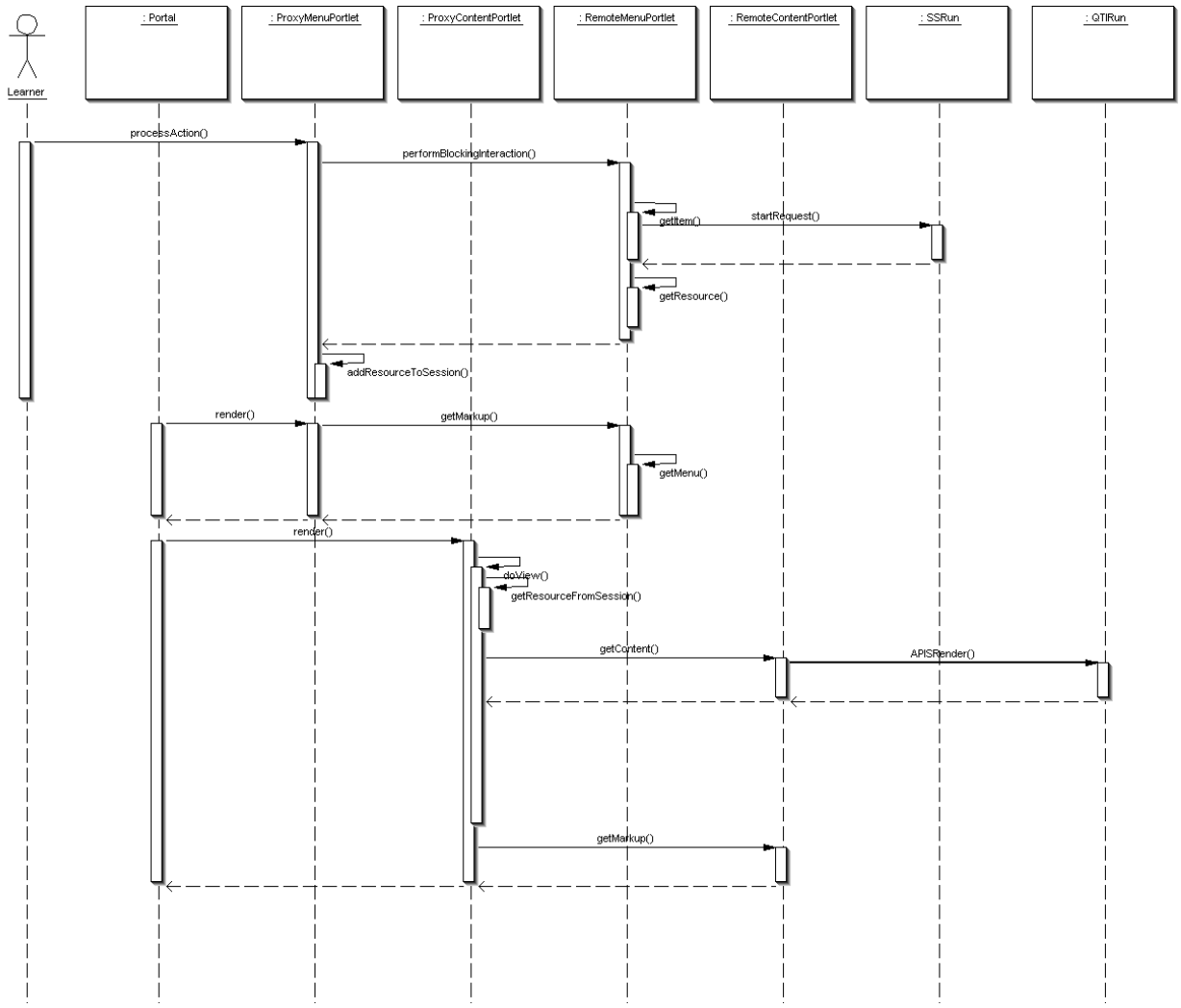
3. Each proxy portlet then invokes the WSRP *getMarkup()* call on its corresponding remote portlet to return an initial view. The *RemoteMenuPortlet* invokes the *getPackageList()* method and returns a menu of available content packages. The *RemoteContentPortlet* invokes the *getContent()* method, which for this initial call simply returns an initial view, such as homepage.

4. The learner is presented with a list of links to content packages in the *ProxyMenuPortlet* and an initial homepage in the *ProxyContentPortlet*.

2. Start Sequencing

In this step, a learner selects a content package from a list in the menu portlet. This triggers a cascade of calls that result in a sequencing engine making an initial traversal of the activity tree to determine the first resource to deliver. This resource is then rendered as an XHTML fragment in the content portlet.

Figure 8. Start Sequencing



1. The learner clicks a link in the *ProxyMenuPortlet* to select a particular content package.

2. The portal calls the *processAction()* method of *ProxyMenuPortlet* to register a user interaction with a specific portlet in the portal.

3. The *ProxyMenuPortlet* then invokes the WSRP *perfomBlockingInteraction()* call on the *RemoteMenuPortlet* to invoke the *getItem()* method of the remote portlet.

