
Filename	YAWL-ObserverGatewayConcepts.doc						
Date Created	1-Mar-04	Last Modified	13/03/2006 16:58:00	Revision	Error! Unknown document property name.	Page	1 of 10

Implementing In-Process Custom Services

REVISION HISTORY

Author	Date	Revision	Reason for Change
Andrew Hastie	15 Jan-06	1.0	Document created
Andrew Hastie	22-Jan-06	1.1	Post review by LA
Lachlan Aldred	27-Jan-06	1.2	General refinements

Filename	YAWL-ObserverGatewayConcepts.doc	Revision		Page	2 of 10
Date Created	1-Mar-04	Last Modified	13/03/2006 16:58:00	Error! Unknown document property name.	

CONTENTS1.....	INTRODUCTION
3	
1.1 WHY UTILISE YAWL?.....	3
1.2 SOFTWARE RELEASE LEVEL.....	3
2. THE INTERFACEB OBSERVER GATEWAY	4
2.1 BACKGROUND.....	4
2.2 INTERFACE DESIGN	4
2.3 CUSTOM SERVICE URLS	5
2.4 IMPLEMENTING THE MANAGER CLASS.....	7
2.4.1 The ObserverGateway Interface	8
2.5 REGISTERING WITH THE YAWL RUNTIME.....	9

Filename	YAWL-ObserverGatewayConcepts.doc						
Date Created	1-Mar-04	Last Modified	13/03/2006 16:58:00	Revision	Error! Unknown document property name.	Page	3 of 10

1. Introduction

This document describes a suggested architecture for connecting free-standing Java applications to an instance of the YAWL runtime in order to control some form of process control flow.

1.1 Why Utilise YAWL?

The justification for using the YAWL system in order to provide a “process flow controller” is simply that of extreme flexibility. There are numerous examples of coordinated software systems where some form of control flow exists, advancing the overall processing requirement from one stage to the next. A good example of such a control flow is that of a web browser session into a web server over the Internet. A user connects to some website in order to order some goods, and subsequently have them delivered. In this scenario, the user typically searches for and locates a number of items from the online stores catalogue, and then is directed to the payment details web form, and finally to the delivery details web form. Whilst there are several web development frameworks that support the simple routing of web pages, it is considered that such frameworks do not provide the rich set of process routing and data manipulation facilities found within the YAWL system.

In Enterprise Application Integration (EAI) scenarios, a separate set of features make the use of YAWL extremely attractive, especially when these are coupled with other Java2 Enterprise Edition (J2EE) features such as the Java Message Service (JMS). Traditionally, back-office or batch style applications are executed in some predefined sequence, controlled either manually or via complex shell scripts. In the situation where a Service Oriented Architecture (SOA) model is adopted¹, it becomes feasible that the sophisticated process control flow features of the YAWL runtime can be utilised to build highly automated and efficient back-office processing systems. In fact the YAWL runtime provides an extremely flexible and powerful routing engine. In an XML-centric integration architecture such as SOAP, the use of YAWL becomes even more beneficial given its comprehensive support for XML with its data manipulation and interrogation layer.

The remainder of this document describes an architectural model utilising a single instance of the YAWL runtime to control multiple YAWL Custom Services. Each custom service effectively services a request into a back-end office application. The technique outlined in this document proposes an architecture where each application may execute as a free-standing Java application with no requirement for an external application server². The approach to connecting some custom services with their own external systems is left as an exercise for the reader although various suggestions are made.

1.2 Software Release Level

The document assumes a release of the YAWL software at release Beta7 or later.

¹ Whereby a formalised interface is developed to allow external yet controlled access to application system functionality.

² This is unlike the default architecture for YAWL wherein a J2EE/Servlet container is used to coordinate a set of integrated Web applications.

Filename	YAWL-ObserverGatewayConcepts.doc			Revision	Error! Unknown document property name.	Page	4 of 10
Date Created	1-Mar-04	Last Modified	13/03/2006 16:58:00				

2. The InterfaceB Observer Gateway

2.1 Background

The YAWL runtime primarily implements two interfaces that, one way or another, are exposed to the business environment:

Interface A	Management interface for loading and unloading process models adding users, and observing case progress.
Interface B	An execution interface for launching cases (process instances) and worklist style operations such as starting and completing tasks within a case.

