Automating Production of Cross Media Content for Multi-channel Distribution

www.AXMEDIS.org

DE3.1.2.3.12 Specification of AXMEDIS Workflow Tools, update of DE3.1.2.2.12

Version: 1.0
Date: 13-07-2007
Responsible: IRC (revised and approved coordinator)

Project Number: IST-2-511299 Project Title: AXMEDIS Deliverable Type: report Visible to User Groups: yes Visible to Affiliated: yes Visible to the Public: yes
Deliverable Number: DE3.1.2.3.12 Contractual Date of Delivery: M34 Actual Date of Delivery: 13-07-2007 Title of Deliverable: Specification of AXMEDIS Workflow Tools, first update of part of DE3.1.2 Work-Package contributing to the Deliverable: WP3.1 Task contributing to the Deliverable: WP3, WP2 Nature of the Deliverable: report Author(s): UR, HP, XIM

Abstract: this part includes the specification of components, formats, databases and protocol related to the AXMEDIS Framework area of Workflow including open flow, biztalk.

Keyword List: workflow management in axmedis, open flow, biztalk.

Please note that:

- You can become affiliated with AXMEDIS. This will give you the access to a huge amount of knowledge, information and source code related to the AXMEDIS Framework. If you are interested please contact P. Nesi at nesi@dsi.unifi.it. Once affiliated with AXMEDIS you will have the possibility of using the AXMEDIS specification and technology for your business.
- You can contribute to the improvement of AXMEDIS documents and specification by sending the contribution to P. Nesi at nesi@dsi.unifi.it
- You can attend AXMEDIS meetings that are open to public, for additional information see WWW.AXMEDIS.org or contact P. Nesi at nesi@dsi.unifi.it

Table of Content

1 Executive Summary and Report Scope

The full AXMEDIS specification document has been decomposed in the following parts:

DE number	Deliverable title	responsible
DE3.1.2.3.1	Specification of General Aspects of AXMEDIS framework AXMEDIS-DE3-1-2-3-1-Spec-of-AX-Gen-Asp-of-AXMEDIS-framework	DSI
DE3.1.2.3.2	Specification of AXMEDIS Command Manager AXMEDIS- DE3-1-2-3-2-Spec-of-AX-Cmd-Man	DSI
DE3.1.2.3.3	Specification of AXMEDIS Object Manager and Protection Processor AXMEDIS-DE3-1-2-3-3-Spec-of-AXOM-and-ProtProc	DSI
DE3.1.2.3.4	Specification of AXMEDIS Editors and Viewers AXMEDIS-DE3-1-2-3-4-Spec-of-AX-Editors-and-Viewers	DSI
DE3.1.2.3.5	Specification of External AXMEDIS Editors/Viewers and Players AXMEDIS-DE3-1-2-3-5-Spec-of-External-Editors-Viewers-Players	DSI
DE3.1.2.3.6	Specification of AXMEDIS Content Processing AXMEDIS-DE3-1-2-3-6-Spec-of-AX-Content-Processing	DSI
DE3.1.2.3.7	Specification of AXMEDIS External Processing Algorithms AXMEDIS-DE3-1-2-3-7-Spec-of-AX-External-Processing-Algorithms	FHGIGD
DE3.1.2.3.8	Specification of AXMEDIS CMS Crawling Capabilities AXMEDIS-DE3-1-2-3-8-Spec-of-AX-CMS-Crawling-Capab	DSI
DE3.1.2.3.9	Specification of AXMEDIS database and query support AXMEDIS-DE3-1-2-3-9-Spec-of-AX-database-and-query-support	EXITECH
DE3.1.2.3.10	Specification of AXMEDIS P2P tools, AXEPTool and AXMEDIS AXMEDIS-DE3-1-2-3-10-Spec-of-AXEPTool-and-AXMEDIA-tools	DSI
DE3.1.2.3.11	Specification of AXMEDIS Programme and Publication tools AXMEDIS-DE3-1-2-3-11-Spec-of-AX-Progr-and-Pub-tool	UNIVLEEDS
DE3.1.2.3.12	Specification of AXMEDIS Workflow Tools AXMEDIS-DE3-1-2-3-12-Spec-of-AX-Workflow-Tools	UR
DE3.1.2.3.13	Specification of AXMEDIS Certifier and Supervisor and networks of AXCS AXMEDIS-DE3-1-2-3-13-Spec-of-AXCS-and-networks	DSI
DE3.1.2.3.14	Specification of AXMEDIS Protection Support AXMEDIS-DE3-1-2-3-14-Spec-of-AX-Protection-Support	UPC
DE3.1.2.3.15	Specification of AXMEDIS accounting and reporting AXMEDIS-DE3-1-2-3-15-Spec-of-AX-Accounting-and-Reporting	EXITECH

1.1 This document concerns

DE3.1.2.3.12 Specification of AXMEDIS Workflow Tools, update of part of DE3.1.2 AXMEDIS of the above list

1.2 List of Modules or Executable Tools Specified in this document

A module is a component that can be or it is reused in other cases or points of the AXMEDIS framework or of other AXMEDIS based solutions.
The modules/tools have to include effective components and/or tools and also testing components and tools.

Module/tool Name	Module/Tool Description and purpose, state also in which other AXMEDIS area is used	Standards exploited if any
Workflow Engine – Openflow	Workflow Engine forms the basis of the workflow support in the AXMEDIS Environment. It will store the rules, roles, activities, etc for the workflow	
Workflow Engine – Biztalk	As above.	
Workflow User Interface – Openflow	This forms the basis for the user to interact with the openflow workflow engine	
Workflow Request Adaptors - Openflow	The Request Adapters are the communication channels used by the WorkFlow Engine for issuing the request to the above mentioned AXMEDIS components. This will ensure that when a task inside an activity has to execute some AXMEDIS component, the proper Adapter will be called for invoking the correct method through the Plug-ins	
Workflow Request Adaptors – BizTalk Server	The function of Request Adaptor for BizTalk server is same as for Openflow. The only difference is the way they are implementated	
Workflow Input Queue Adaptor – Openflow	The Input Queue Adapters are the communication channels used by the WorkFlow Engine for receiving the responses of previously issued requests from the above mentioned AXMEDIS components: that is when a response is received, the awaiting pending activity will be resumed by the WorkFlow Engine and the response properly checked in order to continue with following activities in the process flow instance.	
Workflow Input Queue Adaptors - Biztalk	The function of Input queue Adaptor for BizTalk server is same as for Openflow. The only difference is the way they are implementated	
Workflow Request Gateways	The WF Request Adapter (Openflow) sends the requests via an http GET call. This http GET call is received by a Web Server running Mictosoft IIS and directed to an ASP process called WF Request Gateway. This ASP process decodes the GET requests and formats a WebService request towards the proper AXMEDIS module. The BizTalk WF Request Adapter communicates directly with the AXMEDIS module, via Web Services.	
Workflow Response Gateways	When an AXMEDIS tool/engine wants to send back responses to the WorkFlow (i.e. Notifications), it calls a WebServices in the WF Response Gateway. The WF Response Gateway encodes the response in an XMLRPC call directed to the WF Input Queue Adapter.	
Workflow Plugins	There are three workflow plugins that are loaded using AXOM's Command and Reporting. These plugins forms the interface with AXMEDIS Editor/Viewer, Engines and Rule Editor/Viewer. They form basis for these tools to communicate with the request and response gateways Originally, the were separate versions of these plug-ins for each WfMS, but they are now unified.	

1.3 List of Formats Specified in this document

A format can be (i) an XML content file for modeling some information, (ii) a file format for storing information, (iii) a format that is manipulated by the tools described in this document, etc...

Format Name	Format Description and purpose, state also in which other modules is used	Standards exploited if any
Config.XML	An XML file consisting of various configuration settings for the workflow Gateways to identify various URLs or IP addresses for issues calls.	
AX_adataper.cgf	Text file consisting of URLs for various workflow Request Gateway methods	

1.4 List of Databases Specified in this document

Database Name	database Description and purpose, state also in which other AXMEDIS area is using	Standards exploited if any
AXWFDB	AXWFDB is the internal database for the workflow engine to store various information essentially Process, Activity, Transition, Instance, Workitem, Employee, Role, History.	

1.5 List of Protocols Specified in this document

A protocol is a communication modality among distinct processes that can be located or not on different computers.

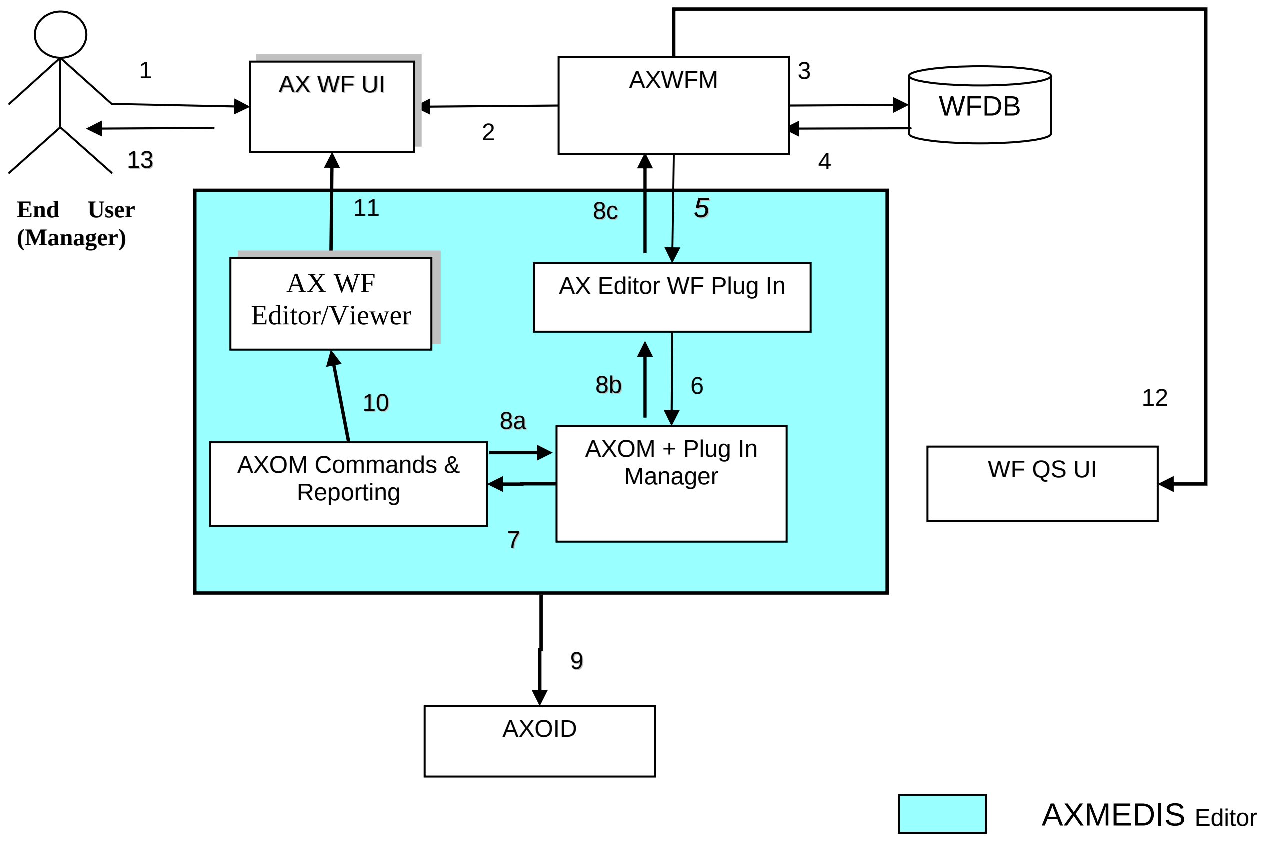
Protocol Name	protocol Description and purpose, state also in which other modules is used	Who is the master and who is the slave	Standards exploited if any
axWFReq	http GET protocol is used to issues a workflow request to activate any AXMEDIS tool or to retrive information about any AXMEDIS Tools. The http GET request is issued from the Workflow reqeust adaptor to workflow request gateway.	Workflow Request Adaptor – Master Workflow Request Gateway – Slave	
AxWfEditorReq	Webservice calls are used to pass on the workflow request coming from the reqeust adaptor and converting them into a generalised function call towards AXMEDIS Object Editor (DRM, Resource, Meta-Data, etc) using Webservices	Reqeust Gateway – Master AXMEDIS Object Editor – Slave	
AxWfEngineReq	Webservice calls are used to pass on the workflow request coming from the reqeust adaptor and converting them into a generalised function call towards AXMEDIS Engines (AXCP, AXEPTool, PnP, etc) using Webservices	Reqeust Gateway – Master AXMEDIS Engines – Slave e.g. AXCP, Pnp	
AxWfReReq	Webservice calls are used to pass on the workflow request coming from the reqeust adaptor and converting them into a generalised function call towards AXMEDIS Rule Editors (AXCP, Program) using Webservices	Reqeust Gateway – Master AXMEDIS Rule Editor– Slave	

AxWfDbReq	Webservice calls are used to pass on the workflow request coming from the reqeust adaptor and converting them into a generalised function call towards AXMEDIS Database Query Support using Webservices	Reqeust Gateway – Master AXMEDIS Database Query Support– Slave	
AxWfEditorRes	Webservice calls are also used to pass on the request coming from the AXMEDIS Object Editor (DRM, Resource, Meta-Data, etc) towards the workflow. The calls are recieved by the response gateways.	Response Gateway – Slave AXMEDIS Editor – Master	
AxWfEngineRes	Webservice calls are also used to pass on the request coming from the AXMEDIS Engines (AXCP, AXEPTool, PnP, etc) towards the workflow. The calls are recieved by the response gateways.	Response Gateway – Slave AXMEDIS Engine – Master	
AxWfReRes	Webservice calls are also used to pass on the request coming from the AXMEDIS Rule Editors (AXCP, Program) towards the workflow. The calls are recieved by the response gateways.	Response Gateway – Slave AXMEDIS Rule Editor – Master	
AxWfDbRes	Webservice calls are also used to pass on the request coming from the AXMEDIS Database Query Support towards the workflow. The calls are recieved by the response gateways.	Response Gateway – Slave AXMEDIS Database Query Support – Master	
axWRFRes	The workflow response gateways, uses XMLRPC calls to initialise a workflow process and serve the request coming form AXMEDIS Tools	Respone Gateway – Master Input Queue Adaptor – Slave	

2 General Use Cases and scenarios

2.1 Scenario for Starting New Instance of an NPD

Scenario 1: Starting a New Instance of an NPD i.e. New NPD Set-up (Managerial Task)



Scenario 1: - Starting a New Instance of an NPD i.e. New NPD Set-up (A Managerial Task)

There are times when a user may wish to cause a new workflow process to be set up by "development and configuration technicians" to support new kinds of NPD (New Product Development) with new business process logics. However this scenario relates to occasions when through the Workflow UI, project managers may wish to start a new NPD instance of an already defined workflow process (e.g. the process for producing a new media content, which has been previously defined and configured).
A project manager can thus subsequently assign work activities to individual users or let the assignments be made automatically by the workflow engine, based on pre-defined rules and roles.

The following scenario describes the process of defining a new NPD within the workflow sub-system. At the end of this scenario, the project manager can expect a fully configured workspace that can be interrogated by users at various levels to give information about all the necessary tasks to be performed, people responsible for performing those tasks, the tools needed to do the tasks, and the location where each task is to be performed, etc, the scenario proceeds as follows:

1) The user (Manager) is already authenticated and logged into the system.

2, 3, 4) He invokes the "create new NPD workspace" function by clicking on this button to define the product and the NPD for its development. A pop-up dialogue box appears to allow him to enter the basic details of the NPD (e.g. name, type, etc) and to select pre-defined templates.

5, 6) The workflow manager (AXWFM) communicates with the AXMEDIS Object Manager (AXOM) through the AXMEDIS editor workflow plug-in, to generate a Process ID which is to be assigned to the new NPD.

7) The workflow editor/viewer is then launched to enable the user to define the workflow for the new NPD.

10) The workflow editor launches a blank page (or a page containing the structure of the selected template) for defining the workflow components.

13) NPD set-up phase: The user can now select and add components to define the new project. These components can be tasks, people, project, products, objects, places, links, etc. This functionality of the workflow editor is similar to a drawing utility provided by the Microsoft Word editor, which allows the user to add shapes and assign properties to them. However, for the workflow editor it is necessary that all the added components must be connected to at least one other component to form a semantic integration of all the components, which when executed in the defined order produces the required product. Whenever any component is added to the NPD the corresponding properties dialogue box appears for the added component. The user can (re)set the required properties in this dialogue box so as to control the behaviour of the components.

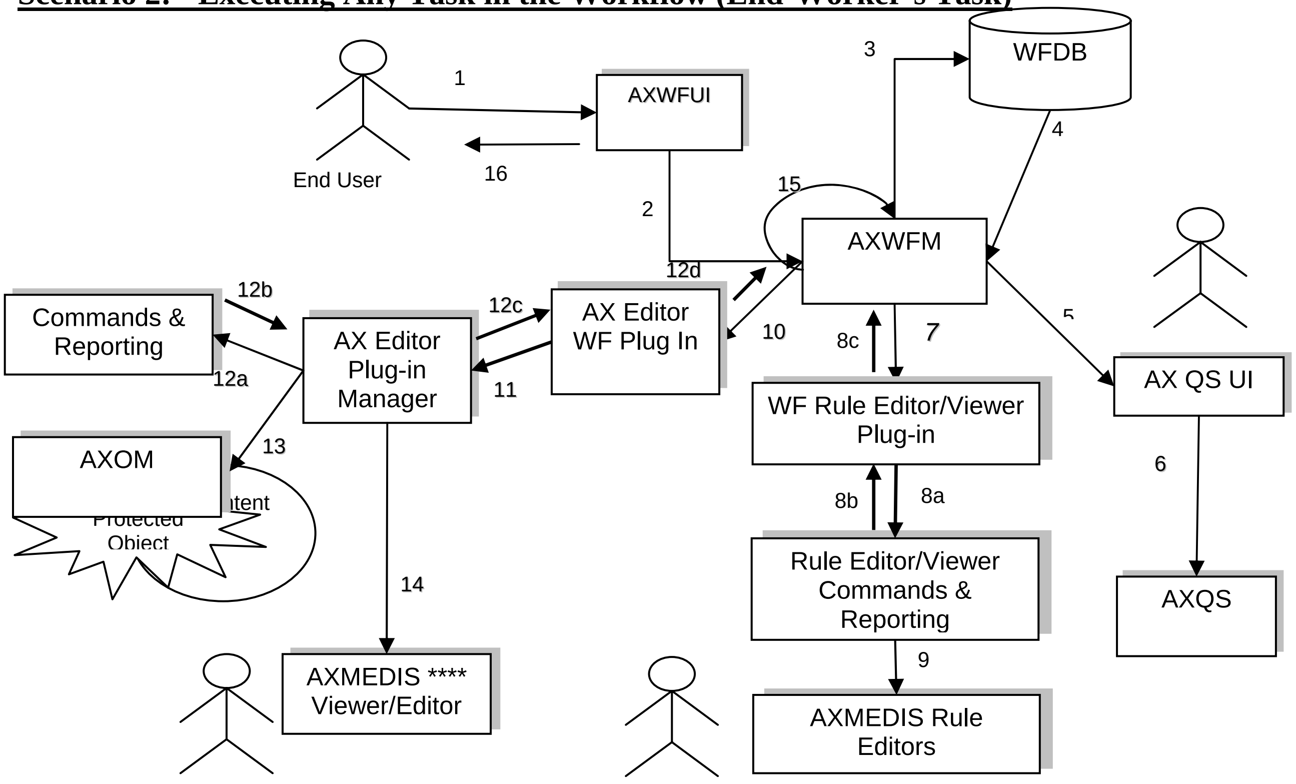
For example the user can add a task to the current NPD workspace and may designate its type as a “Formatting Task”. The user then can add a person to the current workspace and assign his role to be, say, the “Technical Editor” responsible for the Formatting Task. He can then link this person to the “Formatting Task”. The user can then add a tool to the current workspace and assign its role as a “Formatting Tool” and then link this tool to the “Formatting Task”. The workflow will interpret these links as “The AXMEDIS Object(ID---) to be formatted with the specific Formatting Tool by the named Technical Editor” thus assigned this task.

It is also possible for the User to define all the tasks and people working on the project first without creating the links. As mentioned before the workflow system can automatically distribute the work to the people, partners, places, etc based on the saved profiles (roles) of the available participating resources and objects.

There are typically two approaches to defining workflow processes: using a specific User Interface or describing the process via a meta-language (e.g. XPDL); workflow solutions tend to adopt one or the other of the two approaches).

The command and Reporting is shown in this diagram explicitly as the component that is connected to the editor/viewer to notify termination of the editing and viewing task. In practice, the Command and Reporting module can be viewed as an integral component of AXMEDIS Editor.

2.2 Scenario for Executing Any Task in Workflow


Scenario 2: - Executing Any Task in the Workflow (End-Worker's Task)
End User
AXWFUI
WFDB
AXWFM
Commands & Reporting
AX Editor Plug-in Manager
AX Editor WF Plug In
AXOM
Protected Object
WF Rule Editor/Viewer Plug-in
Rule Editor/Viewer Commands & Reporting
AXMEDIS **** Viewer/Editor
AXMEDIS Rule Editors
AX QS UI
AXQS
1
2
3
4
5
6
7
8a
8b
8c
9
10
11
12a
12b
12c
12d
13
14
15
16

Scenario 2: - Executing Any Task in the Workflow (End-Worker's Task)

This scenario describes the process of executing any work Task within the workflow environment. At the end of this scenario, the user can expect the status of the AXMEDIS Object(s) concerned to be updated and the work Task marked as completed thus triggering new sets of tasks as appropriate. This scenario proceeds as follows:

1) The user (Worker) is already authenticated and logged into the system and the workflow system is up and running. We can therefore assume that the client's (i.e. session-owner's) credentials_ID has been made available to be authenticated for sign-on to initiate the required exchanges with the AXMEDIS tools/engines as service providers and thus the information for authentication and billing purposes has been provided.

2, 3, 4) The user invokes the list-work function by clicking on the button list-work supplying a workflow_Instance_ID which effectively represents a given NPD, then selecting a work-item to get the choice of actions to be performed on the work-item for the NPD (or, identically, the workflow-instance) for which he is assigned to perform tasks.

7, 8, 9, 10, 11, 12, 13) For any selected workitem, from any given workflow-instance, the Workflow UI displays to the user a choice of available actions and descriptions/suggestions related to the selected work-item (i.e. viewed dynamically these are potential workspace instantiations). These can include actions such as:

Edit: The user may wish to invoke the AXMEDIS Editor by clicking on Edit and, say, invoke Edit DRM to Edit the DRM of a selected object; this will launch the AXMEDIS DRM Editor.

5, 6) **Search:** The user may wish to search for all objects involved in a particular NPD, by invoking the Search function of the Workflow UI. The user clicks on Search and then supplies the workflow_Instance_ID. The Workflow Manager passes this query via the Workflow Query Interface through to the Query Support Web Services Interface which submits it to the AXMEDIS Query Support

User Interface. This sets up an interaction with the AXMEDIS Object Database to search for all objects involved in the specified process or fulfilling certain criteria.

Show: The user can request the workflow system to show more information on any selected components (AXMEDIS object(s), tool(s), etc) as may be included in the work-list.

Terminate Activity: Users can invoke this functionality to signal to the workflow system their wish to have an activity terminated. Accordingly the workflow system will proceed to the next step in the workflow process instance (It is important to note that this functionality enables an over-ride control action on the part of the human operator if required).

Based on the selected Action the workflow system launches the required tool using the appropriate Interface (e.g. Web-services) or plug-ins associated with that tool. If the tool is in the exclusive access area of the user, the “Check-in” and “Check-out” interfaces will be invoked.

The workflow system assigns a time-stamp to such an Action as the start_time, which is later referred to while tracking the history of the component.

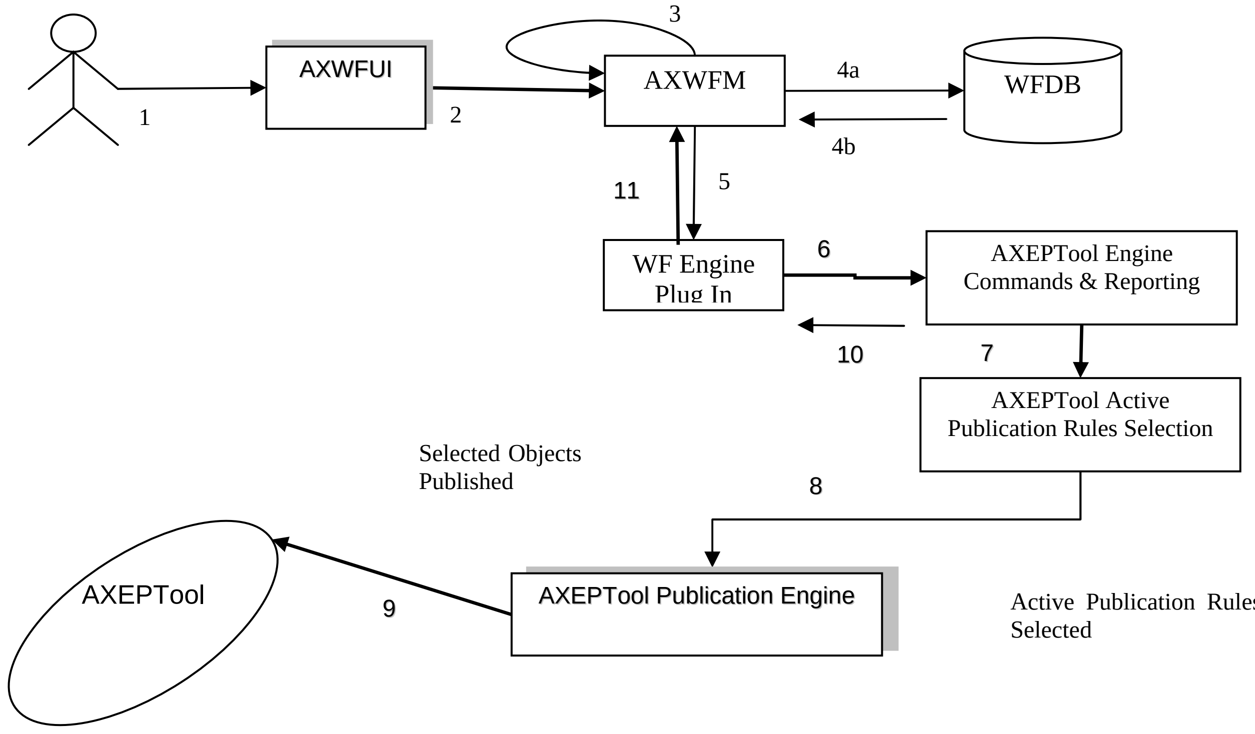
If required the workflow system will also generate new versions of the AXMEDIS Object. Upon the completion of the Task the workflow system will again assign a time-stamp to this Task as the end time.

At the end of the Action, the workflow system will update the status of the AXMEDIS Object, which may trigger various other tasks (e.g. DRM editing, invoking AXEPTool, etc).

In all the following scenario diagrams as presented in this section, the Command and Reporting Module wherever applicable has been shown explicitly as the AXMEDIS component that mediates as a gateway linking the AXWFM to the relevant service provider for NOTIFICATIONS i.e. to notify of the termination of the Requested tasks. In practice, the Command and Reporting Module can be viewed as an integral component of the AXMEDIS Editor.

2.3 Scenario to Invoke AXEPTool (Publish)

Scenario 3: Invoking the AXEPTool (Publish)


3
AXWFUI
AXWFM
4a
WFDB
1
2
4b
11
5
WF Engine Plug In
6
AXEPTool Engine Commands & Reporting
10
7
AXEPTool Active Publication Rules Selection
Selected Objects Published
8
AXEPTool
9
AXEPTool Publication Engine
Active Publication Rules Selected

Scenario 3: Invoking the AXEPTool (Publish)

This scenario describes the interaction between the workflow and the AXEPTool to share any AXMEDIS Object over the P2P network. There are two possible interaction scenarios between the workflow and the AXEPTool. On the one side, the interaction can be for uploading (Publishing) of some AXMEDIS Object(s), while on the other side the interaction can be for downloading (Loading) of an AXMEDIS Object. The Loading operation may involve a Negotiation Phase to procure an appropriate licence. Such Negotiation is controlled by a subsystem of AXMEDIS workflow called Negotiation Workflow. In this section we deal with the AXEPTool Publication Scenario.

It is assumed that the publications tasks are normally carried out asynchronously and autonomously, without the intervention of the user. Moreover, the workflow instance contains the Task for uploading of the AXMEDIS Object on the sender's side and downloading of the AXMEDIS Object on the receiver's side. The Scenario Proceeds as follows:

1) The workflow system is up and running.

2, 3, 4) We can also assume that the client's (i.e. session-owner's) credentials_ID has been made available to be authenticated for sign-on to initiate the required exchanges information for authentication and billing purposes.

5, 6, 7, 8) The workflow system passes the so-called Active Publication Request via the workflow plug-in to the AXEPTool Command and Reporting module to trigger the AXEPTool Active Publications Rule Selection Module which enables the selection and submission of appropriate objects for publication. The publication engine uses the thus activated publication request together with a AXRID to control the publication of the right object(s) from the Active List.

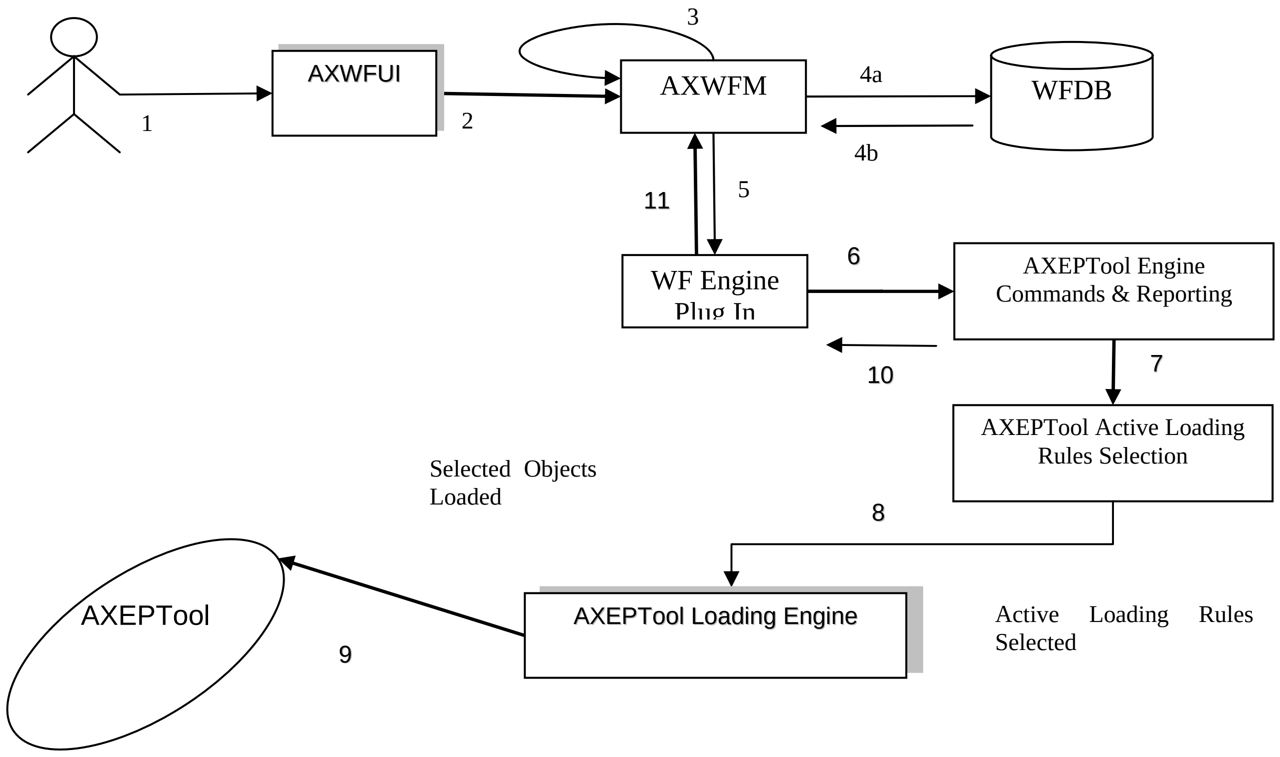
If the AXEPTool is not up and running, it is launched by the workflow system using the appropriate interfaces.

9) The AXEPTool Publication engine then moves the relevant AXMEDIS Object(s) to the “AXEPTool Out AXMEDIS Database Area” for publication on the P2P network under the control of the specified AXRID.

10, 11) Upon completion of the activity, the AXEPTool Publication engine informs the AXWFM via the AXEPTool Command and Reporting Module about the completion of the process, so it can proceed with the next step in the workflow-instance flow.

2.4 Scenario to Invoke AXEPTool (Load)

Scenario 4: Invoking the AXEPTool (Load)


3
AXWFUI
AXWFM
4a
WFDB
1
2
4b
11
5
6
WF Engine Plug In
AXEPTool Engine Commands & Reporting
10
7
AXEPTool Active Loading Rules Selection
Selected Objects Loaded
8
AXEPTool
AXEPTool Loading Engine
Active Loading Rules Selected
9

Scenario 4: Invoking the AXEPTool (Load)

This scenario describes the interaction between the workflow and the AXEPTool to share any AXMEDIS Object over the P2P network. There are two possible interaction scenarios between the workflow and the AXEPTool. On the one hand, the interaction can be for uploading (Publishing) of some AXMEDIS Object(s), while on the other hand the interaction can be for downloading (Loading) of the AXMEDIS Object. The Loading operation may involve a Negotiation Phase to procure an appropriate licence. Such Negotiation is controlled by a subsystem of AXMEDIS workflow called Negotiation Workflow. In this section we deal with a Loading operation not requiring the invocation of the Negotiation workflow for Licence Procurement. The scenario proceeds as follows:

1) It is assumed that the user is interacting with the Workflow Management System.
Thus the user (Worker) is already authenticated and logged into the system and the workflow system is up and running.

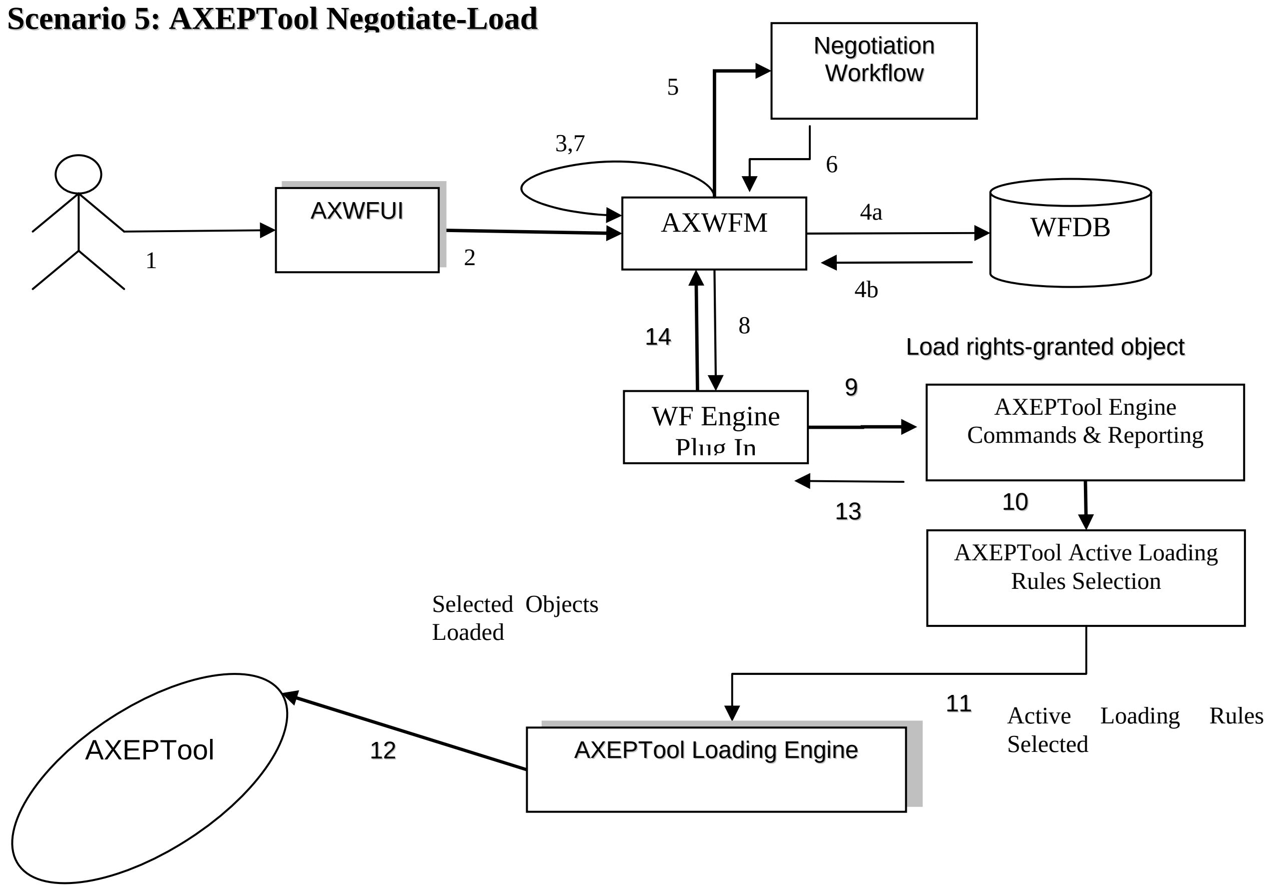
2, 3, 4) We can also assume that the client's (i.e. session-owner's) credentials_ID has been made available to be authenticated for sign-on to initiate the required exchanges information for authentication and billing purposes.

5, 6, 7, 8) If any Task requires downloading of an AXMEDIS Object from the P2P network, the workflow system passes this request via the workflow plug-in to the AXEPTool Command and Reporting module to trigger the AXEPTool Active Loading Rule Selection Module which enables the selection and downloading of appropriate objects. The Loading engine uses the thus activated loading request together with a AXRID to control the downloading of the right object(s) from the Active List.

9) As soon as the required object is available on the P2P network, the Loading Tool Engine of AXEPTool downloads this AXMEDIS Object and moves it to the "AXEPTool In AXMEDIS Database Area".

10, 11) When the AXEPTool Loading engine has completed the transfer, it informs the workflow system via the AXEPTool Command and Reporting module and the AXWFM then moves this component to the appropriate location and proceeds to enable further tasks that could be performed once the object has become available.

2.5 Scenario for AXEPTool Negotiate-Load


Scenario 5: AXEPTool Negotiate-Load
Negotiation Workflow
5
3,7
6
AXWFUI
AXWFM
4a
WFDB
1
2
4b
14
8
Load rights-granted object
9
WF Engine Plug In
AXEPTool Engine Commands & Reporting
13
10
AXEPTool Active Loading Rules Selection
Selected Objects Loaded
11
Active Loading Rules Selected
AXEPTool
12
AXEPTool Loading Engine

Scenario 5: Invoking the AXEPTool (Load Upon Completion of Negotiation)

This scenario describes the interaction between the workflow and the AXEPTool for downloading (Loading) of an AXMEDIS Object when such Loading requires Negotiation as controlled by a subsystem of AXMEDIS workflow called Negotiation Workflow. In this section we deal with a Loading operation requiring the invocation of the Negotiation workflow for Licence Procurement. The Scenario proceeds as follows:

1) It is assumed that the user is in direct interaction with the Workflow.

We can also assume that the client's (i.e. user/session-owner's) credentials_ID has been made available to be authenticated for sign-on to initiate the required exchange information for authentication and billing purposes.

2, 3, 4) Thus the user (Worker) is already authenticated and logged onto the system and the workflow system is up and running. It is assumed that the user is in interaction with the Workflow System and that the AXWF contains the AXMEDIS Object Licence Procurement Negotiation Workflow.

Suppose there is a Task that requires the downloading of an AXMEDIS Object from the P2P network, with the added complication that the user is required to enter into and complete a Negotiation Phase regarding the Licence Procurement of a particular AXMEDIS Object.

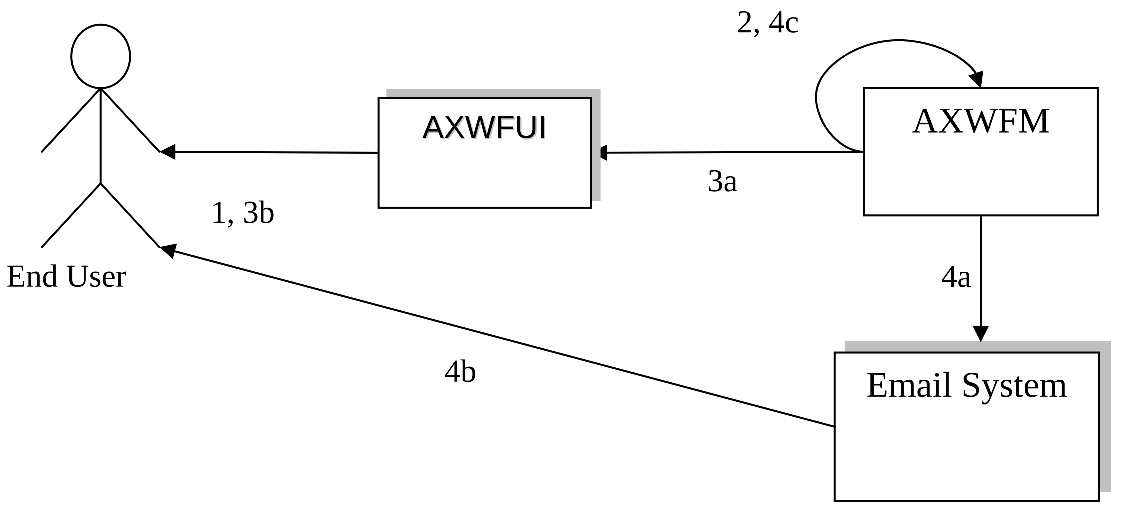
5, 6, 7) The workflow system passes the relevant AXOID through to the AXMEDIS Query Support User Interface to set up an interaction with the AXMEDIS Object Manager (AXOM). The relevant Object Licensing particulars that thus become available are then passed to the Licence Procurement Negotiation Workflow to trigger the start of the Negotiation Phase.

8, 9, 10, 11, 12) Once the Negotiation Phase is completed the AXOID is passed to the AXEPTool Loading Engine as usual, using the WF Plug-in, through the AXEPTool Command and Reporting module which enables a link to the AXEPTool Active Loading Rule Selection list. Once the selection of the relevant rule(s) for the download of the object is completed, then, as soon as the required Object becomes available on the P2P network, the Loading Tool Engine of AXEPTool downloads this AXMEDIS Object and moves it to the "AXEPTool In the AXMEDIS Database Area".

13, 14) When the AXEPTool Loading engine has completed the transfer, it informs the workflow system via the AXEPTool Command and Reporting module and the AXWFM then moves this component to the appropriate location and proceeds to enable further tasks that could be performed once the object download has been completed.

2.6 Scenario for Sending Notifications

Scenario 6: Sending out Notifications to People



Scenario 6: Sending out Notifications to People

This scenario describes the process of sending out Notifications initiated by the workflow system or by the people within the workflow environment. At the end of this scenario, the user can expect that notifications are generated and sent to appropriate target(s). The scenario proceeds as follows:

1) The user (Worker) is already authenticated and logged into the system and the workflow system is up and running. We can thus also assume that the client's (i.e. session-owner's) credentials_ID has been made available to be authenticated for sign-on to initiate the required exchanges information for authentication and billing purposes.

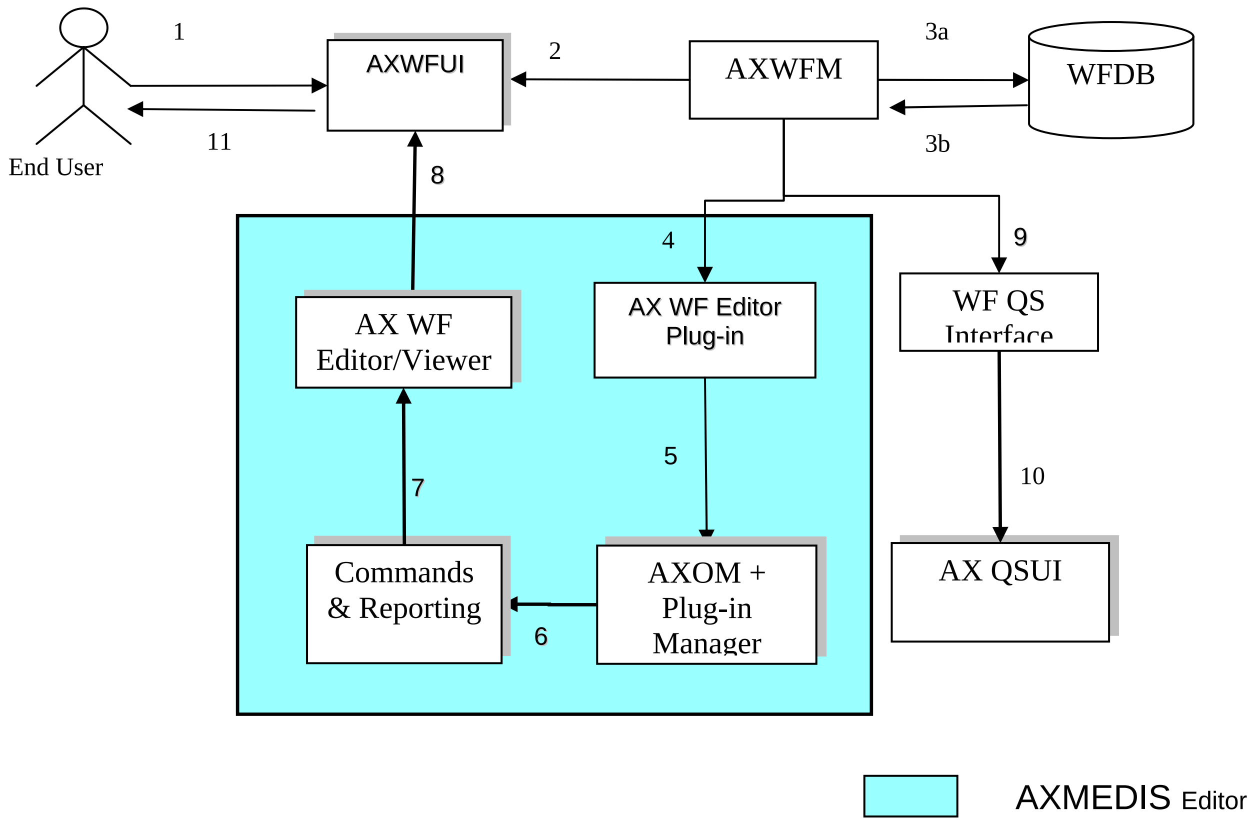
2, 3) Upon completion of any Task, the workflow system will generate appropriate Notifications, e.g. if any Task is waiting for the DRM to be cleared, the workflow system will notify this Task by raising the appropriate signal whenever the required DRMs are cleared.

4) The workflow system can also send out notifications to the users through appropriate tools like e-mailing systems, pop-up messages, etc. e.g. if any actor is waiting for an AXMEDIS object to be downloaded by the AXEPTool, then upon completion of this the Workflow system is notified by the respective Command and Reporting module and it in turn can deliver a pop-up message on the relevant client screen or other designated terminal.

Notifications can also be sent out in the form of e-mails to the user, e.g. if the user has been assigned a new Task, an email will be sent to him regarding this Task and his personal work-list is updated accordingly.

2.7 Scenario for Global View and Tracking

Scenario 7: Global View and Tracking of any Component in the Workflow



Scenario 7: Global View and Tracking of any Component in the Workflow

This scenario describes the process of generating a global view of any NPD and the tracking of any component within the selected NPD. At the end of this scenario, the user can expect to have the up-to-date progress status of the AXMEDIS Objects within the selected NPD.

1) The user (Manager/Worker with appropriate rights) is already authenticated and logged into the system and the workflow system is up and running. Therefore we can assume that the client's (i.e. session-owner's) credentials_ID has been made available to be authenticated for sign-on to initiate the required exchanges information for authentication and billing purposes.

2) The user selects a particular NPD (or identically a workflow_Instance_ID) and clicks on the Global View icon.