4. The *RemoteMenuPortlet* then invokes a SOAP call to the *startRequest()* method of the *SSRun* web service to start a sequencing session. The *SSRun* web service returns a unique identifier of the content package *item* as determined by its *sequencing engine*.

5. The *RemoteMenuPortlet* uses the unique identifier returned by the *SSRun* web service for two operations: first to modify the state of its internal representation of menu, so that the current activity is shown as selected; second to retrieve the resource (XHTML content ot XML QTI item) that corresponds to the unique identifier. This resource is then returned to the *ProxyMenuPortlet* which places the resource in session state and com# pletes the WSRP *perfomBlockingInteraction()* call. Note that because the two proxy portlets are within the same *portlet application* they have access to the same session state. Session state modified by one portlet is ac# cessible to the other portlet.

6. The portal calls the *render()* methods of both proxy portlets for their current view.

7. The call to the *render()* method of the *ProxyMenuPortlet* invokes WSRP *getMarkup()* call on the *Remote# MenuPortlet* which returns the markup for a menu with the current activity as a selected item.

8. The call to the *render()* method of the *ProxyContentPortlet* invokes the overriden *doView()* method of the *ProxyContentPortlet*. This method retrieves the resource placed in shared session state by the *ProxyMenuPort# let* and submits it the *RemoteContentPortlet*.

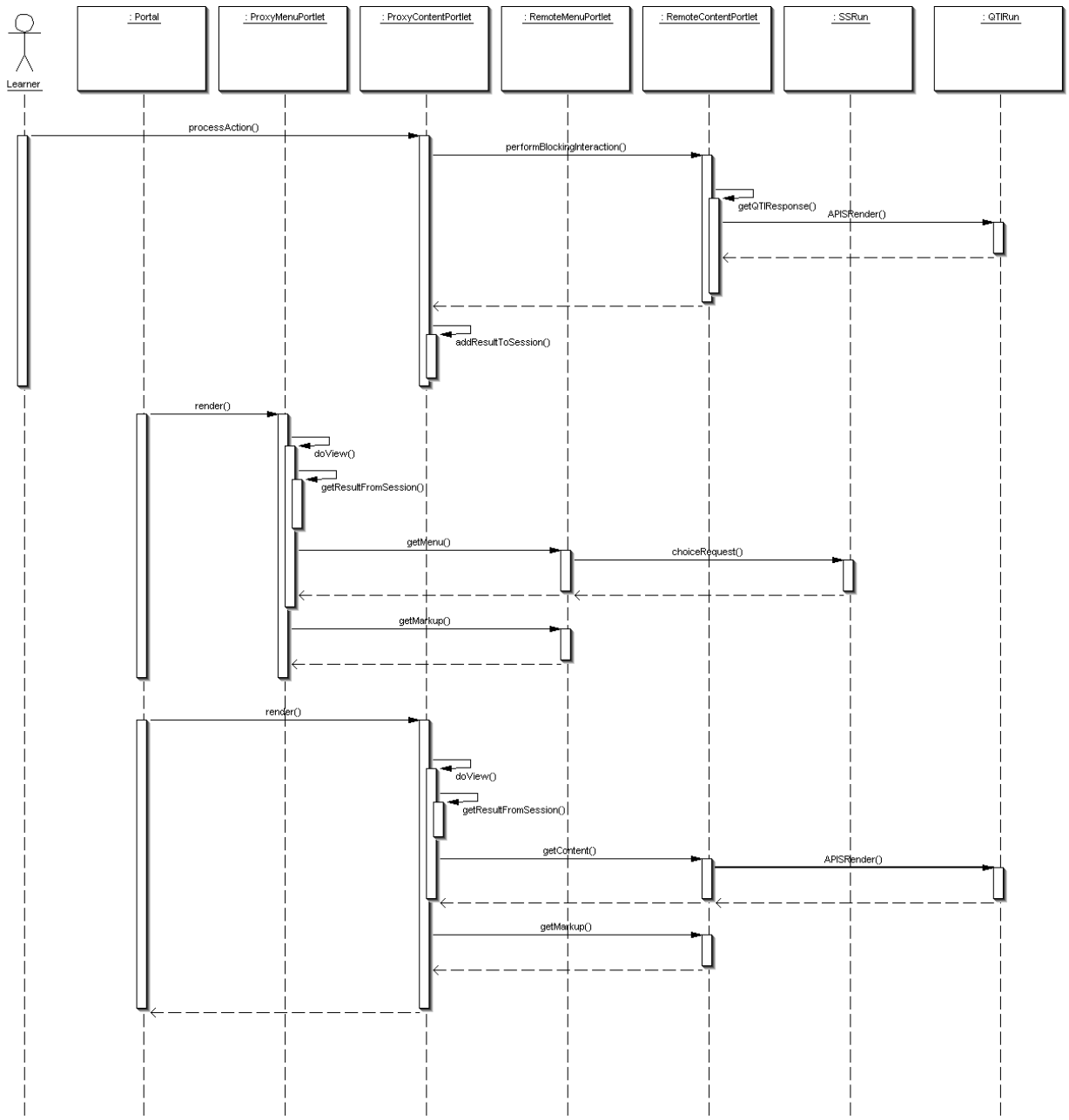
9. The *RemoteContentPortlet* then invokes a call to the *QTIRun* web service which returns an XHTML fragment representing the resource. The XHTML fragment is returned as mark up to the *ProxyContentservlet*.