These interfaces listen and respond, to requests from the business environment. Conversely the “Observer Gateway” interface is designed to allow external systems/modules to listen and respond to the YAWL runtime engine. Once an observer gateway instance is deployed into the runtime engine, the implementation may receive notifications normally associated with YAWL custom services. As YAWL custom services typically implement the logic to process work item requests associated with atomic tasks, the events handled by the observer gateway are principally concerned with the scheduling and execution of tasks - like Interface B.

There are two alternative interaction models in communication with the YAWL runtime engine. These were alluded to above. First there is a “pull style” model. In this model, the custom service polls the YAWL runtime on demand in order to retrieve, and ultimately display to the end user a list of outstanding work item tasks. The alternative is a “push style” model wherein the YAWL runtime engine can notify external systems of newly scheduled tasks etc. Since beta 7 this “push style” model has been extended into a plug-in style architecture. The “Observer Gateway” interface forms a primary foundation to this framework. Now there is a conveniently flexible alternative to the XML/HTTP technique used to connect the YAWL runtime engine with custom services.

2.2 Interface Design

The “Observer Gateway” interface is designed to allow multiple Java objects to register interest with an instance of a YAWL runtime in order to receive notification as to when an atomic task’s work item becomes available or unavailable. The object making the registration needs to provide a number of “listener style” methods as demanded by the observer gateway interface. As is usual for listener interfaces, the call into the listener method executes on the runtime’s thread and should not block execution for an extended period of time. It is suggested that the method implementing the listener interface creates and starts a separate worker thread to handle the request, thus allowing the runtime to continue with minimal delay. Whilst the Observer Gateway interface is used to connect with the custom YAWL services that are shipped with the system, it is open to integration with user developed code.

In the existing HTTP based custom service model, (see YAWL document “Technical Note – Connecting Services to the YAWL Engine”) a Servlet container is used to host one or more custom services. Each custom service needs to be registered with the runtime via the YAWL Administration facility. Once registered, a custom service will be notified of new and cancelled work items via a well defined interface call. The observer gateway interface extends this model in the following ways:

Filename	YAWL-ObserverGatewayConcepts.doc			Revision	Error! Unknown document property name.	Page	5 of 10
Date Created	1-Mar-04	Last Modified	13/03/2006 16:58:00				

1. The requirement to pre-register a service URL with the runtime is removed³.
2. Assuming a suitable manager style application is used to interact with the runtime, custom services can be implemented where communication between it and the runtime (via a suitable manager) can be via any form of wire protocol.
3. Notification for completion of a specific case is supported

2.3 Custom Service URLs

The existing YAWL custom service implementation utilises the configuration of specific URLs against an atomic task in order to route work requests (and hence notifications) to a specific custom service.

The same use of URLs is made within the observer gateway, but instead of routing on a one-to-one basis from an atomic task to a specific custom service, the “scheme” component of the URL is used to identify the object listening on the observer interface. Once consumed by the listener, the remainder of the URL specification can be used to forward processing of the work items data associated with the atomic task using any scheme required by the user.

The observer gateway controller (implemented via class `ObserverGatewayController`) within the runtime supports registration by multiple objects which implement the `ObserverGateway` interface. The gateway then forwards requests to these registered objects using in-process method calls. It is therefore a requirement that both the runtime and all objects registering must be within the JVM instance. Rather than expose the observer gateway to alternate protocols such as RMI, XML-RPC, SOAP, JMS etc., it is left as an implementation issue for the developer of the object which registers with the runtime. It is assumed that developers will design and implement a “Custom Service Manager” style application which is a sophisticated type of “Observer Gateway”. It could perform the following functions:

- Register itself with the runtime specifying some unique “scheme” tag.
- Manage a pool of objects which implement the custom service processing.
- Invoke the require custom service calls using whatever protocol is required.

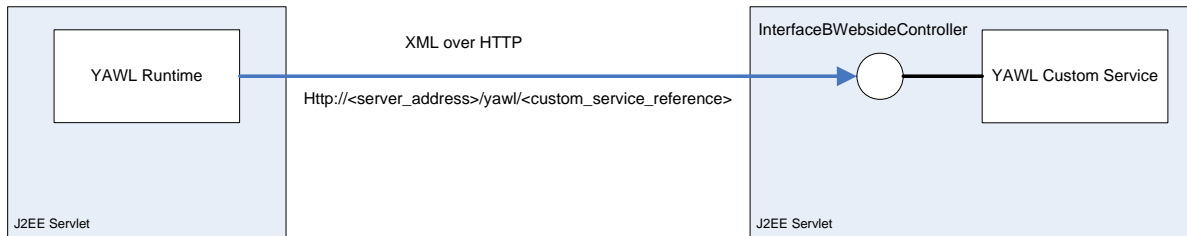
The advantage of the above approach is that developers are free to utilise whatever protocols they require, be they “standard” protocols found within the J2SE and J2EE stable, or custom protocols such as those required to communicate with legacy systems, possibly utilising non-Java based resources via the Java Native Interface language feature.