3, 9, 10) The Workflow system identifies all the components for the selected NPD and launches a set of queries to retrieve information for all of such components from the AXOM through the AXMEDIS Query Support Interface.

4, 5, 6, 7, 8) The workflow systems can then launch an Interactive GUI (Workflow viewer) to show the overall status of the NPD workflow along with its Critical Path Tasks (CPA), based on the results received for the above queries.

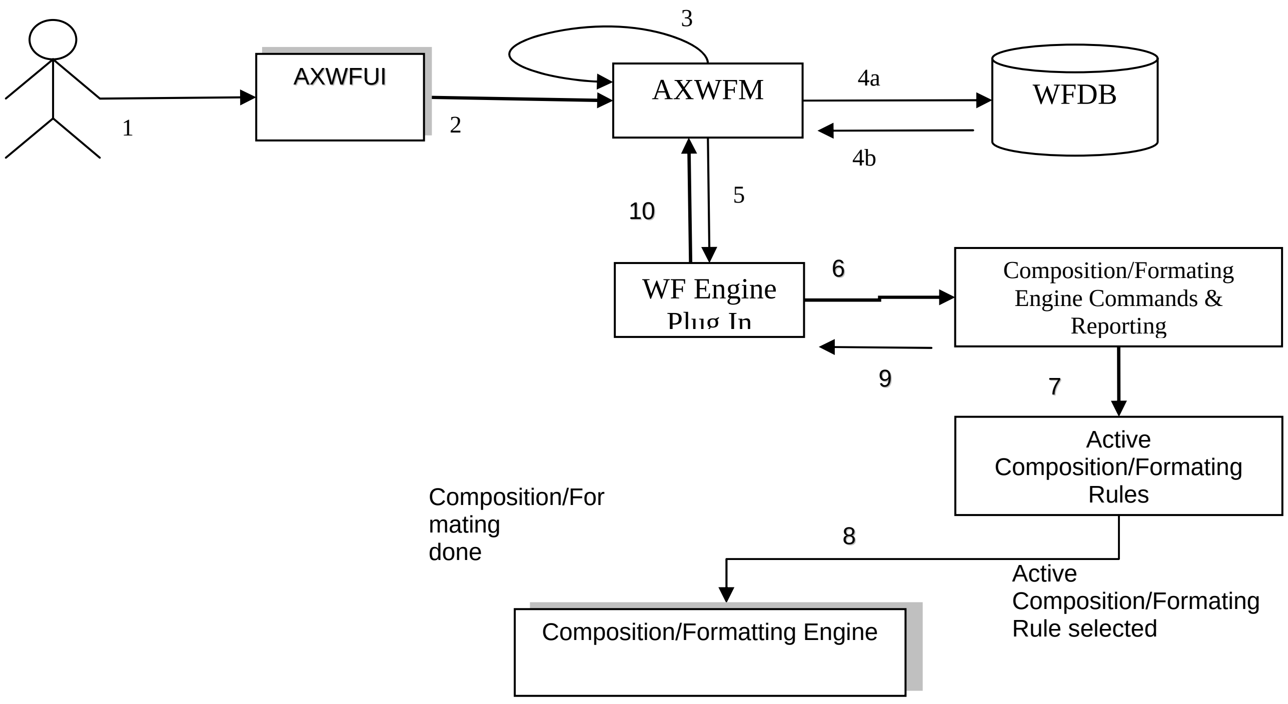
11) Through the interactive GUI, the user can select any individual component and can demand more information on it. This component can be any object, task, person, etc.

Accordingly the workflow system can launch a responsive query to retrieve detailed information regarding the component(s) selected by the user.

The Command and Reporting is shown in the scenario diagram explicitly as the component that is connected to the editor/viewer to notify termination of the editing and viewing task. In practice, the Command and Reporting module can be viewed as an integral component of AXMEDIS Editor.

2.8 Scenario for Invoking the Composition/Formatting Engine

Scenario 8: Invoking the Composition/Formatting Engine



3
AXWFUI
AXWFM
4a
WFDB
1
2
4b
5
10
6
WF Engine Plug In
Composition/Formating Engine Commands & Reporting
9
7
Active Composition/Formating Rules
Composition/Formating done
8
Active Composition/Formating Rule selected
Composition/Formatting Engine

Scenario 8: Invoking the Composition and Formatting Engine

This scenario describes the interaction between the workflow and the Composition and Formatting Engine to compose/format any AXMEDIS Object according to selected composition/formatting rules (Rule-ID).

It is assumed that the composition and/or formatting task(s) can be carried out autonomously, without the intervention of the user but it can be done on an ad hoc basis synchronously at the user's instant request. In any event we can also assume that the client's (i.e. project-owner's) credentials_ID has been made available to be authenticated for sign-on to initiate the required exchanges information for authentication and billing purposes.

1) The workflow system is up and running.

3, 4, 5, 6, 7, 8) The workflow system effects the request to the Composition/Formatting engine via the Workflow Plug-in linking through the Command and Reporting Module through to the Composition and Formatting Active Rules module. In this way the workflow system passes to the Composition and Formatting engine an Activate compose/format request together with a composition/Formatting AXRID and AXOID to control the correct composition/formatting of the right object(s) from the Active List.

The Composition/Formatting Engine then composes/formats the relevant AXMEDIS Object(s) as required per specified (or default) composition/formatting rules

9, 10) Upon completion of the composition/formatting, the AXWFM is informed by the Command and Reporting Module and the metadata of the relevant Object is also updated accordingly.

3 General architecture and relationships among the modules produced

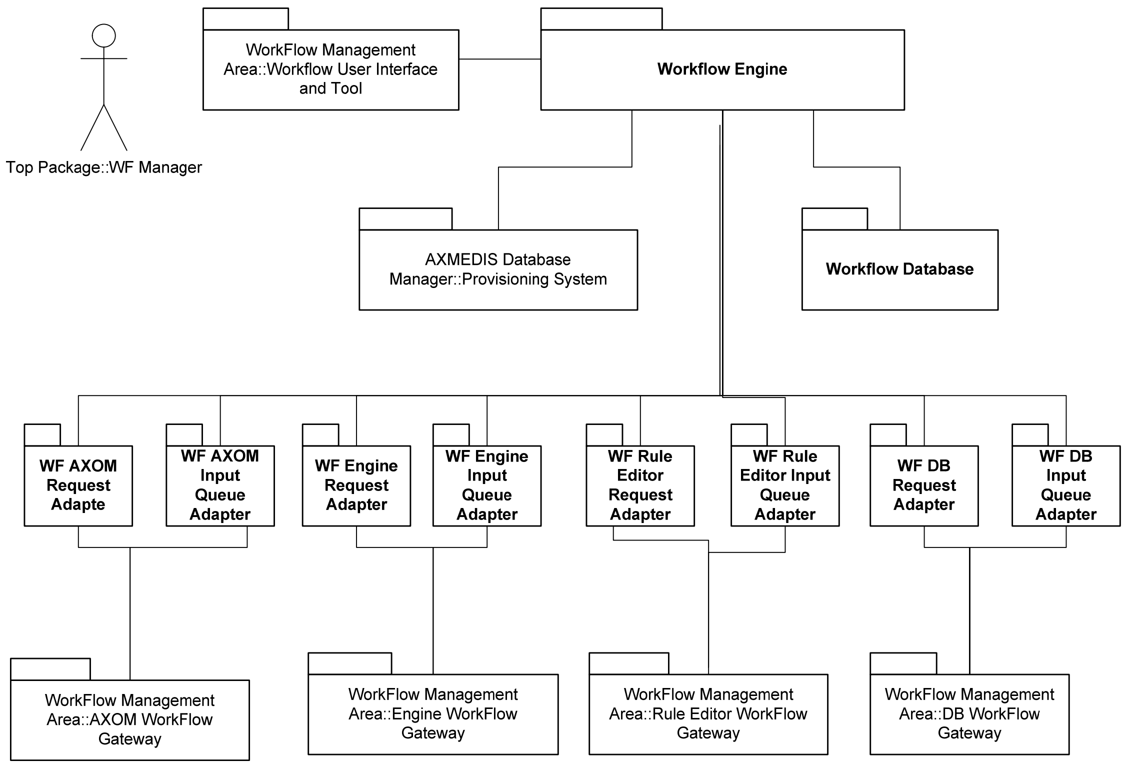
The AXMEDIS WorkFLow Area includes:

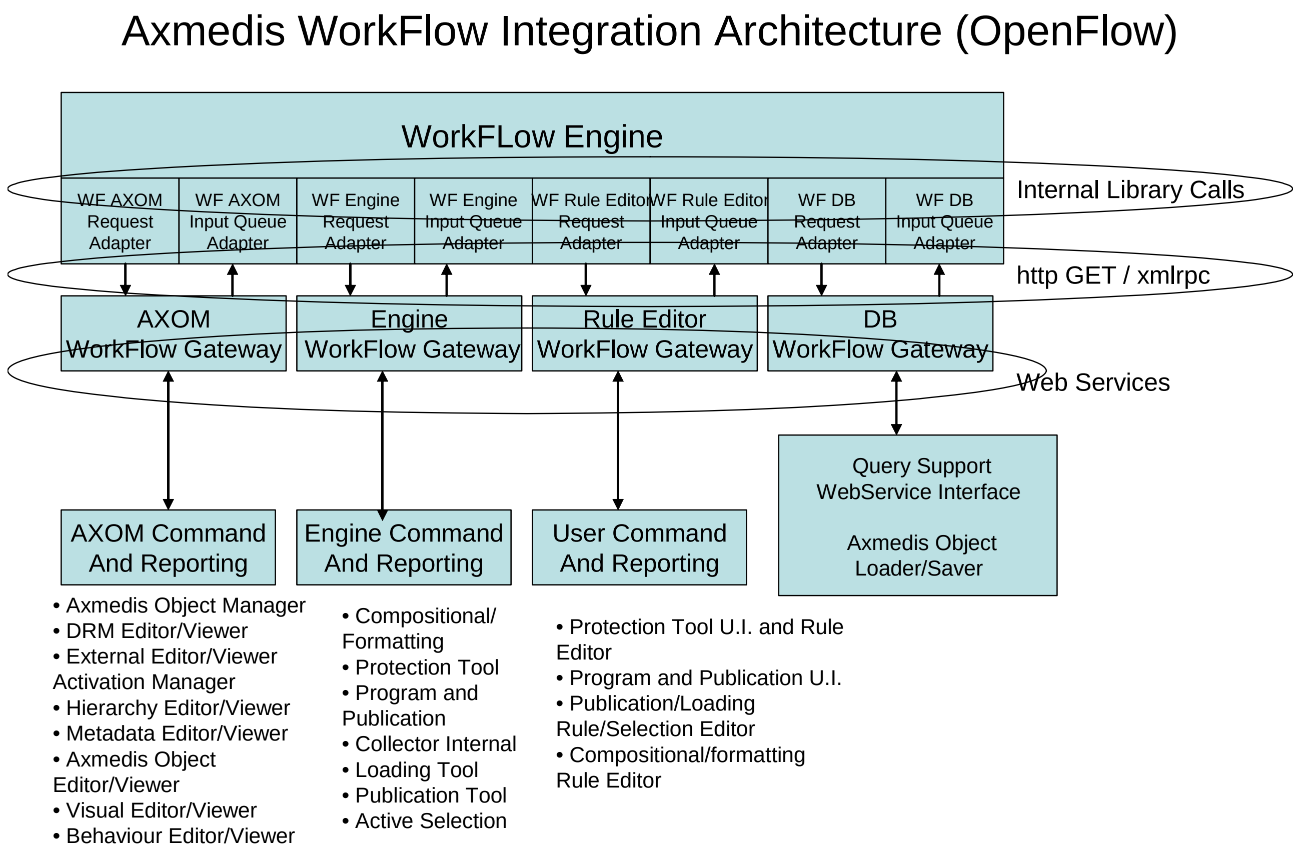
- WorkFLow Management User Interface and Tool
- WorkFlow Engine
- WorkFlow DataBase
- WF AXOM Request Adapter
- WF AXOM Input Queue Adapter
- WF Engine Request Adapter
- WF Engine Input Queue Adapter
- WF Rule Editor Request Adapter
- WF Rule Editor Input Queue Adapter
- WF DB Request Adapter
- WF DB Input Queue Adapter
- AXOM WorkFlow Gateway
- E ngine WorkFlow Gateway
- Rule Editors WorkFlow Gateway
- DB WorkFlow Gateway

The required 4-Channel Workflow Integration Architecture to fulfil the above requirements will be specified in the following UML diagrams illustrating:

0. The AXMEDIS Workflow Manager Integration Global View
1. Workflow Editors Interfaces
2. Workflow Engines Interfaces
3. Workflow Rule Editors/Viewers Interfaces
4. Workflow Query Support Interfaces

AXMEDIS WorkFlow Manager


Top Package::WF Manager
WorkFlow Management Area::Workflow User Interface and Tool
Workflow Engine
AXMEDIS Database Manager::Provisioning System
Workflow Database
WF AXOM Request Adapte
WF AXOM Input Queue Adapter
WF Engine Request Adapter
WF Engine Input Queue Adapter
WF Rule Editor Request Adapter
WF Rule Editor Input Queue Adapter
WF DB Request Adapter
WF DB Input Queue Adapter
WorkFlow Management Area::AXOM WorkFlow Gateway
WorkFlow Management Area::Engine WorkFlow Gateway
WorkFlow Management Area::Rule Editor WorkFlow Gateway
WorkFlow Management Area::DB WorkFlow Gateway


Axmedis WorkFlow Integration Architecture (OpenFlow)
WorkFLow Engine
WF AXOM Request Adapter
WF AXOM Input Queue Adapter
WF Engine Request Adapter
WF Engine Input Queue Adapter
WF Rule Editor Request Adapter
WF Rule Editor Input Queue Adapter
WF DB Request Adapter
WF DB Input Queue Adapter
Internal Library Calls
http GET / xmlrpc
AXOM WorkFlow Gateway
Engine WorkFlow Gateway
Rule Editor WorkFlow Gateway
DB WorkFlow Gateway
Web Services
Query Support WebService Interface
Axmedis Object Loader/Saver
AXOM Command And Reporting
Engine Command And Reporting
User Command And Reporting
• Axmedis Object Manager
• DRM Editor/Viewer
• External Editor/Viewer Activation Manager
• Hierarchy Editor/Viewer
• Metadata Editor/Viewer
• Axmedis Object Editor/Viewer
• Visual Editor/Viewer
• Behaviour Editor/Viewer
• Compositional/ Formatting
• Protection Tool
• Program and Publication
• Collector Internal
• Loading Tool
• Publication Tool
• Active Selection
• Protection Tool U.I. and Rule Editor
• Program and Publication U.I.
• Publication/Loading Rule/Selection Editor
• Compositional/formatting Rule Editor

There are four channels for communicating between WF Engine and the AXMEDIS tools/editors/viewers comprising specific *Request and Response Chains* as follows:

1. The WorkFLow Editor Channel, encompassing:

   The Request Chain:

   - WF AXOM Request Adapter
   - AXOM WorkFlow Gateway
   - AXOM_WebService_Plugin (Previously Listener which is now part of the plugin)
   - AXOM Command and Reporting

   The Response Chain:

   - AXOM Command and Reporting
   - AXOM_WebService_Plugin (Previously Listener which is now part of the plugin)
   - AXOM WorkFlow gateway
   - AXOM Input Queue Adapter

2. The WorkFlow Engine Channel encompassing:

   The Request Chain:

   - WF Engine Request Adapter

- Engine WorkFlow Gateway
- Engine_Plugin
- Engine Command and Reporting

The Response Chain:

- Engine Command and Reporting
- Engine Plugin
- Engine WorkFlow Gateway
- WF Engine Input Queue Adapter

3. The WorkFlow Rule Editor Channel encompassing:

   The Request Chain:

   - WF Rule Editor Request Adapter
   - Rule Editor Gateway
   - Rule Editor Plugin
   - User Command and Reporting

   The Response Chain:

   - User Command and Reporting
   - Rule Editor Plugin
   - Rule Editor Gateway
   - WF Rule Editor Input Queue Adapter

4. The WorkFlow Query and DataBase Channel encompassing:

   The Request Chain:

   - WF DB Request Adapter
   - DB WorkFlow Gatreway
   - AXMEDIS Object Loader/Saver and Query Support WebService Interface

   The Response Chain:

   - AXMEDIS Object Loader/Saver and Query Support WebService Interface
   - DB WorkFlow Gatreway
   - WF DB Input Queue Adapter

The WorkFlow Engine communicates with the WF Adapters via WorkFlow engine specific libraries and communication methods.

The WF Request Adapters communicates towards Gateways via WorkFlow available communuication methods and protocols.

The Gateways communicates towards WF Input Queue Adapters via WorkFlow available communuication methods and protocols.

The Gateways communicates with AXMEDIS modules via WebServices. This interface is standard for AXMEDIS, so will be the same when using any WorkFlow Engine technology. In other terms, for each

WorkFlow Engine technology, new WF Adapters and Gateways have to be developed, while the interface to the AXMEDIS modules remains invariant.

In the first step of AXMEDIS, the OpenFlow technology will be used for the WorkFlow Manager. The integration methods described in the following Sections are then applied to OpenFlow technology.

Later a commercial technology will be used (namely Microsoft BizTalk). Replacing OpenFlow with Microsoft BizTalk will mean developing a new set of WorkFlow Gateways and Adapters . These further modules will have the same interface with the AXMEDIS modules, while their interface with the WorkFlow Adapters will be different.

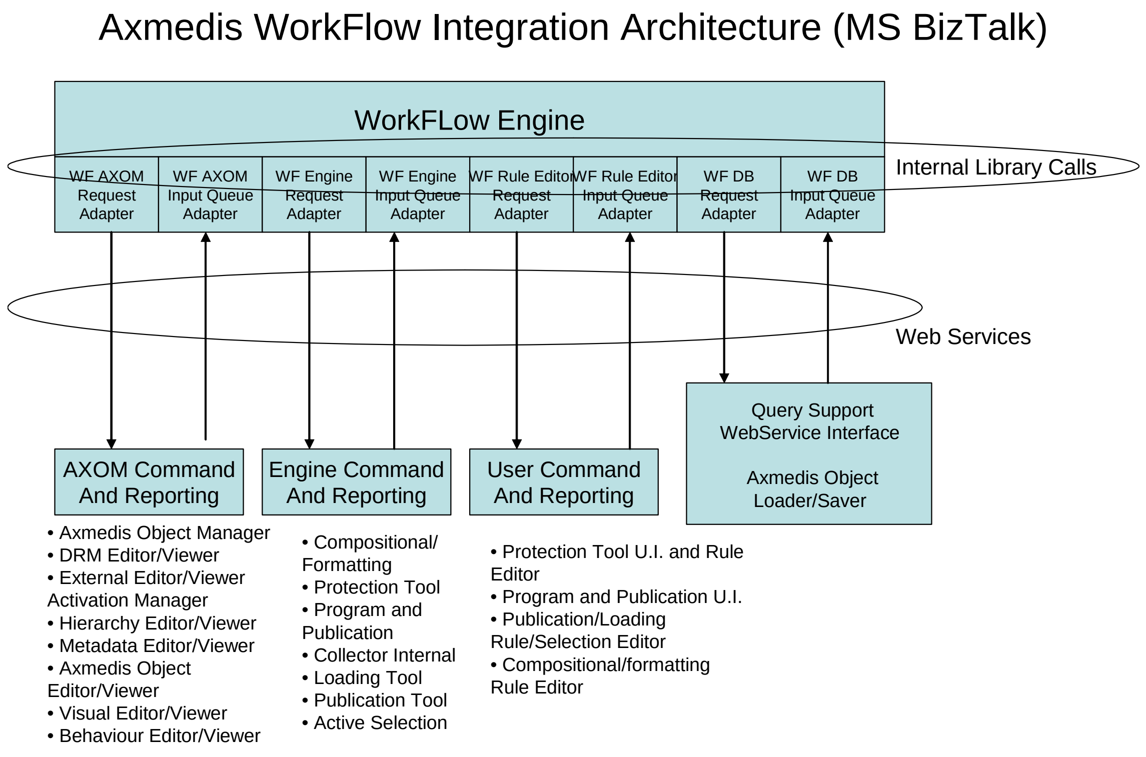
With OpenFlow the WorkFlow Engine communicates with the WF Adapters via library calls internal to the Zope Application Server. This interface will be replaced by the Microsoft BizTalk WebServices adapter internal interface when using BizTalk as the adopted AXMEDIS workflow system.

With OpenFlow the WF Request Adapters communicates with Gateways via GET/POST calls in http protocol.

With OpenFlow the Gateways communicates towards WF Input Queue Adapters via XMLRPC calls, in http protocol.

The Gateways communicates with AXMEDIS modules via WebServices and SOAP. This interface will remain equally applicable when using Microsoft BizTalk.

When Microsoft Biztalk will be used, the following integration architecture will be applied:


Axmedis WorkFlow Integration Architecture (MS BizTalk)
WorkFLow Engine
WF AXOM Request Adapter
WF AXOM Input Queue Adapter
WF Engine Request Adapter
WF Engine Input Queue Adapter
WF Rule Editor Request Adapter
WF Rule Editor Input Queue Adapter
WF DB Request Adapter
WF DB Input Queue Adapter
Internal Library Calls
Web Services
Query Support WebService Interface
Axmedis Object Loader/Saver
AXOM Command And Reporting
Engine Command And Reporting
User Command And Reporting
• Axmedis Object Manager
• DRM Editor/Viewer
• External Editor/Viewer Activation Manager
• Hierarchy Editor/Viewer
• Metadata Editor/Viewer
• Axmedis Object Editor/Viewer
• Visual Editor/Viewer
• Behaviour Editor/Viewer
• Compositional/ Formatting
• Protection Tool
• Program and Publication
• Collector Internal
• Loading Tool
• Publication Tool
• Active Selection
• Protection Tool U.I. and Rule Editor
• Program and Publication U.I.
• Publication/Loading Rule/Selection Editor
• Compositional/formatting Rule Editor

As we can see, the Gateways becomes null modules in this integration architecture, as the Request and Input Queue Adapters are capable of natively communicate with the AXMEDIS modules via WebServices.

4 Module or Executable Tool Workflow Engine - Openflow

Module/Tool Profile		
Workflow Engine - Openflow		
Responsible Name	Badii	
Responsible Partner	IRC	
Status (proposed/approved)	Approved	
Implemented/not implemented	Implemented	
Status of the implementation	First Prototype Completed	
Executable or Library/module (Support)	Executable OpenFlow Package, based upon Zope Application Server	
Single Thread or Multithread	Multi-Threaded	
Language of Development	User interface: Zope DTML (a superset of html) Application logic: Python or DTML	
Platforms supported	Windows, Linux	
Reference to the AXFW location of the source code demonstrator	N.A	
Reference to the AXFW location of the demonstrator executable tool for internal download	http://www.zope.org contains download of Zope http://www.openflow.it contains download of OpenFlow http://www.pyghon.org contains download of Python	
Reference to the AXFW location of the demonstrator executable tool for public download	http://www.zope.org contains download of Zope http://www.openflow.it contains download of OpenFlow http://www.pyghon.org contains download of Python	
Address for accessing to WebServices if any, add accession information (user and Passwd) if any	Not applicable	
Test cases (present/absent)	No	
Test cases location	http:///////////////////	
Usage of the AXMEDIS configuration manager (yes/no)	No	
Usage of the AXMEDIS Error Manager (yes/no)	No	
Major Problems not solved	-- --	
Major pending requirements	-- --	
Interfaces API with other tools, named as	Name of the communicating tools References to other major components needed	Communication model and format (protected or not, etc.)
WorkFlow Data Base		Internal adapter to various DB technologies
WorkFlow User Interface		Via Zope Web Server
WF Adapters		Internal libraries
Formats Used	Shared with	format name or reference to a section

Protocol Used	Shared with	Protocol name or reference to a section
Used Database name		
	See WorkFlow Database	
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
Zope Web-based U.I.	DTML, a superset of html	Idem
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not
Zope 2.7.3	Zope by Zope Corporation Zope 2.7.3 ZPL 2.1 (Zope Public Licence 2.1), open source, GPL compatible	GPL
Python 2.3	Python 2.3 by Stichting Mathematisch Centrum, Amsterdam, The Netherlands. Python 2.3 Free Open Source	GPL
OpenFlow OpenFlow 1.1	OpenFlow by OpenFlow	GPL
xmlrpclib	Xmlrpclib from Python	GPL

4.1 General Description of the Module

Openflow – This is a workflow engine developed by Icube and released as free software under a GNU GPL licence. It is based on an object-oriented structure and has a powerful exception handling system along with dynamic redesign support. These features make OpenFlow much more flexible than any other existing workflow engines. OpenFlow supports most of the open standards (XML/XML-RPC) including also the web standards. It has got a simple access to most of the relational DBMSs and thus it facilitates the integration of heterogeneous systems.

Through an integrated role assignment system, OpenFlow can assign tasks and activities to single users or workgroups and also to automatic applications. At every moment OpenFlow can trace the complete history of a certain situation e.g. participants involved, activities and actions executed and invoked. It is possible to carry out performance and efficiency analysis and verify the correct implementation of the adopted model.

OpenFlow is activity-based, web-based, WFMC inspired, built and integrated with the application server Zope. OpenFlow is capable of running on most operating systems including Linux, Windows 9x, NT/2000, XP, MacOs.

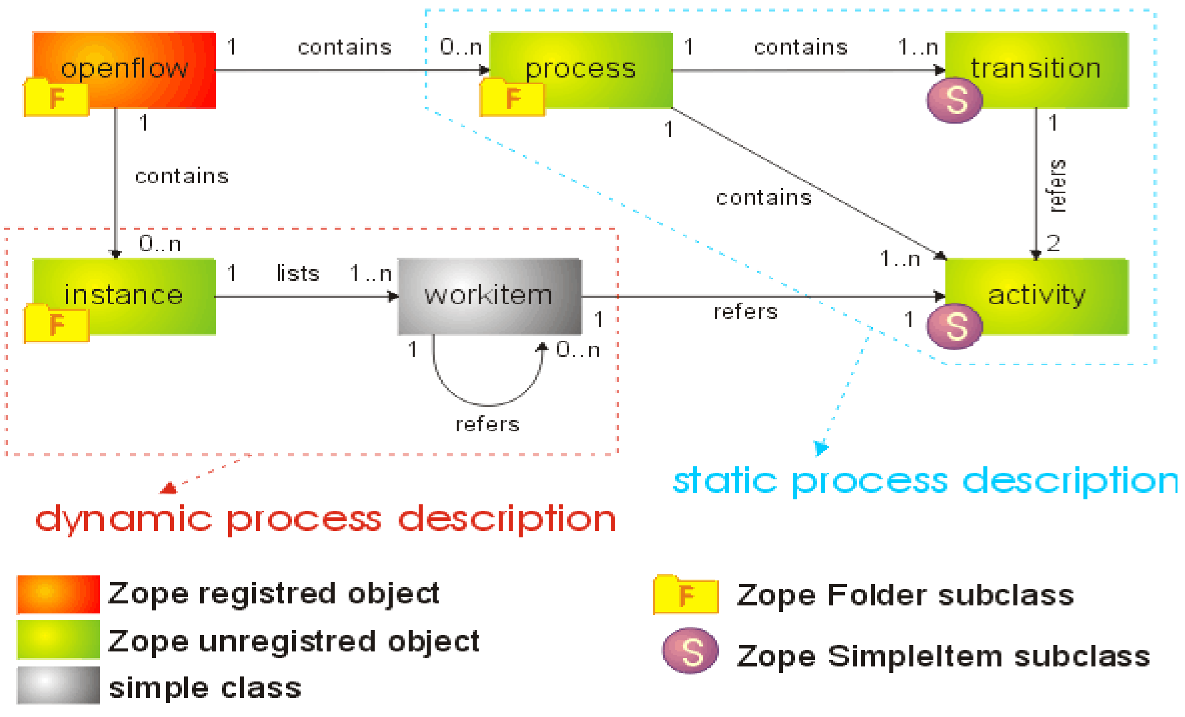
OpenFlow is written in Python, which is an interpreted, interactive, object-oriented programming language. It is often compared to Tcl, Perl, Scheme or Java. Python combines remarkable power with very clear syntax. It has modules, classes, exceptions, very high level dynamic data types, and dynamic typing. There are interfaces to many system calls and libraries, as well as to various windowing systems (X11, Motif, Tk, Mac, MFC). New built-in modules for OpenFlow can be easily developed in C or C++.

OpenFlow is strongly web-oriented and offers complete support for developing and executing workflows via a browser. The interaction with OpenFlow uses simple HTTP requests as in, for example, process modelling, assignment of users to activities, definition of the interaction with the applications. Every user receives his task which interacts with appropriate applications through the network.

4.2 Module Design in terms of Classes

In this section a description of the objects that compose OpenFlow and an overview of the Openflow terminology is provided.

The following picture shows the relations among all the classes in OpenFlow:



OpenFlow Class Diagram

OpenFlow comprises a number of Zope objects; as follows:

- Openflow
- Process
- Activity
- Transition
- Instance
- Employee

An outline description of the above objects, as deployed within the Openflow architecture, is as follows:

OpenFlow

OpenFlow is the container of processes (static description) and their instances (dynamic description). It is subclassed from a normal Zope folder but its contents are managed through three separate views.

Within a workflow it is possible to get the following views:

- The *Contents* view allows the user to see all the normal Zope objects put in Openflow. It is not possible to see any Process or Instance since they can be seen from their respective views.
- The *Processes* view allows the user to see all the processes created within the Workflow. It is possible also to add new processes to the Workflow.
- The *Instances* view allows the user to see all the process instances created within the Workflow. It is possible also to add new instances to the Workflow.
- The *Properties, Security, Undo, Ownership* and *Find* views are just like those in normal Zope folder views.
- The *Worklist* view allows a user to view the to-do list upon logging in. The list of workitems waiting for work to be done on them is ordered firstly by process and secondly by activity.

Process

This may be renamed "process definition" in the future. A process holds the map that describes the flow of work. The process map is made of activities and transitions. The instances you create on the map will begin the flow in the configured *begin* activity. Instances can be moved forward from activity to activity, going through transitions, until they reach the End activity.

This is sub-classed from a normal Zope folder but it can only contain *Activity* and *Transition*. Within a Process it is possible to get the following views:

- The *Contents* view enables the user to see all the normal Zope objects included in the Process.
- The *Map* view enables the user to see all the activity and *Transition* items created within the Process. It is possible here also to add new *Activity* and *Transition* items to the Process.
- The *Properties*, *Security*, *Undo*, *Ownership* and *Find* views are just like a normal Zope folder's views.

Activity

Activities represent any kind of action an employee might wish to perform on an Instance. The action might require attaching a file to the Instance, or may need to set a new field, or change an existing one or simply route the Instance onto a particular path. Activities are the places where any of these actions are resolved by employees.

It is sub-classed from a normal Zope simpleitem contained in a process and has no views. In selecting an activity it is possible to edit its configuration.

Transition

A transition connects two Activities: a *From* and a *To* activity. It represents a link starting from the *From* and ending in the *To* activity. Linking the activities in the process allows a process map to be drawn.

Each transition is associated with a *condition* that will be tested each time an Instance has to choose which path to follow. The only transition whose condition is evaluated as true will be the transition chosen for the forwarding of the Instance concerned.

The transition is implemented as a Zope simpleitem contained in a process and has no views. In selecting a transition it is possible to edit its condition.

Instance

This is a process Instance. A process Instance is created when a user decides to carry out a task, and this requires a start using a process defined in Openflow. The process is a class and each time a user wishes to "do what is defined in this process", this means he/she wishes to create an INSTANCE of the process. In previous versions of Openflow, instances were named "tokens".

So from this point of view, an Instance represents the dynamic part of a process. While the process definition contains the map of the workflow, the Instance stores the usage, the history, the state of this process. The Instance will collect and handle workitems (see Section 5 Glossary) to be passed from activity to activity in the process.

Each Instance can have more than one workitem depending on the number of split actions encountered in the process flow. This means that an Instance is actually a collection of all of the Instance "pieces" (workitems) that arise from the decomposition of the same original process Instance.

Each Instance keeps track of its history through a graph. Each node of the graph represents an activity the Instance has gone through (normal graph nodes) or an activity the Instance is now pending on (a graph leaf node). Tracking the Instance history can be very useful for the operation of Instance monitoring.

Instance is a subclass of a normal Zope folder: it can have a lot of information associated with it (files, attributes, and so on) in the form of contained objects. Within the Instance it is possible to get the following views:

- The *History* view shows the event graph related to the Instance. Each node describes a workitem and its event log.

- The *Contents*, *Properties*, *Security*, *Undo*, *Ownership* and *Find* views are just as in a normal Zope folder's views.

From an implementationally expedient standpoint Openflow defines a *workitem object* as representing an *activity* which is being performed. An Activity object defines the activity, while the workitem object represents the activity which is being performed within/upon an object belonging to the domain of the process. So in Openflow a workitem can be seen as an "instance" of the activity. This arises from the way Openflow needs to store Instances comprising co-occurring state changes involving all the entities that are participating in a given process Instance. This Instance is implemented as a whole dynamic system whose change history is stored in nodes in a graph and is thus re-call-able via a pointer to it (this is essentially the way workspace-instances can be saved and restored efficiently and this process/Instance sub-classing of "workitem *object*" which was defined in Section 2.1.1 as an *Object* class leaves the generic definition given in Section 2.1 as still valid and consistent with the workitem being sub-classed logically with an Object - specifically an object undergoing an activity. In any case for any WFMSs that could be considered as a potential AXWF, irrespective of the way they might sub-class workitems or any other session-specific data, as far as the AXMEDIS platform is concerned all that is required of the WFMSs is that they make available some form of client-session-owner_ID traceable to the user responsible for charges accrued for each session arising from services/granted_rights provided to them by the AXMEDIS Service Providing Components.

How any such WFMSs represent their native client-session-owner_ID data for tracking, control and billing traceability purposes is a matter internal to the WFMSs themselves as long as they can ensure traceability of all AXMEDIS Service Consumers for billing.

Employee

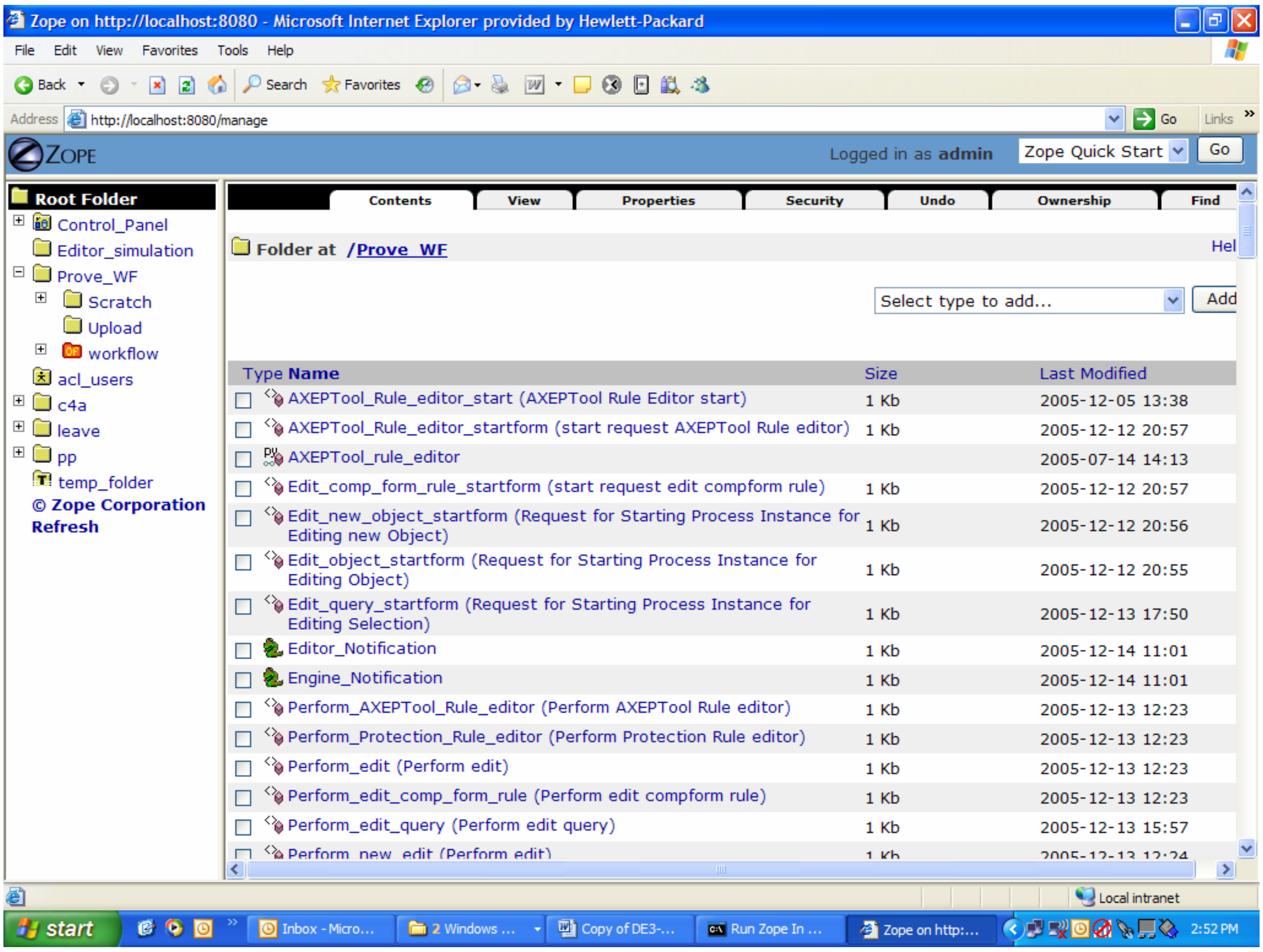
An employee is a Zope user who is responsible for performing the application of one or more activities. By accessing a particular workflow-instance's "to-do-list", the employee is presented with the list of instances pending on activities for which he is responsible in the context of that particular workflow instance.

4.3 User interface description

The OpenFlow User Interface is a traditional Zope web-based interface.

When connecting to Zope, a weak authentication (User / Password) is required from the user. After that, the connection is made to the User Zope Home Page, which was specifically designed and developed for AXMEDIS processes management (see WorkFlow User Interface).

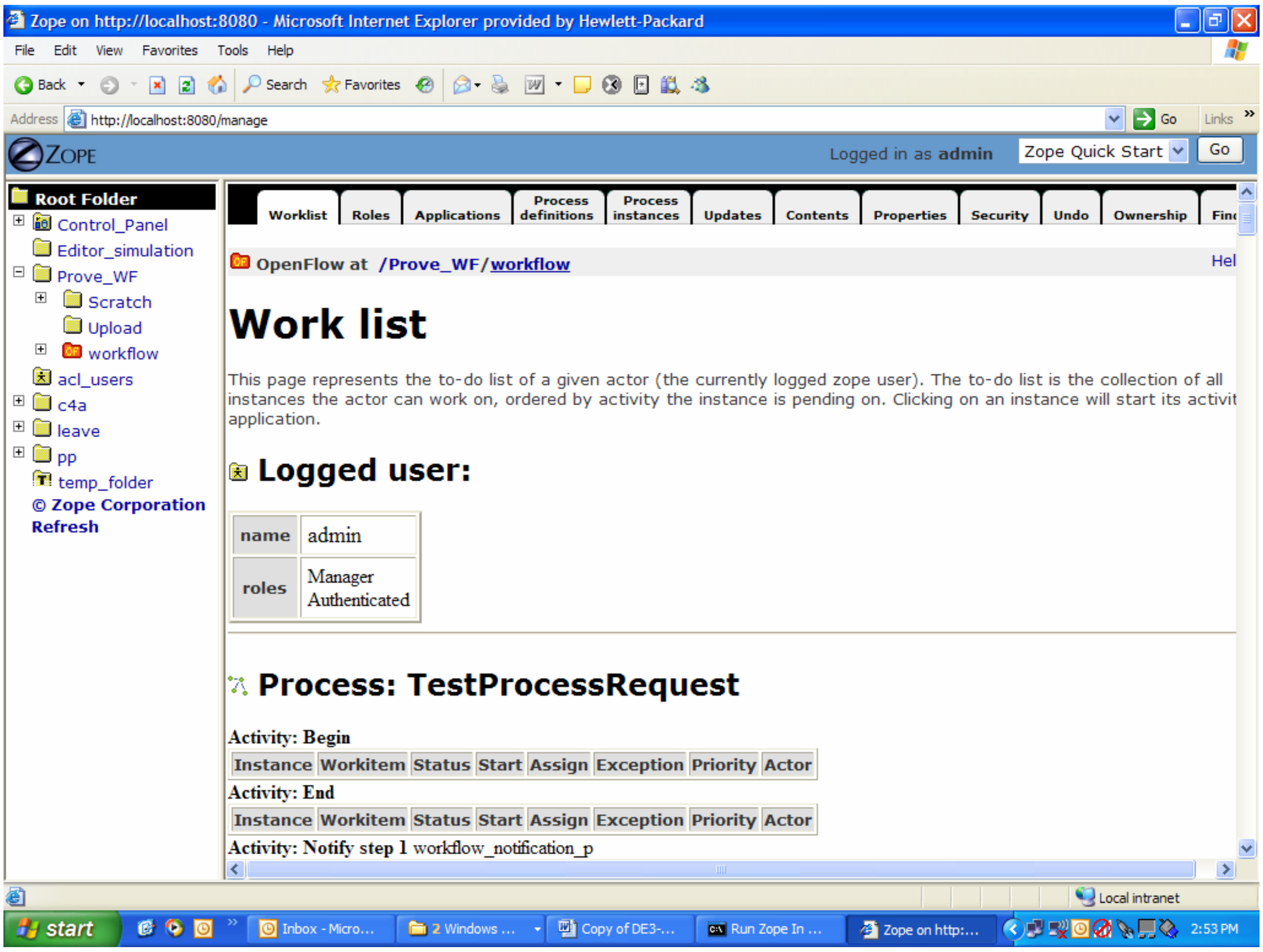
Zope has a standard interface, based on navigation menus (folders/subfolders) like:



Type Name	Size	Last Modified
AXEPTool_Rule_editor_start (AXEPTool Rule Editor start)	1 Kb	2005-12-05 13:38
AXEPTool_Rule_editor_startform (start request AXEPTool Rule editor)	1 Kb	2005-12-12 20:57
AXEPTool_rule_editor		2005-07-14 14:13
Edit_comp_form_rule_startform (start request edit compform rule)	1 Kb	2005-12-12 20:57
Edit_new_object_startform (Request for Starting Process Instance for Editing new Object)	1 Kb	2005-12-12 20:56
Edit_object_startform (Request for Starting Process Instance for Editing Object)	1 Kb	2005-12-12 20:55
Edit_query_startform (Request for Starting Process Instance for Editing Selection)	1 Kb	2005-12-13 17:50
Editor_Notification	1 Kb	2005-12-14 11:01
Engine_Notification	1 Kb	2005-12-14 11:01
Perform_AXEPTool_Rule_editor (Perform AXEPTool Rule editor)	1 Kb	2005-12-13 12:23
Perform_Protection_Rule_editor (Perform Protection Rule editor)	1 Kb	2005-12-13 12:23
Perform_edit (Perform edit)	1 Kb	2005-12-13 12:23
Perform_edit_comp_form_rule (Perform edit compform rule)	1 Kb	2005-12-13 12:23
Perform_edit_query (Perform edit query)	1 Kb	2005-12-13 15:57
Perform_new_edit (Perform edit)	1 Kb	2005-12-13 12:24

This standard interface was designed specifically for Zope administrators and developers.

OpenFlow has as well a "standard" interface for accessing OpenFlow items:



OpenFlow at /Prove_WF/workflow

Work list

This page represents the to-do list of a given actor (the currently logged zope user). The to-do list is the collection of all instances the actor can work on, ordered by activity the instance is pending on. Clicking on an instance will start its activit application.

Logged user:

name	admin
roles	Manager Authenticated

Process: TestProcessRequest

Activity: Begin

Instance	Workitem	Status	Start	Assign	Exception	Priority	Actor

Activity: End

Instance	Workitem	Status	Start	Assign	Exception	Priority	Actor

Activity: Notify step 1 workflow_notification_p

Also this interface was specifically designed for administrators and developers.

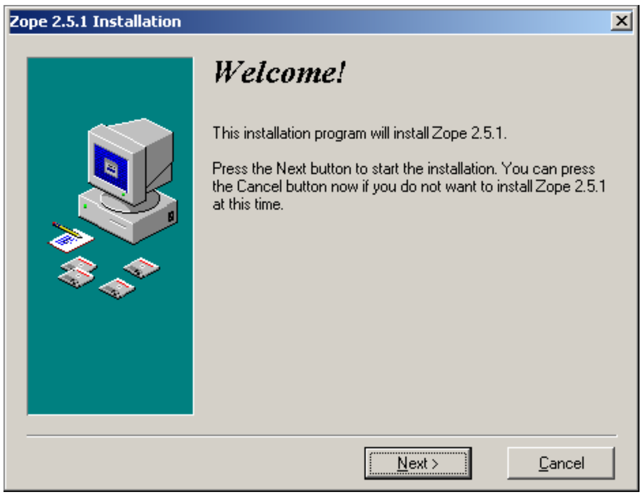
The User Interface for AXMEDIS Users will be designed and developed specifically. (see WorkFlow User Interface).

4.4 Technical and Installation information

- Installation capability, it has to be installable in a very easy manner
- Manual support for technical and user point of views

In order to install the specified openflow Engine, We have to install Zope server first as follows

1. Download the current stable release installer for Windows from Zope.org using your web browser. Place the file in a temporary directory on your hard disk or on your Desktop. Once the installer file has been downloaded, navigate to the folder in which you downloaded the file to, and double-click on the file's icon. The installer then begins to walk you through the installation process.


Zope 2.5.1 Installation

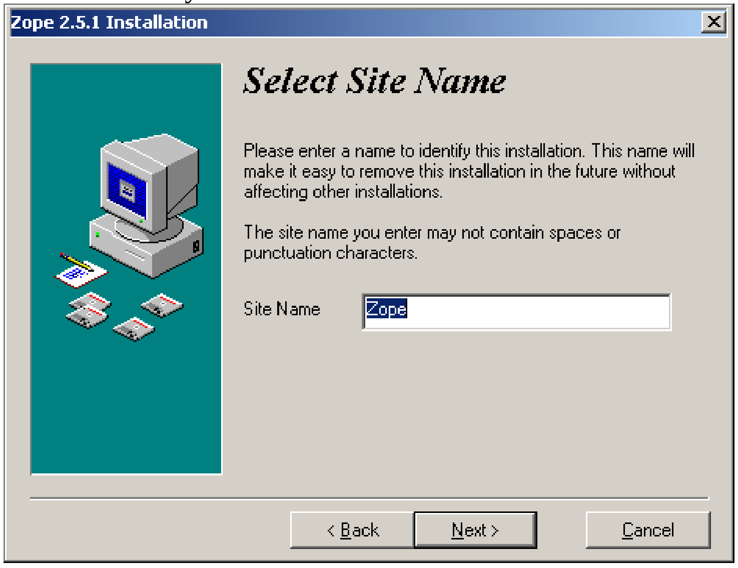
Welcome!

This installation program will install Zope 2.5.1.

Press the Next button to start the installation. You can press the Cancel button now if you do not want to install Zope 2.5.1 at this time.

Figure 2-2 Beginning the installer

2. Click *Next*. You are asked to accept the Zope Public License before installing the product. After you read and accept the license, click *Next* again. Since you can install more than one Zope instance on on any given machine, you are asked to pick a unique "site name" for your Zope instance. The default name is "WebSite". It's recommended that you change this value. "Zope" is a reasonable name, although you are of course free to pick any name you choose.
3. Click *Next* after choosing your site's name. You are then asked to choose a directory in which to install Zope. A reasonable choice for a destination directory is "c:\Program Files\Zope". After filling in the directory name, click *Next*. You will be prompted to create a new Zope user account. This is not an operating system account. It is a user account that is only meaningful to Zope. The account that you specify is called the *initial user* (or "superuser") and is used to log into Zope for the first time. It is also given Zope administrative privileges. You can change this user name and password later if you wish. A reasonable choice for the initial user name is "admin".


