**Workflow Access Service**

**and AWD 10**

**Purpose**

The purpose of this document is to document the design decisions and approach for the Workflow Access service changes for AWD 10.

It provides a reference for consuming AWD SOAP Service. It is intended for developers who are building common service that will use AWD Web services to build custom applications. The document also provides some background on current NetServer/P3 integration.

Our existing Workflow Access service was designed to integrate workflow engine capabilities with the existing architecture. It provides access to workflow engine, passing all the information necessary to create a work and trigger defined business processes.

Additional Features:

* Integrated with Document Management System
* Provides asynchronous communication with AWD

**Problem**

Workflow service is using AWD NetServer API to communicate with AWD work management system. NetServer functionality has been replaced with native AWD10. AWD10 is backward compatible in that existing NetServer processes should continue to work with AWD10 server. However, Netserver 3.1 and 3.2 went out of active vendor support and scheduled for retirement as a product in the near future.

**How do we use AWD NetServer today**

Workflow service connects to AWD/NetServer, sends it XML input data and receives an XML response. TCP/IP protocol is used between Paris 3 and NetServer to send data. NetServer has pre-configured jobs that call one or more tasks. Also, it has ability to create/configure custom jobs with the Knowledge Enabler tool. The P3 Workflow service is using both - pre-configured and custom jobs to send requests to AWD server.

What NetServer calls are used in the integration?

These are NetServer jobs invoked by Workflow service:

CREATE\_JOB = "AWDCreate";

LOOKUP\_AND\_EXPAND\_JOB = "LookupAndExpand";    // custom Paris 3 job

LOOKUP\_JOB = "AWDLookup";

UPDATE\_JOB = "AWDUpdateWork";

ADD\_COMMENT\_JOB = "AWDAddComments";

GET\_HISTORY\_JOB = "AWDGetHistory";

RELATE\_JOB = "AWDRelate";

GET\_IMG\_TYPE\_JOB = "AWDGetImageType";

GET\_SOURCE\_LOB\_JOB = "AWDGetSource";

What specific functionality does the interface provide?

*/\*\**

*\* Adds comments to the workflow object specified by the id*

*\* and the type attributes.*

*\* Creation date: (01/24/2002 11:47:35 AM)*

*\* @param objDesc com.aegon.arch.service.workflow.WorkFlowObjectDescriptor -*

*\* contains object's ID and type.*

*\* @param commentObj com.aegon.arch.service.workflow.Comment - an object*

*\* containing the text.*

*\*/*

void addComment(WorkFlowObjectDescriptor objDesc, Comment commentObj);

*/\*\**

*\* Creates workflow object and links related items together (if any).*

*\* If an error occurs during processing, WorkFlowException will be thrown.*

*\* Object that has already been created will not be rollback.*

*\* Creation date: (01/07/2002 9:55:44 AM)*

*\* @param workFlowObject com.aegon.arch.service.workflow.WorkFlowObject - data*

*\* that will provide all the information necessary to process the request.*

*\* @return java.lang.String - an ID of created workflow object.*

*\*/*

public String createWorkFlowObject(WorkFlowObject workFlowObj);

*/\*\**

*\* Retrieves the list of comments for a particular work object.*

*\* Creation date: (01/24/2002 3:20:35 PM)*

*\* @param objDesc com.aegon.arch.service.workflow.WorkFlowObjectDescriptor -*

*\* contains object's ID and type.*

*\* @return com.aegon.arch.service.workflow.Comment[] - an array of Comment objects*

*\*/*

Comment[] getCommentHistory(WorkFlowObjectDescriptor objDesc);

*/\*\**

*\* Retrieves workflow objects that match criteria passed in.*

*\* Creation date: (01/07/2002 9:55:44 AM)*

*\* @param wfSearchObject com.aegon.arch.service.workflow.WorkFlowSearch - search*

*\* criteria*

*\* @return com.aegon.arch.service.workflow.WorkFlowObject[]*

*\*/*

public WorkFlowObject[] lookup(WorkFlowSearch wfSearchObject);

public WorkFlowObject[] lookupWithMimeType(WorkFlowSearch wfSearchObject);

*/\*\**

*\* Retrieves light workflow items that match criteria passed in.*

*\* Creation date: (01/21/2003 9:36:01 AM)*

*\* @return ArrayList*

*\* @param wfSearchObj com.aegon.arch.service.workflow.WorkFlowSearch*

*\*/*

public ArrayList lookupLightWorkItem(WorkFlowSearch wfSearchObj);

*/\*\**

*\* Creates a parent/child relationship between the workflow objects.*

*\* Creation date: (01/24/2002 11:47:35 AM)*

*\* @param parent com.aegon.arch.service.workflow.WorkFlowObjectDescriptor -*

*\* contains parent object's ID and type.*

*\* @param child com.aegon.arch.service.workflow.WorkFlowObjectDescriptor -*

*\* contains child object's ID and type.*

*\*/*

public void relate(WorkFlowObjectDescriptor parent, WorkFlowObjectDescriptor child);