The URL routing between the standard custom service mechanism and that employed by the observer gateway is summarised in the following diagram:

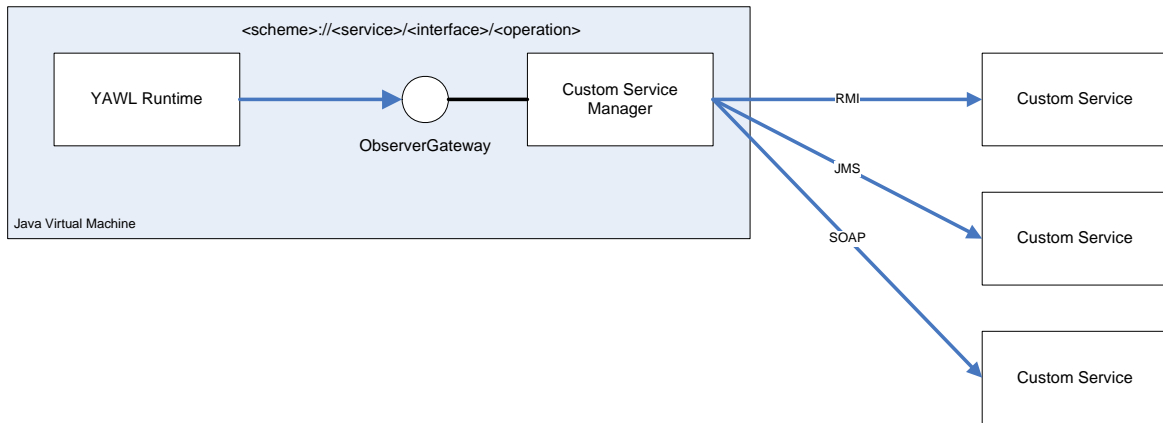
³ Although pre-registration of such services with the runtime is not required, a warning message will be generated at both design time from within the YAWL Editor, and also at runtime when the containing specification is loaded.

Filename	YAWL-ObserverGatewayConcepts.doc			Revision	Error! Unknown document property name.	Page	6 of 10
Date Created	1-Mar-04	Last Modified	13/03/2006 16:58:00				

Classic style YAWL Custom Service Implementation



Observer Gateway YAWL Custom Service Implementation



In the classic style implementation, there is a one-to-one relationship between an atomic task (and hence work item) in the YAWL runtime and a specific custom service implementation. In addition, the custom service must be invoked using XML over HTTP protocol.

In the observer gateway architecture, the relationship between work items and custom services is one-to-many. This is achieved by employing a custom developer manager class which utilises some URL parsing scheme to determine which custom service to delegate the request to. How the URL is constructed in terms of path content and number of elements is completely open ended, the only constraint being that of adherence to standard URL formatting in terms of characters utilised. Such a solution generally adheres to the REST design principle, wherein each URI represent a particular resource.

A suggested URL formatting scheme is as follows:

<scheme>://service/interface/operation

The components are described in the following table:

Component	Description
scheme	This element defines the tag that links this URL to a specific scheme of custom service manager. This is the value that the manager uses when registering itself with the runtime for receipt of notifications on the

Filename	YAWL-ObserverGatewayConcepts.doc			Revision	Error! Unknown document property name.	Page	7 of 10
Date Created	1-Mar-04	Last Modified	13/03/2006 16:58:00				

	observer gateway listener interface.
service	Describes the service as offered by the target custom service. “Order Processing” is a suitable example.
interface	Describes an interface (i.e. a collection of methods or operations) as supported by the “service”. “Provisioning” is a suitable example.
operation	Describes the operation of method to be called within the “interface”. “NewCreditOrder” is a suitable example.

2.4 Implementing the Manager Class

The use of a manager class is optional. Without an intermediate manager class, the object registering with the observer gateway will need to directly implement the business logic required to consume and process all work items where the URL associated with the workitem has the same URL “scheme” as that specified by the registering observer gateway.