Zope 2.5.1 Installation

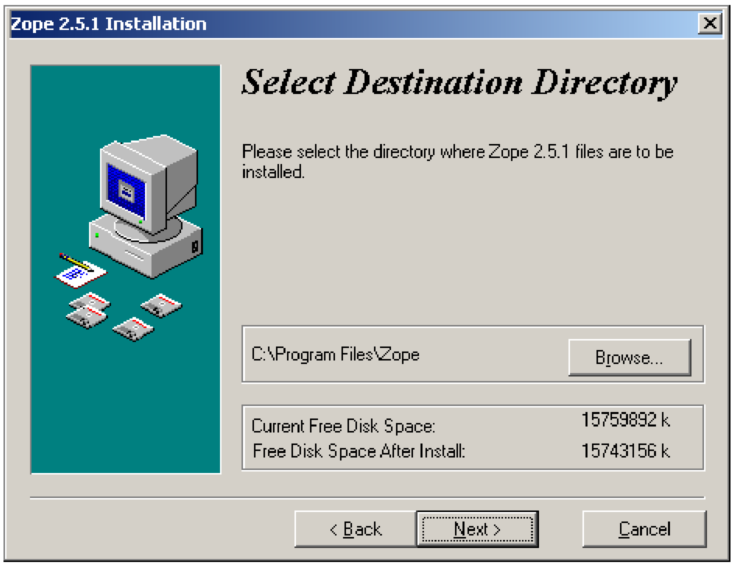
Select Site Name

Please enter a name to identify this installation. This name will make it easy to remove this installation in the future without affecting other installations.

The site name you enter may not contain spaces or punctuation characters.

Site Name: Zope

Figure 2-3 Selecting a Site Name


Zope 2.5.1 Installation

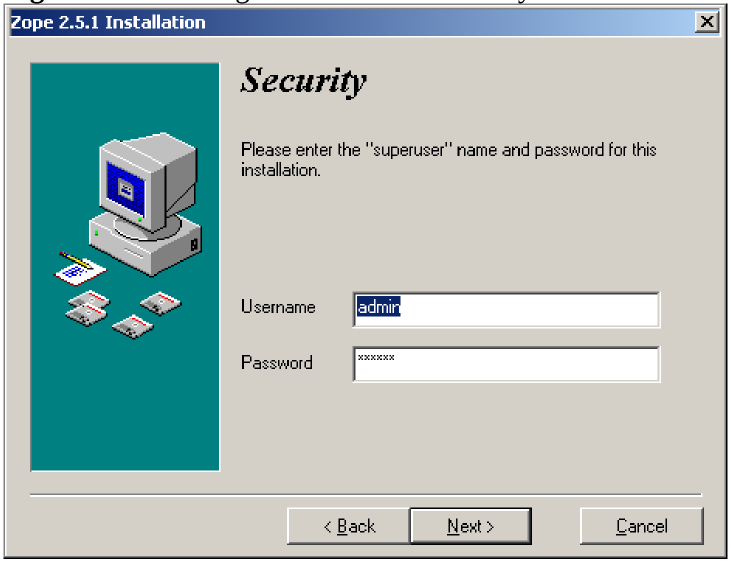
Select Destination Directory

Please select the directory where Zope 2.5.1 files are to be installed.

C:\Program Files\Zope

Current Free Disk Space: 15759892 k
Free Disk Space After Install: 15743156 k

Figure 2-4 Selecting a Destination Directory


Zope 2.5.1 Installation

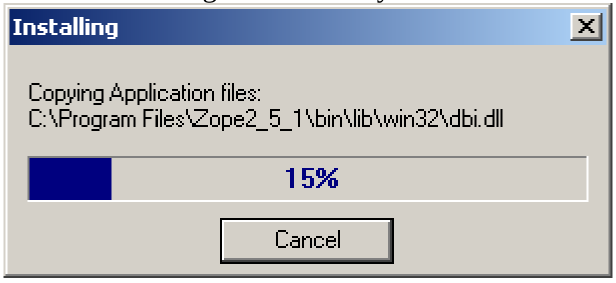
Security

Please enter the "superuser" name and password for this installation.

Username: admin
Password: ******

Figure 2-5 Provide an initial username and password

6. Click *Next* after choosing the initial user name and password. The installer presents a dialog indicating that it is ready to install files. Click *Next* again to begin installing the files.


Installing

Copying Application files:
C:\Program Files\Zope2_5_1\bin\lib\win32\dbi.dll

15%

Figure 2-6 Installing files

7. Once the file copy is finished, if you are using Windows NT, Windows 2000, or Windows XP, you will see a dialog that indicates that you may choose to run Zope as a service. If you are just running Zope for personal use, don't bother running it as a service. If you are running Windows 95, Windows 98, or Windows ME, you cannot run Zope as a service (it is not offered as an option). It is recommended that if you are installing Zope for the first time that you don't choose to run the server manually.

Figure 2-7 Server Options

8. After you click "Next", the installer informs you that the installation was successful. Click "Finish". If you decide to uninstall Zope later you can use the *Unwise.exe* program that resides in the directory in which you chose to install Zope.
9. Note that the Zope installer does not add a program folder entry to your "Start" menu. You will see how to start Zope in an upcoming section.

The following steps are needed for correctly install OpenFlow for AXMEDIS prototype:

1. Install Zope 2.7.3 in default directories (Zope instance in c:\Zope-Instance and Zope program files in c:/program files/Zope-2.7.3-0) by executing the self extracting archive downloaded from www.zope.org
2. unpack the OpenFlow1.2.0 tgz (downloaded from ww.openflow.it) in the lib/python/Products directory of your Zope installation (i.e. c:/program files/Zope-2.7.3-0)
3. Install Zope 2.7.3 and OpenFlow 1.2.0 following the instructions supplied with the products in default directories
4. Install Python 2.3 by executing the self extracting archive downloaded from www.python.org
5. Edit (with IDLE) the file c:\Program Files\Zope-2.7.3-0\lib\python\Products\OpenFlow\__init__.py
6. After the line containing from Globals import InitializeClass insert two new lines:
   - import xmlrpclib
   - import base64
7. After the line containint try: (with the same indentation as the following lines) insert two new lines (correctly indented):
   - allow_module ("xmlrpclib")
   - allow_module ("base64")
8. Save and close the __init__.py file

References to other major components needed	• Zope 2.7.3 (see instructions) • Python 2.3 (see instructions) • Xmlrpclib (included in Python 2.3)
Problems not solved	•
Configuration and execution context	See instructions above

4.5 Draft User Manual

For Zope Server, available online at http://www.zope.org/Documentation/Books/ZopeBook/
For Openflow, available online at
http://www.openflow.it/Documentation/documentation/OpenFlowIntroduction

4.6 Examples of usage

4.7 Integration and compilation issues

NO Issues

4.8 Configuration Parameters

No specific parameters are needed for Zope and OpenFlow: install everything as standard.
See installation instructions.

Config parameter	Possible values

4.9 Errors reported and that may occur

Error code	Description and rationales

4.10 Formal description of algorithm <…………..>

None

name	
Method	
Description	
Input parameters	
Output parameters	

name	
Method	
Description	
Input parameters	
Output parameters	

5 Module or Executable Tool Workflow Engine – BizTalk Server

Module/Tool Profile	
Workflow Engine – BizTalk Server	
Responsible Name	
Responsible Partner	
Status (proposed/approved)	
Implemented/not implemented	Implemented Microsoft BizTalk Server 2004 Product
Status of the implementation	Microsoft Product
Executable or Library/module (Support)	Packaged product.
Single Thread or Multithread	N/A
Language of Development	N/A
Platforms supported	Microsoft Windows
Reference to the AXFW location of the source code demonstrator	N/A
Reference to the AXFW location of the demonstrator executable tool for internal download	N/A: the product have to be purchased from Microsoft
Reference to the AXFW location of the demonstrator executable tool for public download	N/A: the product have to be purchased from Microsoft
Address for accessing to WebServices if any, add accession information (user and Passwd) if any	N/A
Test cases (present/absent)	
Test cases location	
Usage of the AXMEDIS configuration manager (yes/no)	No
Usage of the AXMEDIS Error Manager (yes/no)	No
Major Problems not	No

solved		
Major pending requirements	No	
Interfaces API with other tools, named as	Name of the communicating tools References to other major components needed	Communication model and format (protected or not, etc.)
Formats Used	Shared with	format name or reference to a section
Protocol Used	Shared with	Protocol name or reference to a section
SOAP		
Internal messaging protocol	Request Adaptors	Message queuing used for calling orchestrations from other orchestrations.
	Input Queue Adaptors	
Used Database name		
Microsoft SQL Server 2000	See WorkFlow Database	
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
Microsoft BizTalk Server Administration GUI	N/A	N/A
Microsoft Biztalk HAT (Health and Activity Tracking) GUI	N/A	N/A
Microsoft Biztalk integrated Visual Studio .NET Orchestration Designer	N/A	N/A
Several Wizards (WebServices publishing, deploying, etc)	N/A	N/A
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not

5.1 General Description of the Module

BizTalk Server 2004, an integration server, lets you to develop, deploy, and manage integrated business processes and XML-based Web services.

No application is an island. Even though many are still created with an internal focus, the reality is that tying applications together has become the norm. Yet connecting software is about more than just exchanging bytes. As organizations move toward a service-oriented architecture (SOA), the real goal—creating business processes that unite separate applications into a coherent whole—comes within reach.

Microsoft BizTalk Server 2004 supports this goal. It enables you to connect diverse applications, and then to graphically create and modify business processes that use the services that those applications provide. It also includes mechanisms for specifying business rules, better ways to manage and monitor the applications built on it, and support for single sign-on for those applications.

BizTalk Server 2004 also adds new services for information workers. These include a group of Business Activity Services (BAS), a Business Activity Monitoring (BAM) Framework for analyzing running business processes, support for business process provisioning and configuration, and services that enable information workers to set up and manage interactions with trading partners. BizTalk Server 2004 also adds technology for creating Human Workflow Services (HWS), making possible business processes that people can interact with from Microsoft Outlook and other familiar clients. Given the amount of new technology it contains, it is fair to say that BizTalk Server 2004 is a significant update from its predecessor.

BizTalk Server has changed because the world in which customers use it has changed. The biggest change, one that confronts every organization, is the rapid spread of Web services. Because they allow universal connectivity among applications, Web services will lead most organizations to some form of SOA. Other important changes flow from this, such as the ability to define Web services-based business processes by using the Business Process Execution Language for Web Services (BPEL4WS, commonly called just BPEL). The changes in the Microsoft world itself are also substantial, with the most important flowing from the advent of the Microsoft .NET Framework. Because it provides a new foundation for building applications, the .NET Framework transforms how Microsoft Windows-oriented developers create software.

All of these changes have strongly affected how Microsoft approached building this new BizTalk Server release. Unlike its COM-based predecessors, BizTalk Server 2004 is built completely around the .NET Framework and Microsoft Visual Studio .NET. It also has native support for communicating through Web services, along with the ability to import and export business processes described in BPEL. Designed to work well both with the emerging world of standard Web services and with the large number of applications already in place, BizTalk Server 2004 is a significant step forward for the BizTalk Server product line.

You can use BizTalk Server 2004 in a variety of ways. Traditionally, BizTalk Server has been used for application integration, where the following two scenarios are most important:

- Connecting applications within a single organization, commonly referred to as enterprise application integration (EAI)

- Connecting applications in different organizations, often called business-to-business (B2B) integration

The following figure shows a simple example of the core BizTalk Server 2004 engine applied to an EAI problem.

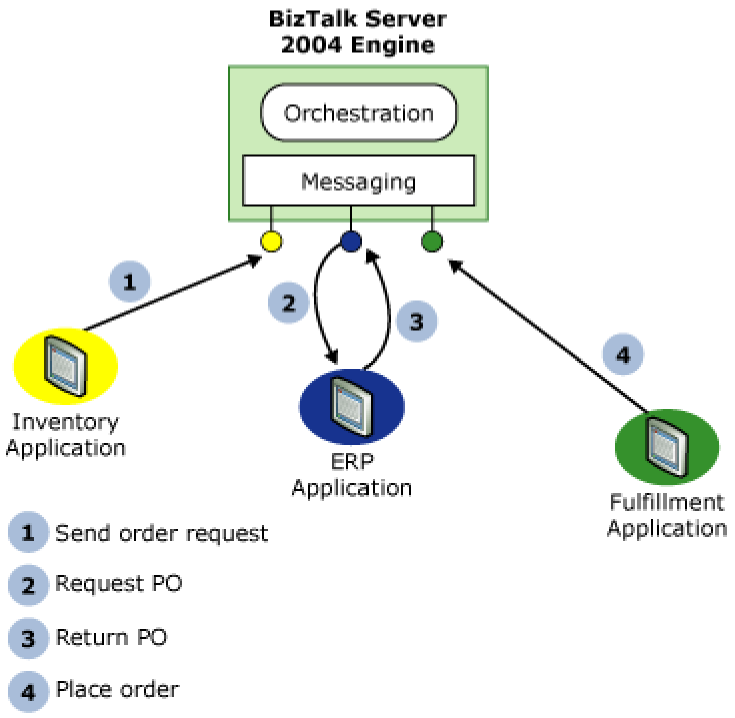


Figure 5.1 BizTalk Server used for EAI

In this scenario, an inventory application, perhaps running on an IBM mainframe, notices that the stock of an item is low and issues a request to order more of that item. The following steps occur:

1. The request is sent to a BizTalk Server 2004 application.
2. The BizTalk Server 2004 application requests a purchase order (PO) from the Enterprise Resource Planning (ERP) application for the organization.
3. The ERP application, which might be running on a UNIX system, sends back the requested PO.
4. The BizTalk Server 2004 application informs a fulfillment application, perhaps built on Microsoft Windows by using the .NET Framework, that the item should be ordered.

In this example, each application communicates by using a different protocol. Accordingly, the messaging infrastructure of the BizTalk Server 2004 engine must be able to talk with each application in its native communication style. Also, notice that no single application is aware of the complete business process. That business process, along with the intelligence required to coordinate all of the parts, is implemented in the BizTalk Server 2004 application.

Connecting applications within an organization is important, but connecting applications that span organizations can sometimes have even more business value. The following figure shows a simple example of a B2B integration scenario.

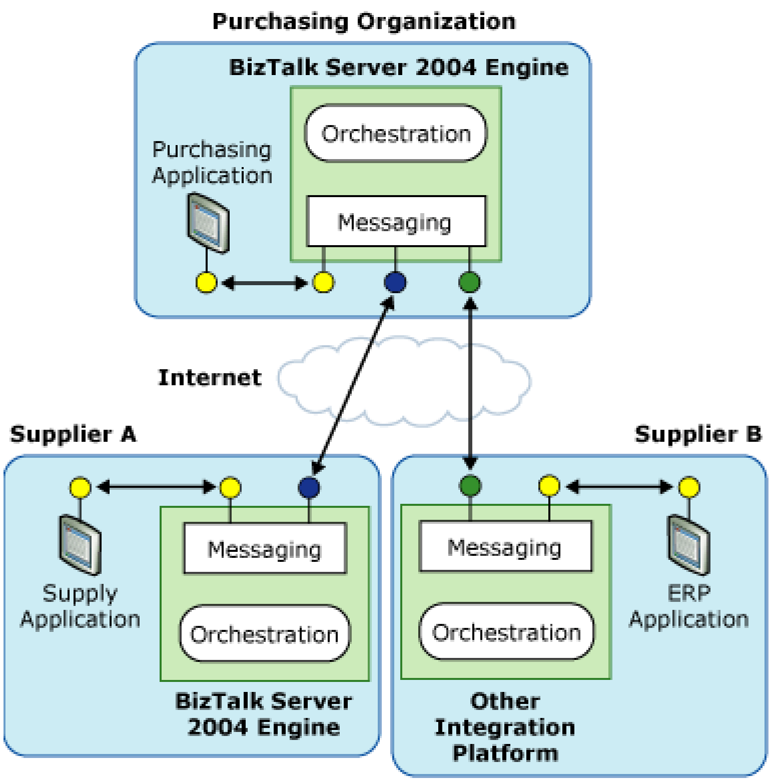


Figure 5.2 BizTalk Server used for B2B

The applications are connected as follows:

- The purchasing organization at the top of the figure runs a BizTalk Server 2004 application that interacts with two supplier organizations.
- Supplier A also uses BizTalk Server 2004, providing indirect access to its Supply application.
- Supplier B uses an integration platform from another vendor, connecting to the BizTalk Server 2004 application for the purchasing organization by using, say, Web services. Supplier B is executing the same business process as the others, and so it may have been sent a BPEL definition of that process by the purchasing organization, which exported this definition from BizTalk Server 2004.

The integration of existing applications—whether they are used in a single company or spread across different organizations—into a single business process is a fundamental goal of BizTalk Server 2004. However, it is not the whole story. Other useful services are built on this foundation. For example, the people who use a business process need to interact with it in various ways. Accordingly, BizTalk Server 2004 provides services that give information workers—and not just technical workers—visibility into running business processes, and let them interact with the processes when necessary.

5.1.1 Human Workflow Services

The BizTalk Server 2004 engine enables you to connect applications to carry out a business process. But what if that process requires human intervention? Think about handling a purchase order (PO), for example. An organization might require human approval at various stages in this process, and the people involved might change depending on the PO size. Or imagine a request for proposal (RFP) arriving in a sales office. Creating a response might require a shifting group of people working together over days or weeks. Human-oriented business processes like these, commonly called "workflows", can certainly benefit from automation, but the BizTalk Server 2004 engine by itself is not sufficient. More is required to enable people to interact with and control the process.

These extras are provided by Human Workflow Services (HWS), a standard part of BizTalk Server 2004. This service provides a workflow infrastructure built on the BizTalk Server 2004 engine. The HWS infrastructure is accessed through Web services, and so it can be used by any client application. Among the most important clients, however, are the applications in Microsoft Office. Word, Outlook, Excel, and InfoPath are fundamental to how many information workers work with information, and so it makes sense to allow these common tools to be the environment from which people participate in workflows. For the case of AXMEDIS, the most important client tools are the AXMEDIS Object Editor, AXMEDIS Engines like AXCP, PnP, AXEPTool, Rule Editors and the AXMEDIS database query support. The workflow plug-ins developed for these tools are based on the web services, and hence they can be easily integrated with the HWS for BizTalk Server

The following figure shows the basic structure of Human Workflow Services. As just described, the technology relies on the BizTalk Server 2004 engine, as well as an Administration database and the Tracking database (also used by Business Activity Services) that holds information about the various actions that make up this workflow. Outlook and Word are likely to be the most common clients, along with custom forms built by using InfoPath. ASP.NET applications and other applications that can access standard Web services can also use this part of BizTalk Server 2004. For clients built on the .NET Framework, Human Workflow Services provides a library that exposes all of its Web services as .NET-based objects. And although it is not shown in the diagram, applications are administered by using the HWS Administration console, an MMC snap-in.

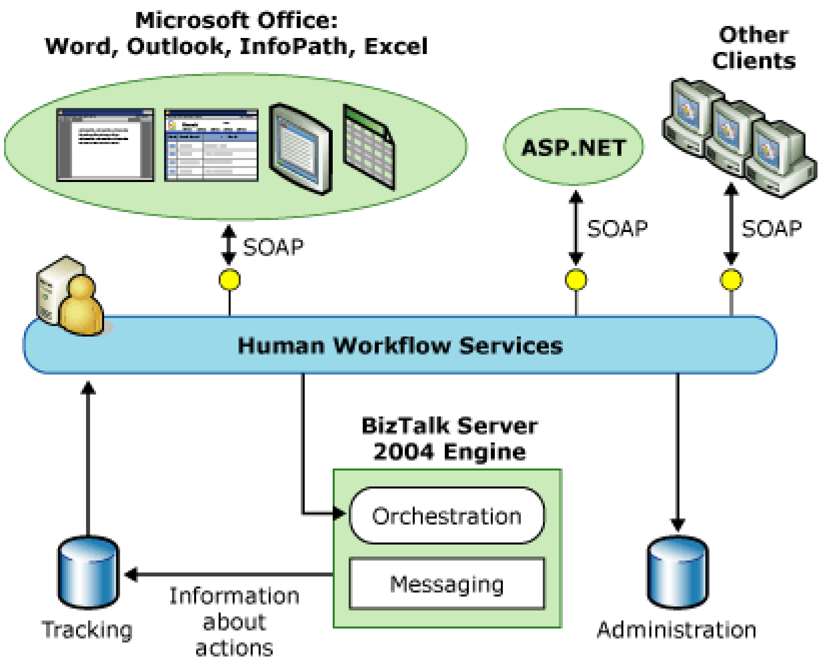


Figure 5.3 Human Workflow Services

To get a sense of how all of this might be used, imagine that an application based on Human Workflow Services has been created to help an organization automate its response to an RFP. The process of handling the RFP might be started by the original recipient of the request invoking a Web service that initiates the workflow. In Microsoft Office 2003, Web services can be invoked from Outlook, from an InfoPath form, from a new feature in Word called SmartDocs that allows a document to contain embedded actions, and in other ways. The RFP and any associated documents, such as a draft response, might be sent to multiple people involved in the process, each of whom takes some action. Several people might need to approve the response, for example, and so each one can directly interact with the workflow application from an Outlook message requesting this approval.

Many other activities are also possible, depending on how the workflow is defined. An Office 2003 SmartDocs document can contain a list of people to whom a task can be delegated, for example, and a user can click the name of the person to whom this task should be assigned. Doing this invokes a Web service that causes the Human Workflow Services application to delegate the task to that person. A user can also access an ASP.NET page that graphically depicts the status of a particular workflow, indicating who has and has not taken various actions, who has delegated tasks (and to whom they were delegated), and more.

To support this quite general approach to workflow applications, Human Workflow Services defines a few core abstractions, all of which are built on the BizTalk Server 2004 engine. Every workflow is built from some number of actions, each of which is actually implemented as an orchestration. For example, a workflow for responding to an RFP might have actions such as Review, Approve, Delegate, and Escalate. Like all orchestrations, those used with Human Workflow Services are created by using Orchestration Designer, but each one has a number of standard behaviors along with the customized behavior created by the developer who builds this application.

The actions in a workflow occur in a defined order, called an activity flow. Each activity flow has an activity model that defines the actions within it and how they relate to one another. The people who use a Human Workflow Services application are called actors, and communication between actors and actions happens through tasks, which are actually XML-defined messages. Each action has one or more tasks associated with it, and so when an actor clicks a button in an Office application that does something in a workflow, that actor is really sending a particular task message to some action (that is, to an orchestration).

A Human Workflow Services application can also impose constraints on the people who use it based on their roles. An application might allow only managers to approve POs of more than a million euros, for example, or limit who is allowed to delegate tasks to a vice president. To support this, the creator of a workflow application can define constraints that rely on roles defined in Microsoft Active Directory®, a SQL Server database, or in other ways.

Developers are certainly required to build Human Workflow Services applications. After they are created, though, those applications can be used solely by information workers. Workflows can be created, new actions can be added to the activity flow, existing actions can be interrupted, and more, all by the business people who use this application. While the BizTalk Server 2004 engine provides a way to connect software together, Human Workflow Services adds what is needed to let people participate directly in the business process.

Human Workflow Services (HWS) is a standard part of BizTalk Server 2004. HWS provides a workflow infrastructure built on the BizTalk Server 2004 engine. You access the HWS infrastructure by using Web services, so it can be used by any client application. The applications in Microsoft Office are among the most important clients that use HWS. Many information workers use Microsoft Word, Microsoft Outlook®, Microsoft Excel, and Microsoft Office InfoPath™, and so it makes sense to allow these common tools to be the environment from which people participate in workflows.

The following figure shows the basic model.

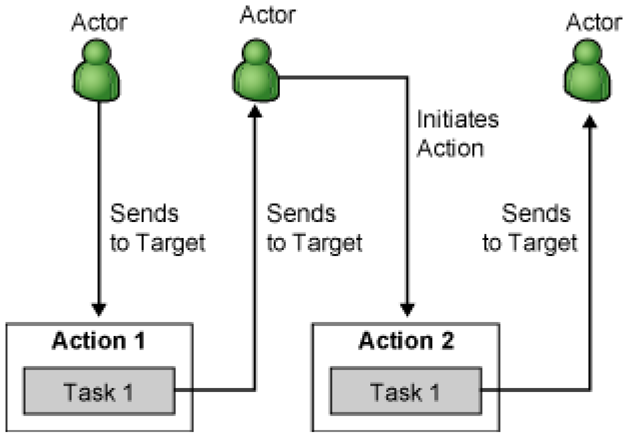


Figure 5.4 Human Workflow Services Basic Model

Activity flows are composed of one or more actions. An action represents a fundamental business process, or unit of workflow, that you cannot reduce to further sub-actions. Actions may contain zero or more tasks defining work.

Actions do not always send tasks. In dependent composition for example, actions send synchronization message and no tasks. A synchronization message is a message that an action sends or receives to or from another action when the action runs. You assign tasks to the participants (called actors) in the workflow, which are external entities that either start an activity flow or participate in an ongoing activity flow. The actor who initiates the first action within an activity flow becomes the owner of the activity flow.

Note HWS implements actions as BizTalk Orchestrations and implements tasks and responses as send or receive ports in a BizTalk Orchestration. Actors represent HWS users who participate in a HWS activity flow

The following figure shows the basic components used in Human Workflow Services.

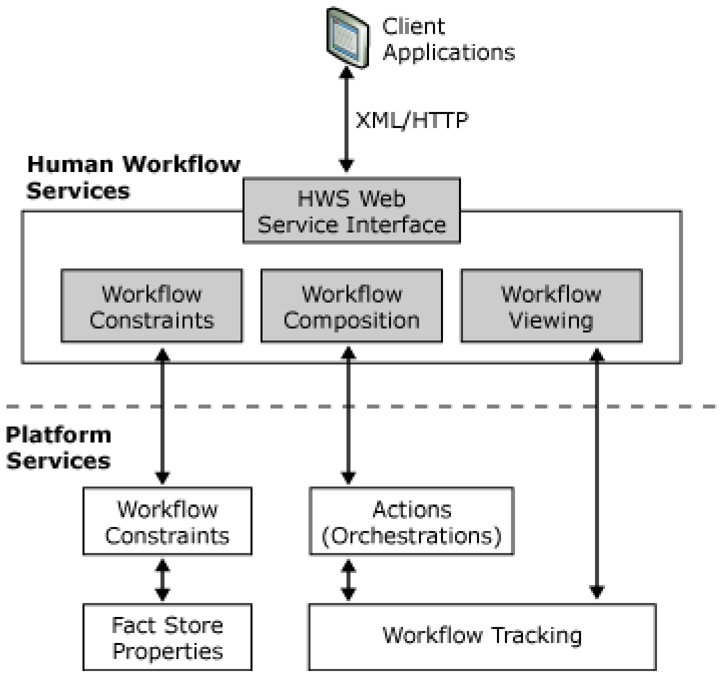


Figure 5.5 Human Workflow Services Basic Components

- **Actors**

An actor is an entity external to the workflow that performs tasks as part of a workflow. For example, an actor could be a human being or an application.

- **Actions**

To support this general approach to workflow applications, Human Workflow Services defines a few core abstractions, all of which are built on the BizTalk Server 2004 engine. Every workflow is built from one or more building blocks called actions, each of which is implemented as an orchestration.

Actions are BizTalk orchestrations that receive and send messages that contain extensions to the task schema. An action assigns a task to an actor by sending the actor an HWS message.

For example, a workflow for responding to a request for proposal might have actions such as Review, Approve, Delegate, and Escalate. Like all orchestrations, those used with HWS are created by using Orchestration Designer. Each HWS orchestration has a number of standard behaviors along with any customized behavior created by the developer who builds the application.

HWS uses this process to keep track of assigned tasks and to provide a representation of actor interactions in specific activity flows.

HWS creates tracked events for every action. These tracked events provide up-to-date views of workflow progress. The Constraint service uses tracked events to determine what workflows each actor can create. Actions track the following events:

- Activation point of action.
- Ending point of an action.
- All HWS messages sent or received

Users can perform actions as needed. Additionally, actions can be part of a stored sequence called an activity model.

Developers use an HWS action project template to construct actions.

- **Activity models**

You use activity models to maximize efficiency. Activity models provide the sequence of actions in an HWS workflow. You use the transitions between each step in an activity model to define the action sequence.

A trusted user can create, perform, and initiate actions in an activity model on behalf of a target participant (HWS enforces system-level constraints by default).

- **Activation blocks**

An activity model may consist of one or more activation blocks. The initiator of the activation block must supply parameters for all actions within the activation block in order to activate them (defaults will be used if parameters are not provided).

Keep the following guidelines in mind when working with activation blocks:

- Each step within an activation block has a unique ID. Two activation blocks cannot share the same step (although they may have actions of the same type).
- The actor providing the activation parameters for the action instance within an activation block must also provide the activation parameters for all dependent actions within the activation block.
- Loops may originate from any action but must reference the first action within an activation block, and therefore by definition are independently composed.

You can define constraints to restrict the initiator of the root step and targets for any other step within the activity model. In addition, the initiator of all independently composed steps other than the root step must receive a task from another step within the activity model.

- **Activity flows**

The actions in a workflow occur in a defined order, called an activity flow. Each activity flow has an activity model that captures actions in a particular order and enables the person designing the activity flow to add structure to the workflow. The users of an HWS application are called actors, and communication between actors and actions happens through tasks, which are XML-defined messages.

Each action has one or more tasks associated with it, and so when an actor clicks a button in an Office application that does something in a workflow, that actor is sending a particular task message to some action (that is, to an orchestration).

- **Constraints**

An HWS application can also impose constraints on the people who use it based on their roles. For example, an application might allow only managers to approve purchase orders of more than a million euros, or might allow only vice presidents to delegate tasks. To support this, the creator of a workflow application can define constraints that rely on roles defined in Active Directory®, in a Microsoft SQL Server™ database, or in other ways.

- **Fact retriever**

The fact retriever implements a standard HWS interface to enable the Constraint service to retrieve facts from the fact store about users (actors) who initiate or are the target of actions within an activity flow

5.2 Module Design in terms of Classes

5.2.1 BizTalk Server 2004 Engine

To enable users to create a business process that spans multiple applications, the BizTalk Server 2004 engine must provide two capabilities: a way to specify the business process, and some mechanism for communicating between the applications the business process uses. The following figure shows the main components of the BizTalk Server 2004 engine that provide these capabilities.

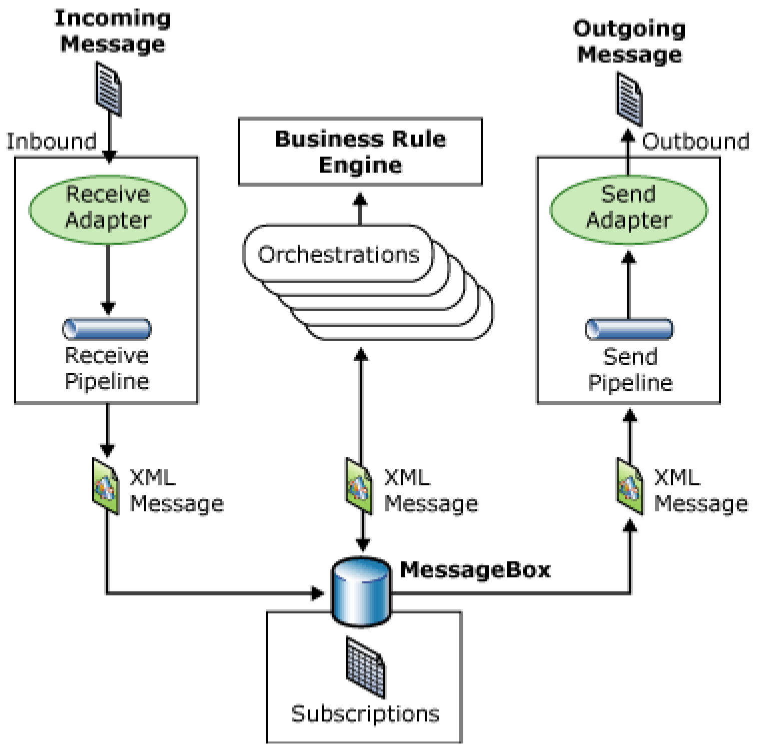


Figure 5.6 BizTalk Server engine

A business process is implemented as one or more orchestrations, each of which consists of executable code. These orchestrations are not created by writing code in a language such as C#, however. Instead, a business analyst uses the Orchestration Designer for Business Analysts (a Microsoft Visio snap-in) to graphically organize a defined group of shapes to express the conditions, loops, and other behavior of the business process. Business processes can also use the Business Rule Engine, which provides a simpler and more easily modified way to express the rules in a business process. Each orchestration creates subscriptions to indicate the kinds of messages it wants to receive.

The message processing shown in the preceding figure is as follows:

1. A message is received through a receive adapter. Different adapters provide different communication mechanisms, so a message might be acquired by accessing a Web service, reading from a file, or in some other way.
2. The message is processed through a receive pipeline. This pipeline can contain components that perform actions such as converting the message from its native format into an XML document or validating the digital signature for the message.
3. The message is delivered into a database called the MessageBox database, which is implemented by using Microsoft SQL Server.
4. The message is dispatched to its target orchestration, which takes whatever action the business process requires.
5. The result of this processing is typically another message, produced by the business process and saved in the MessageBox database.
6. This message, in turn, is processed by a send pipeline, which may convert it from the internal XML format used by BizTalk Server 2004 to the format required by its destination, add a digital signature, and more.
7. The message is sent out through a send adapter, which uses an appropriate mechanism to communicate with the application for which this message is destined.

Three roles are necessary to create and maintain BizTalk Server 2004 applications. These roles, and the functions they perform with the BizTalk Server 2004 engine, are as follows:

- **Business analyst.** Defines the rules and behaviors that make up a business process, and determines the flow of the business process by defining what information is sent to each application and how one business document is mapped into another.
- **Developer.** Implements the business process defined by the business analyst. Implementation includes tasks such as defining the XML schemas for the business documents that will be used, specifying the detailed mapping between them, and creating the orchestrations necessary to implement the business process.
- **Administrator.** Performs tasks such as setting up communication among the parts and deploying the application in an appropriately scalable way.

5.2.2 Connecting Applications

Effectively exchanging messages with applications is an absolute requirement for integration. Given the diversity of communication styles that exist, the BizTalk Server 2004 engine must support a variety of protocols and message formats. As described next, a significant portion of the BizTalk Server 2004 engine is devoted to making this communication work. One important fact to keep in mind, however, is that the engine works only with XML documents internally. Whatever format a message arrives in, it is always converted to an XML document after it is received. Similarly, if the recipient of a document cannot accept that document as XML, the engine converts it into the format expected by the target application.

5.2.3 Sending and Receiving Messages: Adapters

Because the BizTalk Server 2004 engine must talk to a wide range of other software, it relies on a range of adapters to make this possible. An adapter is an implementation of a communication mechanism, such as a particular protocol. BizTalk Server 2004 provides built-in adapters, and adapters have been created for popular applications such as SAP. A developer can determine which adapters to use in a given situation, or can create custom adapters for specific needs. All adapters are built on a standard base called the Adapter Framework. New with BizTalk Server 2004, this framework provides a common way to create and run adapters. It also enables you to use the same tools to manage both standard and custom adapters. While adapters written for earlier versions of BizTalk Server will not work with BizTalk Server 2004, the Adapter Framework makes it easier to create new adapters for this new release.

BizTalk Server 2004 provides the following adapters:

- **SOAP adapter.** Enables sending and receiving messages by using SOAP over HTTP. Because SOAP is the core protocol for Web services, this adapter gives BizTalk Server 2004 the ability to interact in a Web services world. As usual with Web services, URLs are used to identify the sending and receiving systems.

- **BizTalk Message Queuing adapter.** Enables sending and receiving messages by using BizTalk Message Queuing (MSMQT). BizTalk Message Queuing is an implementation of the Microsoft Message Queuing (MSMQ) protocol that can receive and send MSMQ messages from and to the MessageBox database. It is not a replacement for MSMQ, but rather an efficient way to use the MSMQ transport with BizTalk Server.

- **File adapter.** Enables reading from and writing to files in the Windows file system. Because the applications involved in a business process can often access the same file system, either locally or across a network, exchanging messages through files can be a convenient option.

- **HTTP adapter.** Enables sending and receiving information by using HTTP. The BizTalk Server 2004 engine exposes one or more URLs to allow other applications to send data to it, and it can use this adapter to send data to other URLs.

- **SMTP adapter.** Enables sending messages by using Simple Mail Transfer Protocol (SMTP). Standard e-mail addresses are used to identify the parties.

- **SQL adapter.** Enables reading and writing information from and to a SQL Server database.

- **Base EDI adapter.** Enables sending and receiving messages by using the American National Standards Institute (ANSI) X-12 and Electronic Data Interchange for Administration, Commerce, and Trade (EDIFACT) standards.

- **FTP adapter.** Enables exchange of files between BizTalk and FTP servers.

The messages that an adapter receives usually need to be processed before an orchestration can access them. Similarly, outgoing messages produced by an orchestration often need to be processed before an adapter sends them. How this processing is done is described next.

5.2.4 Processing Messages: Pipelines

The applications underlying a business process communicate by exchanging various kinds of documents, such as purchase orders and invoices. For a BizTalk Server 2004 application to execute a business process, it must be able to correctly process the messages that contain these documents. This processing can involve multiple steps, and it is performed by a message pipeline. Incoming messages are processed through a receive pipeline, while outgoing messages go through a send pipeline.

For example, even though more and more applications understand XML documents, many—probably even the majority today—do not. Because the BizTalk Server 2004 engine works only with XML documents internally, it must provide a way to convert other formats to and from XML. Other services may also be required, such as authenticating the sender of a message. To handle these and other tasks in a modular way, a pipeline is constructed from some number of stages, each of which contains one or more .NET or COM components. Each component handles a particular part of the message processing. The BizTalk Server 2004 engine provides several standard components that address the most common cases. If these are not sufficient, developers can also create custom components for both receive and send pipelines.

BizTalk Server 2004 defines some default pipelines, including a simple receive/send pair that you can use for handling messages that are already expressed in XML. A developer can also create custom pipelines by using the Pipeline Designer tool. This tool, which runs inside Visual Studio .NET, provides a graphical interface that enables dragging and dropping components to create receive and send pipelines with the required behavior.

The following figure illustrates the stages in a receive pipeline, along with the standard components provided for each one.

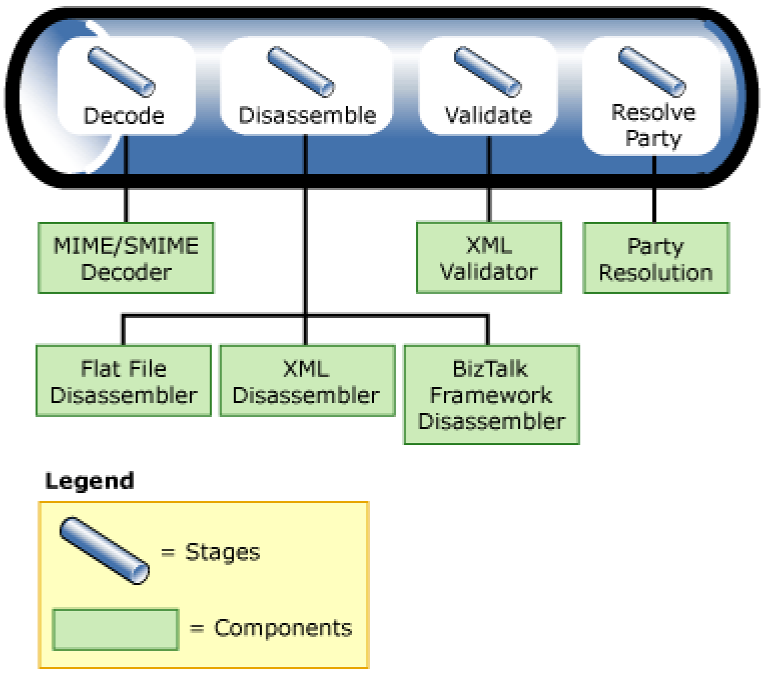


Figure 5.7 Receive pipeline

The stages and their associated components are:

- **Decode.** BizTalk Server 2004 provides one standard component for this stage, the MIME/SMIME decoder. This component can handle messages and any attachments they contain in either MIME or Secure MIME (S/MIME) format. The component converts both kinds of messages into XML, and it can also decrypt S/MIME messages and verify their digital signatures.

- **Disassemble.** BizTalk Server 2004 provides three standard disassembler components. These are:

    - **Flat file disassembler.** Turns flat files into XML documents. Those files can be positional, where each record has the same length and structure, or delimited, with a designated character used to separate records in the file.
    - **XML disassembler.** Parses incoming messages that are already described by using XML.
    - **BizTalk Framework disassembler.** Accepts messages sent by using the reliable messaging mechanism defined by the BizTalk Framework. The BizTalk Framework was implemented in BizTalk Server 2000 and will eventually become less relevant when Web services reliable messaging specifications are widely adopted.

- **Validate.** BizTalk Server 2004 provides an XML validator component for this stage. As its name suggests, this component validates an XML document produced by the Disassemble stage against a specified schema or group of schemas, returning an error if the document does not conform to one of those schemas.

- **Resolve Party.** The only standard component for this stage, party resolution, attempts to determine an identity for the sender of the message. If the message was digitally signed, the signature is used to look up a Windows identity in the BizTalk Server 2004 Configuration database. If the message carries the authenticated security identifier (SID) of a Windows user, this identity is used. If neither mechanism succeeds, the sender of the message is assigned a default anonymous identity.

Outgoing messages can also go through multiple stages, as defined by a send pipeline. The following figure shows the stages and standard components for a send pipeline.

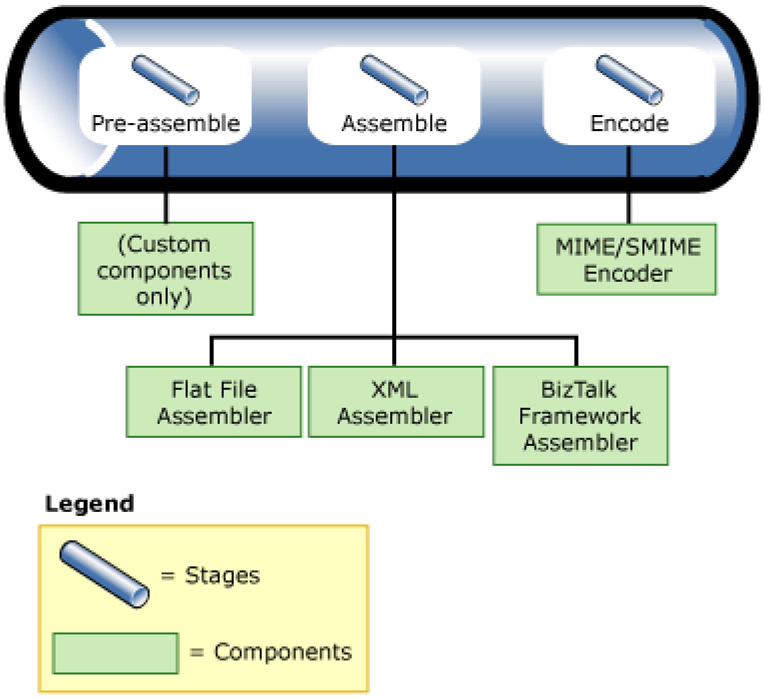


Figure 5.8 Send pipeline

The stages and their associated components are:

- **Pre-assemble.** No standard components are provided. Instead, custom components can be inserted here as needed.

- **Assemble.** Paralleling the Disassemble stage in a receive pipeline, this stage also has three standard components:

    - **Flat file assembler.** Converts an XML message into a positional or delimited flat file.
    - **XML assembler.** Enables adding an envelope and making other changes to an outgoing XML message.
    - **BizTalk Framework assembler.** Packages messages for reliable transmission by using the BizTalk Framework messaging technology.

- **Encode.** BizTalk Server 2004 defines one standard component for this stage, the MIME/SMIME encoder. This component packages outgoing messages in either MIME or S/MIME format. If S/MIME is used, the message can also be digitally signed and/or encrypted.

5.2.5 Choosing Messages: Subscriptions

After a message has passed through an adapter and a receive pipeline, the business process must determine where it should go. A message is most often targeted to an orchestration, but it is also possible for a message to go directly to a send pipeline, using the BizTalk Server 2004 engine purely as a messaging system. In either case, messages are matched with their destinations through subscriptions.

When a receive pipeline processes a message, it creates a message context that contains various properties of the message. An orchestration or a send pipeline can subscribe to messages based on the values of these properties. For example, an orchestration might create a subscription that matches all messages of the type "Invoice", or all messages of the type "Invoice" received from Woodgrove Bank, or all messages of the type "Invoice" received from Woodgrove Bank that are for more than $10,000. However, it is specified, a subscription returns to its subscriber only those messages that match the criteria that the subscription defines. A received message might initiate a business process by instantiating some orchestration or it might activate another step in an already running business process. Similarly, when an orchestration sends a message, that message is matched to a send pipeline based on a subscription that the pipeline has established.

5.3 User interface description

A developer relies on three primary tools for creating an orchestration: BizTalk Editor for creating XML schemas, BizTalk Mapper for defining translations between those schemas, and Orchestration Designer for specifying the flow of business processes. Unlike earlier versions of the product, all of the developer tools in BizTalk Server 2004 are hosted inside Visual Studio .NET, providing a consistent environment. This section describes what each of these tools does and how they work together.

5.3.1 Defining Business Processes

Sending messages between different applications is a necessary part of solving the problems that BizTalk Server 2004 addresses. Yet in most cases, it is only a means to an end. The real goal is to define and execute business processes based on these applications. The BizTalk Server 2004 engine provides two technologies for doing this: orchestration and the Business Rule Engine.

Orchestration

You can implement a business process directly in a language such as C# or Microsoft Visual Basic® .NET. However, creating, maintaining, and managing complex business processes in conventional programming languages can be challenging. Just as important, a process created in this way is likely to act as a set of disparate applications rather than as a coherent business process. Like its predecessors, then, BizTalk Server 2004 does not take this approach. Instead, it enables you to create a business process graphically. Doing this can be faster than building the process directly in a programming language, and it can make the process easier to understand, explain, and change. Business processes built in this fashion can also be monitored more easily, a fact that is exploited by the Business Activity Monitoring (BAM) technology described later in this document.

Successfully creating an automated business process usually requires collaboration between technology-oriented developers and less-technical business people. Accordingly, BizTalk Server 2004 provides appropriate tools for each. The developer tools run inside Visual Studio .NET, an environment in which software professionals feel at home. Most business people do not find Visual Studio especially inviting, however, so BizTalk Server 2004 also provides a subset of the developer tool functionality in the Orchestration Designer for Business Analysts (ODBA), an add-in for Microsoft Visio. Information created in the ODBA tool can be imported into the Visual Studio-based tools and vice-versa, which helps developers and business analysts work together when creating a business process. After it exists, this process, known as an orchestration, is automatically transformed into standard assemblies that run on the .NET Framework.

5.3.2 Creating Scalable Configurations

If the application using it is not too large, the entire BizTalk Server 2004 engine can be installed on a single computer. In many situations, however, this is not the right solution. The number of messages the engine must handle might be too great for one computer, or redundancy might be required to make the system more reliable. To meet requirements like these, the BizTalk Server 2004 engine can be deployed in a number of ways.

A fundamental concept for deploying the engine is the idea of a host. A host can contain various things, including orchestrations, adapters, and pipelines. Hosts are just logical constructs; to use them, a BizTalk Server 2004 administrator must cause actual host instances to be created. Each host instance is a Windows process, and as the following figure shows, it can contain various elements.