*/\*\**

*\* Returns the document associated with the Source object.*

*\* Creation date: (06/19/2002 1:49:30 PM)*

*\* @return byte[] - a binary source document*

*\* @param sourceId java.lang.String -  The ID of the Source object*

*\*/*

public byte[] retrieveDocument(String sourceId);

*/\*\**

*\* Updates the specified work object.*

*\* Creation date: (01/07/2002 9:55:44 AM)*

*\* @param workFlowObject com.aegon.arch.service.workflow.WorkFlowObject - data*

*\* that will provide all the information necessary to process the request.*

*\*/*

public void updateWorkFlowObject(WorkFlowObject workFlowObj);

*/\*\**

*\* Creates a source object and uploads a source file.  If an error occurs*

*\* during processing, WorkFlowException will be thrown. Object that has*

*\* already been created will not be rolledback.*

*\* @param xmlRequest - A byte array representing xmlRequest document.*

*\* @param data - A byte array representing the source to be uploaded.*

*\* @return java.lang.String - an ID of created source object.*

*\*/*

public String uploadDocument (byte[] xmlRequest, byte[] document);

public String uploadDocument (byte[] xmlRequest, byte[] document, String workitemId, String sourceId);

*/\*\**

*\* Replaces the document associated with an AWD/Source*

*\* with the document passed in.*

*\* Creation date: (06/19/2002 1:29:51 PM)*

*\* @param sourceId java.lang.String - The ID of the Source object*

*\* @param document byte[] - A byte array containing the source document*

*\*/*

public void uploadDocument(String sourceId, byte[] document);

**Solution**

Discussed two solutions: one short term to update our service to work with the updated NetServer with AWD 10 and then a longer term solution w/o NetServer.

Based on analysis performed, decided to have our Workflow service ported to AWD’s web services.

**Goals**

The goal would be to keep our API the same, but integrate with new AWD 10 API.

**JAX-WS**

Workflow service is going to utilize JAX-WS 2.2 (Java API for XML Web Services) to access AWD’s web services. JAX-WS is part of standard JDK since version 6.

The JAX-WS 2.2 specification [JSR 224](http://jcp.org/en/jsr/detail?id=224) defines a standard Java-to-WSDL mapping which determines how WSDL operations are bound to Java methods when a SOAP message invokes a WSDL operation.

The WSDL document:

* describes web services and how to access them in XML-based format
* references an XML schema document which controls the data types that appear in SOAP requests and responses

JAX-WS Features:

* part of Java platform
* supports annotations
* web services can be called both synchronously and asynchronously
* supports resource injection
* leverages JAXB 2.0 API for data binding for mappings between Java objects and XML documents
* support for SOAP 1.2
* support for optimized transmission of binary data as specified by MTOM (it is a SOAP version1.2 feature); The raw bytes are appended to the SOAP Message and are separated by a MIME boundary.
* *wsimport* tool

**Use Case for using MTOM in Web Service**

If there is a requirement where the web service has to send and receive large number of documents (images, word documents, pdf etc.) to an enterprise system over internet, then using MTOM feature should be considered. You gain efficient network transmission by using MTOM. Consider sending a large PDF document over the wire using a web service with MTOM, the document will be transmitted outside the envelope as a MIME attachment and it is transmitted as binary data. Here there is no XML BASE64 encoding required for the document.

**Connecting to AWD Web Service**

Will use the following URL to connect to an AWD web service:

[*http://appserver/awdServer/AWDProcessingService?WSDL*](http://appserver/awdServer/AWDProcessingService?WSDL)where *appserver* is the domain name or IP address of JEE application server.

To test use *appserver*: TRSAWDDEV.US.AEGON.COM

Workflow service will make calls through a port, a local object that acts as a proxy for the remote AWD processing service. The port is created at development time by the *wsimport* tool, which generates JAX-WS artifacts based on the service’s WSDL file.

**AWD 10 Processing Web Services**

AWD web services allow client applications to create, retrieve, relate, lookup and update work and source objects. As of today, all functions have been tested using SoapUI, [open-source](http://en.wikipedia.org/wiki/Open-source_software) web service testing application, except retrieving images and files. Retrieve documents is not included into basic AWD10 package and will be provided by vendor as separate component.

**Dependencies**

Logging – Log4 J

Configuration – Apache Commons Configuration

**Packages**

com.aegon.arch.service.workflow

com.aegon.arch.service.workflow.queue

com.aegon.arch.service.workflow.util

com.aegon.arch.service.workflow.ws

com.aegon.arch.service.workflow.test