For example, the following work item requests will be forwarded to observer gateways which register with a scheme of “xyz”:

- xyz://sales/order/newOrder
- xyz://sales/order/cancelOrder
- xyz://inventory/stockitem/addNewStockItem

By making the registering Observer Gateway object a form of “manager”, work item requests can be analysed and the work delegated to some other object or even external system. The routing can be performed by parsing the full URL path defined against the work item, or even by inspection of the XML data associated with the work item (this being available once the work item has been checked-out of the runtime – and hence moved from the “enabled” state to the “executing” state).

If we take the earlier URL examples, the use of a manager allows us for example to route all work requests for the “sales” domain to an external non-Java based system via some JNI wrapper interface. Work requests for the “inventory” domain however could be routed via a JMS interface to some external Java based application.

Internally within the runtime, all calls to register an object implementing the ObserverGateway interface result in a callback reference being stored within a table, and therefore many Observer Gateway objects can register with the runtime. When the runtime enables a new workitem, the atomic task definition associated with that workitem is examined and any associated custom service URL is extracted. The ObserverGatewayController within the runtime then examines all registered Observer Gateways, and if the URL scheme matches, a callback is invoked to inform all gateways for the given scheme (e.g. http) announcing the new workitem.

Filename	YAWL-ObserverGatewayConcepts.doc			Revision	Error! Unknown document property name.	Page	8 of 10
Date Created	1-Mar-04	Last Modified	13/03/2006 16:58:00				

2.4.1 The ObserverGateway Interface

The customer service manager or custom service itself (if no manager class is being utilised, must implement the interface `au.edu.qut.yawl.engine.ObserverGateway`. The Java interface is shown below for YAWL release Beta 7:

```
public interface ObserverGateway
{
    String getScheme();

    void announceWorkItem(YAWLServiceReference yawlServiceReference,
        YWorkItem yWorkItem);

    void cancelAllWorkItemsInGroupOf(YAWLServiceReference yawlServiceReference,
        YWorkItem yWorkItem);

    void announceCaseCompletion(YAWLServiceReference yawlServiceReference,
        YIdentifier yIdentifier,
        Document document);
}
```

Filename	YAWL-ObserverGatewayConcepts.doc			Revision	Error! Unknown document property name.	Page	9 of 10
Date Created	1-Mar-04	Last Modified	13/03/2006 16:58:00				

getScheme

This method should return the scheme tag that this custom service manager supports. Suggested values are as follows, but are not validated in any way by the YAWL runtime:

jms	Java Method Call
rmi	Remote Method Invocation
http	Hyper Text Transfer Protocol
jms	Java Message Service
soap	Simple Object Access Protocol

announceWorkItem

This method will be called by the YAWL runtime for new workitems where the scheme component of the service URL associated with that work item matches getScheme() on an ObserverGateway. The invocation supplies a reference to both the YAWL service URI together with the workitem.

cancelAllWorkItemsInGroupOf

This method will be called by the YAWL runtime whenever a previously announced work item for this service is cancelled. Again the call is only made where the scheme component of the task URL matches getScheme() on an Observer Gateway..

announceCaseCompletion

Upon launching a case the caller may elect any Custom YAWL Service to be notified once the case is complete. When doing so, this method gets called by the runtime once the case has completed. As usual, the call is only made for ObserverGateway instances where the scheme component of the URL matches getScheme(). The call supplies a reference to the YAWL service from which the URL can be obtained, together with a case ID, and the output case data.

2.5 Registering with the YAWL Runtime

Any custom service or custom service manager instance needs to register itself with the runtime once it has completed its initialisation and is ready to accept requests. To achieve registration, the object which implements the ObserverGateway interface needs to obtain a reference to the YAWL runtime and invoke the registration method as shown in the following code fragment:

```
YEngine engine = YEngine.getInstance(false);
engine.registerInterfaceBObserverGateway(this);
```

Once registered, the object will receive notifications for work items as they are generated. It should be noted that work items already posted against the customer service will not be re-notified, it being the responsibility of the manager to retrieve any outstanding work items and re-process these if required. To avoid such a situation, a reliable messaging framework such as JMS could be utilised to ensure requests in the form of messages are delivered to the target custom services, and avoid any requirement to re-post messages after a restart of the YAWL runtime and/or custom service manager instances.

ERROR! UNKNOWN DOCUMENT PROPERTY NAME.

Filename	YAWL-ObserverGatewayConcepts.doc						
Date Created	1-Mar-04	Last Modified	13/03/2006 16:58:00	Revision	Erro r! Unk now n docu ment prop erty nam e.	Page	10 of 10

END OF DOCUMENT