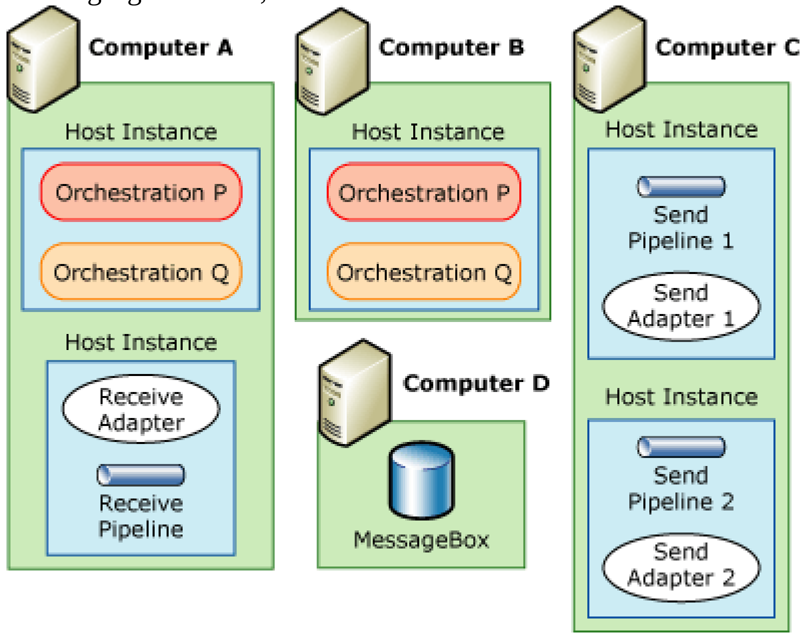


Figure 5.9 Host computers

In this figure:

- Computer A runs two host instances. One contains a receive adapter and receive pipeline, while the other contains the orchestrations P and Q.
- Computer B runs just one host instance, also containing the orchestrations P and Q.
- Computer C, like computer A, runs two host instances, but neither of them contains an orchestration. Instead, each of these instances contains a send pipeline and send adapter.
- Computer D houses the MessageBox database that is used by all of the host instances in this configuration.

This example illustrates several ways in which hosts might be used. For example, because both computers A and B run host instances that contain the orchestrations P and Q, BizTalk Server 2004 can automatically assign requests to these orchestrations based on the availability and current load on each computer. This allows a business process to scale up as needed for high-volume applications. Notice also that computer C contains two different ways to handle outgoing messages. One way might rely on a standard BizTalk Server 2004 adapter, such as the HTTP adapter, while the other might use a custom adapter to communicate with a particular application. Grouping all output processing on a single computer like this can make good sense in some situations. And because each host instance is isolated from every other host instance—they are different processes—it is safer to run code that is not completely trusted, such as a new custom adapter, in a separate instance. It is also worth pointing out that even though this example contains only a single instance of the MessageBox database, you can also replicate or cluster the databases to avoid creating a single point of failure.

5.3.3 Creating Schemas: BizTalk Editor

Orchestration works with XML documents, each of which conforms to some XML schema. Accordingly, there must be a way to define these schemas. To do this, the BizTalk Server 2004 engine provides BizTalk Editor. This tool enables you to use the XML Schema definition language (XSD) to create schemas, which are essentially definitions of the types and structure of the information in a document. While BizTalk Editor is not new with this release, this complete reliance on standard XSD is. Because the final Worldwide Web Consortium (W3C) standard for XSD was not complete when they were developed, previous versions of BizTalk Server used XML Data Reduced (XDR), a Microsoft-specific language for defining schemas. Now that XSD has been finalized, BizTalk Server 2004 relies on this standard approach for describing the structure of XML documents.

Creating raw XSD schemas without some tool support is not simple. To make this necessary step more approachable, BizTalk Editor enables its user—probably a developer—to build a schema by defining its elements in a graphical hierarchy. Existing schemas can also be imported from either files or accessible Web services. However they are acquired, schemas are used as the basis for BizTalk maps, which are described next.

5.3.4 Mapping Between Schemas: BizTalk Mapper

An orchestration implementing a business process typically receives some documents and sends others. It is common for part of the information in the received documents to be transferred to the sent documents, perhaps transformed in some way. For example, an order fulfillment process might receive an order for some number of items, and then send back an acknowledgment indicating that the order was sent. It is possible that information from the order, such as the name and address of the purchaser, might be copied from fields in the received order into fields in the order acknowledgment. You can use BizTalk Mapper to define a transformation—a map—from one document to the other.

To the developer creating it, each map is expressed as a graphical correlation between two XML schemas that defines a relationship between elements in those schemas. The W3C has defined the Extensible Stylesheet Language Transformation (XSLT) as a standard way to express these kinds of transformations between XML schemas; therefore, as in previous versions of BizTalk Server, maps in BizTalk Server 2004 are implemented as XSLT transformations.

The transformation defined in a map can be simple, such as copying a name and address from one document to another. Direct data copies like this are expressed by using a link, which is shown in BizTalk Mapper as a line connecting the appropriate elements in the source schema with their counterparts in the destination schema. More complex transformations are also possible by using functoids. A functoid is a chunk of executable code that can define arbitrarily complex mappings between XML schemas, and BizTalk Mapper represents it as a box on the line connecting the elements being transformed. Because some of those transformations are fairly common, BizTalk Server 2004 includes a number of built-in functoids. These built-in functoids are grouped into categories that include the following:

- **Mathematical functoids.** Perform operations such as adding, multiplying, and dividing the values of fields in the source document and storing the result in a field in the target document.

- **Conversion functoids.** Convert a numeric value to its ASCII equivalent and vice-versa.

- **Logical functoids.** Used to determine whether an element or attribute should be created in the target document based on a logical comparison between specified values in the source document. Those values can be compared for equality, greater than/less than, and in other ways.

- **Cumulative functoids.** Compute averages, sums, or other values from various fields in the source document, and then store the result in a single field in the target document.

- **Database functoids.** Access information stored in a database.

You can create custom functoids directly in XSLT or by using .NET languages like C# and Visual Basic .NET. (Earlier versions of BizTalk Server used Visual Basic Scripting Edition (VBScript) or Microsoft JScript® to create custom functoids, and you may need to modify maps that use these to work with BizTalk Server 2004.) Functoids can also be combined in sequences, cascading the output of one into the input of another.

Having a way to define the XML schema of a document is essential, as is a mechanism for mapping information across documents with different schemas. BizTalk Editor and BizTalk Mapper address these two functions. Yet defining schemas and maps is not enough. You must also specify the business process that will use the schemas and invoke the maps. How this is done is described next.

5.3.5 Defining Business Processes: Orchestration Designer

A business process is a set of actions that together meet some useful business need. With the BizTalk Server 2004 engine, you can use Orchestration Designer to define these actions graphically. This tool enables you to create an orchestration by connecting a series of shapes in a logical way, rather than expressing the steps in a programming language. Some of the most commonly used shapes are:

- **Receive.** Enables the orchestration to receive messages. A Receive shape can have a filter that defines exactly what kinds of messages should be received, and it can be configured to start a new instance of an orchestration when a new message arrives.

- **Send.** Enables the orchestration to send messages.

- **Port.** Defines how messages are transmitted. Each instance of a Port shape is connected to either a Send or Receive shape. Each port also has a type, which defines things such as what kinds of messages this port can receive; a direction, such as send or receive; and a binding, which determines how a message is sent or received by, for example, specifying a particular URL and other information.

- **Decide.** Represents an if-then-else statement that allows an orchestration to perform different tasks based on Boolean conditions. You can use the Expression Editor, part of Orchestration Designer, to specify this conditional statement.

- **Loop.** Enables performing an action repeatedly while some condition is true.

- **Construct Message.** Enables building a message.

- **Transform.** Enables transferring information from one document to another, transforming it on the way by invoking maps defined with BizTalk Mapper.

- **Parallel Actions.** Enables specifying that multiple operations should be performed in parallel rather than in sequence. The shape that follows this one will not be executed until all of the parallel actions have completed.

- **Scope.** Enables grouping operations into transactions and defining exception handlers for error handling. Both traditional atomic transactions and long-running transactions are supported. Unlike atomic transactions, long-running transactions rely on compensating logic rather than rollback to handle unexpected events.

- **Message Assignment.** Enables assigning values to orchestration variables. These variables can be used to store state information used by the orchestration, such as a message being created or a character string.

After you have defined a business process in this way, the group of shapes and relations between them is converted into the Microsoft intermediate language (MSIL) that is used by the .NET Framework common language runtime (CLR). Ultimately, the group of shapes that you define in BizTalk Server 2004 becomes just a standard .NET assembly. And of course, you can still add explicit code to an orchestration when necessary by calling a COM or .NET object from inside a shape.

5.3.6 Additional Orchestration Support

The rise of Web services is having an impact on how business processes are defined. For example, think about the case where two organizations interact by using Web services. To interoperate effectively, it might be necessary for each side of the interaction to know something about the business process the other is using. If both organizations use BizTalk Server 2004, this is not a big problem; tools such as the Trading Partner Management technology described later in this document can be used to distribute this knowledge. But what if the organizations use different software? What if one organization uses BizTalk Server 2004 and the other uses software from another vendor? For cases like this, it is useful to have a way to describe the business process interactions in a cross-vendor way.

To support this type of interaction, Microsoft, IBM, and others have created Business Process Execution Language (BPEL). A business process defined by using Orchestration Designer can be exported to BPEL, and BizTalk Server 2004 can also import processes defined in BPEL. While the language is useful for describing and sharing the external-facing parts of a business process between trading partners, it is important to realize that BPEL is not focused on cross-platform execution of business processes. It is also important to understand that BPEL is built entirely on Web services, while BizTalk Server 2004 and other products that support this language provide a broader set of services. For example, BizTalk Server 2004 provides support for mapping between different XML schemas, calling methods in local objects, executing transactions, and other features that are not available in BPEL.

Web services enable applications to exchange XML documents through SOAP, and they have had a big impact on integration platforms. Unlike previous versions of the product, BizTalk Server 2004 has built-in support for Web services. To access an external Web service, the creator of an orchestration can use the Add Web Reference option in Visual Studio .NET along with the SOAP adapter to directly invoke operations, just as with any other .NET assembly. Similarly, BizTalk Server 2004 provides the Web Services Publishing Wizard that can generate an ASP.NET Web service project exposing one or more of operations of an orchestration as SOAP-callable Web services. These two options enable you to both access existing Web services from within a business process and expose the functionality of an orchestration as a Web service to other business processes.

Like the other developer tools provided by BizTalk Server 2004, Orchestration Designer runs inside Visual Studio .NET. In some cases, however, a business analyst rather than a developer may wish to graphically define business processes. Because business analysts may not be comfortable using Visual Studio .NET, BizTalk Server 2004 also includes the Orchestration Designer for the Business Analyst (ODBA) tool, an add-in for Microsoft Visio that enables you to define a business process and then import that process into Orchestration Designer.

Orchestrations are the fundamental mechanism for creating business processes in BizTalk Server 2004. Some aspects of an orchestration tend to change more often than others, however. In particular, the decisions embedded in a business process—the business rules—are commonly its most volatile aspect. The spending limit for a manager was $100,000 last week, but a promotion bumps this up to $500,000, or the maximum allowed order for a slow-paying customer decreases from 100 units to only 10. Why not provide an explicit way to specify and update these rules? This is exactly what the Business Rule Engine does, as described next.

5.3.7 Business Rule Engine

Orchestration Designer, together with BizTalk Editor and BizTalk Mapper, is an effective way to define a business process and the rules it uses. It is sometimes useful, though, to have an easier way to define and change business rules. To allow this, BizTalk Server 2004 provides the Business Rule Engine to enable more business-oriented users to directly create and modify sets of business rules. These rules are created by using a tool called the Business Rule Composer, and then executed directly by the engine. This technology is new in BizTalk Server 2004, and it is one of the most interesting features in the product.

To see why this approach is useful, think about what is required to change a business rule that is implemented within an orchestration. A developer must first open the orchestration in Visual Studio .NET, modify the appropriate shapes (and perhaps the .NET or COM objects they invoke), and then build and deploy the modified assembly. If instead this business rule is implemented by using the Business Rule Engine, it can be modified without recompiling or restarting anything. All that is needed is to use the Business Rule Composer to change the desired rule, and then redeploy the new set of rules. The change takes effect immediately. And while orchestrations are typically created and maintained by developers, business rules are readable enough to be modified by business analysts without the need to involve more technical people.

When you create a set of business rules, you typically begin by using the Business Rule Composer to define a vocabulary for use in specifying those rules. Each term in the vocabulary provides a user-friendly name for some information. For example, a vocabulary might define terms such as Number Shipped or Maximum Quantity of Items or Approval Limit. Each of these terms can be set to a constant or be mapped to a particular element or attribute in some XML schema (and thus in an incoming message), or to the result of an SQL query against some database, or even to a value in a .NET object.

Note: You can also use XML element and attribute names directly in definitions of business rules, so creating a vocabulary is not strictly required. It just makes business rules easier to create and understand, especially if business analysts are doing this rather than developers.

After you have defined a vocabulary, you can use the Business Rule Composer to create business policies that use the vocabulary. Each policy can contain one or more business rules. A rule uses the terms defined in some vocabulary together with logical operators such as Greater Than, Less Than, or Is Equal To to define how a business process operates. A business rule can define how values contained in a received XML document should affect the values created in an XML document that is sent, or how those received values should affect what value is written in a database, or other things.

Imagine, for example, a simple vocabulary that defines the term Maximum Allowed Order Quantity, whose value is set to 100, and the term Quantity Requested, whose value is derived from a specified element in received XML documents that correspond to the schema used for placing orders. A business analyst might create a rule stating that if the Quantity Requested in an incoming order is greater than the Maximum Allowed Order Quantity, the order should be rejected, perhaps resulting in an appropriate XML document being sent back to the originator of this order.

To execute a business policy, an orchestration can contain a Call Rules shape that creates an instance of the Business Rule Engine, identifies which policy to execute, and then passes in the information this rule needs, such as a received XML document. The Business Rule Engine can also be invoked programmatically through a .NET-based object model, which allows it to be called from applications that do not use the BizTalk Server 2004 engine. This means that Windows Forms applications, software exposing Web services, and anything else built on the .NET Framework can use the Business Rule Engine whenever it helps to solve the problem at hand.

Both vocabularies and business rules can be much more complicated—and much more powerful—than the simple examples described here. But the core idea of defining a vocabulary and then defining sets of rules that use that vocabulary is the heart of the Business Rule Engine. The goal is to provide a straightforward way for BizTalk Server 2004 users of all kinds to create and work with the rules that define business processes.

Orchestration abstracts the flow of a business process, expressing it graphically rather than in code. Similarly, the Business Rule Engine enables you to express the rules of a business process in a higher-level way. Together, these two technologies provide an effective approach to creating business logic.

5.3.8 Managing Applications

The BizTalk Server 2004 engine provides a range of services, and several different tools are used to manage this environment. The primary tool is the BizTalk Administration console, a Microsoft Management Console (MMC) snap-in. This tool enables you to create hosts, assign hosts to computers, start and stop orchestrations, and perform many other administration tasks. You can even dynamically add computers and specify what hosts should be assigned to them while an application is running—there is no need to shut the application down to make these changes. You can also access the functions of the Administration console programmatically through Windows Management Instrumentation (WMI), which enables you to create scripts that automate management functions.

Other management tools are also available for more specific purposes. For example, you can use the BizTalk Deployment Wizard to deploy assemblies to computers, and you can use the Subscription Viewer to examine subscriptions in the BizTalk Server 2004 engine. All of these tools, along with the BizTalk Administration console, rely on a common Configuration database to store the information that they work with.

One more important tool for working with BizTalk Server 2004 applications is BizTalk Explorer. It is new with BizTalk Server 2004, and to understand what it does, think about how a developer builds an application. Initially, a developer creating an application works mainly in logical terms, without defining details. For example, a developer might specify that the BizTalk Server 2004 engine will communicate with a particular application through the HTTP adapter without defining the URL that will be used, or specify that the send pipeline should add a digital signature to outgoing messages without defining the key that will be used to create this signature.

Yet to make the application work, these details must be specified, which is where BizTalk Explorer comes in. Relying on the same Configuration database that is used by the other BizTalk Server 2004 management tools, this tool enables mapping between the logical view of an orchestration and the concrete physical view required to actually use that orchestration. BizTalk Explorer can also be helpful in working with partners, because it enables you to create a single application configuration, and then deploy this configuration differently for each partner. And because it exposes its services through a .NET-based object model as well as a graphical interface, you can create scripts for configuring applications programmatically.

5.3.9 Monitoring Applications: Health and Activity Tracking

BizTalk Server 2004 applications do many things: send and receive messages, process messages within orchestrations, communicate with various applications using different protocols, and more. It is essential to keep track of what is going on, especially when failures occur. Doing this requires a way to look into running applications and to monitor what is happening with the system, which exactly describes the services provided by the Health and Activity Tracking (HAT) component of BizTalk Server 2004.

The HAT tool provides graphical access to detailed information, including when an orchestration starts and ends, when each shape within it is executed, when each of its messages is sent and received, what is in those messages, and much more. You can even set breakpoints, enabling you to stop the orchestration and examine it at predetermined places. You can also use the HAT tool to examine archived data, looking for patterns and trends in the execution of a business process. This information is useful for debugging, answering business questions (such as verifying that a message really was sent to a customer), and keeping ongoing statistics that can be used to improve performance.

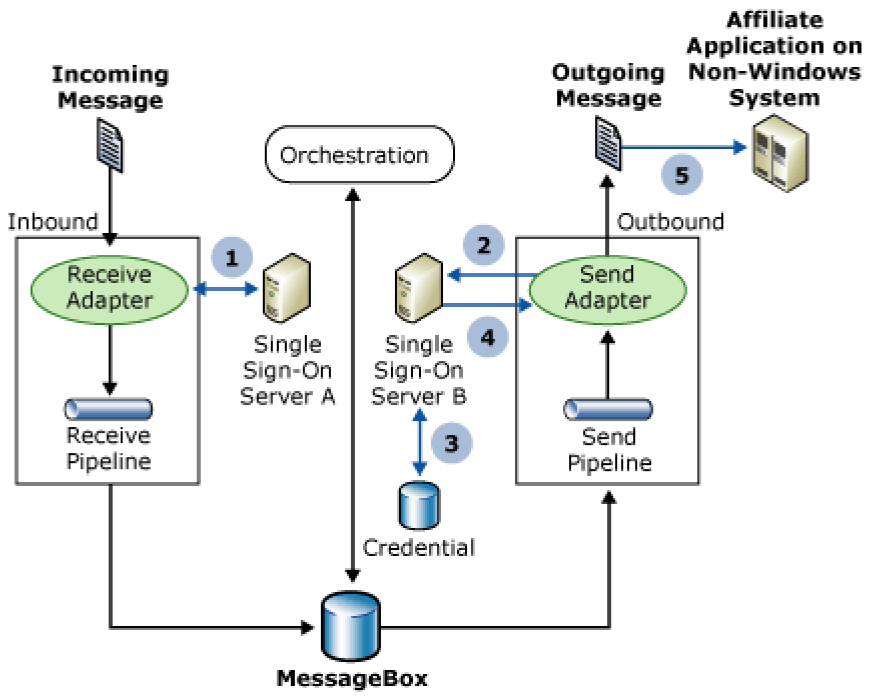
5.3.10 Enterprise Single Sign-On

A business process that relies on several different applications is likely to face the challenge of dealing with several different security domains. Accessing an application on a Windows system may require one set of security credentials, while accessing an application on an IBM mainframe may require different credentials, such as an RACF user name and password. Dealing with this profusion of credentials is hard for users, and it can be even harder for automated processes. To address this problem, BizTalk Server 2004 includes Enterprise Single Sign-On (SSO).

Enterprise Single Sign-On provides a way to map a Windows user ID to non-Windows user credentials. It will not solve all of enterprise sign-on problems for an organization, but this service can make things simpler for business processes that use applications on diverse systems.

To use SSO, you define affiliate applications, each of which represents a non-Windows system or application. For example, an affiliate application might be a CICS application running on an IBM mainframe, an SAP ERP system running on UNIX, or any other kind of software. Each of these applications has its own mechanism for authentication, and so each requires its own unique credentials.

SSO stores an encrypted mapping between a users Windows user ID and the credentials for that user for one or more affiliate applications in the Credential database. When a user needs to access an affiliate application, an SSO server can look up the credentials for that user for the application in the Credential database. The following figure shows how this works.



1 =Get SSO ticket for user X
2 =Redeem SSO ticket
3 =Get user X's credentials for affiliate application
4 =Return user X's credentials for affiliate application
5 =Send message with user X's credentials for affiliate application

Figure 5.10 Enterprise Single Sign-On

In this example, a message sent by some application to BizTalk Server 2004 is processed by an orchestration and then sent to an affiliate application running on an IBM mainframe. The job of Enterprise Single Sign-On is to make sure that the correct credentials (for example, the right user name and password) are sent with the message when it is passed to the affiliate application.

The figure illustrates a five-step process:

1. When a receive adapter gets a message, the adapter can request an SSO ticket from SSO server A. This encrypted ticket contains the Windows identity of the user that made the request and a time-out period. (Do not confuse this with a Kerberos ticket—it is not the same thing.) After it is acquired, the SSO ticket is added as a property to the incoming message. The message then takes its normal path through the BizTalk Server 2004 engine, which in this example means being handled by an orchestration. When this orchestration generates an outgoing message, that message also contains the SSO ticket acquired earlier.

2. This new message is destined for the application running on an IBM mainframe, and so it must contain the appropriate credentials for this user to access that application. To get these credentials, the send adapter contacts SSO server B, supplying the message (which contains the SSO ticket) that it just received and the name of the affiliate application for which it wants to retrieve the credentials.

3. This operation, called redemption, causes SSO server B to verify the SSO ticket, and then look up the credentials for this user for that application.

4. SSO server B returns those credentials to the send adapter.

5. The send adapter uses the credentials to send an appropriately authenticated message to the affiliate application.

Enterprise Single Sign-On also includes administration tools to perform various operations. All operations performed on the Credential database are audited, for example, so tools are provided that enable you to monitor these operations and set various audit levels. Other tools enable you to disable a particular affiliate application, turn on and off an individual mapping for a user, and perform other functions. There is also a client utility that enables end users to configure their own credentials and mappings. And like other parts of BizTalk Server 2004, Enterprise Single Sign-On exposes its services through a programmable API. The creators of third-party BizTalk Server adapters use this API to access the single sign-on services, and administrators can use it to create scripts for automating common tasks.

The preceding example shows what is likely to be a typical use of Enterprise Single Sign-On, but it is not the only option. A smaller BizTalk Server 2004 installation might have only a single SSO server, for example, and it is even possible to use Enterprise Single Sign-On independently from the BizTalk Server 2004 engine. Because business processes implemented by using BizTalk Server 2004 need to interact with diverse applications, however, making this service part of the product makes good sense.

5.4 Technical and Installation information

Minimum Software Requirements

This section contains the minimum software requirements for a server computer. This section also provides additional information for the client computers needed for Business Activity Services (BAS), and Single Sign-On (SSO) client utility.

Server Computer

The minimum software requirements for a complete installation of BizTalk Server 2004 on a single computer include:

- Microsoft Windows® 2000 Server with Service Pack 4, Windows XP Professional with Service Pack 1 and NTFS file system, or Microsoft Windows Server™ 2003 Standard, Enterprise, or Datacenter Edition.

Important Windows Server 2003 is required for Business Activity Services and Windows SharePoint™ Services .
Important Windows Server 2003 Web Edition is not compatible with BizTalk Server 2004.

- Microsoft Visual Studio® .NET 2003 with Microsoft Visual C#® .NET
- Microsoft SQL Server™ 2000 Enterprise, Standard, or Developer Edition with Service Pack 3a

Important SQL Server 2000 Personal Edition is not compatible with BizTalk Server 2004.

- Microsoft SQL Server 2000 Analysis Services with Service Pack 3a

Note SQL Server and Analysis Server can be remote installations.

- Windows SharePoint™ Services (required for Business Activity Services only)

Install or update the following Windows components prior to installing BizTalk Server and all its components.

Windows XP Windows Update	**Windows 2000 Server Windows Update**	**Windows Server 2003 Windows Update**
Windows XP Service Pack 1a	Windows 2000 Service Pack 4	
Microsoft XML Core Services (MSXML) 4.0 with Service Pack 2	Microsoft XML Core Services (MSXML) 4.0 with Service Pack 2	Microsoft XML Core Services (MSXML) 4.0 with Service Pack 2

Microsoft XML Core Services (MSXML)3.0 with Service Pack 4	Microsoft XML Core Services (MSXML)3.0 with Service Pack 4	Microsoft XML Core Services (MSXML)3.0 with Service Pack 4
Microsoft .NET Framework 1.1	Microsoft .NET Framework 1.1	Microsoft .NET Framework 1.1
Microsoft Data Access Components (MDAC) 2.8	Microsoft Data Access Components (MDAC) 2.8	
SQLXML 3.0 with Service Pack 2*	SQLXML 3.0 with Service Pack 2*	SQLXML 3.0 with Service Pack 2*
Microsoft Internet Explorer 6 Service Pack 1	Microsoft Internet Explorer 6 Service Pack 1	
Microsoft Office Web Controls 10	Microsoft Office Web Controls 10	Microsoft Office Web Controls 10
Updates Q828758, Q329433, and Q831950**	Updates Q828748, Q329433, Q821887, and Q831950**	Update Q831950**

Client Computer

If you are configuring Business Activity Services (BAS) the minimum software requirement for a client computer includes:

- Microsoft Office InfoPath™ 2003 (This must be installed on every computer that accesses Business Activity Services.)
- Microsoft Internet Explorer 6 Service Pack 1

If you are using the Single Sign-On (SSO) client utility (ssoclient.exe), the minimum requirement for a client computer includes:

- Windows XP Professional with Service Pack 1, Windows 2000 with Service Pack 4, or Windows Server 2003

The SSO client package is available as a self-extracting file (SSOClientInstall.exe) installed with the SSO administration feature. The SSOClient.exe enables users to configure their client mappings in the credential database.

The installation of the stand-alone SSO client (ssoclient.exe) and SSO administrative utilities (ssomanage.exe and ssoconfig.exe) does not create shortcuts on the **Start** menu to access the command line utilities. To run the SSO client or SSO administrative utilities after installation, you must open a command prompt and navigate to the SSO directory: \Program Files\Common Files\Enterprise Single Sign-On.

References to other major components needed	•
Problems not solved	•
Configuration and execution context	See instructions above

Note: - Please visit following URLs for installation problems that might occur because of Microsoft Windows Security updates

Link describing solution of .NET framework 1.1 updating problem
http://www.ureader.com/message/1093268.aspx

Link that describes the solution to Single Sign-On problem
http://geekswithblogs.net/darko/archive/2005/11/21/60711.aspx

http://support.microsoft.com/default.aspx?scid=k;en-us;841893

Link describing the issues related to user accounts for MS SQL 2000 server
http://geekswithblogs.net/mhamilton/archive/2005/05/01/38784.aspx

Link describing solutions to pipeline problem
http://firechewy.com/blog/archive/2005/02/27/551.aspx

5.5 Draft User Manual

Microsoft Biztalk Server documentation is downloaded together with Microsoft Biztalk product.

5.6 Examples of usage

Microsoft Biztalk Server documentation contains Tutorials and Samples.

5.7 Integration and compilation issues

5.8 Configuration Parameters

Configuration Parameters and Settings are defined during Microsoft Biztalk Installation Wizard: use default parameters and settings.

Config parameter	Possible values

5.9 Errors reported and that may occur

Error code	Description and rationales

5.10 Formal description of algorithm <……………>

None

name	
Method	
Description	
Input parameters	
Output parameters	

name	
Method	
Description	
Input parameters	
Output parameters	

6 Module or Executable Tool Workflow User Interface - Openflow

Module/Tool Profile		
Workflow User Interface - Openflow		
Responsible Name		
Responsible Partner		
Status (proposed/approved)	Approved	
Implemented/not implemented	Implemented	
Status of the implementation	First Prototype Completed	
Executable or Library/module (Support)	Executable OpenFlow User Interface based upon Zope User Interface	
Single Thread or Multithread	Multithreaded	
Language of Development	User interface: Zope DTML (a superset of html) Application logic: Python or DTML	
Platforms supported	Windows, Linux	
Reference to the AXFW location of the source code demonstrator	https://cvs.AXMEDIS.org/repos/.....................	
Reference to the AXFW location of the demonstrator executable tool for internal download	https://cvs.	
Reference to the AXFW location of the demonstrator executable tool for public download		
Address for accessing to WebServices if any, add accession information (user aNd Passwd) if any		
Test cases (present/absent)		
Test cases location	http://////////////////	
Usage of the AXMEDIS configuration manager (yes/no)	No	
Usage of the AXMEDIS Error Manager (yes/no)	No	
Major Problems not solved	-- --	
Major pending requirements	-- --	
Interfaces API with other tools, named as	Name of the communicating tools References to other major components needed	Communication model and format (protected or not, etc.)
WorkFlow Engine		Via Zope Web Server
Formats Used	Shared with	format name or reference to a section

Protocol Used	Shared with	Protocol name or reference to a section
Used Database name		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
Zope Web	based U.I. DTML, a superset of html Idem	Zope Web
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not
OpenFlow OpenFlow 1.1 GPL 2.0	Zope by Zope Corporation Zope 2.7.3 ZPL 2.0 (Zope Public Licence 2.0), open source, GPL compatible	Python by Stichting Mathematisch Centrum, Amsterdam, The Netherlands. Python 2.3 Free Open Source by Stichting Mathematisch Centrum, Amsterdam, The Netherlands.
Xmlrpclib by Secret Labs AB	and by Fredrik Lundh Xmlrpclib Free Open source by Secret Labs AB	and by Fredrik Lundh
C expat Library by James Clark C expat Mozilla Public Licence Version 1.1		
OpenFlow OpenFlow 1.1 GPL 2.0	Zope by Zope Corporation Zope 2.7.3 ZPL 2.0 (Zope Public Licence 2.0), open source, GPL compatible	Python by Stichting Mathematisch Centrum, Amsterdam, The Netherlands. Python 2.3 Free Open Source by Stichting Mathematisch Centrum, Amsterdam, The Netherlands.

6.1 General Description of the Module

main purpose and description, major functionalities, etc.

The workflow editor and viewer is the gateway interface for creating and changing new project workspaces referred to as NPDs in the terminology adopted for the AXMEDIS Workflow and object life cycle analysis elsewhere in our document.

Naturally the functionality of this editor/viewer at the level of NPD editing will be a subset of the use cases already set-out for the AXMEDIS workflow management system particularly focusing on the global management requirements of the NPD workspace including Actors, Objects, Processes, etc. as follows

A) Search is a generic use case that can search for anything at any time both synchronously (during user interaction online; i.e. whilst the client workspace-instance is online and running, or alternatively whilst it is in-pause/offline (asynchronous search). The search can be blind/exploratory or more specific as in a special case that can be inherited to search for eligible components to be worked on; as follows:
   - Search in the AXWF Database for (sub)processes that have to be worked by the user; this is based on specific search criteria
   - Retrieve from AXWF DB, the AXMEDIS Components associated with (sub)processes
   - Invoke the AXMEDIS Object Manager (or AXMEDIS DB Manager) to search for the components based on search criteria (to be used by the search engine deployed by the AXMEDIS Object Manager)

B) Create a New Product Development, which typically entails creating new AXMEDIS Object instance and a new workflow process instance; as follows:
   - Create a new process instance in the AXWF DB
   - Invoke the AXMEDIS Object Manager to create new AXMEDIS Object
   - Link the process instance to the AXMEDIS Object instance

C) Discard an NPD, which implies the termination of the process instance:
   - Cancel the process instance from the AXWF DB
   - Invoke the AXMEDIS Object Manager to remove the associated AXMEDIS Object

D) Add a Component to a specified NPD; this can imply the creation of a new sub-process instance:
   - Invoke the AXMEDIS Object Manager to create a new component in the AXMEDIS Object instance
   - If the process flow requires it, create a sub-process instance in the AXWF DB
   - Link the sub-process instance to the AXMEDIS Object component

E) Remove a Component from a specified NPD; this can imply the termination of a sub-process instance:
   - Invoke the AXMEDIS Object Manager to delete the component
   - If the process flow requires it, also cancel the associated sub-process instance

F) Start an Activity in the process flow instance (work_item) selected by the user (for example start editing a component):
   - The user selects from the work_items list and the choice of activity to start
   - The WF manager automatically performs the selected actions in the process flow:
   - Launching compositional/ formatting/ loading tool /publication tool /protection tool/ Program and publications engine; or launching the local PC editor tool (see below for details)

G) Group is responsible for bundling components, people, processes, partners, projects, teams, packets, digital assets products, etc into one entity which may be further referred to.

H) Show the Component in which the user has to work:
   - Invokes the local PC viewing tool for showing the component associated with a work-item in the work-item list (see below for details)

I) Track Component: shows the history of what has so far been performed on the component
   - Invokes the AXMEDIS Object Manager (or the AXMEDIS DB Manager) for retrieving the history of an AXMEDIS component associated with a work-item in the work-items list

J) Track CPA identifies the Critical Path Activities (CPA) and produces all the information regarding those activities e.g. people involved, components being worked on, processes needing attention, possibly implicitly or explicitly also tracks CPA-slack-critical objects/processes, etc.

K) Time-stamp Generate: it is an internal AXWFM Object Manager function, not visible to the user, typically invoked by the Check-in/Check-out function

L) The AXWFM Object Manager invokes the AXMEDIS Object Manager (or AXMEDIS DB Manager) to update the tracking information

M) Generate Version: again a function internal to the AXWFM Object Manager; can be explicitly executed by the user or automatically generated by the AXWF process
- The WF Object Manager invokes the AXMEDIS Object Manager (or AXMEDIS DB Manager) for updating the Object/component revision

N) List Work: lists in a hierarchical view the work-items in which the user can or has to perform activities
- The WF Object Manager retrieves from the AXWF DB the list of the work-items in which the user or team is involved

O) Select a workitem is responsible for selecting a workitem from the work-list

P) Complete a Task sends the WorkFlow engine the trigger that causes it to have the respective user activity recorded as completed and then to go to the next activity in the process-instance flow

Q) Distribute Work: AXWF function used for assigning activities to users
- The AXWF Object Manager invokes the AXWF DB for changing the process-instance information related to users

R) Change State/Phase: again a function internal to AXWF Object Manager; can be explicitly executed by the user or automatically generated by the AXWF process
- The AXWF Object Manager invokes the AXMEDIS Object Manager (or AXMEDIS DB Manager) to update the Object State/phase; phase change typically occurs as a result of developmental changes as reflected in the workflow-instance/NPD-instance

S) Global Viewer: shows details about the NPD
- Invokes the local PC viewing tool for showing the AXMEDIS Object associated to a work-item in the work-item list (see below for details)

T) Notification: used to send notifications to other users in the project
- This may or may not invoke an external Notification engine

U) Check-in: used to lock an AXMEDIS component, and copy it to a user exclusive access area, ready for download:
- Invoke the AXMEDIS Object Manager for locking the Object/component
- Copy the Object component in a Server area of exclusive access to the user, so that the user can download it to his PC disk
- Invoke the AXMEDIS Object Manager for updating tracking information

V) Check-out: Applicable only to previously checked-in Objects. After having uploaded the modified AXMEDIS component, re-loads it onto the AXMEDIS Object Manager and updates tracking data
- Copy to the AXMEDIS Object Manager the selected Object component from the Server area of exclusive access of the user concerned
- Invoke the AXMEDIS Object Manager for unlocking the Object/component
- Invoke the AXMEDIS Object Manager for updating tracking information

6.1.1 Workflow Editor Business Logic

It will be possible for the AXMEDIS workflow management system to support inter-factory workflow. An example of this would be collaborating content producers who work jointly on common objects. Content Factory A would create an object, then Factory B would perform some activities to add value to the object, then returning it to Factory A for completion. Conceptually, this process is identical to the normal, intra-factory scenario where activities are carried out in one content factory. In the inter-factory scenario, the collaborating factories will need to establish an agreed workflow in order to manage their division of work productively and efficiently. This workflow agreement can be modelled in the same manner as a conventional intra-factory workflow. We are not proposing centralised single server architecture; rather each partner will have their workflow running with their part of the project workflow definition. The transitions resulting into change of partner will be defined in the workflow to reflect the collaborative workflow logic as agreed between the collaborators. The waiting period for the factories can be defined as “Idle/wait” activities within the workflow which are completed upon receiving the workitem from the external factory. For example when the workitem is handed to Factory B from Factory A, as defined in the workflow, Factory A will then start an “Idle/wait” activity which will end upon receiving the workitem back from Factory B.

It is important that collaborating factories therefore share common WFMS tools in order to manage and track the progress of an NPD across their combined activities. This enables dynamic planning and scheduling of resources across the factories, much in the way that automotive companies operating just-in-time policies use integrated logistics systems to track components through their value chain.

This dynamic visibility would not be possible if separate WFMS tools were employed in each factory and the only communication available were some embedded historic metadata within objects passed between factories.

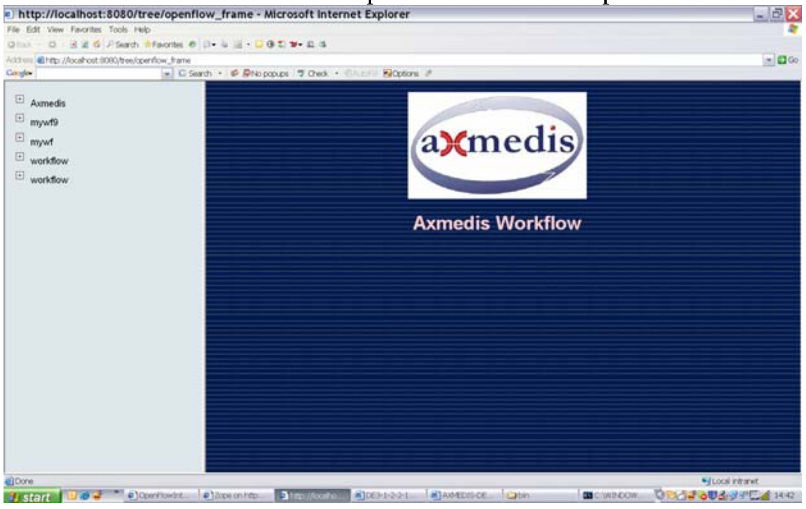
For this reason, a common web-based editor will be used for the AXMEDIS Workflow user interface, which will be capable of being accessed from multiple collaborating content producers, integrators and distributors sharing a common inter-factory workflow.

6.2 Module Design in terms of Classes

NONE

6.3 User interface description

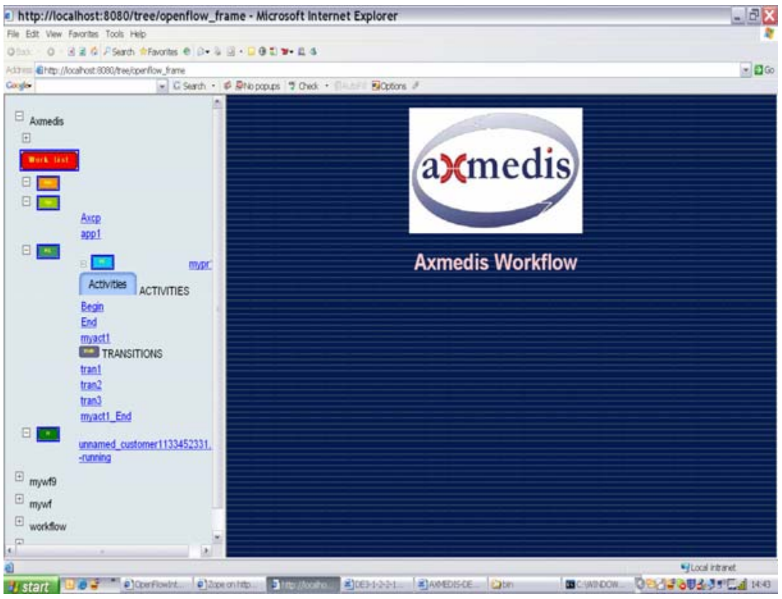
The User Interface that was implemented for the Openflow Workflow is as shown below.


http://localhost:8080/tree/openflow_frame - Microsoft Internet Explorer
Axmedis
mywf9
mywf
workflow
workflow
axmedis
Axmedis Workflow

The Interface contains two main areas.

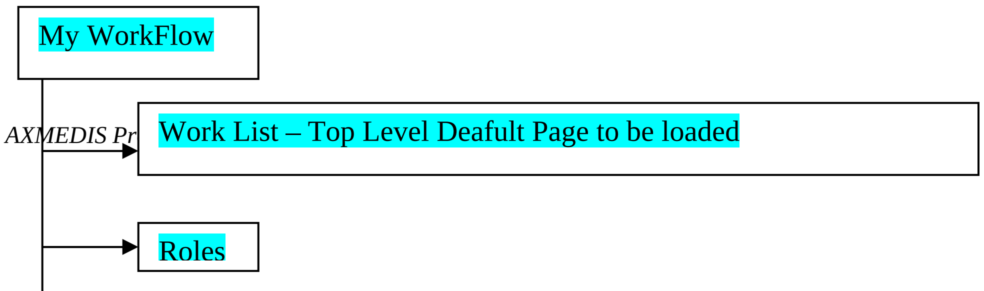
- On the left is a tree based navigation menu, used to navigated through various workflow specific details.
- On the right is the main visualization area, where the information specific to the currently selected node is displayed.

The following screen shows all the node in their expanded state.



The Schematics for the above UI is as shown below

Node Name	Description
My Workflow/Worklist	Is the instance of the Openflow Products installed in the Zope environment. For each instances there will be one distinct node. On selecting this node, the user's personal worklist for that instance will be displayed in the right frame.
Roles	The roles of all the Zope users.
App	This is the application node, displaying all the applications that are initialized in the corresponding openflow instance. Each application in turns displays the url for activation.
Process Definitions	This displays the list of all the processes defined in the corresponding openflow instance. Each processes can then be selected by clicking on their names
Process	This displays the list of all the activities and transitions for the current process
Act	This displays the details for the activities
Tans	This displays the details for the transition
Process Instance	This displays the list of all the process instances in the current instance of the workflow. Each process instance can be clicked to know the details of it.



6.4 Technical and Installation information

- Installation capability, it has to be installable in a very easy manner
- Manual support for technical and user point of views

In Order to install the User Interface correctly, Please follow these steps

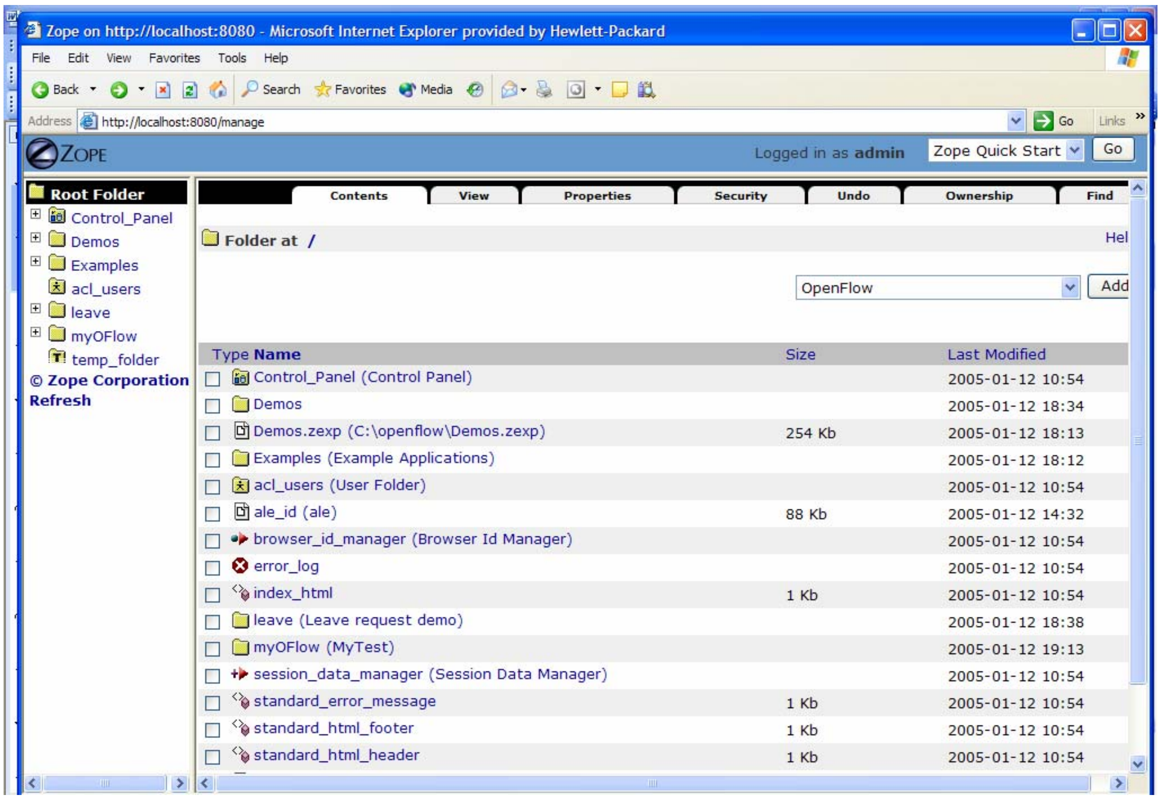
1) Install Zope from www.zope.org
2) Install Openflow from www.openflow.it
3) Extract the zip folder in some known location.
4) Open Zope management interface.
5) Import the three ZEXP files into the zope installation. These files are returnOpen.zexp, returnOpenOnly.zexp and tree.zexp
6) Open tree folder in the Zope interface.
7) Open openflow_frame file in the browser.
8) It will show you a list of all the openflow installations on your machine. You can directly browse it using this tree.

References to other major components needed	Zope Server, Openflow workflow
Problems not solved	•
Configuration and execution context	

6.5 Draft User Manual

Openflow runs on the Zope platform which is managed through the “Zope Management Interface” using industry standard browsers, typically by logging on as the administrator (admin) at URL http://localhost:8080/manage. The screen shot below shows an example of this management interface.

Creating a new process in openflow is a multi-step process which begins with adding an OpenFlow container using the Zope management interface as shown below (delineated by a an ellipse in red).

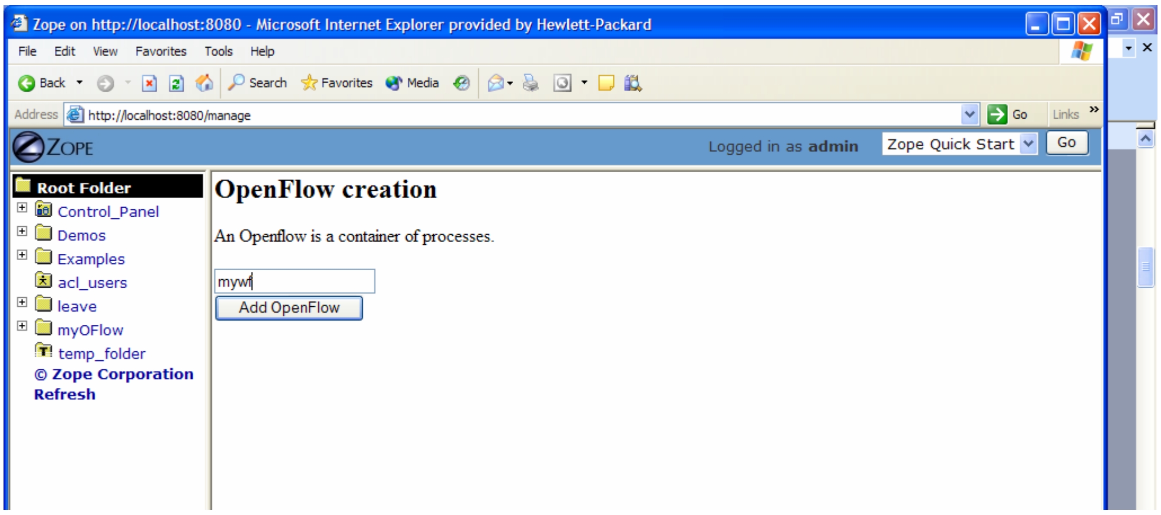

Zope on http://localhost:8080 - Microsoft Internet Explorer provided by Hewlett-Packard
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8080/manage
Go Links
ZOPE
Logged in as admin
Zope Quick Start
Go
Root Folder
Control_Panel
Demos
Examples
acl_users
leave
myOFlow
temp_folder
© Zope Corporation
Refresh
Contents View Properties Security Undo Ownership Find
Folder at /
Hel
OpenFlow
Add

Type	Name	Size	Last Modified
☐	Control_Panel (Control Panel)		2005-01-12 10:54
☐	Demos		2005-01-12 18:34
☐	Demos.zexp (C:\openflow\Demos.zexp)	254 Kb	2005-01-12 18:13
☐	Examples (Example Applications)		2005-01-12 18:12
☐	acl_users (User Folder)		2005-01-12 10:54
☐	ale_id (ale)	88 Kb	2005-01-12 14:32
☐	browser_id_manager (Browser Id Manager)		2005-01-12 10:54
☐	error_log		2005-01-12 10:54
☐	index_html	1 Kb	2005-01-12 10:54
☐	leave (Leave request demo)		2005-01-12 18:38
☐	myOFlow (MyTest)		2005-01-12 19:13
☐	session_data_manager (Session Data Manager)		2005-01-12 10:54
☐	standard_error_message	1 Kb	2005-01-12 10:54
☐	standard_html_footer	1 Kb	2005-01-12 10:54
☐	standard_html_header	1 Kb	2005-01-12 10:54

Figure 1: Adding an OpenFlow container through the Zope Management Interface

During the creation of the OpenFlow container, the name of the container must be specified as shown in the next screen-shot.