10. The learner is presented with a menu of items in a content package and the first item from the content package according to the sequencing rules within the content package.

3. Submit QTI Item

In this step, a learner selects a form element in the content portlet to indicate their choice of response to the question in the QTI item. On submitting the form, the menu portlet is updated and the content portlet displays the QTI feed# back.

Figure 9. Submit QTI Item



1. The learner selects a form element in the *ProxyContentPortlet* to indicate a choice of answer in a QTI item and then presses a form submit button.

2. The portal calls the *processAction()* method of *ProxyContentPortlet* to register a user interaction with a specific portlet in the portal.

3. The *ProxyContentPortlet* then invokes the WSRP *perfomBlockingInteraction()* call on the *RemoteContentPort# let* to invoke the *getQTIResponse()* method of the remote portlet.

4. The *RemoteContentPortlet* then invokes a call to the *APISRender()* method of the *QTIRun* web service to de# termine the result of the learner's interaction with the QTI item. The *QTIRun* web service returns the result of

the QTI response processor, either a boolean or float.

5. The *RemoteContentPortlet* returns the boolean or float QTI result to the *ProxyContentPortlet* which places the result in session state and completes the WSRP *perfomBlockingInteraction()* call. Note that because the two proxy portlets are within the same *portlet application* they have access to the same session state. Session state modified by one portlet is accessible to the other portlet.

6. The portal calls the *render()* methods of both proxy portlets for their current view.

7. The call to the *render()* method of the *ProxyMenuPortlet* invokes the overriden *doView()* method of the *Proxy# MenuPortlet*. This method retrieves the boolean or float QTI result placed in shared session state by the *Proxy# ContentPortlet* and submits it the *RemoteMenuPortlet* to update the markup of the menu. This operation may also include further calls to the *SSRun* web service to update the state of the internal representation of the menu maintained by the *RemoteMenuPortlet*.

8. The call to the *render()* method of the *ProxyContentPortlet* invokes the overriden *doView()* method of the *ProxyContentPortlet*. This method retrieves the QTI boolean or float result placed earlier in shared session state submits it to the *RemoteContentPortlet*. The remote portlet makes a further call to the *QTIRun* web service which returns the XHTML fragment representing the QTI feedback.

9. The *RemoteContentPortlet* then invokes a call to the *QTIRun* web service which returns an XHTML fragment representing the resource. The XHTML fragment is returned as mark up to the *ProxyContentPortlet*.

10. The learner is presented with an updated menu of items in a content package and the QTI feedback from the first item from the content package.

In summary, the UML sequence diagrams illustrate the addition of business logic to the portlet application contain# ing the two proxy portlets. This runs counter to the design philosophy of WSRP, which is presentation orientated, not data or business logic orientated. Further to this, the JSR 168 specification only has very limited support for in# terportlet communication through shared session state. This results in a less than elegant method for managing the interactions between the two portlets and perhaps an inappropriate technology choice for assembling applications from remote web services.

Conclusions

This paper has contrasted two standards based approaches to assembling applications from discrete web services components: BPEL4WS and WSRP. The comparison is summarized in the table below:

Table 2. Comparison of BPEL4WS and WSRP

Feature	**BPEL4WS**	**WSRP**
Integration Strategy	Business Process Management	Aggregation of Content
Design Philosophy	Assembly of high level business pro# cesses from component web services at the level of business logic. Design is logic orientated.	Aggregation of content markup from remote portlets hosted as web ser# vices. Design is presentation orient# ated.
Open Standard ?	Yes, version 1.1	Yes, version 1.0
Java Run Time ?	Yes, freely available as binary under commercial licence. BPWS4J from IBM. Version 2.1	Yes, freely available as open source and binary. WSRP4J from IBM, now in Apache Incubator. Version below 1.0
Hosted as a web application ?	Yes, tested in Tomcat on Linux and Windows	Yes, tested in Tomcat on Linux and Windows

Feature	**BPEL4WS**	**WSRP**
Sophistication for Web Service Or# chestration	High level language for describing synchronous or asynchronous trans# actions and work flows between web services.	Limited interportlet communication via access to objects held in shared session state.
Visual environment for assembling web service orchestration?	Yes, Eclipse IDE plug in.	No.
Supporting Technologies	WSDL, SOAP. Apache Tomcat, WSIF, Xerces, Xalan, and the Axis and SOAP engines.	JSR 168, WSDL, SOAP. Apache Tomcat, Xerces, Xalan, Pluto and the Axis engine.

The natural conclusion to this comparison is that BPEL4WS, designed as a solution to web service orchestration, is a superior technology choice compared to WSRP, for assembling applications from web services in the ASSIS project. WSRP was designed to solve a different technical problem: the hosting of remote portlets.

The final licensing model of BPWS4J run time is currently unclear and needs to be resolved before recommendation of BPWS4J for inclusion in ASSIS application development.