.


Zope on http://localhost:8080 - Microsoft Internet Explorer provided by Hewlett-Packard
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8080/manage
Go Links
ZOPE
Logged in as admin
Zope Quick Start
Go
Root Folder
Control_Panel
Demos
Examples
acl_users
leave
myOFlow
temp_folder
© Zope Corporation
Refresh
Add OpenFlow

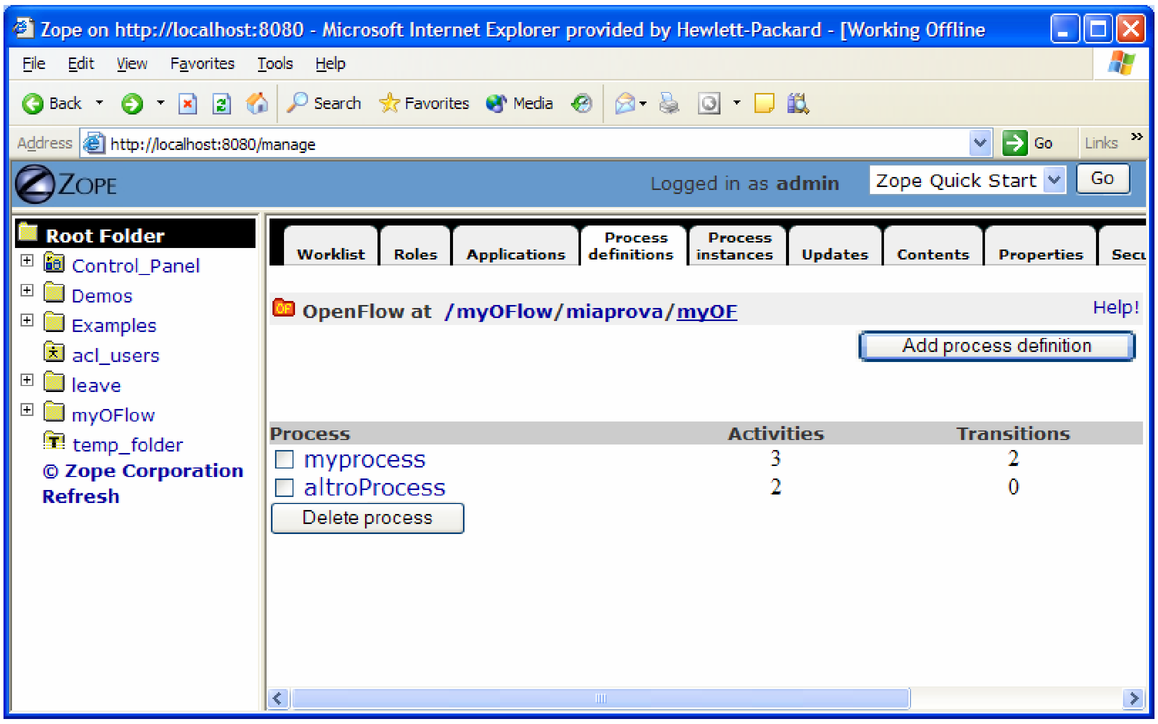
OpenFlow creation

An Openflow is a container of processes.

mywf

Figure 2: Creating the OpenFlow container

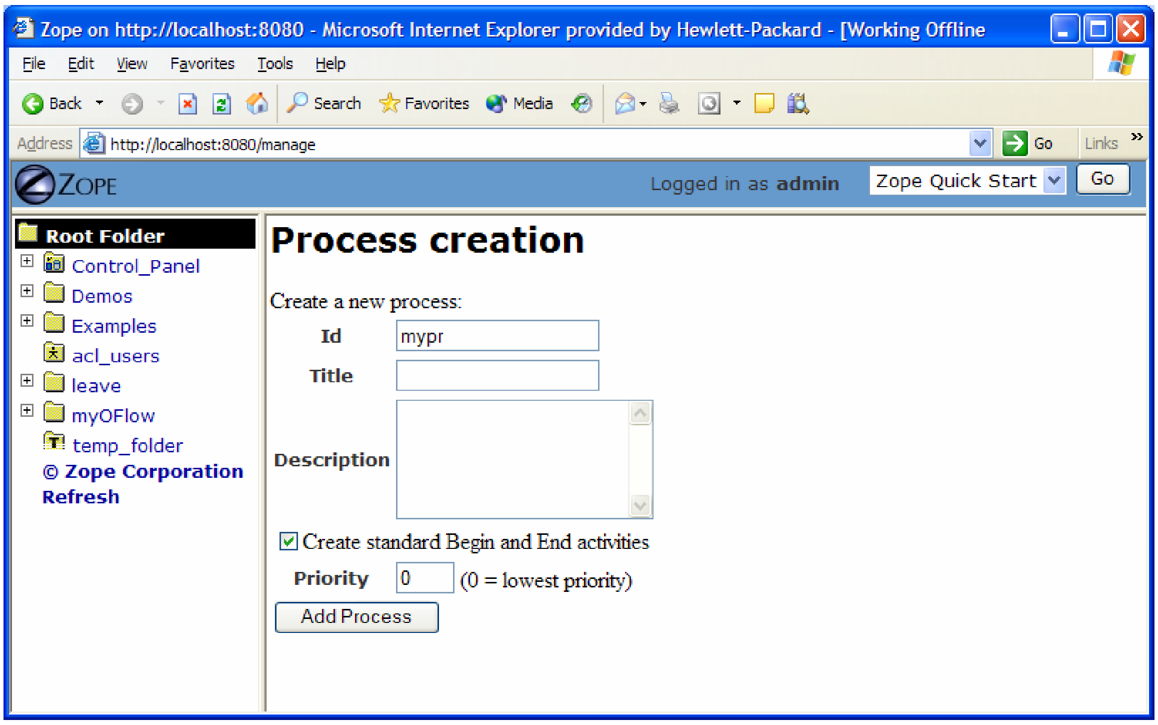
Next it is necessary to define the process and the activities pertaining to the process, together with their transitions (From Activity and To Activity). These operations are performed by accessing the tabs in the Openflow container as shown in the following screen-shots:



OpenFlow at /myOFlow/miaprova/myOF

Process	Activities	Transitions
☐ myprocess	3	2
☐ altroProcess	2	0

Figure 3: The process definition tab


Zope on http://localhost:8080 - Microsoft Internet Explorer provided by Hewlett-Packard - [Working Offline
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8080/manage
Go Links
Logged in as admin
Zope Quick Start
Root Folder
Control_Panel
Demos
Examples
acl_users
leave
myOFlow
temp_folder
© Zope Corporation
Refresh
Add Process

Process creation

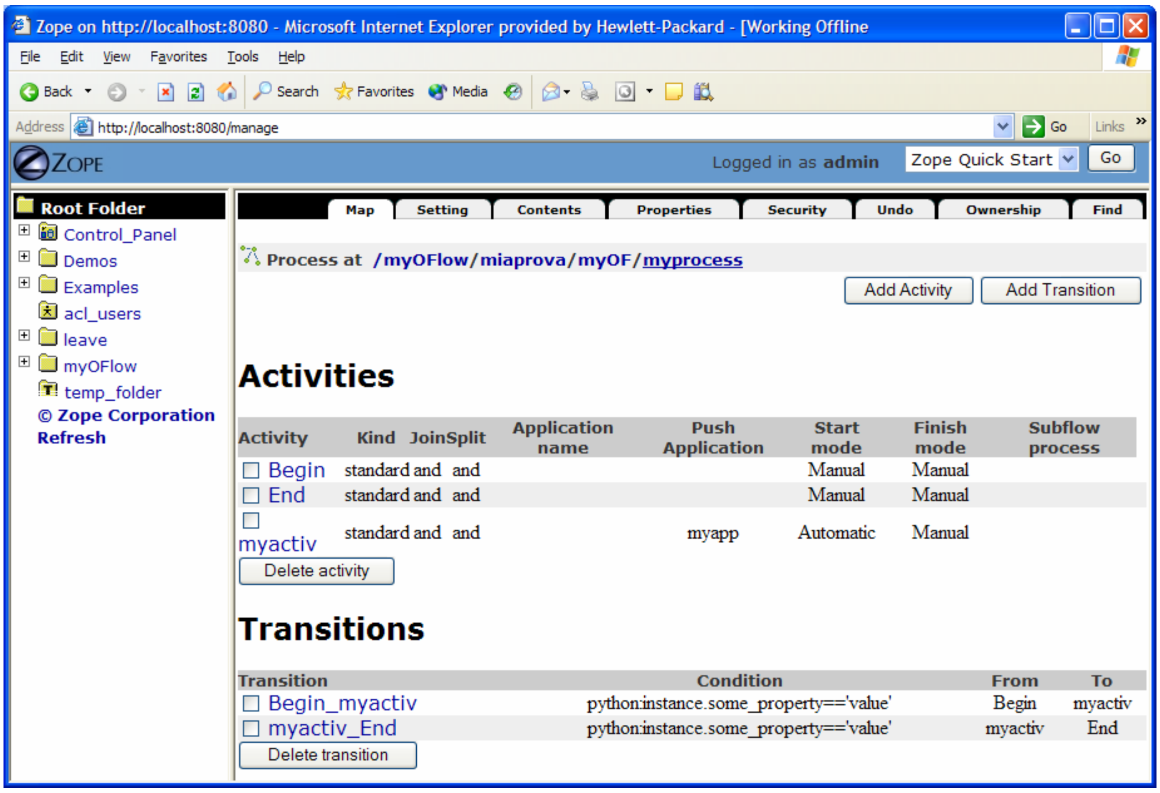
Create a new process:

Id	mypr
Title	
Description	

☑ Create standard Begin and End activities

Priority 0 (0 = lowest priority)

Figure 4: Creating a new Process definition



Process at /myOFlow/miaprova/myOF/myprocess

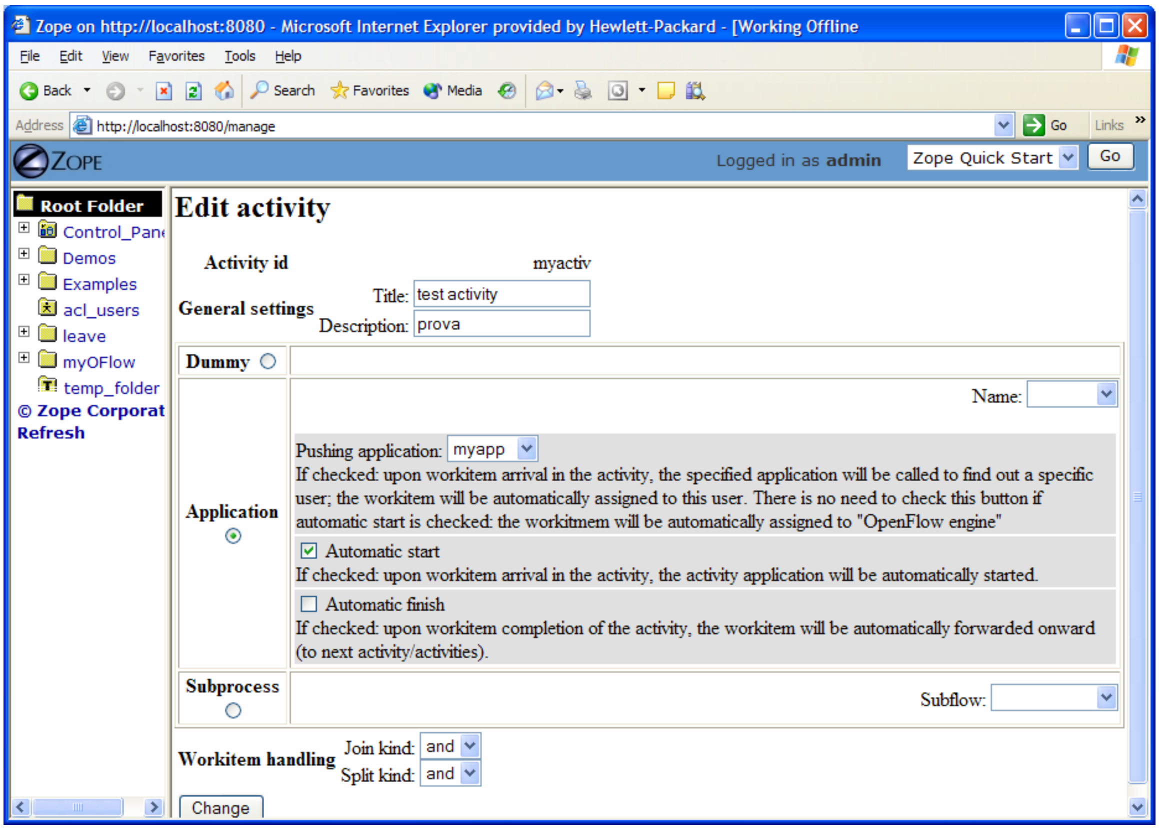
Activities

Activity	Kind	JoinSplit	Application name	Push Application	Start mode	Finish mode	Subflow process
☐ Begin	standard	and and			Manual	Manual	
☐ End	standard	and and			Manual	Manual	
☐ myactiv	standard	and and		myapp	Automatic	Manual	

Transitions

Transition	Condition	From	To
☐ Begin_myactiv	python:instance.some_property=='value'	Begin	myactiv
☐ myactiv_End	python:instance.some_property=='value'	myactiv	End

Figure 5: Management of activity and transitions of a process


Zope on http://localhost:8080 - Microsoft Internet Explorer provided by Hewlett-Packard - [Working Offline
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8080/manage
Logged in as admin
Zope Quick Start
Go
Root Folder
Control_Pane
Demos
Examples
acl_users
leave
myOFlow
temp_folder
© Zope Corporat
Refresh
Change

Edit activity

Activity id: myactiv

General settings — Title: test activity; Description: prova

Dummy ○

Application ● — Name:

Pushing application: myapp

If checked: upon workitem arrival in the activity, the specified application will be called to find out a specific user; the workitem will be automatically assigned to this user. There is no need to check this button if automatic start is checked: the workitmem will be automatically assigned to "OpenFlow engine"

☑ Automatic start

If checked: upon workitem arrival in the activity, the activity application will be automatically started.

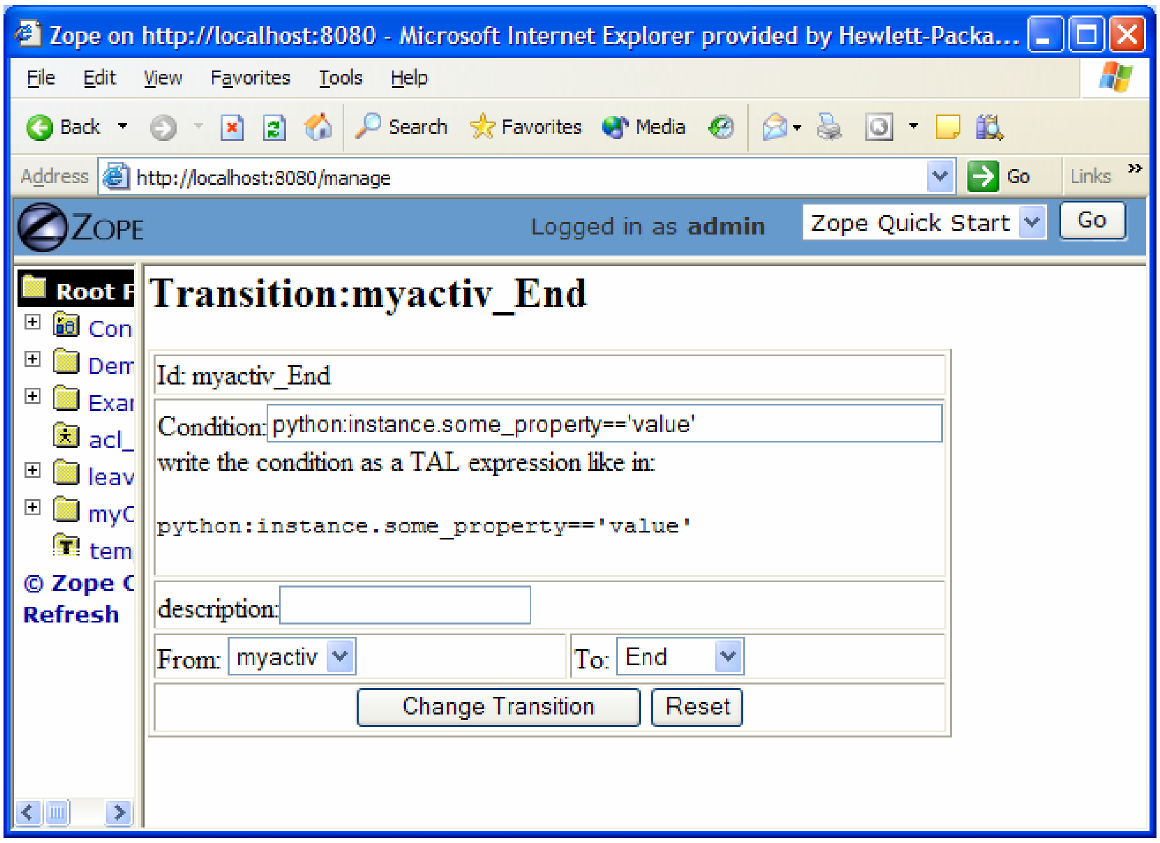
☐ Automatic finish

If checked: upon workitem completion of the activity, the workitem will be automatically forwarded onward (to next activity/activities).

Subprocess ○ — Subflow:

Workitem handling — Join kind: and; Split kind: and

Figure 6: Editing a process activity


Zope on http://localhost:8080 - Microsoft Internet Explorer provided by Hewlett-Packa...
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8080/manage
Go Links
Logged in as admin
Zope Quick Start
Go
Root F
Con
Dem
Exar
acl_
leav
myC
tem
© Zope C
Refresh

Transition:myactiv_End

Id: myactiv_End

Condition: python:instance.some_property=='value'

write the condition as a TAL expression like in:

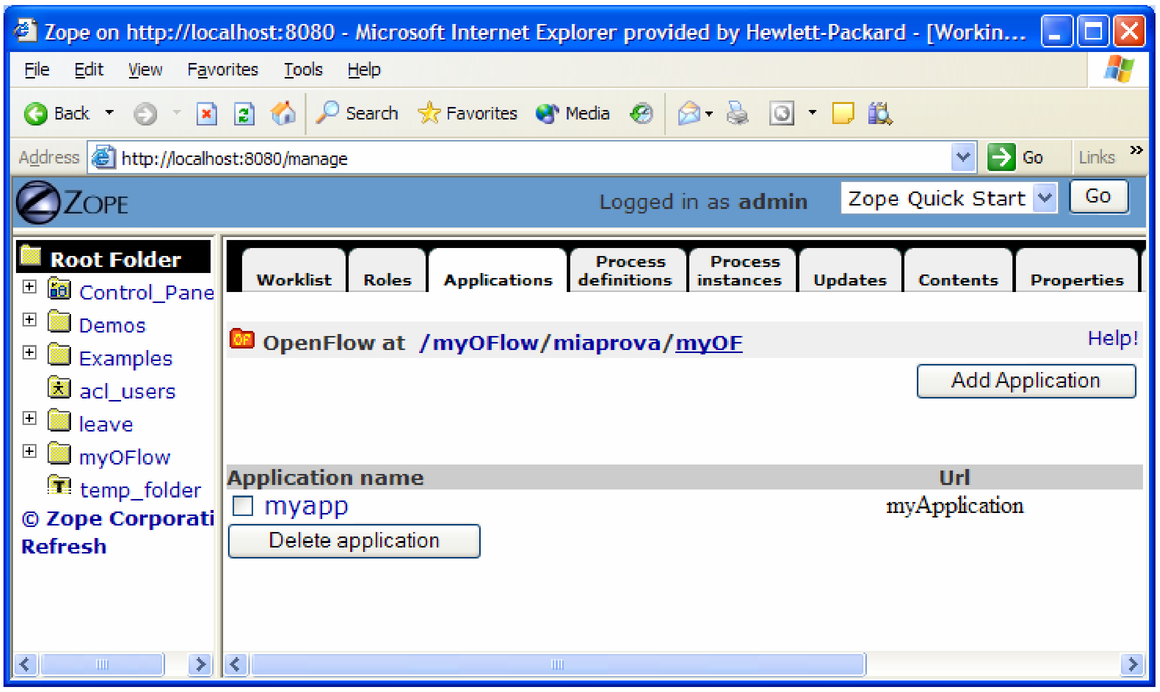
```
python:instance.some_property=='value'
```

description:

From: myactiv To: End

Figure 7: Defining process transition and related conditions

Applications associated to the activities are then specified selecting the Applications Tab.

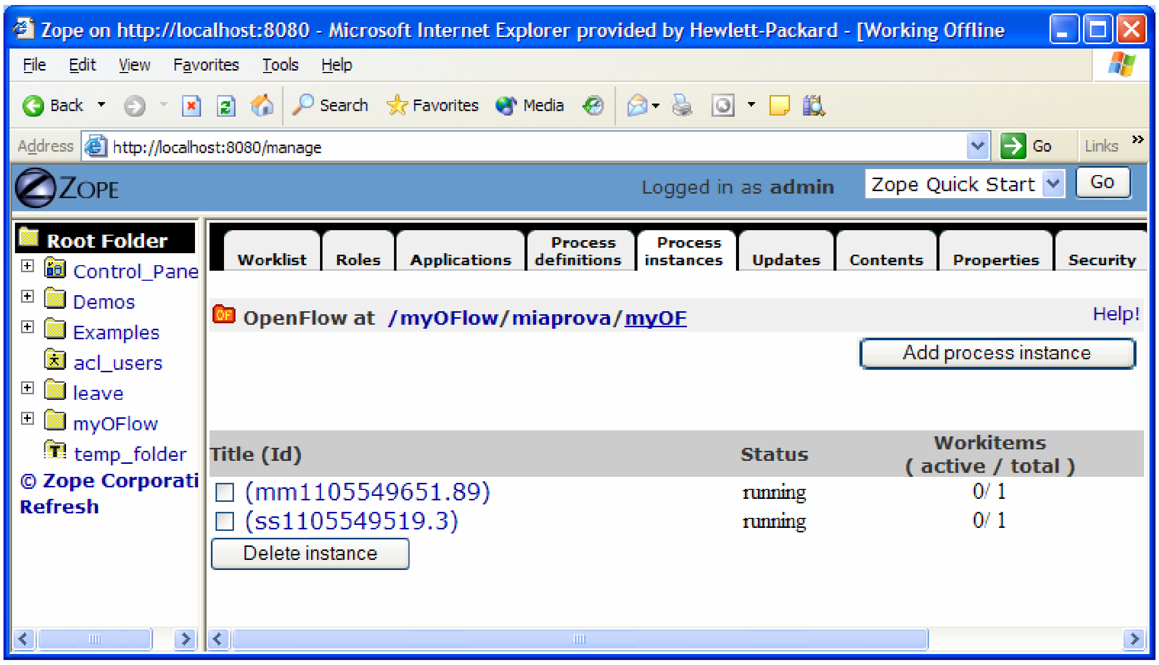


Application name	Url
myapp	myApplication

Figure 8: Defining process applications

The users and roles are configured as Zope users and roles as access control list (acl_users).

Once a process has been defined it can be tested. An instance of the process can be created and executed directly in the processflow-instance management tab shown below.

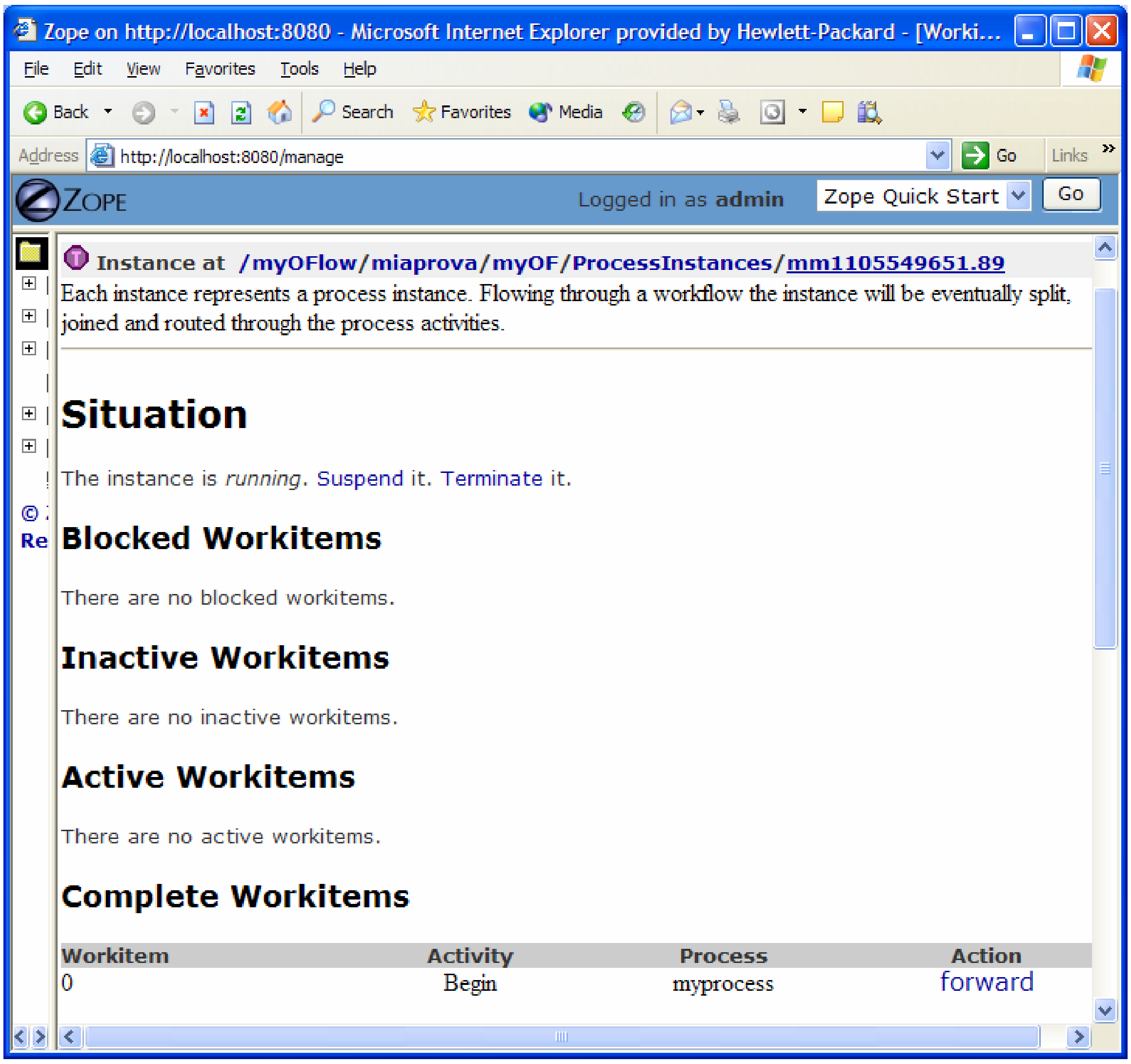

Zope on http://localhost:8080 - Microsoft Internet Explorer provided by Hewlett-Packard - [Working Offline
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8080/manage Go Links
ZOPE Logged in as admin Zope Quick Start Go
Root Folder
Control_Pane
Demos
Examples
acl_users
leave
myOFlow
temp_folder
© Zope Corporati
Refresh
Worklist Roles Applications Process definitions Process instances Updates Contents Properties Security
Help!
Add process instance
Delete instance

OpenFlow at /myOFlow/miaprova/myOF

Title (Id)	Status	Workitems (active / total)
☐ (mm1105549651.89)	running	0/ 1
☐ (ss1105549519.3)	running	0/ 1

Figure 9: Process instance management tab

The following Figure shows the of the workitems involved in the process instance that has been created.


Zope on http://localhost:8080 - Microsoft Internet Explorer provided by Hewlett-Packard - [Worki...
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8080/manage
Go Links
ZOPE
Logged in as admin
Zope Quick Start
Go

Instance at /myOFlow/miaprova/myOF/ProcessInstances/mm1105549651.89

Each instance represents a process instance. Flowing through a workflow the instance will be eventually split, joined and routed through the process activities.

Situation

The instance is *running*. Suspend it. Terminate it.

Blocked Workitems

There are no blocked workitems.

Inactive Workitems

There are no inactive workitems.

Active Workitems

There are no active workitems.

Complete Workitems

Workitem	Activity	Process	Action
0	Begin	myprocess	forward

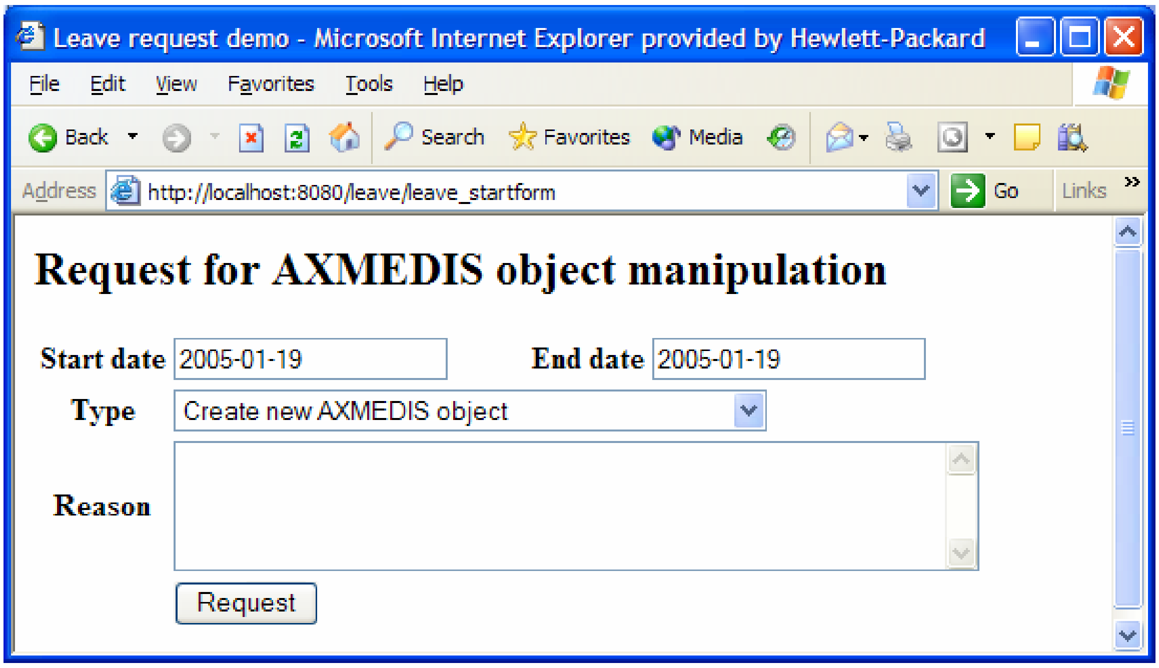
Figure 10: Monitoring and management of a specific process instance

Process Example:

The following simple example illustrates a process to request a AXMEDIS object manipulation (*a mock-up process*). This is an example of explicit forwarding to different actors having different roles. The first actor requests the creation of a new AXMEDIS object by filling out a form. The request goes to the second actor (called Socius) who checks that the request is acceptable. The request is then forwarded to the third actor (called Prefectus) for approval.

The following steps are necessary for the above example process to be enacted:

The first actor (called Tertius) enters an AXMEDIS object manipulation request by filling out the following form as shown in the screen-shot below:


Leave request demo - Microsoft Internet Explorer provided by Hewlett-Packard
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8080/leave/leave_startform Go Links

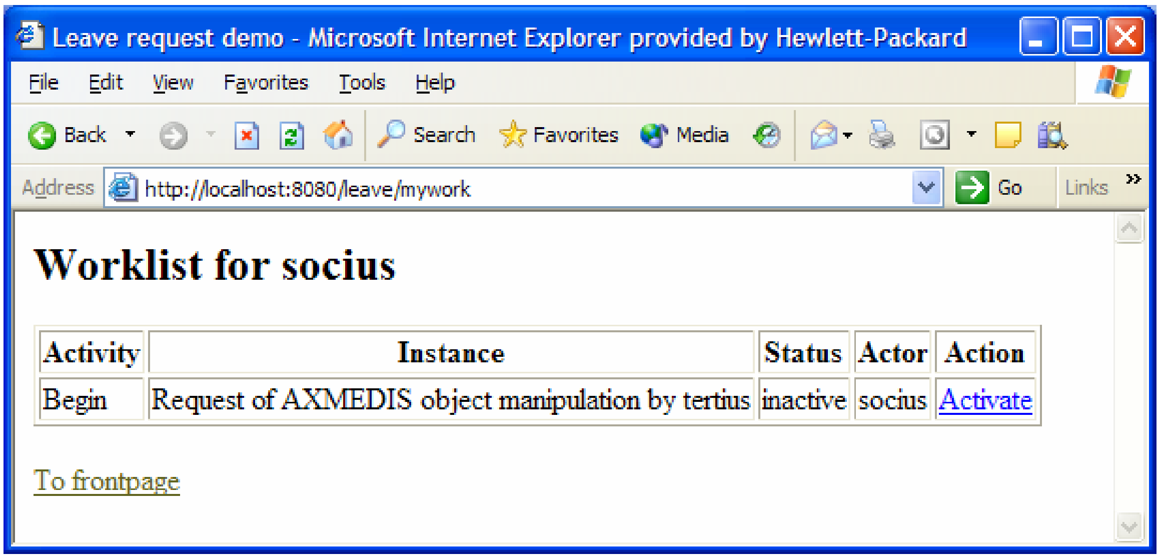
Request for AXMEDIS object manipulation

Start date	2005-01-19	End date	2005-01-19
Type	Create new AXMEDIS object		
Reason			

Request

Figure 11: Tertius' AXMEDIS object manipulation form

According to the processflow, the request goes to the next actor (called Socius). When Socius logs in, his work list shows that there is a workitem in his worklist as shown in the screen-shot below:


Leave request demo - Microsoft Internet Explorer provided by Hewlett-Packard
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8080/leave/mywork Go Links

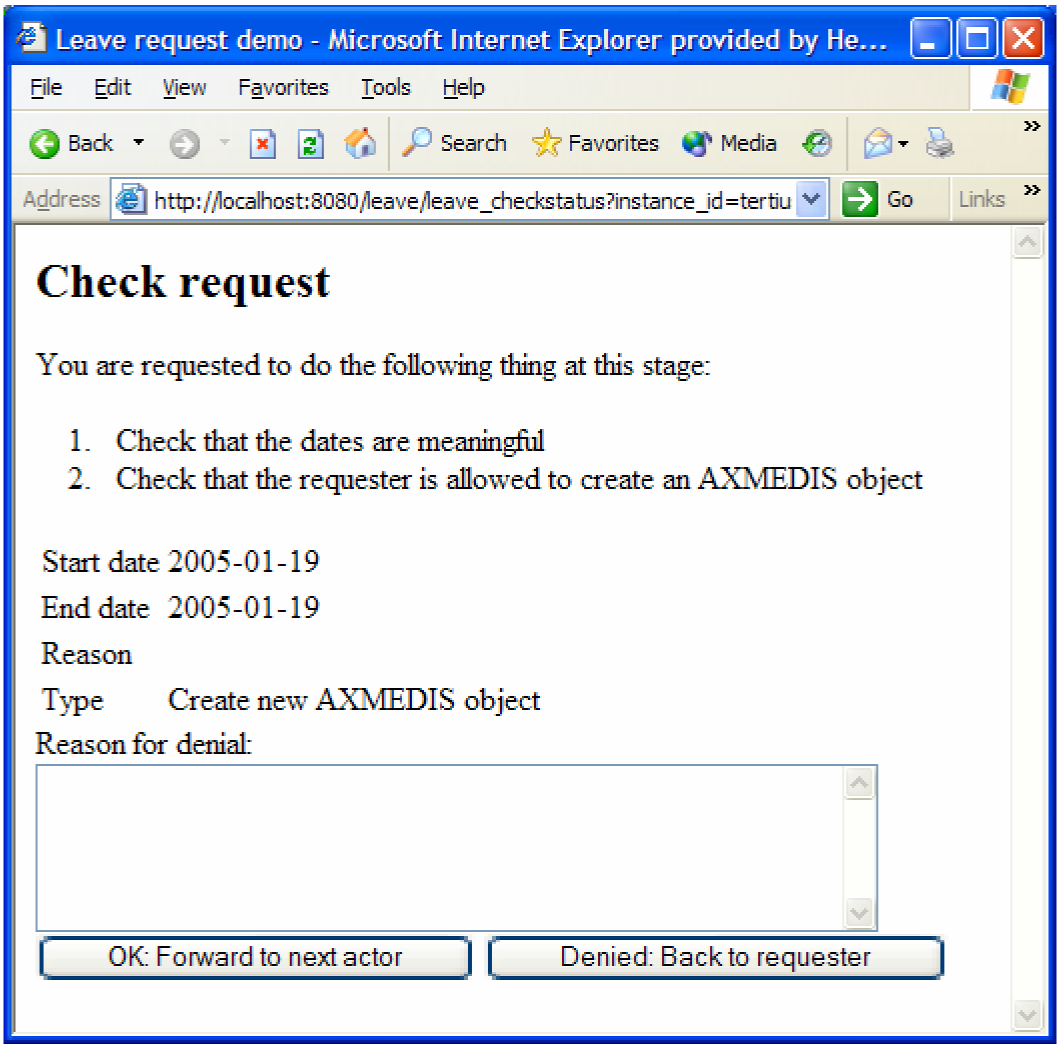
Worklist for socius

Activity	Instance	Status	Actor	Action
Begin	Request of AXMEDIS object manipulation by tertius	inactive	socius	Activate

To frontpage

Figure 12: Socius' worklist and workitem activation

To execute the workitem, the actor (Socius) has to activate the workitem (Begin) and perform the related activities. Next this actor either forwards the workitem to the next actor, which in this case is the supervisor (called Prefectus), or rejects the request; as illustrated by the screen-shot below:


Leave request demo - Microsoft Internet Explorer provided by He...
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8080/leave/leave_checkstatus?instance_id=tertiu
Go Links

Check request

You are requested to do the following thing at this stage:

1. Check that the dates are meaningful
2. Check that the requester is allowed to create an AXMEDIS object

Start date	2005-01-19
End date	2005-01-19
Reason	
Type	Create new AXMEDIS object

Reason for denial:

OK: Forward to next actor
Denied: Back to requester

Figure 13: Socius' workitem execution and forwarding

Then the activity is forwarded to the last actor and the process ends.

6.6 Examples of usage

As Above

6.7 Integration and compilation issues

None

6.8 Configuration Parameters

Config parameter	Possible values

6.9 Errors reported and that may occur

Error code	Description and rationales

6.10 Formal description of algorithm <……………>

NONE

name	
Method	
Description	
Input parameters	
Output parameters	

name	
Method	
Description	
Input parameters	
Output parameters	

7 Module or Executable Tool Workflow Request Adaptor - Openflow

Module/Tool Profile	
Workflow Request Adaptor - Openflow	
Responsible Name	
Responsible Partner	
Status (proposed/approved)	Approved
Implemented/not implemented	Implemented
Status of the implementation	First Prototype Completed
Executable or Library/module (Support)	library
Single Thread or Multithread	Multithreaded
Language of Development	Python and DTML
Platforms supported	Windows, Linux
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedig.org/repos/Framework/source/workflow_editor_channel/ openflow/input_queue_adapter/Prove_WF.zexp contains: prototype process definitions prototype dtml scripts for User Interface and process logic prototype python scripts for process logic https://cvs.axmedig.org/repos/Framework/source/workflow_editor_channel/ openflow/input_queue_adapter/read_config_file.py contains configuration file read method used by all invocations https://cvs.axmedig.org/repos/Framework/source/workflow_editor_channel/ openflow/input_queue_adapter/parse_editor_response.py contains xml parser for the Editor channel https://cvs.axmedig.org/repos/Framework/source/workflow_rule_editor_channel/ openflow/input_queue_adapter/parse_rule_editor_response.py contains xml parser for the Rule Editor channel https://cvs.axmedig.org/repos/Framework/source/workflow_database_channel/ openflow/input_queue_adapter/parse_query_response.py contains xml parser for the Database channel https://cvs.axmedig.org/repos/Framework/source/workflow_engine_channel/ openflow/input_queue_adapter/parse_rule_editor_response.py contains xml parser for the Engine channel https://cvs.axmedig.org/repos/Framework/source/workflow_editor_channel/ openflow/input_queue_adapter/read_config_file.py contains the Adapter read configuration file method https://cvs.axmedig.org/repos/Framework/source/workflow_editor_channel/ openflow/request_adapter contains all request adapter methods (python scripts) for the editor channel:

	edit_object.py view_object_attribute.py add_history_info.py https://cvs.axmedig.org/repos/Framework/source/workflow_engine_channel/ openflow/request_adapter contains all request adapter methods (python scripts) for the rule editor channel: edit_comp_form_rule_post.py pandp_user_interface.py activate_pnp.py list_of_axprograms.py AXEPTool_Rule_editor.py Protection_Rule_editor.py https://cvs.axmedig.org/repos/Framework/source/workflow_rule_editor_channel/ openflow/request_adapter contains all request adapter methods (python scripts) for the engine channel: axinstall_and_activate_post.py axrun_rule.py axdeactivate_rule.py axsuspend_rule.py axpause_rule.py axkill_rule.py axremove _rule.py axresume_rule.py axget_rule_status.py axget_rule_logs.py axget_list_of_rules.py axget_rule_schema.py axpnp_status_request.py axsuspend_pnp_program.py axabort_pnp_program.py axresume_pnp_program.py axactivate_pnp_program.py axworkflow_notification.py https://cvs.axmedig.org/repos/Framework/source/workflow_database_channel/ openflow/request_adapter contains all request adapter methods (python scripts) for the database channel: edit_query.py axactivate_selection_sync.py axactivate_selection_async.py axdelete_selection.py axload_selection.py axsave_selection_post.py axlist_user_selection.py axlist_entitled_selection.py axcheck_out_sync.py url_object_download.py axcheck_out_async.py axcommit_sync.py read_file.py strip_file.py

	axcommit_async.py axlock_object.py axunlock_object.py	
Reference to the AXFW location of the demonstrator executable tool for internal download	same as above	
Reference to the AXFW location of the demonstrator executable tool for public download		
Address for accessing to WebServices if any, add accession information (user aNd Passwd) if any	Not applicable	
Test cases (present/absent)	Yes	
Test cases location	Test cases are just the execution of all the simple processes implemented in prototype which tests all methods for invoking AXMEDIS.	
Usage of the AXMEDIS configuration manager (yes/no)	No	
Usage of the AXMEDIS Error Manager (yes/no)	No	
Major Problems not solved	-- --	
Major pending requirements	-- --	
Interfaces API with other tools, named as	Name of the communicating tools References to other major components needed	Communication model and format (protected or not, etc.)
WorkFlow Engine API		Internal library calls
WorkFlow Request Gateway		GET/POST calls in http
Formats Used	Shared with	format name or reference to a section
Protocol Used	Shared with	Protocol name or reference to a section
axWFReq	WorkFlow Request Gateway	axWFReq
Used Database name		
None		

User Interface	Development model, language, etc.	Library used for the development, platform, etc.
None		
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not
Zope 2.7.3	Zope by Zope Corporation Zope 2.7.3 ZPL 2.1 (Zope Public Licence 2.1), open source, GPL compatible	GPL
Python 2.3	Python 2.3 by Stichting Mathematisch Centrum, Amsterdam, The Netherlands. Python 2.3 Free Open Source	GPL
OpenFlow OpenFlow 1.1	OpenFlow by OpenFlow	GPL
xmlrpclib	Xmlrpclib from Python	GPL

7.1 General Description of the Module

There will be a WorkFlow Request Adapter for each communication channel with the other AXMEDIS tools:

1. WF AXOM Request Adatper for integrating the AXMEDIS Object Manager and all AXOM Editors (WorkFlow Editor Channel)
2. WF Rule Editor Request Adapter for integrating all AXMEDIS Rule Editors (WorkFlow Rule Editor Channel)
3. WF Engine Request Adapter for integrating all AXMEDIS Engines (WorkFlow Engine Channel)
4. WF DB Request Adapter for integrating the AXMEDIS DataBase and the Query Support functions (WorkFlow Query and DataBase Channel)

The Request Adapters are the communication channels used by the WorkFlow Engine for issuing the request to the above mentioned AXMEDIS components. This will ensure that when a task inside an activity has to execute some AXMEDIS component, the proper Adapter will be called for invoking the correct method through the Plug-ins.

7.2 Module Design in terms of Classes

For **each channel** (*Editor, Rule Editor and Engine*) a Python External module with different methods was developed for **parsing** the xml Response coming from the Request Gateway:

- **parse_editor_response.py** contains xml parsing for Editor channel Responses
- **parse_rule_editor_response.py** contains xml parsing for Rule Editor channel Responses

- **parse_engine_response.py** contains xml parsing for Engine channel Responses
- **parse_query_response.py** contains xml parsing for database channel Responses

One Python script (**read_config_file.py**) was developed to let invocation methods to read the configuration file.

One Python script was developed for **creating a new OpenFlow process instance**, with the needed properties: **create_axwf_instance**. The create_axwf_instance creates a process instance with a list of properties later used by the workflow AXMEDIS methods:

- *title*: the title of the process instances, free string
- *AXOID*: the AXMEDIS Object ID being dealt by the process instance
- *Credentials*: the credential string to be sent in the invocation towards AXMEDIS tools, for user indentification and authentication
- *attribute_sequence*: used for storing parameters of the following invocation
- *arguments*: used for storing additional parameters
- *result*: used for storing the result of the received response or notification
- *engine_status*: used for storing the status returned by a notification
- *editing_completed*: used for informing the editor processes that editing is completed

Other properties can be added as needed by Openflow processes and scripts.

One DTML script was developed for presenting the pending workitems assigned to an OpenFlow logged user: **mywork**.

One DTML script was developed for assigning a workitem to the user *admin*, used for the tests: **route_to_admin**.

As a general flow, to each method is assigned a test/example process.
In order to start a process instance, the first **input Zope DTML script** has to be executed. This scrip, in turn, executes the corresponding **Zope DTML logic script** which creates (using the Python script **create_axwf_instance**) a new process instance, adds the needed properties values and starts it.
If the process logic is simpler, a **launching script** is automatically started in order to execute the **External Python Script** which invokes the proper method.
If the process logic is more complicated, the next workitem is assigned to *admin* (with **route_to_admin** script). The logged *admin* user gets the list of his assigned workitems through **mywork** script and executes the corresponding **input Zope DTML** and **Zope DTML logic scripts**. These scripts, in turn, execute some **launching scripts** for invoking the proper methods via the **External Python Scripts**.
The method invocation is performed towards some test xml simulators in **Editor_simulation** folder. The response xml is returned by the **External Python Scripts** to the **launching scripts** which parse the reponses calling the **Python External parsing modules**.
The result extracted from the response is stored in proper variables in the process instance properties and used by the OpenFlow process to take decision about the next workitem to execute.

The following tables summarize, for each method, the process developed in the prototype as well as the scripts used:

Editor Channel Methods:

AXMEDIS invoked Method	**OpenFlow Process(es)**	**Input scipt(s)**	**Logic script(s)**	**Launching script**	**External me**
edit_axobject	edit_object edit_new_object	Edit_object_start form Edit_new_object	Edit_object_start Edit_new_object_ start	Launch_edit_object	edit_object.p

AXMEDIS invoked Method	OpenFlow Process(es)	Input scipt(s)	Logic script(s)	Launching script	External me
		_startform Perform_edit Perform_new_edit Continue_perform_edit			
View_object_attribute	View_object	View_object_startform	View_object_start	Launch_view_object_p	view_object_
Add_history_info	Add_history	Add_history_startform	Add_history_start	Launch_add_history_p	add_history_

Engine Channel Methods

AXMEDIS invoked Method	OpenFlow Process(es)	Input scipt(s)	Logic script(s)	Launching script	External me
Axinstall_and_activate_post	Install_and_activate	Install_and_activate_startform	Install_and_activate_start	Launch_install_and_activate_p	axinstall_and e_post.py
Axrun_rule	Run_rule	Run_rule_startform	Run_rul_start	Launch_run_rule_p	axrun_rule.p
Axdeactivate_rule	Deactivate_rule	Deactivate_rule_startform	Deactivate_rule_start	Launch_deactivate_rule_p	axdeactivate_
Axsuspend_rule	Suspend_rule	Suspend_rule_startform	Suspend_rule_start	Launch_suspend_rule_p	axsuspend_ru
Axpause_rule	Pause_rule	Pause_rule_startform	Pause_rule_start	Launch_pause_rule_p	axpause_rule
Axkill_rule	Kill_rule	Kill_rule_startform	Kill_rule_start	Launch_kill_rule_p	axkill_rule.p
Axremove_rule	Remove_rule	Remove_rule_startform	Remove_rule_start	Launch_remove_rule_p	axremove _ru
Axresume_rule	Resume_rule	Resume_rule_startform	Resume_rule_start	Launch_resume_rule_p	axresume_rul
Axget_rule_status	Get_rule_status	Get_rule_status_startform	Get_rule_status_start	Launch_get_rule_status_p	axget_rule_st
Axget_rule_logs	Get_rule_logs	Get_rule_logs_startform	Get_rule_logs_start	Launch_get_rule_logs_p	axget_rule_lo
Axget_list_of_rules	Get_list_of_rules	Get_list_of_rules_startform	Get_list_of_rules_start	Launch_get_list_of_rules_p	axget_list_of_ y
Axget_rule_schema	Get_rule_schema	Get_rule_schema_startform	Get_rule_schema_start	Launch_get_rule_schema_p	axget_rule_sc y
Axpnp_status_request	Pnp_status_request	Status_request_pnp_startform	Status_request_pnp_start	Launch_pnp_status_request_p	axpnp_status_ .py
Axsuspend_pnp_program	Suspend_pnp_program	Suspend_pnp_program_startform	Suspend_pnp_program_start	Launch_suspend_pnp_program_p	axsuspend_pr ram.py
Axabort_pnp_program	Abort_pnp_program	Abort_pnp_program_startform	Abort_pnp_program_start	Launch_abort_pnp_program_p	axabort_pnp_ m.py
Axresume_pnp_program	Resume_pnp_program	Resume_pnp_program_startform	Resume_pnp_program_start	Launch_resume_pnp_program_p	axresume_pn am.py
Axactivate_pnp_program	Activate_pnp_program	Activate_pnp_program_startform	Activate_pnp_program_start	Launch_activate_pnp_program_p	axactivate_pr ram.py
Axworkflow_notification	TestProcessRequest	N/A: the process is started by	N/A: the process is started by	Workflow_notification_p	axworkflow_ ion.py

AXMEDIS invoked Method	OpenFlow Process(es)	Input scipt(s)	Logic script(s)	Launching script	External me
		workflow_process_request from Response Adapter	workflow_process_request from Response Adapter		

Rule Editor Channel:

AXMEDIS invoked Method	OpenFlow Process(es)	Input scipt(s)	Logic script(s)	Launching script	External script
Edit_ax_comp_form_rule_post	edit_comp_form_rule	Edit_comp_form_rule_startform Perform_edit_comp_form_rule Continue_perform_rule_edit	Edit_comp_form_rule_start	Launch_edit_comp_form_rule	edit_comp_fo post.py
Pandp_ax_user_interface	Pandp_user_interface	Pandp_user_interface_startform Perform_pandp_user_interface Continue_pandp_user_interface	Pandp_user_interface_start	Launch_pandp_user_interface	pandp_user_i py
Activate_axpnp	Activate_pnp	Pandp_activate_program_startform	Pandp_activate_program_start	Launch_activate_program_p	activate_pnp.
List_of_axprograms	List_of_programs	List_of_programs_startform	List_of_programs_start	Launch_list_of_programs_p	list_of_axpro
AXEPTool_Rule_editor	AXEPTool_editor	AXEPTool_Rule_editor_startform Perform_AXEPTool_Rule_editor continue_AXEPTool_Rule_editor	AXEPTool_Rule_editor_start	launch_AXEPTool_Rule_editor	AXEPTool_I or.py
Protection_rule_editor	Protection_editor	Protection_Rule_editor_startform Perform_Protection_Rule_editor continue_Protection_Rule_editor	Protection_Rule_editor_start	launch_Protection_Rule_editor	Protection_R r.py

DataBase Channel:

AXMEDIS invoked Method	OpenFlow Process(es)	Input scipt(s)	Logic script(s)	Launching script	External script
Edit_axquery	Edit_query	Edit_query_startform Perform_edit_query	edit_query_start	Launch_edit_query	edit_query.p

AXMEDIS invoked Method	**OpenFlow Process(es)**	**Input scipt(s)**	**Logic script(s)**	**Launching script**	**External script**
		Continue_perform_query			
Axactivate_selection_sync	activate_selection_sync	Activate_selection_sync_startform	activate_selection_sync_start	launch_activate_selection_sync_p	axactivate_se ync.py
Axactivate_selection_async	activate_selection_async	Activate_selection_async_startform	activate_selection_async_start	launch_activate_selection_async_p	axactivate_se sync.py
Axdelete_selction	delete_selection	delete_selection_startform	Delete_selection_start	Launch_delete_selection_p	axdelete_sele
Axload_selection	Load_selection	load_selection_startform	load_selection_start	launch_load_selection_p	axload_select
Axsave_selection_post	Save_selection	save_selection_startform	save_selection_start	launch_save_selection_p	axsave_selec py
Axlist_user_selection	List_user_selection	List_user_selection_startform	List_user_selection_start	Launch_list_user_selection_p	axlist_user_s y
Axlist_entitled_selection	List_entitled_selection	List_entitled_selection_startform	List_entitled_selection_start	Launch_list_entitled_selection_p	axlist_entitle n.py
Axcheck_out_sync url_object_download	Check_out_sync	Check_out_sync_startform	Check_out_sync_start	Launch_check_out_sync_p Launch__download_sync_p	axcheck_out_ url_object_do p
Axcheck_out_async	Check_out_async	Check_out_async_startform	Check_out_async_start	Launch_check_out_async_p	axcheck_out_
Axcommit_sync Read_file Strip_file	Commit_sync	Commit_sync_startform	Commit_sync_start	Launch_commit_sync_p	axcommit_sy read_file.py strip_file.py
Axcommit_async	Commit_async	Commit_async_startform	Commit_async_start	Launch_commit_async_p	axcommit_as
Axlock_object	Lock_object	Lock_object_startform	Lock_object_start	Launch_lock_object_p	axlock_objec
Axunlock_object	unlock_object	unlock_object_startform	unock_object_start	Launch_unlock_object_p	axunlock_obj

7.3 User interface description

None

7.4 Technical and Installation information

1. Start Zope
2. Put the file Prove_WF.zexp in c:\Zope-Instance\import
3. log in to Zope as admin
4. from root directory of Zope import: Prove_WF.zexp
5. go in the folder Prove_WF/workflow/ProcessInstances and select tag 'Security'
6. check that the Manage Properties checkbox for Anonymous is checked
7. check that, in folder Upload, the delete objects checkbox for Anonymous is checked
8. Copy read_config_file.py in c:/Program Files/Zope-2.7.3-0/lib/python
9. copy .py files in c:\Zope-Instance\Extensions directory
10. Copy and edit with correct parameters the configuration file (see below)

References to other major components needed	WorkFlow Engine
Problems not solved	•
Configuration and execution context	Configuration file is AX_WF_Adapter.cfg in Extensions\ directory of Zope Instance (i.e. c:\Zope-Instance)

7.5 Draft User Manual

None

7.6 Examples of usage

None

7.7 Integration and compilation issues

7.8 Configuration Parameters

Configuration parameters are in AX_WF_Adapter.cfg configuration file:

```
ListenerURI=<URI where the adapter waits Notifications>
activate_axpnp=<URI>
add_history_info=<URI>
axabort_pnp_program=<URI>
axactivate_pnp_program=<URI>
axactivate_rule=<URI>
axdeactivate_rule=<URI>
AXEPTool_rule_editor=<URI>
axget_list_of_rules=<URI>
axget_rule_logs=<URI>
axget_rule_schema=<URI>
axget_rule_status=<URI>
axinstall_and_activate_post=<URI>
axkill_rule=<URI>
axpause_rule=<URI>
axpnp_status_request=<URI>
axremove_rule=<URI>
axresume_pnp_program=<URI>
axresume_rule=<URI>
axrun_rule=<URI>
axsuspend_pnp_program=<URI>
axsuspend_rule=<URI>
axworkflow_notification=<URI>
edit_ax_comp_form_rule_post=<URI>
edit_axobject=<URI>
list_of_axprograms=<URI>
pandp_ax_user_interface=<URI>
Protection_rule_editor=<URI>
```

```
view_object_attribute=<URI>
edit_axquery=<URI>
axactivate_selection_sync=<URI>
axactivate_selection_async=<URI>
axdelete_selection=<URI>
axload_selection=<URI>
axsave_selection_post=<URI>
axlist_user_selection=<URI>
axlist_entitled_selection=<URI>
axcheck_out_sync=<URI>
axcheck_out_async=<URI>
axcommit_sync=<URI>
axcommit_sync=<URI>
axlock_object=<URI>
axunlock_object=<URI>
```

Config parameter	Possible values
ListenerURI	URI where the WF Response Adapter waits for Notifications coming from Response Gateway
activate_axpnp	URI of the Request Gateway where the Request Adapters sends this method
add_history_info	URI where the WF Response Adapter waits for Notifications coming from Response Gateway
axabort_pnp_program	URI of the Request Gateway where the Request Adapters sends this method
axactivate_pnp_program	URI where the WF Response Adapter waits for Notifications coming from Response Gateway
axactivate_rule	URI of the Request Gateway where the Request Adapters sends this method
axdeactivate_rule	URI where the WF Response Adapter waits for Notifications coming from Response Gateway
AXEPTool_rule_editor	URI of the Request Gateway where the Request Adapters sends this method
axget_list_of_rules	URI where the WF Response Adapter waits for Notifications coming from Response Gateway
axget_rule_logs	URI of the Request Gateway where the Request Adapters sends this method
axget_rule_schema	URI where the WF Response Adapter waits for Notifications coming from Response Gateway
axget_rule_status	URI of the Request Gateway where the Request Adapters sends this method
axinstall_and_activate_post	URI where the WF Response Adapter waits for Notifications coming from Response Gateway
axkill_rule	URI of the Request Gateway where the Request Adapters sends this method
axpause_rule	URI where the WF Response Adapter waits for Notifications coming from Response Gateway
axpnp_status_request	URI of the Request Gateway where the Request Adapters sends this method
axremove_rule	URI where the WF Response Adapter waits for Notifications coming from Response Gateway
axresume_pnp_program	URI of the Request Gateway where the Request Adapters sends this method
axresume_rule	URI where the WF Response Adapter waits for Notifications coming from Response Gateway
axrun_rule	URI of the Request Gateway where the Request Adapters sends this method
axsuspend_pnp_program	URI where the WF Response Adapter waits for Notifications coming from Response Gateway
axsuspend_rule	URI of the Request Gateway where the Request Adapters sends this method
axworkflow_notification	URI where the WF Response Adapter waits for Notifications coming from Response Gateway
edit_ax_comp_form_rule_post	URI of the Request Gateway where the Request Adapters sends this method
edit_axobject	URI where the WF Response Adapter waits for Notifications coming from Response Gateway
list_of_axprograms	URI of the Request Gateway where the Request Adapters sends this method
pandp_ax_user_interface	URI where the WF Response Adapter waits for Notifications coming from Response Gateway
Protection_rule_editor	URI of the Request Gateway where the Request Adapters sends this method
view_object_attribute	URI where the WF Response Adapter waits for Notifications coming from Response Gateway
edit_axquery	URI of the Request Gateway where the Request Adapters sends this method
axactivate_selection_sync	URI where the WF Response Adapter waits for Notifications coming from Response Gateway
axactivate_selection_async	URI of the Request Gateway where the Request Adapters sends this method
axdelete_selection	URI where the WF Response Adapter waits for Notifications coming from Response Gateway
axload_selection	URI of the Request Gateway where the Request Adapters sends this method
axsave_selection_post	URI where the WF Response Adapter waits for Notifications coming from Response Gateway
axlist_user_selection	URI of the Request Gateway where the Request Adapters sends this method
axlist_entitled_selection	URI where the WF Response Adapter waits for Notifications coming from Response Gateway
axcheck_out_sync>	URI of the Request Gateway where the Request Adapters sends this method
axcheck_out_async	URI where the WF Response Adapter waits for Notifications coming from Response Gateway

axcommit_sync	URI of the Request Gateway where the Request Adapters sends this method
axcommit_sync	URI where the WF Response Adapter waits for Notifications coming from Response Gateway
axlock_object	URI of the Request Gateway where the Request Adapters sends this method
axunlock_object	URI where the WF Response Adapter waits for Notifications coming from Response Gateway

7.9 Errors reported and that may occur

Error code	Description and rationales

7.10 Formal description of algorithm <……………>

name	
Method	
Description	
Input parameters	
Output parameters	

name	
Method	
Description	
Input parameters	
Output parameters	

8 Module or Executable Tool Workflow Request Adaptor – BizTalk Server

Module/Tool Profile	
Workflow Request & Input Queue Adaptor – Biztalk Server	
Responsible Name	
Responsible Partner	
Status (proposed/approved)	
Implemented/not implemented	Not implemented
Status of the implementation	
Executable or Library/module	Library (dll) files

(Support)		
Single Thread or Multithread		
Language of Development	Development is performed using MicroSoft Visual Studio 2003 with Biztalk SDK (with Biztalk Wizards).	
Platforms supported	Microsoft Windows	
Reference to the AXFW location of the source code demonstrator	https://cvs.AXMEDIS.org/repos/.................... See in table below the list of the files.	
Reference to the AXFW location of the demonstrator executable tool for internal download	https://cvs. See in table below the list of dll files for Schemas and Orchestration Assemblies	
Reference to the AXFW location of the demonstrator executable tool for public download	https://cvs.AXMEDIS.org./repos/.... See in table below the list of dll files for Schemas and Orchestration Assemblies	
Address for accessing to WebServices if any, add accession information (user and Passwd) if any		
Test cases (present/absent)		
Test cases location	http:///////////////////	
Usage of the AXMEDIS configuration manager (yes/no)	No	
Usage of the AXMEDIS Error Manager (yes/no)	No	
Major Problems not solved	-- --	
Major pending requirements	-- --	
Interfaces API with other tools, named as	Name of the communicating tools References to other major components needed	Communication model and format (protected or not, etc.)
Biztalk Engine API		Internal Library Calls
editorPlugin		Web Services
enginePlugin		Web Services
rePlugin		Web Services
AXMEDIS database query support		Web Services
Formats Used	Shared with	format name or reference to a section

Protocol Used	Shared with	Protocol name or reference to a section
SOAP/WebServices	Biztalk HWS (Human WorkFlow Services)	
Internal Messaging	Calling Biztalk orchestrations	
axWFEditorReq	editorPlugin	axWFEditorReq
axWFEngineReq	enginePlugin	axWFEngineReq
axWFREReq	rePlugin	axWFREReq
axWFDBReq	AXMEDIS Database Query Support	axWFDBReq
axWFEditorRes	editorPlugin	axWFEditorRes
axWFEngineRes	enginePlugin	axWFEngineRes
axWFRERes	rePlugin	axWFRERes
axWFDBRes	AXMEDIS Database Query Support	axWFDBRes
Used Database name		
Microsoft SQL Server 2000		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
Microsoft Biztalk	Microsoft Visual Studio 2003 Biztalk wizards and ASP.NET wizards/C#	Microsoft .NET libraries
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not
Microsoft .NET libraries	.NET Version 1.1.4322	Microsoft Proprietary
Microsoft BizTalk libraries	Version included in Biztalk Server 2004	Microsoft Proprietary

8.1 General Description of the Module

For each of the methods exposed by the WorkFlow Plug-in WebServices (see details in the WSDLs of the AXMEDIS WorkFlow Plug-in WebServices), a Biztalk orchestration will be developed which can be called by other orchestrations implementing the user processes.

The main orchestrations implementing the user process, will call the AXMEDIS orchestrations implementing the WorkFlow Adaptors for accessing the AXMEDIS methods though the Plug-ins, passing the needed parameters and receiving the response parameters through specific BizTalk messages.

The AXMEDIS orchestrations implementing the WorkFlow Adaptors will be called always synchronously and the AXMEDIS orchestrations will perform all the needed requests and will receive the response and notifications (then performing also the Input Queue Adaptors) even when the called requests are implicitly asynchronous. As an example the EditObject method used for calling the AXMEDIS Editor will be implemented via an AXMEDIS orchestration which will call the EditObject WebService receiving the Response and then will wait for the EditorNotification when the Editor is terminated; at this point the AXMEDIS orchestration will give back the AXOID returned in the EditorNotification to the calling BizTalk orchestration.

This approach will simplify the Biztalk orchestration design as in general it has just to call the AXMEDIS orchestrations waiting for the answers. However, if needed, it will be in any case possible to manage also all the Requests and Notifications from the Biztalk orchestrations using directly the messages, ports and Web References implemented in the WorkFlow Adaptors.

8.2 Module Design in terms of Classes

For each method of the WorkFlow Plug-ins in the Editor, Rule Editor, Engine and DataBase Channels, exposed as WebServices, an AXMEDIS orchestration constitutes the WorkFlow Adaptor, implementing also the corresponding Input Queue Adaptor for the related Notifications.

Each orchestration will be composed by:

- A BizTalk schema assembly (a dll file) containing:
  - Schema (xsd file) for the message received from the calling orchestration, containing the request parameters
  - Schema (xsd file) for the message sent back to the calling orchestration, containing the return parameters
  - Property schema (xsd file) for specifying the correlation between the request sent to the WorkFlow plug-in and the received Notifications (i.e. AXRQID)
  - Web References of the Plug-in (request) WebService and of the Notification WebService (osd, xsd, wsdl and disco files)
  - Mapping file mapping the message schema received from the calling orchestration and the called WebService
  - Mapping file mapping the response or the notification (whichever the appropriate case) received from the WebService and the message returned to the calling orchestration
- A Biztalk orchestration assembly (a dll file) containing:
  - The AXMEDIS orchestration design (odx file) containing:
    - The message and message types definitions for:
      - The messages received / sent back from/to the calling orchestration
      - The request WebService
      - The Notification WebService
    - The port and port types for the request WebService and the Notification WebService
    - The correlation type and correlation set used for correlating the WebService request with the following Notifications (i.e. AXRQID correlation)

The following table summarizes the names of the various files for all the AXMEDIS methods:

Editor Channel Methods:

Plug-in invoked WebService	**Schema Assembly**	**Orchestration Assembly**	**Orchestration Design**	**Schema Files**	**Property Schema**	**Web References**	**Mappings**
EditObject	EditObjectSchemas.dll	EditObjectOrchestrations.dll	EditObjectOrchestration.odx	EditObjectRequestSchema.xsd	AXRQID.xsd	WFEditorService.disco WFEditorService.wsdl	EditObjectRequest2WS.map EditObject

Plug-in invoked WebService	**Schema Assembly**	**Orchestration Assembly**	**Orchestration Design**	**Schema Files**	**Property Schema**	**Web References**	**Mappings**
				EditObjectResponseSchema.xsd		Reference.odx Reference.xsd	WS2Response.map
ViewObjectAttribute	ViewObjectAttributeSchemas.dll	ViewObjectAttributeOrchestrations.dll	ViewObjectAttributeOrchestration.odx	ViewObjectAttributeRequestSchema.xsd ViewObjectAttributeResponseSchema.xsd	AXRQID.xsd	WFEditorService.disco WFEditorService.wsdl Reference.odx Reference.xsd	ViewObjectAttributeRequest2WS.map ViewObjectAttributeWS2Response.map
AddHistoryInfo	AddHistoryInfoSchemas.dll	AddHistoryInfoOrchestrations.dll	AddHistoryInfoOrchestration.odx	AddHistoryInfoRequestSchema.xsd AddHistoryInfoResponseSchema.xsd	AXRQID.xsd	WFEditorService.disco WFEditorService.wsdl Reference.odx Reference.xsd	AddHistoryInfoRequest2WS.map AddHistoryInfoWS2Response.map

Engine Channel Methods

Plug-in invoked WebService	**Schema Assembly**	**Orchestration Assembly**	**Orchestration Design**	**Schema Files**	**Property Schema**	**Web References**	**Mappings**
InstallAndActivate	InstallAndActivateSchemas.dll	InstallAndActivateOrchestrations.dll	InstallAndActivateOrchestration.odx	InstallAndActivateRequestSchema.xsd InstallAndActivateResponseSchema.xsd	AXRQID.xsd	EngineService.disco EngineService.wsdl Reference.odx Reference.xsd	InstallAndActivateRequest2WS.map InstallAndActivateWS2Response.map
RunRule	RunRuleSchemas.dll	RunRuleOrchestrations.dll	RunRuleOrchestration.odx	RunRuleRequestSchema.xsd RunRuleResponseSchema.xsd	AXRQID.xsd	EngineService.disco EngineService.wsdl Reference.odx Reference.xsd	RunRuleRequest2WS.map RunRuleWS2Response.map
DeactivateR	Deactivate	DeactivateRul	DeactivateRul	Deactivat	AXRQID.	EngineServ	Deactivate

Plug-in invoked WebService	Schema Assembly	Orchestration Assembly	Orchestration Design	Schema Files	Property Schema	Web References	Mappings
ule	RuleSchemas.dll	eOrchestrations.dll	eOrchestration.odx	eRuleRequestSchema.xsd DeactivateRuleResponseSchema.xsd	xsd	ice.disco EngineService.wsdl Reference.odx Reference.xsd	RuleRequest2WS.map DeactivateRuleWS2Response.map
SuspendRule	SuspendRuleSchemas.dll	SuspendRuleOrchestrations.dll	SuspendRuleOrchestration.odx	SuspendRuleRequestSchema.xsd SuspendRuleResponseSchema.xsd	AXRQID.xsd	EngineService.disco EngineService.wsdl Reference.odx Reference.xsd	SuspendRuleRequest2WS.map SuspendRuleWS2Response.map
PauseRule	PauseRuleSchemas.dll	PauseRuleOrchestrations.dll	PauseRuleOrchestration.odx	PauseRuleRequestSchema.xsd PauseRuleResponseSchema.xsd	AXRQID.xsd	EngineService.disco EngineService.wsdl Reference.odx Reference.xsd	PauseRuleRequest2WS.map PauseRuleWS2Response.map
KillRule	KillRuleSchemas.dll	KillRuleOrchestrations.dll	KillRuleOrchestration.odx	KillRuleRequestSchema.xsd KillRuleResponseSchema.xsd	AXRQID.xsd	EngineService.disco EngineService.wsdl Reference.odx Reference.xsd	KillRuleRequest2WS.map KillRuleWS2Response.map
RemoveRule	RemoveRuleSchemas.dll	RemoveRuleOrchestrations.dll	RemoveRuleOrchestration.odx	RemoveRuleRequestSchema.xsd RemoveRuleResponseSchema.xsd	AXRQID.xsd	EngineService.disco EngineService.wsdl Reference.odx Reference.xsd	RemoveRuleRequest2WS.map RemoveRuleWS2Response.map
ResumeRule	ResumeRuleSchemas.dll	ResumeRuleOrchestrations.dll	ResumeRuleOrchestration.odx	ResumeRuleRequestSchema.xsd ResumeRuleResponseSchema.xsd	AXRQID.xsd	EngineService.disco EngineService.wsdl Reference.odx Reference.xsd	ResumeRuleRequest2WS.map ResumeRuleWS2Response.map
GetRuleStatus	GetRuleStatusSchemas.dll	GetRuleStatusOrchestrations.dll	GetRuleStatusOrchestration.odx	GetRuleStatusRequestSchema.xsd	AXRQID.xsd	EngineService.disco EngineService.wsdl	GetRuleStatusRequest2WS.map GetRuleSta

Plug-in invoked WebService	**Schema Assembly**	**Orchestration Assembly**	**Orchestration Design**	**Schema Files**	**Property Schema**	**Web References**	**Mappings**
				GetRuleStatusResponseSchema.xsd		Reference.odx Reference.xsd	tusWS2Response.map
GetRuleLog	GetRuleLogSchemas.dll	GetRuleLogOrchestrations.dll	GetRuleLogOrchestration.odx	GetRuleLogRequestSchema.xsd GetRuleLogResponseSchema.xsd	AXRQID.xsd	EngineService.disco EngineService.wsdl Reference.odx Reference.xsd	GetRuleLogRequest2WS.map GetRuleLogWS2Response.map
GetListofRules	GetListofRulesSchemas.dll	GetListofRulesOrchestrations.dll	GetListofRulesOrchestration.odx	GetListofRulesRequestSchema.xsd GetListofRulesResponseSchema.xsd	AXRQID.xsd	EngineService.disco EngineService.wsdl Reference.odx Reference.xsd	GetListofRulesRequest2WS.map GetListofRulesWS2Response.map
GetRule	GetRuleSchemas.dll	GetRuleOrchestrations.dll	GetRuleOrchestration.odx	GetRuleRequestSchema.xsd GetRuleResponseSchema.xsd	AXRQID.xsd	EngineService.disco EngineService.wsdl Reference.odx Reference.xsd	GetRuleRequest2WS.map GetRuleWS2Response.map
StatusRequestPnP	StatusRequestPnPSchemas.dll	StatusRequestPnPOrchestrations.dll	StatusRequestPnPOrchestration.odx	StatusRequestPnPRequestSchema.xsd StatusRequestPnPResponseSchema.xsd	AXRQID.xsd	EngineService.disco EngineService.wsdl Reference.odx Reference.xsd	StatusRequestPnPRequest2WS.map StatusRequestPnPWS2Response.map
SuspendPnPProgram	SuspendPnPProgramSchemas.dll	SuspendPnPProgramOrchestrations.dll	SuspendPnPProgramOrchestration.odx	SuspendPnPProgramRequestSchema.xsd SuspendPnPProgramResponseSchema.xsd	AXRQID.xsd	EngineService.disco EngineService.wsdl Reference.odx Reference.xsd	SuspendPnPProgramRequest2WS.map SuspendPnPProgram WS2Response.map
AbortPnPProgram	AbortPnPProgramSchemas.dll	AbortPnPProgramOrchestrations.dll	AbortPnPProgramOrchestration.odx	AbortPnPProgramRequestSchema.xsd AbortPnPProgramR	AXRQID.xsd	EngineService.disco EngineService.wsdl Reference.odx	AbortPnPProgramRequest2WS.map AbortPnPProgramWS

Plug-in invoked WebService	Schema Assembly	Orchestration Assembly	Orchestration Design	Schema Files	Property Schema	Web References	Mappings
				esponseSchema.xsd		Reference.xsd	2Response.map
ResumePnPProgram	ResumePnPProgramSchemas.dll	ResumePnPProgramOrchestrations.dll	ResumePnPProgramOrchestration.odx	ResumePnPProgramRequestSchema.xsd ResumePnPProgramResponseSchema.xsd	AXRQID.xsd	EngineService.disco EngineService.wsdl Reference.odx Reference.xsd	ResumePnPProgramRequest2WS.map ResumePnPProgramWS2Response.map
ActivatePnPProgram	ActivatePnPProgramSchemas.dll	ActivatePnPProgramOrchestrations.dll	ActivatePnPProgramOrchestration.odx	ActivatePnPProgramRequestSchema.xsd ActivatePnPProgramResponseSchema.xsd	AXRQID.xsd	EngineService.disco EngineService.wsdl Reference.odx Reference.xsd	ActivatePnPProgramRequest2WS.map ActivatePnPProgramWS2Response.map
WFNotification	WFNotificationSchemas.dll	WFNotificationOrchestrations.dll	WFNotificationOrchestration.odx	WFNotificationRequestSchema.xsd WFNotificationResponseSchema.xsd	AXRQID.xsd	EngineService.disco EngineService.wsdl Reference.odx Reference.xsd	WFNotificationRequest2WS.map WFNotificationWS2Response.map

Rule Editor Channel Methods:

Plug-in invoked WebService	Schema Assembly	Orchestration Assembly	Orchestration Design	Schema Files	Property Schema	Web References	Mappings
EditCompositionFormatingRule	EditCompositionFormatingRuleSchemas.dll	EditCompositionFormatingRuleOrchestrations.dll	EditCompositionFormatingRuleOrchestration.odx	EditCompositionFormatingRuleRequestSchema.xsd EditComp	AXRQID.xsd	RuleEditorService.disco RuleEditorService.wsdl Reference.o	EditCompositionFormatingRuleRequest2WS.map EditCompositionForm

Plug-in invoked WebService	**Schema Assembly**	**Orchestration Assembly**	**Orchestration Design**	**Schema Files**	**Property Schema**	**Web References**	**Mappings**
				ositionFormatingRuleResponseSchema.xsd		dx Reference.xsd	atingRuleWS2Response.map
PnPUI	PnPUISchemas.dll	PnPUIOrchestrations.dll	PnPUIOrchestration.odx	PnPUIRequestSchema.xsd PnPUIResponseSchema.xsd	AXRQID.xsd	RuleEditorService.disco RuleEditorService.wsdl Reference.odx Reference.xsd	PnPUIRequest2WS.map PnPUIWS2Response.map
ActivatePnP	ActivatePnPSchemas.dll	ActivatePnPOrchestrations.dll	ActivatePnPOrchestration.odx	ActivatePnPRequestSchema.xsd ActivatePnPResponseSchema.xsd	AXRQID.xsd	RuleEditorService.disco RuleEditorService.wsdl Reference.odx Reference.xsd	ActivatePnPRequest2WS.map ActivatePnPWS2Response.map
GetListofPrograms	GetListofProgramsSchemas.dll	GetListofProgramsOrchestrations.dll	GetListofProgramsOrchestration.odx	GetListofProgramsRequestSchema.xsd GetListofProgramsResponseSchema.xsd	AXRQID.xsd	RuleEditorService.disco RuleEditorService.wsdl Reference.odx Reference.xsd	GetListofProgramsRequest2WS.map GetListofProgramsWS2Response.map
EditAXEPToolRule	EditAXEPToolRuleSchemas.dll	EditAXEPToolRuleOrchestrations.dll	EditAXEPToolRuleOrchestration.odx	EditAXEPToolRuleRequestSchema.xsd EditAXEPToolRuleResponseSchema.xsd	AXRQID.xsd	RuleEditorService.disco RuleEditorService.wsdl Reference.odx Reference.xsd	EditAXEPToolRuleRequest2WS.map EditAXEPToolRuleWS2Response.map
EditProtectionRule	EditProtectionRuleSchemas.dll	EditProtectionRuleOrchestrations.dll	EditProtectionRuleOrchestration.odx	EditProtectionRuleRequestSchema.xsd EditProtectionRule	AXRQID.xsd	RuleEditorService.disco RuleEditorService.wsdl Reference.o	EditProtectionRuleRequest2WS.map EditProtectionRuleWS2Response.

Plug-in invoked WebService	Schema Assembly	Orchestration Assembly	Orchestration Design	Schema Files	Property Schema	Web References	Mappings
				Response Schema.xsd		dx Reference.xsd	map

DataBase Channel:

Plug-in invoked WebService	Schema Assembly	Orchestration Assembly	Orchestration Design	Schema Files	Property Schema	Web References	Mappings
EditQuery	EditQuerySchemas.dll	EditQueryOrchestrations.dll	EditQueryOrchestration.odx	EditQueryRequestSchema.xsd EditQueryResponseSchema.xsd	AXRQID.xsd	WFDatabaseService.disco WFDatabaseService.wsdl Reference.odx Reference.xsd	EditQueryRequest2WS.map EditQueryWS2Response.map
ActivateSelectionSync	ActivateSelectionSyncSchemas.dll	ActivateSelectionSyncOrchestrations.dll	ActivateSelectionSyncOrchestration.odx	ActivateSelectionSyncRequestSchema.xsd ActivateSelectionSyncResponseSchema.xsd	AXRQID.xsd	WFDatabaseService.disco WFDatabaseService.wsdl Reference.odx Reference.xsd	ActivateSelectionSyncRequest2WS.map ActivateSelectionSyncWS2Response.map
ActivateSelectionAsync	ActivateSelectionAsyncSchemas.dll	ActivateSelectionAsyncOrchestrations.dll	ActivateSelectionAsyncOrchestration.odx	ActivateSelectionAsyncRequestSchema.xsd ActivateSelectionAsyncResponseSchema.xsd	AXRQID.xsd	WFDatabaseService.disco WFDatabaseService.wsdl Reference.odx Reference.xsd	ActivateSelectionAsyncRequest2WS.map ActivateSelectionAsyncWS2Response.map
DeleteSelction	DeleteSelectionSchemas.dll	DeleteSelectionOrchestrations.dll	DeleteSelectionOrchestration.odx	DeleteSelectionRequestSchema.xsd DeleteSelectionResponseSchema.xsd	AXRQID.xsd	WFDatabaseService.disco WFDatabaseService.wsdl Reference.odx Reference.xsd	DeleteSelectionRequest2WS.map DeleteSelectionWS2Response.map

Plug-in invoked WebService	Schema Assembly	Orchestration Assembly	Orchestration Design	Schema Files	Property Schema	Web References	Mappings
LoadSelection	LoadSelectionSchemas.dll	LoadSelectionOrchestrations.dll	LoadSelectionOrchestration.odx	LoadSelectionRequestSchema.xsd LoadSelectionResponseSchema.xsd	AXRQID.xsd	WFDatabaseService.disco WFDatabaseService.wsdl Reference.odx Reference.xsd	LoadSelectionRequest2WS.map LoadSelectionWS2Response.map
SaveSelection	SaveSelectionSchemas.dll	SaveSelectionOrchestrations.dll	SaveSelectionOrchestration.odx	SaveSelectionRequestSchema.xsd SaveSelectionResponseSchema.xsd	AXRQID.xsd	WFDatabaseService.disco WFDatabaseService.wsdl Reference.odx Reference.xsd	SaveSelectionRequest2WS.map SaveSelectionWS2Response.map
ListUserSelection	ListUserSelectionSchemas.dll	ListUserSelectionOrchestrations.dll	ListUserSelectionOrchestration.odx	ListUserSelectionRequestSchema.xsd ListUserSelectionResponseSchema.xsd	AXRQID.xsd	WFDatabaseService.disco WFDatabaseService.wsdl Reference.odx Reference.xsd	ListUserSelectionRequest2WS.map ListUserSelectionWS2Response.map
ListEntitledSelection	ListEntitledSelectionSchemas.dll	ListEntitledSelectionOrchestrations.dll	ListEntitledSelectionOrchestration.odx	ListEntitledSelectionRequestSchema.xsd ListEntitledSelectionResponseSchema.xsd	AXRQID.xsd	WFDatabaseService.disco WFDatabaseService.wsdl Reference.odx Reference.xsd	ListEntitledSelectionRequest2WS.map ListEntitledSelectionWS2Response.map
CheckOutSync	CheckOutSyncSchemas.dll	CheckOutSyncOrchestrations.dll	CheckOutSyncOrchestration.odx	CheckOutSyncRequestSchema.xsd CheckOutSyncResponseSchema.xsd	AXRQID.xsd	WFDatabaseService.disco WFDatabaseService.wsdl Reference.odx Reference.xsd	CheckOutSyncRequest2WS.map CheckOutSyncWS2Response.map
CheckOutAsync	CheckOutAsyncSchem	CheckOutAsyncOrchestratio	CheckOutAsyncOrchestratio	CheckOutAsyncReq	AXRQID.xsd	WFDatabaseService.di	CheckOutAsyncReques

Plug-in invoked WebService	Schema Assembly	Orchestration Assembly	Orchestration Design	Schema Files	Property Schema	Web References	Mappings
	as.dll	ns.dll	n.odx	uestSchema.xsd CheckOutAsyncResponseSchema.xsd		sco WFDatabaseService.wsdl Reference.odx Reference.xsd	t2WS.map CheckOutAsyncWS2Response.map
CommitSync	CommitSyncSchemas.dll	CommitSyncOrchestrations.dll	CommitSyncOrchestration.odx	CommitSyncRequestSchema.xsd CommitSyncResponseSchema.xsd	AXRQID.xsd	WFDatabaseService.disco WFDatabaseService.wsdl Reference.odx Reference.xsd	CommitSyncRequest2WS.map CommitSyncWS2Response.map
CommitAsync	CommitAsyncSchemas.dll	CommitAsyncOrchestrations.dll	CommitAsyncOrchestration.odx	CommitAsyncRequestSchema.xsd CommitAsyncResponseSchema.xsd	AXRQID.xsd	WFDatabaseService.disco WFDatabaseService.wsdl Reference.odx Reference.xsd	CommitAsyncRequest2WS.map CommitAsyncWS2Response.map
LockObject	LockObjectSchemas.dll	LockObjectOrchestrations.dll	LockObjectOrchestration.odx	LockObjectRequestSchema.xsd LockObjectResponseSchema.xsd	AXRQID.xsd	WFDatabaseService.disco WFDatabaseService.wsdl Reference.odx Reference.xsd	LockObjectRequest2WS.map LockObjectWS2Response.map
UnlockObject	UnlockObjectSchemas.dll	UnlockObjectOrchestrations.dll	UnlockObjectOrchestration.odx	UnlockObjectRequestSchema.xsd UnlockObjectResponseSchema.xsd	AXRQID.xsd	WFDatabaseService.disco WFDatabaseService.wsdl Reference.odx Reference.xsd	UnlockObjectRequest2WS.map UnlockObjectWS2Response.map

8.3 User interface description

No User Interface is implemented in the Adaptors. The calling orchestrations User Interface will be developed using the BizTalk HWS (Human WorkFlow Services).

8.4 Technical and Installation information

References to other major components needed	
Problems not solved	•
Configuration and execution context	

8.5 Draft User Manual

none

8.6 Examples of usage

none

8.7 Integration and compilation issues

none

8.8 Configuration Parameters

Config parameter	Possible values
	Xxxx : Yyyyy : ….. :

8.9 Errors reported and that may occur

Error code	Description and rationales

8.10 Formal description of algorithm <……………>

name	
Method	
Description	
Input parameters	
Output parameters	

name	
Method	
Description	
Input parameters	
Output parameters	

9 Module or Executable Tool Workflow Input Queue Adaptor – Openflow

Module/Tool Profile		
Workflow Input Queue Adaptor - Openflow		
Responsible Name		
Responsible Partner		
Status (proposed/approved)	Approved	
Implemented/not implemented	Implemented	
Status of the implementation	First Prototype Completed	
Executable or Library/module (Support)	library	
Single Thread or Multithread	Multithreaded	
Language of Development	Python or DTML	
Platforms supported	Windows, Linux	
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedig.org/repos/Framework/source/workflow_editor_channel/ openflow/input_queue_adapter/Prove_WF.zexp contains all the libraries for maange the Notifications coming from the Response Gateway	
Reference to the AXFW location of the demonstrator executable tool for internal download	same as above	
Reference to the AXFW location of the demonstrator executable tool for public download		
Address for accessing to WebServices if any, add accession information (user aNd Passwd) if any		
Test cases (present/absent)	yes	
Test cases location	xmlrpc Notifications can be simulated by calling the python shell and issuing: • Import xmlrpclib • d=xmlrpclib.Server("http address of the AXMEDIS Workflow instance") • d.Notification_method(parameters expected in the notification)	
Usage of the AXMEDIS configuration manager (yes/no)	No	
Usage of the AXMEDIS Error Manager (yes/no)	No	
Major Problems not solved	-- --	
Major pending requirements	-- --	
Interfaces API with other	Name of the communicating tools	Communication model and format

tools, named as	References to other major components needed	(protected or not, etc.)
WorkFlow Engine API		Internal librariy calls
Response Gateway		Xmlrpc calls in http
Formats Used	Shared with	format name or reference to a section
Protocol Used	Shared with	Protocol name or reference to a section
axWFRes	WorkFlow Response Gateway	See Section: Formal Description of Communication Protocol between WorkFlow and Gateways
Used Database name		
None		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
None		
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not
Zope 2.7.3	Zope by Zope Corporation Zope 2.7.3 ZPL 2.1 (Zope Public Licence 2.1), open source, GPL compatible	GPL
Python 2.3	Python 2.3 by Stichting Mathematisch Centrum, Amsterdam, The Netherlands. Python 2.3 Free Open Source	GPL
OpenFlow OpenFlow 1.1	OpenFlow by OpenFlow	GPL
xmlrpclib	Xmlrpclib from Python	GPL

9.1 General Description of the Module

There will be a WorkFlow Input Queue Adapter for each communication channel with the other AXMEDIS tools:

1. WF AXOM Input Queue Adatper for integrating the AXMEDIS Object Manager and all AXOM Editors (WorkFlow Editor Channel)
2. WF Rule Editor Input Queue Adapter for integrating all AXMEDIS Rule Editors (WorkFlow Rule Editor Channel)
3. WF Engine Input Queue Adapter for integrating all AXMEDIS Engines (WorkFlow Engine Channel)
4. WF DB Input Queue Adapter for integrating the AXMEDIS DataBase and the Query Support functions (WorkFlow Query and DataBase Channel)

The Input Queue Adapters are the communication channels used by the WorkFlow Engine for receiving the responses of previously issued requests from the above mentioned AXMEDIS components: that is when a response is received, the awaiting pending activity will be resumed by the WorkFlow Engine and the response properly checked in order to continue with following activities in the process flow instance.

9.2 Module Design in terms of Classes

For **each channel**, a Python script was developed for receiving the **Notifications** from the Response Gateway:

- Editor_Notification for receiving the Notifications from the Editor Channel
- Rule_Editor_Notification for receicing the Notifications from the Rule Editor Channel
- Engine_Notification for receiving the Notifications from the Engine Channel
- Query_Notification for receiving the Notifications from the Database Channel

One Python script was developed for receiving the XML-RPC **requests for starting an OpenFlow process** from the Response gateway: **Workflow_Process_Request**. The invoking parameters are stored in proper variables inside the process instance properties.

One Python script was developed for receiving the **WorkFlow Information Request** from the Response gateway: **get_workflow_information**.

The Workflow_Process_Request uses two additional properties in the instance:

- *AXRQID*: the original request ID sent by the AXMEDIS engine for requesting a process instance start
- *EngineListenerService*: the URI where to send the Workflow Notifications

9.3 User interface description

None

9.4 Technical and Installation information

References to other major components needed	WorkFlow Engine
Problems not solved	•
Configuration and execution context	Configuration file is AX_WF_Adapter.cfg in Extensions\ directory of Zope Instance (i.e. c:\Zope-Instance)

9.5 Draft User Manual

None

9.6 Examples of usage

None

9.7 Integration and compilation issues

9.8 Configuration Parameters

See configuration parameters in Request Adapters

Config parameter	Possible values
	Xxxx : Yyyyy : ….. :

9.9 Errors reported and that may occur

Error code	Description and rationales

9.10 Formal description of algorithm <……………>

name	
Method	
Description	
Input parameters	
Output parameters	

name	
Method	
Description	
Input parameters	

Output parameters	

10 Module or Executable Tool Workflow Input Queue Adaptors – BizTalk Server

Please Refer to Section 8.

11 Module or Executable Tool Workflow Request Gateways

Module/Tool Profile		
Workflow Request Gateways		
Responsible Name		
Responsible Partner		
Status (proposed/approved)	Approved	
Implemented/not implemented	Implemented	
Status of the implementation	First Prototype Completed	
Executable or Library/module (Support)	ASP Process	
Single Thread or Multithread	Multithreaded	
Language of Development	C++	
Platforms supported	Windows	
Reference to the AXFW location of the source code demonstrator	https://cvs.AXMEDIS.org/repos/.....................	
Reference to the AXFW location of the demonstrator executable tool for internal download	https://cvs.	
Reference to the AXFW location of the demonstrator executable tool for public download		
Address for accessing to WebServices if any, add accession information (user aNd Passwd) if any		
Test cases (present/absent)		
Test cases location	http://////////////////	
Usage of the AXMEDIS configuration manager (yes/no)	No	
Usage of the AXMEDIS Error Manager (yes/no)	No	
Major Problems not solved	-- --	
Major pending requirements	-- --	
Interfaces API with other tools, named as	Name of the communicating tools References to other major components needed	Communication model and format (protected or not, etc.)
WF Request Adapters		GET calls in http

AXMEDIS modules		WebServices
Formats Used	Shared with	format name or reference to a section
Protocol Used	Shared with	Protocol name or reference to a section
axWFReq	Openflow Workflow	axWFReq
axWFEditorReq	editorPlugin	axWFEditorReq
axWFEngineReq	enginePlugin	axWFEngineReq
axWFREReq	rePlugin	axWFREReq
axWFDBReq	AXMEDIS Database Query Support	axWFDBReq
Used Database name		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
None		
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not
Microsoft ASP		Microsoft .license terms

11.1 General Description of the Module

There are four Request Gatways:

- AXOM WorkFlow Gateway (for the WorkFlow Editor Channel)
- Rule Editor WorkFlow Gateway (for the WorkFlow Rule Editor Channel)
- Engine WorkFlow Gateway (for the WorkFlow Engine Channel)
- DB WorkFlow Gateway (for Query and DataBase Channel)

As previously described, the WF Request Adapter sends the requests via an http GET call. This http GET call is received by a Web Server running Mictosoft IIS and directed to an ASP process called WF Request

Gateway. This ASP process decodes the GET requests and formats a WebService request towards the proper AXMEDIS module.

11.2 Module Design in terms of Classes

11.3 User interface description

None

11.4 Technical and Installation information

The Request Gateways are to be installed as an ASP process running on Microsoft IIS server 6.0 or later. There will be four virtual folders to be created in the IIS server's virtual directory as follows

- requestGateway\editorChannel
- requestGateway\engineChannel
- requestGateway\reChannel
- requestGateway\dbChannel

References to other major components needed	
Problems not solved	•
Configuration and execution context	

11.5 Draft User Manual

None

11.6 Examples of usage

None

11.7 Integration and compilation issues

The Module in not portable. It is developed to work on Microsoft IIS Server only.

Problem with Selection Archive Web Service, with conflict between ListUserSelection and ListEntitledSelection functions. This has been over come by have two separate Selection Archvie dlls, one in which ListUserSelection is renamed (to allow ListEntitledSelection to work), and the other vice versa.

11.8 Configuration Parameters

Config parameter	Possible values
URL for editorChannel	http://localhost:8000/requestGateway/editorChannel/editorChannel.asmx
URL for engineChannel	http://localhost:8000/requestGateway/engineChannel/engineChannel.asmx
URL for reChannel	http://localhost:8000/requestGateway/reChannel/reChannel.asmx
URL for dbChannel	http://localhost:8000/requestGateway/dbChannel/dbChannel.asmx

11.9 Errors reported and that may occur

Error code	Description and rationales
1001	“parameters count less than expected” When the number of parameters received by the gateways are less than expected.
XXXX (Generated by the Systems.Exception Class)	Any Exception that is generated by the faulty behaviour of the Reqeuest Gateway. E.g. server connection unavailable, remote host unavailable, etc.

11.10 Formal description of algorithm <……………>

NONE

name	
Method	
Description	
Input parameters	
Output parameters	

name	
Method	
Description	
Input parameters	
Output parameters	

12 Module or Executable Tool Workflow Response Gateways

Module/Tool Profile	
Workflow Response Gateways	
Responsible Name	
Responsible Partner	
Status (proposed/approved)	Approved
Implemented/not implemented	Implemented
Status of the implementation	First Prototype Completed
Executable or Library/module (Support)	ASP Process
Single Thread or Multithread	Multithreaded
Language of Development	C++

Platforms supported	Windows	
Reference to the AXFW location of the source code demonstrator	https://cvs.AXMEDIS.org/repos/.....................	
Reference to the AXFW location of the demonstrator executable tool for internal download	https://cvs.	
Reference to the AXFW location of the demonstrator executable tool for public download		
Address for accessing to WebServices if any, add accession information (user aNd Passwd) if any		
Test cases (present/absent)		
Test cases location	http://////////////////	
Usage of the AXMEDIS configuration manager (yes/no)	No	
Usage of the AXMEDIS Error Manager (yes/no)	No	
Major Problems not solved	-- --	
Major pending requirements	-- --	
Interfaces API with other tools, named as	Name of the communicating tools References to other major components needed	Communication model and format (protected or not, etc.)
WF Input Queue Adapters		Xmlrpc calls
AXMEDIS modules		WebServices
Formats Used	Shared with	format name or reference to a section
Protocol Used	Shared with	Protocol name or reference to a section
axWFEditorRes	editorPlugin	axWFEditorRes
axWFEngineRes	enginePlugin	axWFEngineRes
axWFRERes	rePlugin	axWFRERes
axWFDBRes	AXMEDIS Database Query Support	axWFDBRes
axWFRes	Openflow Workflow	axWFRes
Used Database name		

User Interface	Development model, language, etc.	Library used for the development, platform, etc.
None		
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not
Microsoft .net	TBD	Microsoft .license terms
C expat Library by James Clark	C expat	Mozilla Public License Version 1.1

12.1 General Description of the Module

There are four Response Gatways:

- AXOM WorkFlow Gateway (for the WorkFlow Editor Channel)
- Rule Editor WorkFlow Gateway (for the WorkFlow Rule Editor Channel)
- Engine WorkFlow Gateway (for the WorkFlow Engine Channel)
- DB WorkFlow Gateway (for Query and DataBase Channel)

When an AXMEDIS tool/engine wants to send back responses to the WorkFlow (i.e. Notifications), it calls a WebServices in the WF Response Gateway. The WF Response Gateway encodes the response in an XMLRPC call directed to the WF Input Queue Adapter.

12.2 Module Design in terms of Classes

12.3 User interface description

None

12.4 Technical and Installation information

The Response Gateways are to be installed as an ASP process running on Microsoft IIS server 6.0 or later. There will be four virtual folders to be created in the IIS server’s virtual directory as follows

- responseGateway\editorChannel
- responseGateway\engineChannel
- responseGateway\reChannel
- responseGateway\dbChannel

References to other major components needed	
Problems not solved	•
Configuration and execution	

context	

12.5 Draft User Manual

None

12.6 Examples of usage

None

12.7 Integration and compilation issues

The Module in not portable. It is developed to work on Microsoft IIS Server only.

12.8 Configuration Parameters

Config parameter	Possible values
URL for editorChannel	http://localhost:8000/responseGateway/editorChannel/editorChannel.asmx
URL for engineChannel	http://localhost:8000/responseGateway/engineChannel/engineChannel.asmx
URL for reChannel	http://localhost:8000/responseGateway/reChannel/reChannel.asmx
URL for dbChannel	http://localhost:8000/responseGateway/dbChannel/dbChannel.asmx

12.9 Errors reported and that may occur

Error code	Description and rationales
1001	“parameters count less than expected” When the number of parameters received by the gateways are less than expected.
XXXX (Generated by the Systems.Exception Class)	Any Exception that is generated by the faulty behaviour of the Reqeuest Gateway. E.g. server connection unavailable, remote host unavailable, etc.
1010	“AXOID value is NULL ” When the request recieved by the gateways doesnt contains Non-Null AXOID value.

12.10 Formal description of algorithm <……………>

NONE

name	
Method	
Description	
Input parameters	
Output parameters	

name	
Method	

Description	
Input parameters	
Output parameters	

13 Module or Executable Tool Workflow Plug-ins

Module/Tool Profile		
Workflow Plug-ins		
Responsible Name		
Responsible Partner		
Status (proposed/approved)	Approved	
Implemented/not implemented	Implemented	
Status of the implementation	First Prototype Completed	
Executable or Library/module (Support)	ASP Process	
Single Thread or Multithread	Multithreaded	
Language of Development	C++	
Platforms supported	Windows	
Reference to the AXFW location of the source code demonstrator	https://cvs.AXMEDIS.org/repos/....................	
Reference to the AXFW location of the demonstrator executable tool for internal download	https://cvs.	
Reference to the AXFW location of the demonstrator executable tool for public download		
Address for accessing to WebServices if any, add accession information (user aNd Passwd) if any		
Test cases (present/absent)		
Test cases location	http://////////////////	
Usage of the AXMEDIS configuration manager (yes/no)	No	
Usage of the AXMEDIS Error Manager (yes/no)	No	
Major Problems not solved	-- --	
Major pending requirements	-- --	
Interfaces API with other tools, named as	Name of the communicating tools References to other major components needed	Communication model and format (protected or not, etc.)
Workflow Request Gateway		WebServices
AXMEDIS modules	AXOM Command & Reporting	C++ function calls

Workflow Response Gateways		Web Services
Formats Used	Shared with	format name or reference to a section
Protocol Used	Shared with	Protocol name or reference to a section
axWFEditorRes	Response Gateway	
axWFEngineRes	Response Gateway	
axWFRERes	Response Gateway	
axWFDBRes	Response Gateway	
axWFEditorReq	Request Gateway	
axWFEngineReq	Request Gateway	
axWFREReq	Request Gateway	
axWFDBReq	Request Gateway	
Used Database name		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
None		
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not
Microsoft .net		Microsoft .license terms
pthread	Pthreadvce2.dll	GNU
Soapcpp2	Soapcpp2.dll	Gsoap GNU Licence

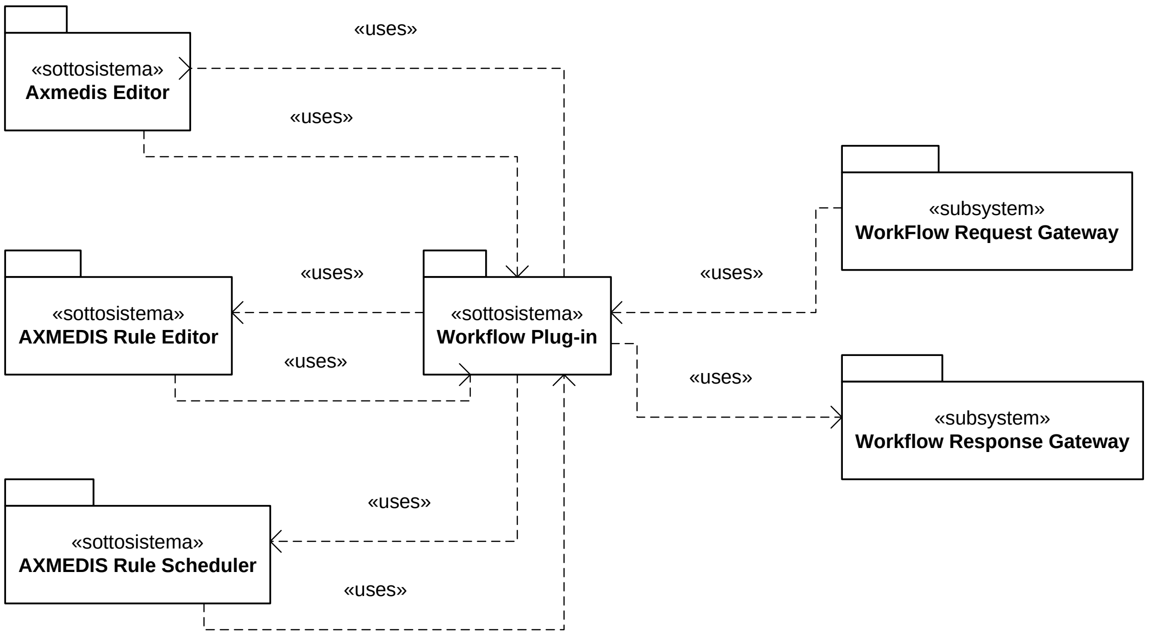
13.1 General Description of the Module

In order to enable invocation of the various workflow commands inside the AXMEDIS Tools, Workflow Plug-ins are developed. The plug-ins will interact with the tools through AXOM's Command & Reporting module. The AXMEDIS Workflow Manager will communicate to AXOM's Command and Reporting through WF Request Adapter. The WF Request Adapter sends the requests via an http GET call. This http GET call is received by a Web Server running Microsoft IIS and directed to an ASP process called WF Request Gateway. This ASP process decodes the GET requests and formats a WebService request towards the proper AXMEDIS Tool. The plug-in contains an active service, listening to incoming webservice request.. Upon receipt of the request, the plug-in will invoke appropriate functions inside the AXOM's command and reporting, which are then delivered to proper tool. An appropriate notification is send back to the workflow to complete the request. In case of asynchronous workflow request, the plug-ins also acts as a client for the Workflow Response Gateways. The AXMEDIS tools can invoke methods inside the workflow plug-ins, which will then make appropriate webservice calls towards the Workflow Response Gateways.

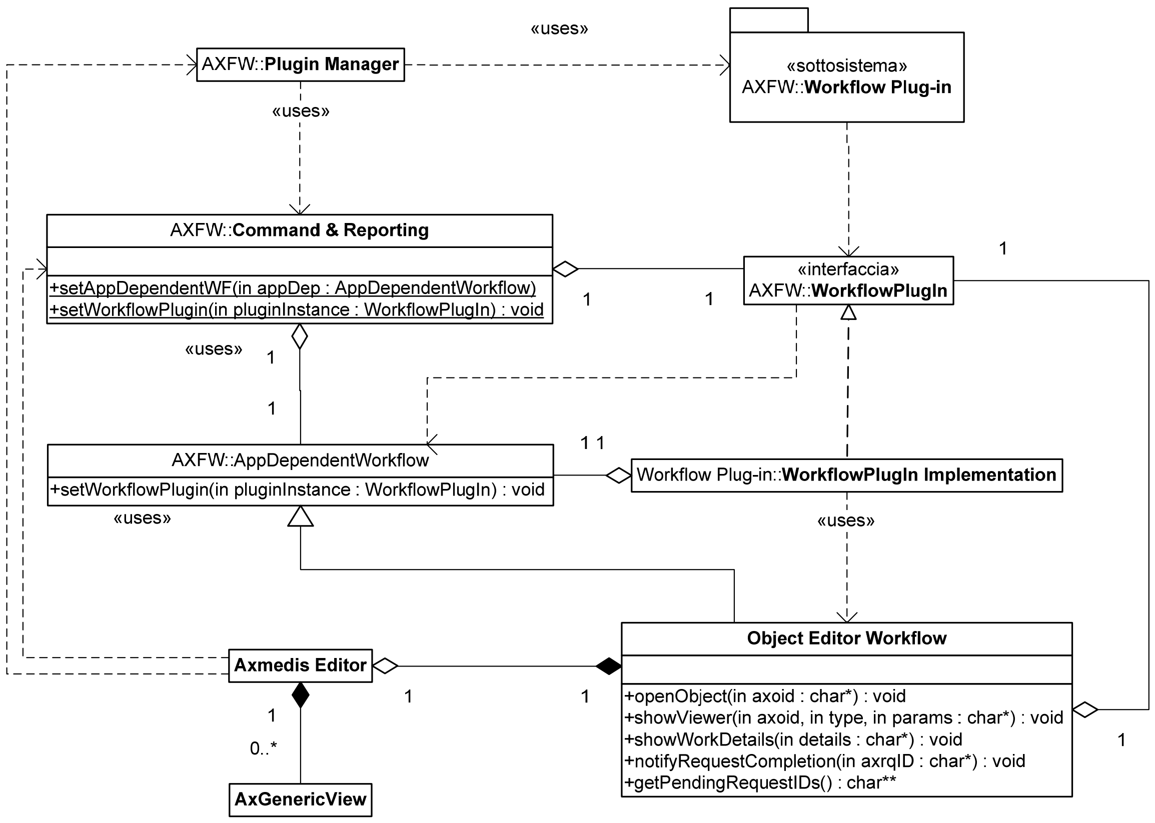
There are in total three plug-ins developed for the case of AXMEDIS as follows

- Workflow editor plug-in
- Workflow engine plug-in
- Workflow rule editor plug-in

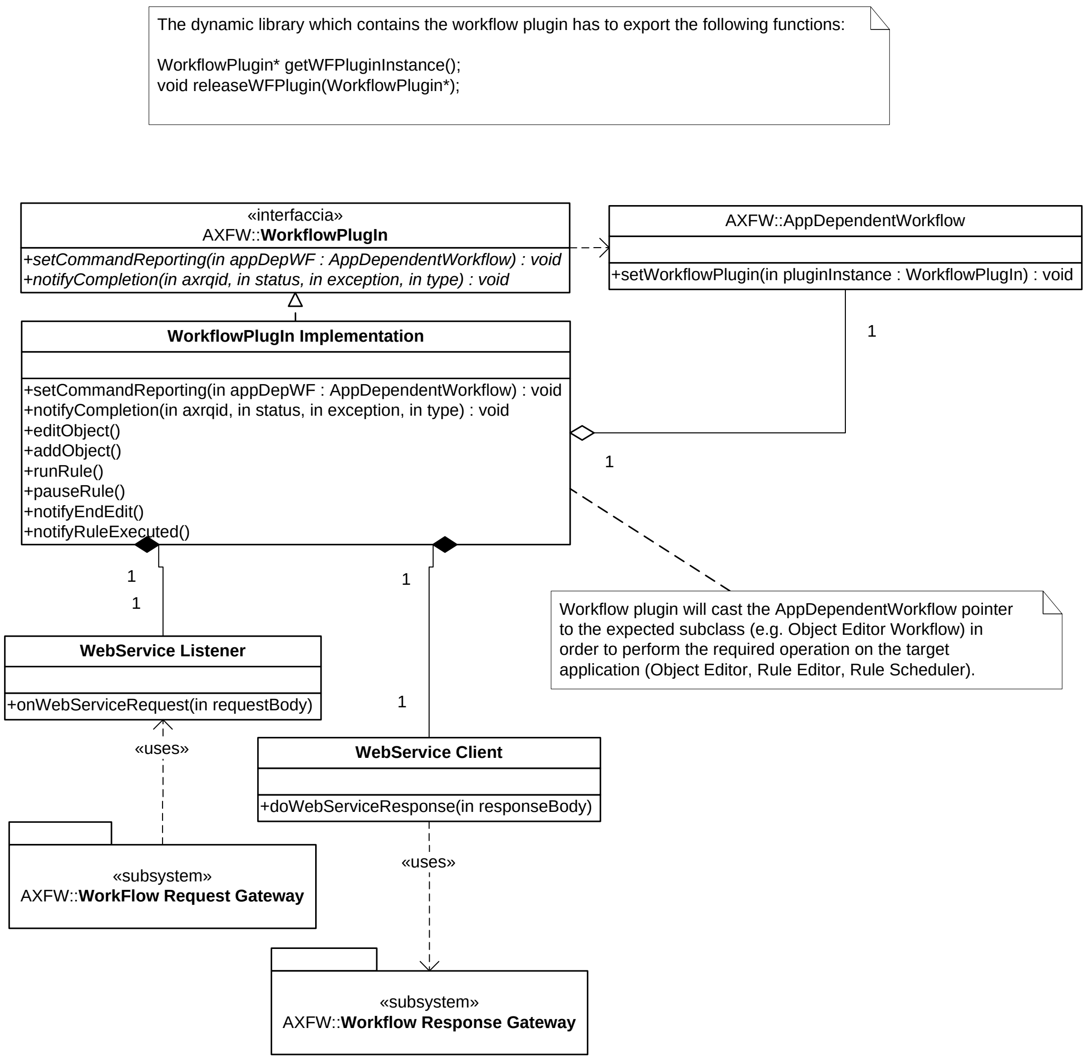
13.2 Module Design in terms of Classes



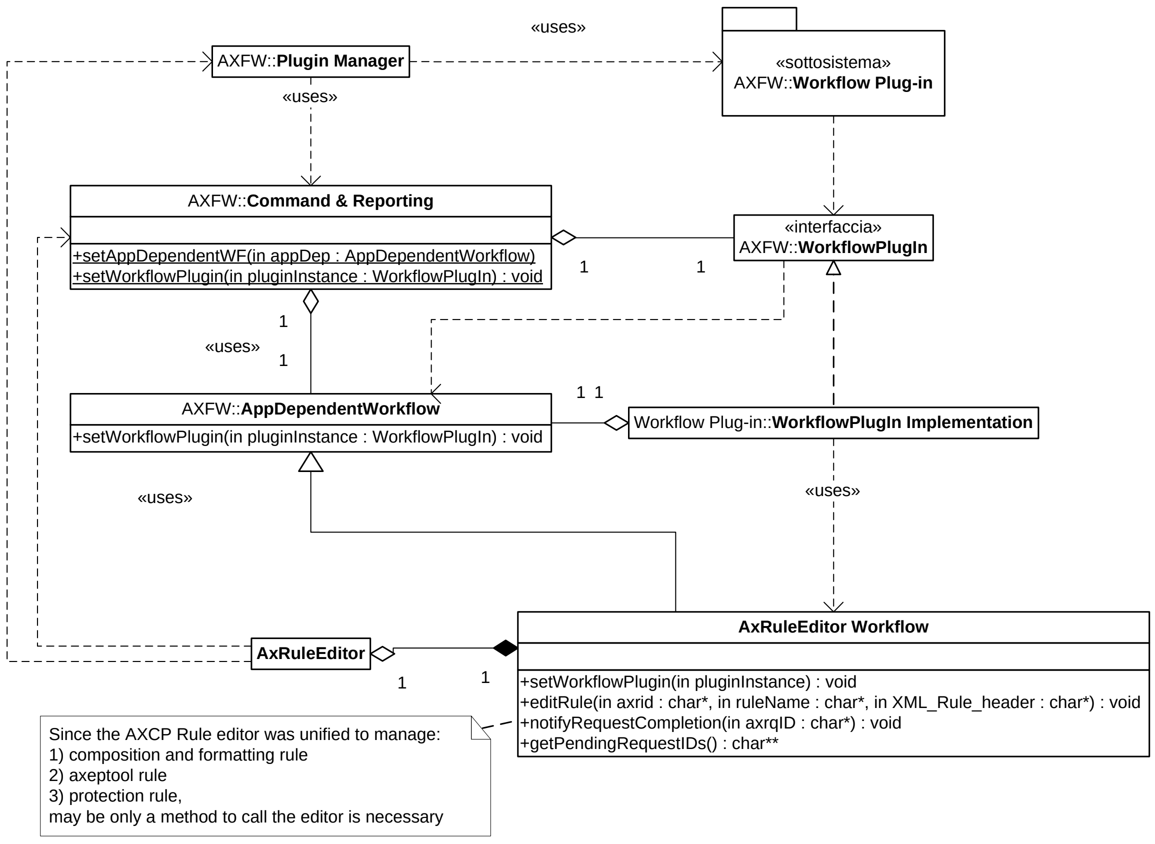
Workflow Plugins Overall Architecture


«uses»
AXFW::Plugin Manager
«sottosistema»
AXFW::Workflow Plug-in
«uses»
AXFW::Command & Reporting
+setAppDependentWF(in appDep : AppDependentWorkflow)
+setWorkflowPlugin(in pluginInstance : WorkflowPlugIn) : void
«interfaccia»
AXFW::WorkflowPlugIn
1
1
1
«uses»
1
1
AXFW::AppDependentWorkflow
+setWorkflowPlugin(in pluginInstance : WorkflowPlugIn) : void
1 1
Workflow Plug-in::WorkflowPlugIn Implementation
«uses»
«uses»
Object Editor Workflow
+openObject(in axoid : char*) : void
+showViewer(in axoid, in type, in params : char*) : void
+showWorkDetails(in details : char*) : void
+notifyRequestCompletion(in axrqID : char*) : void
+getPendingRequestIDs() : char**
Axmedis Editor
1
1
1
0..*
1
AxGenericView

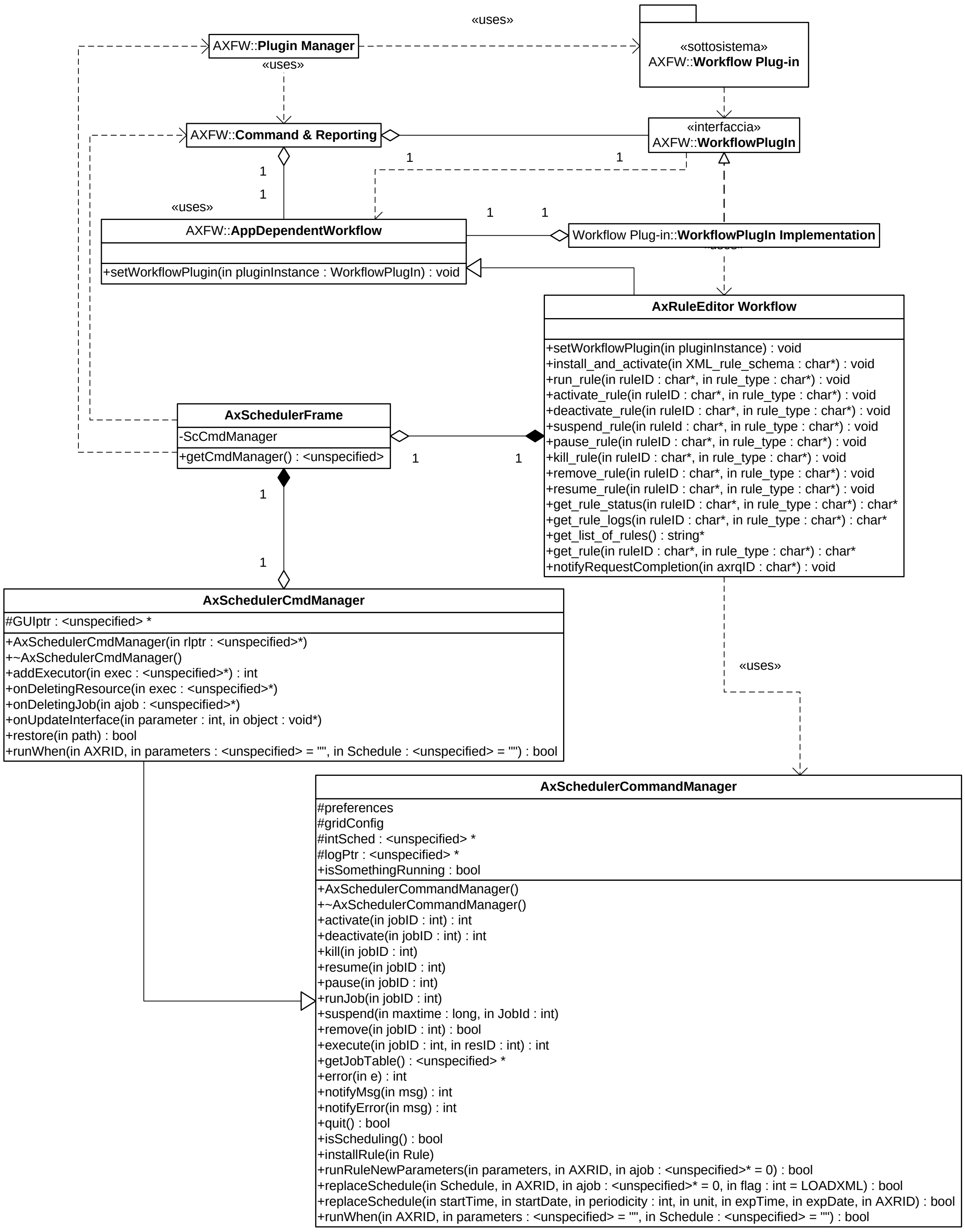
AXMEDIS Editor and Workflow Plugin


The dynamic library which contains the workflow plugin has to export the following functions:
WorkflowPlugin* getWFPluginInstance();
void releaseWFPlugin(WorkflowPlugin*);
«interfaccia»
AXFW::WorkflowPlugIn
+setCommandReporting(in appDepWF : AppDependentWorkflow) : void
+notifyCompletion(in axrqid, in status, in exception, in type) : void
AXFW::AppDependentWorkflow
+setWorkflowPlugin(in pluginInstance : WorkflowPlugIn) : void
1
WorkflowPlugIn Implementation
+setCommandReporting(in appDepWF : AppDependentWorkflow) : void
+notifyCompletion(in axrqid, in status, in exception, in type) : void
+editObject()
+addObject()
+runRule()
+pauseRule()
+notifyEndEdit()
+notifyRuleExecuted()
1
1
1
1
Workflow plugin will cast the AppDependentWorkflow pointer to the expected subclass (e.g. Object Editor Workflow) in order to perform the required operation on the target application (Object Editor, Rule Editor, Rule Scheduler).
WebService Listener
+onWebServiceRequest(in requestBody)
1
«uses»
WebService Client
+doWebServiceResponse(in responseBody)
«uses»
«subsystem»
AXFW::WorkFlow Request Gateway
«subsystem»
AXFW::Workflow Response Gateway

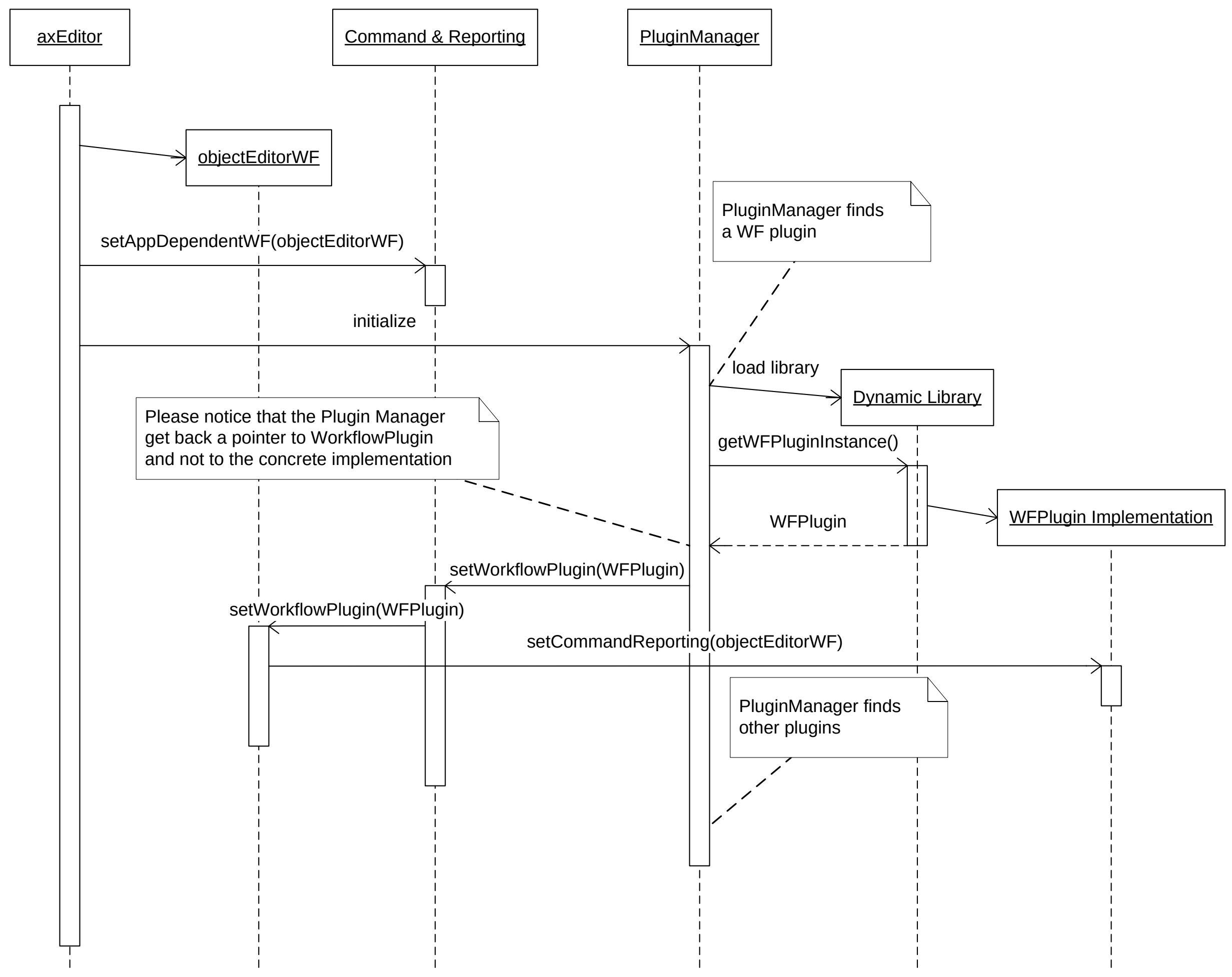
Workflow Plugin General Architecture


«uses»
AXFW::Plugin Manager
«sottosistema»
AXFW::Workflow Plug-in
«uses»
AXFW::Command & Reporting
+setAppDependentWF(in appDep : AppDependentWorkflow)
+setWorkflowPlugin(in pluginInstance : WorkflowPlugIn) : void
«interfaccia»
AXFW::WorkflowPlugIn
1
1
1
«uses»
1
1 1
AXFW::AppDependentWorkflow
+setWorkflowPlugin(in pluginInstance : WorkflowPlugIn) : void
Workflow Plug-in::WorkflowPlugIn Implementation
«uses»
«uses»
AxRuleEditor Workflow
AxRuleEditor
1
1
+setWorkflowPlugin(in pluginInstance) : void
+editRule(in axrid : char*, in ruleName : char*, in XML_Rule_header : char*) : void
+notifyRequestCompletion(in axrqID : char*) : void
+getPendingRequestIDs() : char**
Since the AXCP Rule editor was unified to manage:
1) composition and formatting rule
2) axeptool rule
3) protection rule,
may be only a method to call the editor is necessary

Rule Editor and Workflow Plug-in Architecture


«uses»
AXFW::Plugin Manager
«uses»
«sottosistema»
AXFW::Workflow Plug-in
AXFW::Command & Reporting
«interfaccia»
AXFW::WorkflowPlugIn
1
1
1
1
«uses»
1
1
AXFW::AppDependentWorkflow
+setWorkflowPlugin(in pluginInstance : WorkflowPlugIn) : void
Workflow Plug-in::WorkflowPlugIn Implementation
«uses»
AxRuleEditor Workflow
+setWorkflowPlugin(in pluginInstance) : void
+install_and_activate(in XML_rule_schema : char*) : void
+run_rule(in ruleID : char*, in rule_type : char*) : void
+activate_rule(in ruleID : char*, in rule_type : char*) : void
+deactivate_rule(in ruleID : char*, in rule_type : char*) : void
+suspend_rule(in ruleId : char*, in rule_type : char*) : void
+pause_rule(in ruleID : char*, in rule_type : char*) : void
+kill_rule(in ruleID : char*, in rule_type : char*) : void
+remove_rule(in ruleID : char*, in rule_type : char*) : void
+resume_rule(in ruleID : char*, in rule_type : char*) : void
+get_rule_status(in ruleID : char*, in rule_type : char*) : char*
+get_rule_logs(in ruleID : char*, in rule_type : char*) : char*
+get_list_of_rules() : string*
+get_rule(in ruleID : char*, in rule_type : char*) : char*
+notifyRequestCompletion(in axrqID : char*) : void
AxSchedulerFrame
-ScCmdManager
+getCmdManager() : <unspecified>
1
1
1
1
AxSchedulerCmdManager
#GUIptr : <unspecified> *
+AxSchedulerCmdManager(in rlptr : <unspecified>*)
+~AxSchedulerCmdManager()
+addExecutor(in exec : <unspecified>*) : int
+onDeletingResource(in exec : <unspecified>*)
+onDeletingJob(in ajob : <unspecified>*)
+onUpdateInterface(in parameter : int, in object : void*)
+restore(in path) : bool
+runWhen(in AXRID, in parameters : <unspecified> = "", in Schedule : <unspecified> = "") : bool
«uses»
AxSchedulerCommandManager
#preferences
#gridConfig
#intSched : <unspecified> *
#logPtr : <unspecified> *
+isSomethingRunning : bool
+AxSchedulerCommandManager()
+~AxSchedulerCommandManager()
+activate(in jobID : int) : int
+deactivate(in jobID : int) : int
+kill(in jobID : int)
+resume(in jobID : int)
+pause(in jobID : int)
+runJob(in jobID : int)
+suspend(in maxtime : long, in JobId : int)
+remove(in jobID : int) : bool
+execute(in jobID : int, in resID : int) : int
+getJobTable() : <unspecified> *
+error(in e) : int
+notifyMsg(in msg) : int
+notifyError(in msg) : int
+quit() : bool
+isScheduling() : bool
+installRule(in Rule)
+runRuleNewParameters(in parameters, in AXRID, in ajob : <unspecified>* = 0) : bool
+replaceSchedule(in Schedule, in AXRID, in ajob : <unspecified>* = 0, in flag : int = LOADXML) : bool
+replaceSchedule(in startTime, in startDate, in periodicity : int, in unit, in expTime, in expDate, in AXRID) : bool
+runWhen(in AXRID, in parameters : <unspecified> = "", in Schedule : <unspecified> = "") : bool

AXCP and Workflow Plug-in Architecture


axEditor
Command & Reporting
PluginManager
objectEditorWF
PluginManager finds a WF plugin
setAppDependentWF(objectEditorWF)
initialize
load library
Dynamic Library
Please notice that the Plugin Manager get back a pointer to WorkflowPlugin and not to the concrete implementation
getWFPluginInstance()
WFPlugin Implementation
WFPlugin
setWorkflowPlugin(WFPlugin)
setWorkflowPlugin(WFPlugin)
setCommandReporting(objectEditorWF)
PluginManager finds other plugins

General Sequence Diagram

13.3 User interface description

None

13.4 Technical and Installation information

References to other major components needed	AXOM's Command & Reporting.
Problems not solved	•
Configuration and execution context	

13.5 Draft User Manual

None

13.6 Examples of usage

None

13.7 Integration and compilation issues

Only one instance of the plug-in will be active at a time for each AXMEDIS Tools. E.g. If two AXMEDIS Object Editors are running, only one of the Editor will be able to communicate with the workflow. This is because only one plug-in can bind to port in order to receive workflow requests.

Invocation of tools requires another tool (“AXMEDIS Invoker,perhaps”) , which has knowledge of the paths for each tools’ executable, which can then be called by the Workflow (by Web Service) for a specific tool to be invoked. This is because the plugins now run inside the tools themselves, and so any attempt to communicate with them, requires the tools to already be running.

13.8 Configuration Parameters

Config parameter	Possible values
Incoming Request Port	Any Port that is not blocked by any other application. Appropriate settings needs to be done for the request gateways to send the request at the specified port. Please see above in the document

13.9 Errors reported and that may occur

Error code	Description and rationales
0101	« Cannot Invoke the tool » This message is returned to the workflow, if the plugin is not able to load appropriate methids inside the specified tool.

13.10 Formal description of algorithm <……………>

NONE

name	
Method	
Description	
Input parameters	
Output parameters	

name	
Method	
Description	
Input parameters	
Output	

parameters	

14 Provided API named Workflow Editor Plugin

The editorPlugin, developed for the workflow integration with the AXMEDIS tools, will be loaded by the AXMEDIS Object Editor using AXMEDIS Plugin Manager. The plugin is capable of communication with either Workflow Engine (Openflow or BizTalk).The functions that are exposed by this plugin are as follows:

Call name	
Method	void setCommandReporting(AppDependentWorkflow* appDepWF);
Description	To set the application pointer received from AXOM's Command & Reporting
Input parameters	AppDependentWorkflow* appDepWF// Pointer to AXMEDIS tool handling workflow methods.
Output parameters	void
Request Sample Message	
Response Sample Message	

Call name	
Method	bool notifyCompletion();
Description	This method is invoked by the editor to send back the notification towards workflow engine for the completion of previously issues asynchronous request.
Input parameters	char* AXRQID, //The original request ID char* URI, //URI to be used for sending notifications char* endpURL, //URI for the response gateway bool result=true, //The result of operation true = successful or false = failure char* errorCode="NULL", //Error Code in case of failure char* errorMSG="NULL", //Error MSG in case of failure char* AXOID="NULL", // Not NULL if new object created
Output parameters	bool
Request Sample Message	
Response Sample Message	

Call name	
Method	bool getWFInfo();
Description	This method is invoked by the application to request workflow information from workflow engine.
Input parameters	char* AXOID, char* endpURL,bool *Result, char *FaultCode,char *FaultString, char *AXRQID, char *Title, char *ProcessName, char *ActivityName, char *Status, char *Priority, char *Actor

	The only input parameter is AXOID, others are the return parameters whose values are obtained from workflow's response gateway
Output parameters	bool
Request Sample Message	
Response Sample Message	

Call name	
Method	bool editObject();
Description	This method is invoked through a webservice call coming from Workflow Request Gateway. This method will load the specified object (AXOID) and will allow the user to edit the object in the AXMEDIS Editor.
Input parameters	char *AXOID, //The object ID, NULL if creating new object char *userCredentials, //users permissions char *executionParameters, //parameters for execution char *editingType, //type of editing char *AXRQID, //request ID to be used for notification purpose char *URI//URI to use for notification
Output parameters	bool
Request Sample Message	
Response Sample Message	
Call name	
Method	bool viewObjectAttribute();
Description	This method is invoked through a webservice call coming from Workflow Request Gateway. This method will allow the workflow engine to retrive the object attributes for the object specified by the AXOID.
Input parameters	char* AXOID, //Object ID char* userCredentials, //Users permissions char* attributeList, //Comma Seperated Attribute IDs char* AXRQID//request ID to be used for notification purpose
Output parameters	bool
Request Sample Message	
Response Sample Message	
Call name	
Method	bool addHistoryInfo();

Description	This method is invoked through a webservice call coming from Workflow Request Gateway. This method will allow the workflow engine to add the object history for the object specified by the AXOID.
Input parameters	char* AXOID, //Object ID char* userCredentials, //Users permissions char* historyInfo, //history log info string char* AXRQID//request ID to be used for notification purpose
Output parameters	bool
Request Sample Message	
Response Sample Message	

15 Provided API named Workflow Engine Plugin

The enginePlugin, developed for the workflow integration with the AXMEDIS tools, will be loaded by the various AXMEDIS Engines like AXCP, AXEPTool, PnP, etc using AXMEDIS Plugin Manager. The plugin is capable of communication with either Workflow Engine (Openflow or BizTalk).The functions that are exposed by this plugin are as follows:

Call name	
Method	void setCommandReporting();
Description	To set the application pointer received from AXOM's Command & Reporting
Input parameters	AppDependentWorkflow* appDepWF
Output parameters	void
Request Sample Message	
Response Sample Message	

Call name	
Method	notifyCompletion()
Description	This method is invoked by the engine to send back the notification towards workflow engine for the completion of previously issues asynchronous request.
Input parameters	char* AXRQID,char* URI, char* endpURL,bool result=true, char* errorCode="NULL",char* errorMSG="NULL",char* Status="NULL"
Output parameters	bool
Request Sample Message	
Response Sample Message	

Call name	
Method	bool wfProcessRequest();
Description	This method is invoked by the engine to send request for activation of a workflow process identified the supplied processed.
Input parameters	char* AXRQID,char* URI, char* endpURL, char* processID
Output parameters	bool
Request Sample Message	
Response Sample Message	

Call name	
Method	InstallAndActivate();
Description	This method is invoked through a webservice call coming from Workflow Request Gateway. This method will allow the workflow engine to install a new rule in the AXCP engine. The ruleID of the newly installed rule will be returned as ruleID.
Input parameters	char *AXRQID,char *userCredentials, char *xmlRuleSchema, char *URI,std::string &ruleID
Output parameters	bool
Request Sample Message	
Response Sample Message	

Call name	
Method	RunRule();
Description	This method is invoked through a webservice call coming from Workflow Request Gateway. This method will allow the workflow engine to run a rule in the AXCP engine as per the supplied parameters.
Input parameters	char *AXRQID,char *userCredentials, char *ruleID, char *ruleType, char *URI, char *time, char *arguments char*
Output parameters	bool
Request Sample Message	
Response Sample Message	

Call name	
Method	DeactivateRule();
Description	This method is invoked through a webservice call coming from Workflow Request Gateway. This method will allow the workflow engine to deactivate a rule in the AXCP engine as per the supplied parameters.
Input parameters	char *AXRQID, char *userCredentials, char *ruleID, char *ruleType
Output parameters	bool
Request Sample Message	
Response Sample Message	

Call name	
Method	bool SuspendRule()
Description	This method is invoked through a webservice call coming from Workflow Request Gateway. This method will allow the workflow engine to suspend a rule in the AXCP engine as per the

	supplied parameters.
Input parameters	char *AXRQID,char *userCredentials, char *ruleID, char *ruleType, char *suspendTime
Output parameters	bool
Request Sample Message	
Response Sample Message	

Call name	
Method	bool PauseRule(;
Description	This method is invoked through a webservice call coming from Workflow Request Gateway. This method will allow the workflow engine to pause a rule in the AXCP engine as per the supplied parameters.
Input parameters	char *AXRQID, char *userCredentials, char *ruleID,char *ruleType
Output parameters	bool
Request Sample Message	
Response Sample Message	

Call name	
Method	bool KillRule( );
Description	This method is invoked through a webservice call coming from Workflow Request Gateway. This method will allow the workflow engine to kill a rule in the AXCP engine as per the supplied parameters.
Input parameters	char *AXRQID, char *userCredentials, char *ruleID, char *ruleType
Output parameters	bool
Request Sample Message	
Response Sample Message	

Call name	
Method	bool RemoveRule();
Description	This method is invoked through a webservice call coming from Workflow Request Gateway. This method will allow the workflow engine to remove a rule from the AXCP engine as per the supplied parameters.
Input parameters	char *AXRQID, char *userCredentials, char *ruleID, char *ruleType
Output parameters	bool
Request	

Sample Message	
Response Sample Message	

Call name	
Method	bool ResumeRule();
Description	This method is invoked through a webservice call coming from Workflow Request Gateway. This method will allow the workflow engine to resume a rule in the AXCP engine as per the supplied parameters.
Input parameters	char *AXRQID, char *userCredentials, char *ruleID, char *ruleType, char *URI,char *arguments
Output parameters	bool
Request Sample Message	
Response Sample Message	

Call name	
Method	bool GetRuleStatus();
Description	This method is invoked through a webservice call coming from Workflow Request Gateway. This method will allow the workflow engine to know the status of the a rule in the AXCP engine as per the supplied parameters.
Input parameters	char *AXRQID, char *userCredentials, char *ruleID, char *ruleType,std::string &status
Output parameters	bool
Request Sample Message	
Response Sample Message	

Call name	
Method	bool GetRuleLog();
Description	This method is invoked through a webservice call coming from Workflow Request Gateway. This method will allow the workflow engine to know the run log for a rule in the AXCP engine as per the supplied parameters.
Input parameters	char *AXRQID,char *userCredentials, char *ruleID, char *ruleType, std::string &ruleLogs
Output parameters	bool
Request Sample Message	
Response Sample Message	

Call name	
Method	bool GetListofRules();
Description	This method is invoked through a webservice call coming from Workflow Request Gateway. This method will allow the workflow engine to retrieve the list of currently installed rules in the AXCP engine as per the supplied parameters.
Input parameters	char *AXRQID,char *userCredentials, char *ruleType, std::string &ruleIDList
Output parameters	bool
Request Sample Message	
Response Sample Message	

Call name	
Method	bool GetRule();
Description	This method is invoked through a webservice call coming from Workflow Request Gateway. This method will allow the workflow engine to retrieve the rule schema from the AXCP engine as per the supplied parameters.
Input parameters	char *AXRQID,char *userCredentials, char *ruleID, char *ruleType,std::string &ruleSchema
Output parameters	bool
Request Sample Message	
Response Sample Message	

Call name	
Method	bool StatusRequestPnP();
Description	This method is invoked through a webservice call coming from Workflow Request Gateway. This method will allow the workflow engine to retrieve the status of the PnP engine.
Input parameters	char *AXRQID, char *userCredentials, char *ProgramID, std::string &status
Output parameters	bool
Request Sample Message	
Response Sample Message	

Call name	
Method	bool getListofPrograms();
Description	This method is invoked through a webservice call coming from Workflow Request Gateway.

	This method will allow the workflow engine to retrieve the list of the program from the PnP Editor.
Input parameters	char *AXRQID, char *userCredentials, std::string programList
Output parameters	bool
Request Sample Message	
Response Sample Message	
Call name	
Method	bool SuspendPnPProgram();
Description	This method is invoked through a webservice call coming from Workflow Request Gateway. This method will allow the workflow engine to suspend a program in the PnP engine.
Input parameters	char *AXRQID, char *userCredentials,char *programID
Output parameters	bool
Request Sample Message	
Response Sample Message	

Call name	
Method	bool AbortPnPProgram();
Description	This method is invoked through a webservice call coming from Workflow Request Gateway. This method will allow the workflow engine to abort a program in the PnP engine.
Input parameters	char *AXRQID, char *userCredentials, char *ProgramID
Output parameters	bool
Request Sample Message	
Response Sample Message	

Call name	
Method	bool ResumePnPProgram( );
Description	This method is invoked through a webservice call coming from Workflow Request Gateway. This method will allow the workflow engine to resume a program in the PnP engine.
Input parameters	char *AXRQID, char *userCredentials, char *ProgramID, char *URI
Output parameters	bool
Request Sample Message	
Response	

Sample Message	

Call name	
Method	bool ActivatePnPProgram();
Description	This method is invoked through a webservice call coming from Workflow Request Gateway. This method will allow the workflow engine to activate a program in the PnP engine.
Input parameters	char *AXRQID,char *userCredentials, char *ProgramID, char *URI
Output parameters	bool
Request Sample Message	
Response Sample Message	

Call name	
Method	bool WFNotification();
Description	This method is invoked through a webservice call coming from Workflow Request Gateway. This method will allow the workflow engine to send the notification to the PnP engine for the previously issues request to activate a process.
Input parameters	char *AXRQID, char *userCredentials,char *status
Output parameters	bool
Request Sample Message	
Response Sample Message	

16 Provided API named Workflow Rule Editor Plugin

The rePlugin, developed for the workflow integration with the AXMEDIS tools, will be loaded by the AXMEDIS Rule Editor using AXMEDIS Plugin Manager. The plugin is capable of communication with either Workflow Engine (Openflow or BizTalk). The functions that are exposed by this plugin are as follows:

Call name	
Method	void setCommandReporting();
Description	To set the application pointer received from AXOM’s Command & Reporting
Input parameters	AppDependentWorkflow* appDepWF
Output parameters	void
Request Sample Message	

Response Sample Message	

Call name	
Method	bool notifyCompletion();
Description	This method is invoked by the rule editor to send back the notification towards workflow engine for the completion of previously issues asynchronous request.
Input parameters	char* AXRQID,char* URI, char* endpURL,bool result=true, char* errorCode="NULL",char* errorMSG="NULL", char* xmlRuleSchema="NULL"
Output parameters	bool
Request Sample Message	
Response Sample Message	

Call name	
Method	bool editRule();
Description	This method is invoked through a webservice call coming from Workflow Request Gateway. This method will allow the workflow engine to invoke the rule editor and edit the rule. The rule header will be supplied by the workflow engine.
Input parameters	char* AXRQID, char* userCredentials, char *xmlRuleHeader,char *URI, char* editingType
Output parameters	bool
Request Sample Message	
Response Sample Message	

Call name	
Method	bool PnPUI();
Description	This method is invoked through a webservice call coming from Workflow Request Gateway. This method will allow the workflow engine to invoke the PnP Program Editor to edit a program as specified by its name.
Input parameters	char *AXRQID, char *userCredentials,char *programName, char *URI
Output parameters	bool
Request Sample Message	
Response Sample Message	

17 Table description for database for Workflow Engine

For Openflow as a candidate workflow for integration with AXMEDIS, these are as follows:

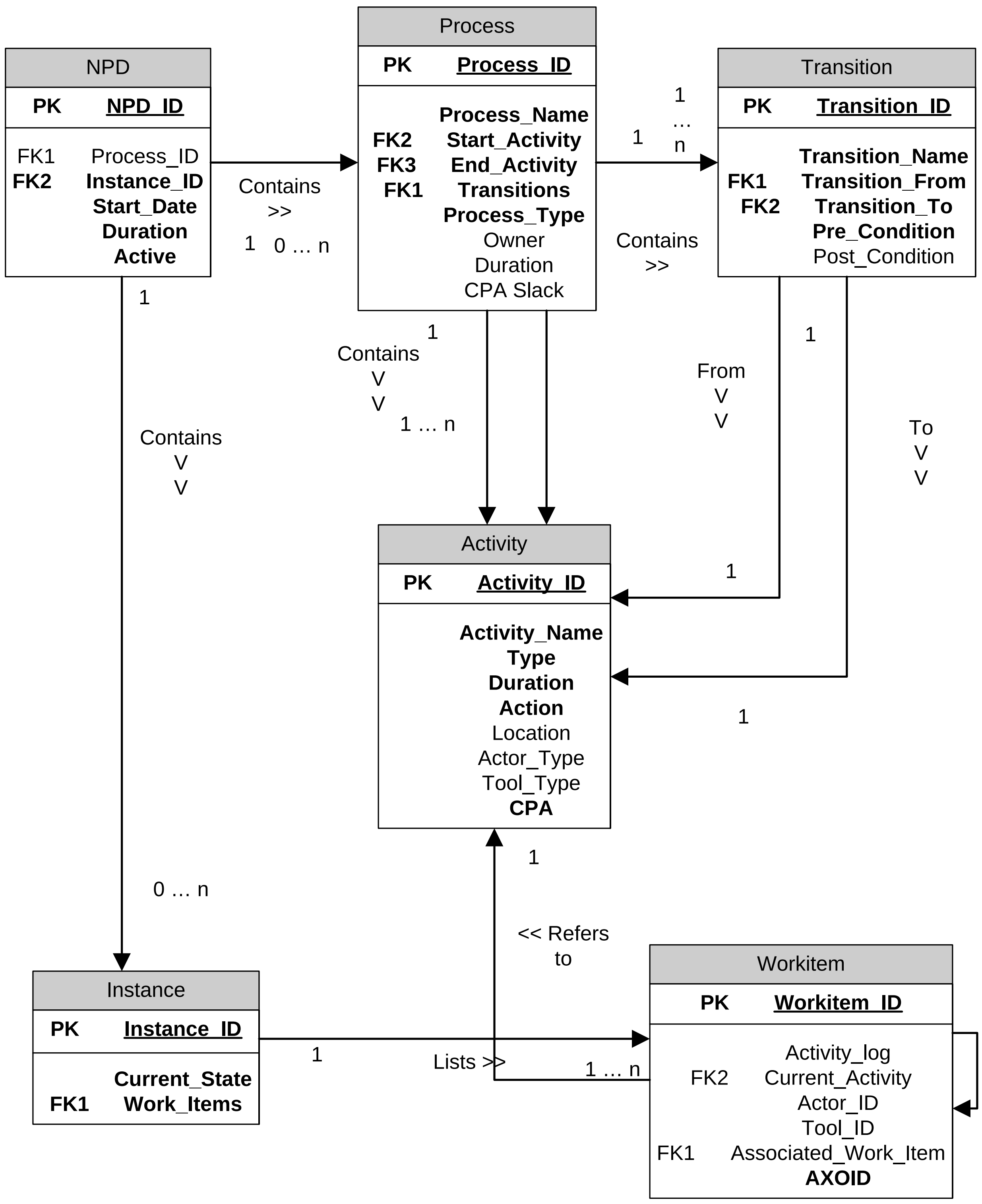
Openflow
Process
Activity
Transition
Instance
Workitem
Employee
Role
History

Where in the context of Openflow deployed as an AXWF, the field *History* from the above list could include the following sub-fields:

History (processing_history, progress, locked, stuck-at, urgency, queue-position authorisation_status)

Thus as far as the design of the AXWFDB is concerned, the Standard OpenFlow Data Model will be adapted so that it can integrate efficiently with AXMEDIS. The primary adaptation will be to associate to every OpenFlow Process Instance the related AXMEDIS AXOID and to associate such additional history sub-fields from the above list as may be required. However as most WFMSs maintain an AcxtionLog which is normally associated with Workitem within the workflow Database, we can safely conclude that the associating an AXMEDIS Object_ID (AXOID) with the Workflow Instance will prove an adequate adaptation in most cases as follows:

AXMEDIS Workflow Database (AXWFDB)



18 Formal description of format Config.XML

For the case of the workflow gateways, the following configuration file (config.xml) is used.

The URLs for various workflow componenents are stored in this file. These components are

1. Editor Request Gateway
2. Editor Response Gateway
3. Editor Plugin
4. Editor Response Adaptor
5. Editor Invoker
6. AXCP Engine Request Gateway
7. AXCP Engine Response Gateway
8. AXCP Engine Plugin
9. AXCP Engine Response Adaptor
10. PnP Engine Request Gateway
11. PnP Engine Response Gateway
12. PnP Engine Plugin
13. PnP Engine Response Adaptor
14. Rule Editor Request Gateway
15. Rule Editor Response Gateway
16. Rule Editor Engine Plugin
17. Rule Editor Response Adaptor
18. Rule Editor Invoker
19. PnP Editor Request Gateway
20. PnP Editor Response Gateway
21. PnP Editor Engine Plugin
22. PnP Editor Response Adaptor
23. PnP Editor Invoker
24. AXDB Request Gateway
25. AXDB Response Gateway

The gateways uses this file to retrieve the URL for the plug-ins to communicate with.

```
<?xml version="1.0" encoding="UTF-8"?>
<CONFIGURATION xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation=".\config.xsd">
      <GATEWAYS>
            <REQUEST>

      <EDITOR>http://localhost:8000/requestGateway/editorChannel/editorChannel.a
smx</EDITOR>

      <AXCPENGINE>http://localhost:8000/requestGateway/engineChannel/engineChann
el.asmx</AXCPENGINE>

      <RULE_EDITOR>http://localhost:8000/requestGateway/reChannel/reChannel.asmx
</RULE_EDITOR>

      <AXDB>http://localhost:8000/requestGateway/dbChannel/dbChannel.asmx</AXDB>
            </REQUEST>
            <RESPONSE>

      <EDITOR>http://localhost:8000/responseGateway/editorChannel/editorChannel.
asmx</EDITOR>
```

```
      <AXCPENGINE>http://localhost:8000/responseGateway/engineChannel/engineChan
nel.asmx</AXCPENGINE>

      <RULE_EDITOR>http://localhost:8000/responseGateway/reChannel/reChannel.asm
x</RULE_EDITOR>

      <LOADLISTENER>http://192.168.0.240:8000/responseGateway/dbChannel/loadList
ener/loadListener.asmx</LOADLISTENER>

      <SAVELISTENER>http://192.168.0.240:8000/responseGateway/dbChannel/saveList
ener/saveListener.asmx</SAVELISTENER>

      <SALISTENER>http://192.168.0.240:8000/responseGateway/dbChannel/saListener
/saListener.asmx</SALISTENER>
            </RESPONSE>
      </GATEWAYS>
      <PLUGINS>
            <EDITOR>http://localhost:9001/</EDITOR>
            <EDITOR_INVOKER>http://localhost:9090/</EDITOR_INVOKER>
            <AXCPENGINE>http://192.168.0.191:9000/</AXCPENGINE>
            <PnPENGINE>http://localhost:9000/</PnPENGINE>
            <RULE_EDITOR>http://localhost:9002/</RULE_EDITOR>
            <RE_INVOKER>http://localhost:9090/</RE_INVOKER>
            <PnP_EDITOR>http://localhost:9002/</PnP_EDITOR>
            <PnP_INVOKER>http://localhost:9090/</PnP_INVOKER>
            <LOADER>http://192.168.0.108:8080/LoaderSaver/load</LOADER>
            <SAVER>http://192.168.0.108:8080/LoaderSaver/save</SAVER>

      <LOCKUNLOCK>http://87.24.151.2:8080/LockUnlockWS/lockunlock</LOCKUNLOCK>

      <SELECTIONARCHIVE>http://192.168.0.108:8080/UserSelectionArchive/sa</SELEC
TIONARCHIVE>
      </PLUGINS>
      <ADAPTORS>
            <RESPONSE>
                  <EDITOR>http://localhost:8080/Prove_WF</EDITOR>
                  <AXCPENGINE>http://localhost:8080/Prove_WF</AXCPENGINE>
                  <RULE_EDITOR>http://localhost:8080/Prove_WF</RULE_EDITOR>
                  <AXDB>http://localhost:8080/Prove_WF</AXDB>
            </RESPONSE>
      </ADAPTORS>
</CONFIGURATION>
```

19 Formal description of format AX_WF_Adapter.cfg

Configuration parameters are in AX_WF_Adapter.cfg configuration file:

```
ListenerURI=<URI where the adapter waits Notifications>
activate_axpnp=<URI>
add_history_info=<URI>
axabort_pnp_program=<URI>
axactivate_pnp_program=<URI>
axactivate_rule=<URI>
axdeactivate_rule=<URI>
AXEPTool_rule_editor=<URI>
axget_list_of_rules=<URI>
axget_rule_logs=<URI>
axget_rule_schema=<URI>
axget_rule_status=<URI>
axinstall_and_activate_post=<URI>
axkill_rule=<URI>
axpause_rule=<URI>
axpnp_status_request=<URI>
axremove_rule=<URI>
axresume_pnp_program=<URI>
axresume_rule=<URI>
axrun_rule=<URI>
axsuspend_pnp_program=<URI>
axsuspend_rule=<URI>
axworkflow_notification=<URI>
edit_ax_comp_form_rule_post=<URI>
edit_axobject=<URI>
list_of_axprograms=<URI>
pandp_ax_user_interface=<URI>
Protection_rule_editor=<URI>
view_object_attribute=<URI>
edit_axquery=<URI>
axactivate_selection_sync=<URI>
axactivate_selection_async=<URI>
axdelete_selection=<URI>
axload_selection=<URI>
axsave_selection_post=<URI>
axlist_user_selection=<URI>
axlist_entitled_selection=<URI>
axcheck_out_sync=<URI>
axcheck_out_async=<URI>
axcommit_sync=<URI>
axcommit_sync=<URI>
axlock_object=<URI>
axunlock_object=<URI>
```

20 Formal description of communication protocol Between Workflow and Gateways

20.1 Webservice Communication Protocol for methods from Request Adapters and Request Gateways (axWFReq)

20.1.1 Edit Object method (Editor Channel)

GET{ServiceURI}?AXOID="Object ID string"&Credentials="Credential string"&AXRQID="Request ID string"&editing_type="editing_type string"&execution_parameters="execution parameter string"&URI="EditorListenerService string"

AXOID is the Object ID to be edited *(Note: AXOID can be void while requesting editing a new object)*
Credentials contains user Credentials.
AXRQID is the unique identifier ot the Request, to be returned in following notifications from the Response Gateway
Editing_type and *execution_parameters* are optional parameters to be sent to the Editor
URI is the URI where the Response Adapter waits for notifications from the Response Gateway

The response to the invoked method is sent via an http GET response. The response is XML coded, following the schema:

```
<xs:element name="Editor_Response">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="result" type="xs:boolean"/>
                        <xs:element name="errormsg" type="xs:string" nillable="true" minOccurs="0"/>
                        <xs:element name="errorcode" type="xs:int" minOccurs="0"/>
                        <xs:element name="AXOID" type="xs:string" minOccurs="0"/>
                        <xs:element name="historylog" type="xs:string" minOccurs="0"
maxOccurs="100"/>
                        <xs:element name="attributes" minOccurs="0" maxOccurs="20">
                                <xs:complexType>
                                        <xs:sequence>
                                                <xs:element name="attributeid" type="xs:string"/>
                                                <xs:element name="attributevalue" type="xs:string"/>
                                        </xs:sequence>
                                </xs:complexType>
                        </xs:element>
                </xs:sequence>
        </xs:complexType>
</xs:element>
```

20.1.2 View Object method (Editor Channel)

GET{ServiceURI}?AXOID="AXOID string"&Credentials="Credential string"&AXRQID="Request ID string"&attributes="list of attributeids comma separated"

AXOID is the Object ID to be edited *(Note: AXOID can be void while requesting editing a new object)*
Credentials contains user Credentials.

AXRQID is the unique identifier ot the Request, to be returned in following notifications from the Response Gateway

The response to the invoked method is sent via an http GET response. The response is XML coded, following the schema:

```
<xs:element name="Editor_Response">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="result" type="xs:boolean"/>
                        <xs:element name="errormsg" type="xs:string" nillable="true" minOccurs="0"/>
                        <xs:element name="errorcode" type="xs:int"/>
                        <xs:element name="AXOID" type="xs:string" minOccurs="0"/>
                        <xs:element        name="historylog"        type="xs:string"        minOccurs="0"
maxOccurs="100"/>
                        <xs:element name="attributes" minOccurs="0" maxOccurs="20">
                                <xs:complexType>
                                        <xs:sequence>
                                                <xs:element name="attributeid" type="xs:string"/>
                                                <xs:element name="attributevalue" type="xs:string"/>
                                        </xs:sequence>
                                </xs:complexType>
                        </xs:element>
                </xs:sequence>
        </xs:complexType>
</xs:element>
```

20.1.3 Add History Info method (Editor Channel)

GET{ServiceURI}?AXOID=”AXOID string”&Credentials=”Credential string”&AXRQID=”Request ID string”&loginfo=”log info string”

AXOID is the Object ID to be edited *(Note: AXOID can be void while requesting editing a new object)*
Credentials contains user Credentials.
AXRQID is the unique identifier ot the Request, to be returned in following notifications from the Response Gateway

The response to the invoked method is sent via an http GET response. The response is XML coded, following the schema:

```
<xs:element name="Editor_Response">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="result" type="xs:boolean"/>
                        <xs:element name="errormsg" type="xs:string" nillable="true" minOccurs="0"/>
                        <xs:element name="errorcode" type="xs:int"/>
                        <xs:element name="AXOID" type="xs:string" minOccurs="0"/>
                        <xs:element        name="historylog"        type="xs:string"        minOccurs="0"
maxOccurs="100"/>
                        <xs:element name="attributes" minOccurs="0" maxOccurs="20">
                                <xs:complexType>
                                        <xs:sequence>
                                                <xs:element name="attributeid" type="xs:string"/>
                                        <xs:element name="attributevalue" type="xs:string"/>
                                    </xs:sequence>
                                </xs:complexType>
                            </xs:element>
                        </xs:sequence>
            </xs:complexType>
</xs:element>
```

20.1.4 Edit Composition Formatting Rule method (Rule Editor Channel)

POST{ServiceURI}

The Request Header contains:
Credentials=”Credential string”
AXRQID=”Request ID string
URI=”UserListenerService string”

And the Request body contains the XML_Rule_header in xml/text format

Credentials contains user Credentials.
AXRQID is the unique identifier ot the Request, to be returned in following notifications from the Response Gateway
URI is the URI where the Response Adapter waits for notifications from the Response Gateway

The response to the invoked method is sent via an http POST response. The response is XML coded, following the schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
            <xs:element name="Rule_editor_Response">
                        <xs:complexType>
                                    <xs:sequence>
                                                <xs:element name="result" type="xs:boolean"/>
                                                <xs:element name="errormsg" type="xs:string" nillable="true"
minOccurs="0"/>
                                                <xs:element name="errorcode" type="xs:int"/>
                                                <xs:element name="programid" type="xs:string" minOccurs="0"
maxOccurs="20"/>
                                    </xs:sequence>
                        </xs:complexType>
            </xs:element>
</xs:schema>
```

20.1.5 Program Publication User Interface method (Rule Editor Channel)

GET{ServiceURI}? Credentials=”Credential string”&AXRQID=”Request ID string”&Program_Name=”Program Name string”&URI=”UserListenerService string”

Program_Name contains the name of the Program to be edited
Credentials contains user Credentials.
AXRQID is the unique identifier ot the Request, to be returned in following notifications from the Response Gateway
URI is the URI where the Response Adapter waits for notifications from the Response Gateway

The response to the invoked method is sent via an http GET response. The response is XML coded, following the schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" attributeFormDefault="unqualified">
	<xs:element name="Rule_editor_Response">
		<xs:complexType>
			<xs:sequence>
				<xs:element name="result" type="xs:boolean"/>
				<xs:element name="errormsg" type="xs:string" nillable="true" minOccurs="0"/>
				<xs:element name="errorcode" type="xs:int"/>
				<xs:element name="programid" type="xs:string" minOccurs="0" maxOccurs="20"/>
			</xs:sequence>
		</xs:complexType>
	</xs:element>
</xs:schema>
```

20.1.6 Activate Program Publication method (Rule Editor Channel)

GET{ServiceURI}? Credentials="Credential string"&AXRQID="Request ID string"&ProgramID="ProgramID string"&URI="EngineListenerService string"

ProgramID contains the ID of the program to be activated.
Credentials contains user Credentials.
AXRQID is the unique identifier ot the Request, to be returned in following notifications from the Response Gateway
URI is the URI where the Response Adapter waits for notifications from the Response Gateway

The response to the invoked method is sent via an http GET response. The response is XML coded, following the schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" attributeFormDefault="unqualified">
	<xs:element name="Rule_editor_Response">
		<xs:complexType>
			<xs:sequence>
				<xs:element name="result" type="xs:boolean"/>
				<xs:element name="errormsg" type="xs:string" nillable="true" minOccurs="0"/>
				<xs:element name="errorcode" type="xs:int"/>
				<xs:element name="programid" type="xs:string" minOccurs="0" maxOccurs="20"/>
			</xs:sequence>
		</xs:complexType>
	</xs:element>
</xs:schema>
```

20.1.7 List of Programs method (Rule Editor Channel)

GET{ServiceURI}? Credentials=”Credential string”&AXRQID=”Request ID string”

Credentials contains user Credentials.
AXRQID is the unique identifier ot the Request, to be returned in following notifications from the Response Gateway

The response to the invoked method is sent via an http GET response. The response is XML coded, following the schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
	<xs:element name="Rule_editor_Response">
		<xs:complexType>
			<xs:sequence>
				<xs:element name="result" type="xs:boolean"/>
				<xs:element name="errormsg" type="xs:string" nillable="true"
minOccurs="0"/>
				<xs:element name="errorcode" type="xs:int"/>
				<xs:element name="programid" type="xs:string" minOccurs="0"
maxOccurs="20"/>
			</xs:sequence>
		</xs:complexType>
	</xs:element>
</xs:schema>
```

20.1.8 Edit AXEPTool Rule method (Rule Editor Channel)

GET{ServiceURI}? Credentials=”Credential string”&AXRQID=”Request ID string”&Rule_name=”Rule Name string”&URI=”UserListenerService string” HTTP/1.1" 200 368 "http://15.156.120.195:8080/Control_Panel/Products/OpenFlow/AXMEDIS/edit_AXEPTool_rule" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR 1.0.3705; .NET CLR 1.1.4322)"

Rule_Name contains the name of the Rule to be edited
Credentials contains user Credentials.
AXRQID is the unique identifier ot the Request, to be returned in following notifications from the Response Gateway
URI is the URI where the Response Adapter waits for notifications from the Response Gateway

The response to the invoked method is sent via an http GET response. The response is XML coded, following the schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
	<xs:element name="Rule_editor_Response">
		<xs:complexType>
			<xs:sequence>
				<xs:element name="result" type="xs:boolean"/>
				<xs:element name="errormsg" type="xs:string" nillable="true"
minOccurs="0"/>
				<xs:element name="errorcode" type="xs:int"/>
```

```
                <xs:element name="programid" type="xs:string" minOccurs="0" maxOccurs="20"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

20.1.9 Edit Protection Rule method (Rule Editor Channel)

GET{ServiceURI}? Credentials=”Credential string”&AXRQID=”Request ID string”&Rule_name=”Rule Name string”&URI=”UserListenerService string”

Rule_Name contains the name of the Rule to be edited
Credentials contains user Credentials.
AXRQID is the unique identifier ot the Request, to be returned in following notifications from the Response Gateway
URI is the URI where the Response Adapter waits for notifications from the Response Gateway

The response to the invoked method is sent via an http GET response. The response is XML coded, following the schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xs:element name="Rule_editor_Response">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="result" type="xs:boolean"/>
                <xs:element name="errormsg" type="xs:string" nillable="true" minOccurs="0"/>
                <xs:element name="errorcode" type="xs:int"/>
                <xs:element name="programid" type="xs:string" minOccurs="0" maxOccurs="20"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

20.1.10 Install and Activate method (Engine Channel)

POST{ServiceURI}

The Request Header contains:
Credentials=”Credential string”
AXRQID=”Request ID string
URI=”UserListenerService string”

And the Request body contains the XML_Rule_in xml/text format

Credentials contains user Credentials.
AXRQID is the unique identifier ot the Request, to be returned in following notifications from the Response Gateway
URI is the URI where the Response Adapter waits for notifications from the Response Gateway

The response to the invoked method is sent via an http POST response. The response is XML coded, following the schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" attributeFormDefault="unqualified">
	<xs:element name="Engine_Response">
		<xs:complexType>
			<xs:sequence>
				<xs:element name="result" type="xs:boolean"/>
				<xs:element name="errormsg" type="xs:string" nillable="true" minOccurs="0"/>
				<xs:element name="errorcode" type="xs:int"/>
				<xs:element name="ruleid" type="xs:string" minOccurs="0" maxOccurs="20"/>
				<xs:element name="status" type="xs:string" minOccurs="0"/>
				<xs:element name="xml_rule_schema" type="xs:string" minOccurs="0"/>
				<xs:element name="rulelog" type="xs:string" minOccurs="0" maxOccurs="100"/>
			</xs:sequence>
		</xs:complexType>
	</xs:element>
</xs:schema>
```

20.1.11 Run Rule method (Engine Channel)

GET{ServiceURI}? Credentials=”Credential string”&AXRQID=”Request ID string”&Rule_ID=”Rule ID string”&Rule_type=”rule type string”&Time=”Time”&Arguments=”Argument string”URI=”EngineListenerService string”

Credentials contains user Credentials.
AXRQID is the unique identifier ot the Request, to be returned in following notifications from the Response Gateway
URI is the URI where the Response Adapter waits for notifications from the Response Gateway
Rule_ID and *Rule_type* are the ID and the type of the rule to be run
Time: When (if missing original time from Rule Schema) the rule shall be run
Arguments: for overriding original arguments in rule schema

The response to the invoked method is sent via an http GET response. The response is XML coded, following the schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" attributeFormDefault="unqualified">
	<xs:element name="Engine_Response">
		<xs:complexType>
			<xs:sequence>
				<xs:element name="result" type="xs:boolean"/>
				<xs:element name="errormsg" type="xs:string" nillable="true" minOccurs="0"/>
				<xs:element name="errorcode" type="xs:int"/>
```

```
                    <xs:element name="ruleid" type="xs:string" minOccurs="0"
maxOccurs="20"/>
                    <xs:element name="status" type="xs:string" minOccurs="0"/>
                    <xs:element name="xml_rule_schema" type="xs:string" minOccurs="0"/>
                    <xs:element name="rulelog" type="xs:string" minOccurs="0"
maxOccurs="100"/>
               </xs:sequence>
          </xs:complexType>
     </xs:element>
</xs:schema>
```

20.1.12 Deactivate rule method (Engine Channel)

GET{ServiceURI}? Credentials=”Credential string”&AXRQID=”Request ID string”&Rule_ID=”Rule ID string”&Rule_type=”rule type string”

Credentials contains user Credentials.
AXRQID is the unique identifier ot the Request, to be returned in following notifications from the Response Gateway
Rule_ID and *Rule_type* are the ID and the type of the rule to be deactivated

The response to the invoked method is sent via an http GET response. The response is XML coded, following the schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
     <xs:element name="Engine_Response">
          <xs:complexType>
               <xs:sequence>
                    <xs:element name="result" type="xs:boolean"/>
                    <xs:element name="errormsg" type="xs:string" nillable="true"
minOccurs="0"/>
                    <xs:element name="errorcode" type="xs:int"/>
                    <xs:element name="ruleid" type="xs:string" minOccurs="0"
maxOccurs="20"/>
                    <xs:element name="status" type="xs:string" minOccurs="0"/>
                    <xs:element name="xml_rule_schema" type="xs:string" minOccurs="0"/>
                    <xs:element name="rulelog" type="xs:string" minOccurs="0"
maxOccurs="100"/>
               </xs:sequence>
          </xs:complexType>
     </xs:element>
</xs:schema>
```

20.1.13 Suspend Rule method (Engine Channel)

GET{ServiceURI}? Credentials=”Credential string”&AXRQID=”Request ID string”&Rule_ID=”Rule ID string”&Rule_type=”rule type string”&Max_time=”string containing max time”

Credentials contains user Credentials.
AXRQID is the unique identifier ot the Request, to be returned in following notifications from the Response Gateway

Rule_ID and *Rule_type* are the ID and the type of the rule to be suspended
Max_time is the maximum time the rule will remain suspended

The response to the invoked method is sent via an http GET response. The response is XML coded, following the schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
	<xs:element name="Engine_Response">
		<xs:complexType>
			<xs:sequence>
				<xs:element name="result" type="xs:boolean"/>
				<xs:element name="errormsg" type="xs:string" nillable="true"
minOccurs="0"/>
				<xs:element name="errorcode" type="xs:int"/>
				<xs:element name="ruleid" type="xs:string" minOccurs="0"
maxOccurs="20"/>
				<xs:element name="status" type="xs:string" minOccurs="0"/>
				<xs:element name="xml_rule_schema" type="xs:string" minOccurs="0"/>
				<xs:element name="rulelog" type="xs:string" minOccurs="0"
maxOccurs="100"/>
			</xs:sequence>
		</xs:complexType>
	</xs:element>
</xs:schema>
```

20.1.14 Pause Rule method (Engine Channel)

GET{ServiceURI}? Credentials=”Credential string”&AXRQID=”Request ID string”&Rule_ID=”Rule ID string”&Rule_type=”rule type string”

Credentials contains user Credentials.
AXRQID is the unique identifier ot the Request, to be returned in following notifications from the Response Gateway
Rule_ID and *Rule_type* are the ID and the type of the rule to be paused

The response to the invoked method is sent via an http GET response. The response is XML coded, following the schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
	<xs:element name="Engine_Response">
		<xs:complexType>
			<xs:sequence>
				<xs:element name="result" type="xs:boolean"/>
				<xs:element name="errormsg" type="xs:string" nillable="true"
minOccurs="0"/>
				<xs:element name="errorcode" type="xs:int"/>
				<xs:element name="ruleid" type="xs:string" minOccurs="0"
maxOccurs="20"/>
				<xs:element name="status" type="xs:string" minOccurs="0"/>
				<xs:element name="xml_rule_schema" type="xs:string" minOccurs="0"/>
```

```
                <xs:element name="rulelog" type="xs:string" minOccurs="0" maxOccurs="100"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

20.1.15 Kill Rule method (Engine Channel)

GET{ServiceURI}? Credentials=”Credential string”&AXRQID=”Request ID string”&Rule_ID=”Rule ID string”&Rule_type=”rule type string”

Credentials contains user Credentials.
AXRQID is the unique identifier ot the Request, to be returned in following notifications from the Response Gateway
Rule_ID and *Rule_type* are the ID and the type of the rule to be killed

The response to the invoked method is sent via an http GET response. The response is XML coded, following the schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xs:element name="Engine_Response">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="result" type="xs:boolean"/>
                <xs:element name="errormsg" type="xs:string" nillable="true" minOccurs="0"/>
                <xs:element name="errorcode" type="xs:int"/>
                <xs:element name="ruleid" type="xs:string" minOccurs="0" maxOccurs="20"/>
                <xs:element name="status" type="xs:string" minOccurs="0"/>
                <xs:element name="xml_rule_schema" type="xs:string" minOccurs="0"/>
                <xs:element name="rulelog" type="xs:string" minOccurs="0" maxOccurs="100"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

20.1.16 Remove Rule method (Engine Channel)

GET{ServiceURI}? Credentials=”Credential string”&AXRQID=”Request ID string”&Rule_ID=”Rule ID string”&Rule_type=”rule type string”

Credentials contains user Credentials.
AXRQID is the unique identifier ot the Request, to be returned in following notifications from the Response Gateway
Rule_ID and *Rule_type* are the ID and the type of the rule to be removed

The response to the invoked method is sent via an http GET response. The response is XML coded, following the schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
	<xs:element name="Engine_Response">
		<xs:complexType>
			<xs:sequence>
				<xs:element name="result" type="xs:boolean"/>
				<xs:element name="errormsg" type="xs:string" nillable="true"
minOccurs="0"/>
				<xs:element name="errorcode" type="xs:int"/>
				<xs:element name="ruleid" type="xs:string" minOccurs="0"
maxOccurs="20"/>
				<xs:element name="status" type="xs:string" minOccurs="0"/>
				<xs:element name="xml_rule_schema" type="xs:string" minOccurs="0"/>
				<xs:element name="rulelog" type="xs:string" minOccurs="0"
maxOccurs="100"/>
			</xs:sequence>
		</xs:complexType>
	</xs:element>
</xs:schema>
```

20.1.17 Resume Rule method (Engine Channel)

GET{ServiceURI}? Credentials=”Credential string”&AXRQID=”Request ID string”&Rule_ID=”Rule ID string”&Rule_type=”rule type string”&URI=”EngineListenerService string”

Credentials contains user Credentials.
AXRQID is the unique identifier ot the Request, to be returned in following notifications from the Response Gateway
URI is the URI where the Response Adapter waits for notifications from the Response Gateway
Rule_ID and *Rule_type* are the ID and the type of the rule to be resumed

The response to the invoked method is sent via an http GET response. The response is XML coded, following the schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
	<xs:element name="Engine_Response">
		<xs:complexType>
			<xs:sequence>
				<xs:element name="result" type="xs:boolean"/>
				<xs:element name="errormsg" type="xs:string" nillable="true"
minOccurs="0"/>
				<xs:element name="errorcode" type="xs:int"/>
				<xs:element name="ruleid" type="xs:string" minOccurs="0"
maxOccurs="20"/>
				<xs:element name="status" type="xs:string" minOccurs="0"/>
				<xs:element name="xml_rule_schema" type="xs:string" minOccurs="0"/>
				<xs:element name="rulelog" type="xs:string" minOccurs="0"
maxOccurs="100"/>
			</xs:sequence>
		</xs:complexType>
```

```
        </xs:element>
</xs:schema>
```

20.1.18 Get Rule Status method (Engine Channel)

GET{ServiceURI}? Credentials=”Credential string”&AXRQID=”Request ID string”&Rule_ID=”Rule ID string”&Rule_type=”rule type string”

Credentials contains user Credentials.
AXRQID is the unique identifier ot the Request, to be returned in following notifications from the Response Gateway
Rule_ID and *Rule_type* are the ID and the type of the rule the status of which is returned

The response to the invoked method is sent via an http GET response. The response is XML coded, following the schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" attributeFormDefault="unqualified">
        <xs:element name="Engine_Response">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element name="result" type="xs:boolean"/>
                                <xs:element name="errormsg" type="xs:string" nillable="true" minOccurs="0"/>
                                <xs:element name="errorcode" type="xs:int"/>
                                <xs:element name="ruleid" type="xs:string" minOccurs="0" maxOccurs="20"/>
                                <xs:element name="status" type="xs:string" minOccurs="0"/>
                                <xs:element name="xml_rule_schema" type="xs:string" minOccurs="0"/>
                                <xs:element name="rulelog" type="xs:string" minOccurs="0" maxOccurs="100"/>
                        </xs:sequence>
                </xs:complexType>
        </xs:element>
</xs:schema>
```

20.1.19 Get Rule Logs method (Engine Channel)

GET{ServiceURI}? Credentials=”Credential string”&AXRQID=”Request ID string”&Rule_ID=”Rule ID string”&Rule_type=”rule type string”

Credentials contains user Credentials.
AXRQID is the unique identifier ot the Request, to be returned in following notifications from the Response Gateway
Rule_ID and *Rule_type* are the ID and the type of the rule the logs of which are returned

The response to the invoked method is sent via an http GET response. The response is XML coded, following the schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" attributeFormDefault="unqualified">
        <xs:element name="Engine_Response">
                <xs:complexType>
```

```
				<xs:sequence>
					<xs:element name="result" type="xs:boolean"/>
					<xs:element name="errormsg" type="xs:string" nillable="true" minOccurs="0"/>
					<xs:element name="errorcode" type="xs:int"/>
					<xs:element name="ruleid" type="xs:string" minOccurs="0" maxOccurs="20"/>
					<xs:element name="status" type="xs:string" minOccurs="0"/>
					<xs:element name="xml_rule_schema" type="xs:string" minOccurs="0"/>
					<xs:element name="rulelog" type="xs:string" minOccurs="0" maxOccurs="100"/>
				</xs:sequence>
			</xs:complexType>
		</xs:element>
</xs:schema>
```

20.1.20 Get List of Rules method (Engine Channel)

GET{ServiceURI}? Credentials=”Credential string”&AXRQID=”Request ID string”&Rule_type=”rule type string”

Credentials contains user Credentials.
AXRQID is the unique identifier ot the Request, to be returned in following notifications from the Response Gateway
Rule_type is the type of the rule to be returned

The response to the invoked method is sent via an http GET response. The response is XML coded, following the schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" attributeFormDefault="unqualified">
		<xs:element name="Engine_Response">
			<xs:complexType>
				<xs:sequence>
					<xs:element name="result" type="xs:boolean"/>
					<xs:element name="errormsg" type="xs:string" nillable="true" minOccurs="0"/>
					<xs:element name="errorcode" type="xs:int"/>
					<xs:element name="ruleid" type="xs:string" minOccurs="0" maxOccurs="20"/>
					<xs:element name="status" type="xs:string" minOccurs="0"/>
					<xs:element name="xml_rule_schema" type="xs:string" minOccurs="0"/>
					<xs:element name="rulelog" type="xs:string" minOccurs="0" maxOccurs="100"/>
				</xs:sequence>
			</xs:complexType>
		</xs:element>
</xs:schema>
```

20.1.21 Get Rule method (Engine Channel)

GET{ServiceURI}? Credentials=”Credential string”&AXRQID=”Request ID string”&Rule_ID=”Rule ID string”&Rule_type=”rule type string”

Credentials contains user Credentials.
AXRQID is the unique identifier ot the Request, to be returned in following notifications from the Response Gateway
Rule_ID and *Rule_type* are the ID and the type of the rule the schema of which is returned

The response to the invoked method is sent via an http GET response. The response is XML coded, following the schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" attributeFormDefault="unqualified">
	<xs:element name="Engine_Response">
		<xs:complexType>
			<xs:sequence>
				<xs:element name="result" type="xs:boolean"/>
				<xs:element name="errormsg" type="xs:string" nillable="true" minOccurs="0"/>
				<xs:element name="errorcode" type="xs:int"/>
				<xs:element name="ruleid" type="xs:string" minOccurs="0" maxOccurs="20"/>
				<xs:element name="status" type="xs:string" minOccurs="0"/>
				<xs:element name="base 64 encoded xml_rule_schema" type="xs:string" minOccurs="0"/>
				<xs:element name="rulelog" type="xs:string" minOccurs="0" maxOccurs="100"/>
			</xs:sequence>
		</xs:complexType>
	</xs:element>
</xs:schema>
```

20.1.22 Status Request to Program and Publication method (Engine Channel)

GET{ServiceURI}? Credentials=”Credential string”&AXRQID=”Request ID string”&ProgramID=”Program ID string”

Credentials contains user Credentials.
AXRQID is the unique identifier ot the Request, to be returned in following notifications from the Response Gateway
ProgramID is the ID of the Program whose status is to be returned

The response to the invoked method is sent via an http GET response. The response is XML coded, following the schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" attributeFormDefault="unqualified">
	<xs:element name="Engine_Response">
		<xs:complexType>
			<xs:sequence>
				<xs:element name="result" type="xs:boolean"/>
				<xs:element name="errormsg" type="xs:string" nillable="true" minOccurs="0"/>
```

```
                        <xs:element name="errorcode" type="xs:int"/>
                        <xs:element name="ruleid" type="xs:string" minOccurs="0"
maxOccurs="20"/>
                        <xs:element name="status" type="xs:string" minOccurs="0"/>
                        <xs:element name="xml_rule_schema" type="xs:string" minOccurs="0"/>
                        <xs:element name="rulelog" type="xs:string" minOccurs="0"
maxOccurs="100"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
</xs:schema>
```

20.1.23 Suspend Program and Publication method (Engine Channel)

GET{ServiceURI}? Credentials=”Credential string”&AXRQID=”Request ID string”&ProgramID=”Program ID string”

Credentials contains user Credentials.
AXRQID is the unique identifier ot the Request, to be returned in following notifications from the Response Gateway
ProgramID is the ID of the Program to be suspended

The response to the invoked method is sent via an http GET response. The response is XML coded, following the schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
      <xs:element name="Engine_Response">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element name="result" type="xs:boolean"/>
                        <xs:element name="errormsg" type="xs:string" nillable="true"
minOccurs="0"/>
                        <xs:element name="errorcode" type="xs:int"/>
                        <xs:element name="ruleid" type="xs:string" minOccurs="0"
maxOccurs="20"/>
                        <xs:element name="status" type="xs:string" minOccurs="0"/>
                        <xs:element name="xml_rule_schema" type="xs:string" minOccurs="0"/>
                        <xs:element name="rulelog" type="xs:string" minOccurs="0"
maxOccurs="100"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
</xs:schema>
```

20.1.24 Abort Program Publication method (Engine Channel)

GET{ServiceURI}? Credentials=”Credential string”&AXRQID=”Request ID string”&ProgramID=”Program ID string”

Credentials contains user Credentials.

AXRQID is the unique identifier ot the Request, to be returned in following notifications from the Response Gateway
ProgramID is the ID of the Program to be aborted

The response to the invoked method is sent via an http GET response. The response is XML coded, following the schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" attributeFormDefault="unqualified">
	<xs:element name="Engine_Response">
		<xs:complexType>
			<xs:sequence>
				<xs:element name="result" type="xs:boolean"/>
				<xs:element name="errormsg" type="xs:string" nillable="true" minOccurs="0"/>
				<xs:element name="errorcode" type="xs:int"/>
				<xs:element name="ruleid" type="xs:string" minOccurs="0" maxOccurs="20"/>
				<xs:element name="status" type="xs:string" minOccurs="0"/>
				<xs:element name="xml_rule_schema" type="xs:string" minOccurs="0"/>
				<xs:element name="rulelog" type="xs:string" minOccurs="0" maxOccurs="100"/>
			</xs:sequence>
		</xs:complexType>
	</xs:element>
</xs:schema>
```

20.1.25 Resume Program Publication method (Engine Channel)

GET{ServiceURI}? Credentials=”Credential string”&AXRQID=”Request ID string”&ProgramID=”Program ID string”&URI=”EngineListenerService string”

Credentials contains user Credentials.
AXRQID is the unique identifier ot the Request, to be returned in following notifications from the Response Gateway
ProgramID is the ID of the Program to be resumed
URI is the URI where the Response Adapter waits for notifications from the Response Gateway

The response to the invoked method is sent via an http GET response. The response is XML coded, following the schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" attributeFormDefault="unqualified">
	<xs:element name="Engine_Response">
		<xs:complexType>
			<xs:sequence>
				<xs:element name="result" type="xs:boolean"/>
				<xs:element name="errormsg" type="xs:string" nillable="true" minOccurs="0"/>
				<xs:element name="errorcode" type="xs:int"/>
				<xs:element name="ruleid" type="xs:string" minOccurs="0" maxOccurs="20"/>
				<xs:element name="status" type="xs:string" minOccurs="0"/>
```

```
                                        <xs:element name="xml_rule_schema" type="xs:string" minOccurs="0"/>
                                        <xs:element name="rulelog" type="xs:string" minOccurs="0"
maxOccurs="100"/>
                              </xs:sequence>
                    </xs:complexType>
          </xs:element>
</xs:schema>
```

20.1.26 Activate Program Publication method (Engine Channel)

GET{ServiceURI}? Credentials=”Credential string”&AXRQID=”Request ID string”&ProgramID=”Program ID string”&URI=”EngineListenerService string

Credentials contains user Credentials.
AXRQID is the unique identifier ot the Request, to be returned in following notifications from the Response Gateway
ProgramID is the ID of the Program to be activated
URI is the URI where the Response Adapter waits for notifications from the Response Gateway

The response to the invoked method is sent via an http GET response. The response is XML coded, following the schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
          <xs:element name="Engine_Response">
                    <xs:complexType>
                              <xs:sequence>
                                        <xs:element name="result" type="xs:boolean"/>
                                        <xs:element name="errormsg" type="xs:string" nillable="true"
minOccurs="0"/>
                                        <xs:element name="errorcode" type="xs:int"/>
                                        <xs:element name="ruleid" type="xs:string" minOccurs="0"
maxOccurs="20"/>
                                        <xs:element name="status" type="xs:string" minOccurs="0"/>
                                        <xs:element name="xml_rule_schema" type="xs:string" minOccurs="0"/>
                                        <xs:element name="rulelog" type="xs:string" minOccurs="0"
maxOccurs="100"/>
                              </xs:sequence>
                    </xs:complexType>
          </xs:element>
</xs:schema>
```

20.1.27 Workflow Notification method (Engine Channel)

GET{EngineListenerServiceURI}? Credentials=”Credential string”&AXRQID=”Request ID string”&Status=”Status string”

Credentials contains user Credentials.
AXRQID is the unique identifier ot the Request, to be returned in following notifications from the Response Gateway
Status is a string containing the Notification from the WorkFlow process

The response to the invoked method is sent via an http GET response. The response is XML coded, following the schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
	<xs:element name="Engine_Response">
		<xs:complexType>
			<xs:sequence>
				<xs:element name="result" type="xs:boolean"/>
				<xs:element name="errormsg" type="xs:string" nillable="true"
minOccurs="0"/>
				<xs:element name="errorcode" type="xs:int"/>
				<xs:element name="ruleid" type="xs:string" minOccurs="0"
maxOccurs="20"/>
				<xs:element name="status" type="xs:string" minOccurs="0"/>
				<xs:element name="xml_rule_schema" type="xs:string" minOccurs="0"/>
				<xs:element name="rulelog" type="xs:string" minOccurs="0"
maxOccurs="100"/>
			</xs:sequence>
		</xs:complexType>
	</xs:element>
</xs:schema>
```

20.1.28 Edit Query method (Database Channel)

GET{ServiceURI}?SelectionID=”Selection ID string”&Credentials=”Credential string”&AXRQID=”Request ID string” &URI=”QueryListenerService string

SelectionID is the ID of the selection to be edited.
(Note: SelectionID can be void while requesting editing a new selection)
Credentials contains user Credentials.
AXRQID is the unique identifier ot the Request, to be returned in following notifications from the Response Gateway
URI is the URI where the Response Adapter waits for notifications from the Response Gateway

The response to the invoked method is sent via an http GET response. The response is XML coded, following the schema:

```
<xs:element name="Query_Response">
	<xs:complexType>
		<xs:sequence>
			<xs:element name="result" type="xs:boolean"/>
			<xs:element name="errormsg" type="xs:string" nillable="true" minOccurs="0"/>
			<xs:element name="errorcode" type="xs:int” minOccurs=”0"/>
			<xs:element name="AXOID" type="xs:string" minOccurs="0" maxOccurs=
100”/>
			<xs:element name="SelectionID" type="xs:string" minOccurs="0"
maxOccurs="100"/>
			<xs:element name="version" type="xs:string” minOccurs=”0"/>
			<xs:element name="XML_Selection" type="xs:string” minOccurs=”0"/>
		</xs:sequence>
	</xs:complexType>
```

```
</xs:element>
```

20.1.29 Activate Selection sync method (Database Channel)

GET{ServiceURI}? Credentials=”Credential string” &AXRQID=”Request ID string” & SelectionID=”Selection ID string”

SelectionID is the ID of the selection to be activated.
Credentials contains user Credentials.
AXRQID is the unique identifier ot the Request, to be returned in following notifications from the Response Gateway

The response to the invoked method is sent via an http GET response. The response is XML coded, following the schema:

```
<xs:element name="Query_Response">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="result" type="xs:boolean"/>
                        <xs:element name="errormsg" type="xs:string" nillable="true" minOccurs="0"/>
                        <xs:element name="errorcode" type="xs:int” minOccurs=”0"/>
                        <xs:element name="AXOID" type="xs:string" minOccurs="0" maxOccurs=
100”/>
                        <xs:element name="SelectionID" type="xs:string" minOccurs="0"
maxOccurs="100"/>
                        <xs:element name="version" type="xs:string” minOccurs=”0"/>
                        <xs:element name="XML_Selection" type="xs:string” minOccurs=”0"/>
                </xs:sequence>
        </xs:complexType>
</xs:element>
```

20.1.30 Activate Selection async method (Database Channel)

GET{ServiceURI}? Credentials=”Credential string” &AXRQID=”Request ID string” & SelectionID=”Selection ID string” &URI=”ListenerURI string”

SelectionID is the ID of the selection to be activated.
Credentials contains user Credentials.
AXRQID is the unique identifier ot the Request, to be returned in following notifications from the Response Gateway
URI is the URI where the Response Adapter waits for notifications from the Response Gateway

The response to the invoked method is sent via an http GET response. The response is XML coded, following the schema:

```
<xs:element name="Query_Response">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="result" type="xs:boolean"/>
                        <xs:element name="errormsg" type="xs:string" nillable="true" minOccurs="0"/>
                        <xs:element name="errorcode" type="xs:int” minOccurs=”0"/>
                        <xs:element name="AXOID" type="xs:string" minOccurs="0" maxOccurs=
100”/>
```

```
                    <xs:element name="SelectionID" type="xs:string" minOccurs="0"
maxOccurs="100"/>
                    <xs:element name="version" type="xs:string” minOccurs=”0"/>
                    <xs:element name="XML_Selection" type="xs:string” minOccurs=”0"/>
            </xs:sequence>
        </xs:complexType>
</xs:element>
```

20.1.31 Delete Selection method (Database Channel)

GET{ServiceURI}? Credentials=”Credential string” &AXRQID=”Request ID string” & SelectionID=”Selection ID string”

SelectionID is the ID of the selection to be deleted.
Credentials contains user Credentials.
AXRQID is the unique identifier ot the Request, to be returned in following notifications from the Response Gateway

The response to the invoked method is sent via an http GET response. The response is XML coded, following the schema:

```
<xs:element name="Query_Response">
        <xs:complexType>
            <xs:sequence>
                    <xs:element name="result" type="xs:boolean"/>
                    <xs:element name="errormsg" type="xs:string" nillable="true" minOccurs="0"/>
                    <xs:element name="errorcode" type="xs:int” minOccurs=”0"/>
                    <xs:element name="AXOID" type="xs:string" minOccurs="0" maxOccurs=
100”/>
                    <xs:element name="SelectionID" type="xs:string" minOccurs="0"
maxOccurs="100"/>
                    <xs:element name="version" type="xs:string” minOccurs=”0"/>
                    <xs:element name="XML_Selection" type="xs:string” minOccurs=”0"/>
            </xs:sequence>
        </xs:complexType>
</xs:element>
```

20.1.32 Load Selection method (Database Channel)

GET{ServiceURI}? Credentials=”Credential string” &AXRQID=”Request ID string” & SelectionID=”Selection ID string”

SelectionID is the ID of the selection to be loaded.
Credentials contains user Credentials.
AXRQID is the unique identifier ot the Request, to be returned in following notifications from the Response Gateway

The response to the invoked method is sent via an http GET response. The response is XML coded, following the schema:

```
<xs:element name="Query_Response">
        <xs:complexType>
            <xs:sequence>
                    <xs:element name="result" type="xs:boolean"/>
```

```
                        <xs:element name="errormsg" type="xs:string" nillable="true" minOccurs="0"/>
                        <xs:element name="errorcode" type="xs:int” minOccurs=”0"/>
                        <xs:element name="AXOID" type="xs:string" minOccurs="0" maxOccurs=
100”/>
                        <xs:element name="SelectionID" type="xs:string" minOccurs="0"
maxOccurs="100"/>
                        <xs:element name="version" type="xs:string” minOccurs=”0"/>
                        <xs:element name="XML_Selection" type="xs:string” minOccurs=”0"/>
                </xs:sequence>
        </xs:complexType>
</xs:element>
```

XML_Selection is base64 encoded containing the Selection XML

20.1.33 Save Selection method (Database Channel)

POST{ServiceURI}

The Request Header contains:
Credentials=”Credential string”
Axrqid=”Request ID string
Selectionid=”SelectionID string”

And the Request body contains the XML_Selection in xml/text format

SelectionID is the ID of the selection to be loaded.
Credentials contains user Credentials.
AXRQID is the unique identifier ot the Request, to be returned in following notifications from the Response Gateway

The response to the invoked method is sent via an http POST response. The response is XML coded, following the schema:

```
<xs:element name="Query_Response">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="result" type="xs:boolean"/>
                        <xs:element name="errormsg" type="xs:string" nillable="true" minOccurs="0"/>
                        <xs:element name="errorcode" type="xs:int” minOccurs=”0"/>
                        <xs:element name="AXOID" type="xs:string" minOccurs="0" maxOccurs=
100”/>
                        <xs:element name="SelectionID" type="xs:string" minOccurs="0"
maxOccurs="100"/>
                        <xs:element name="version" type="xs:string” minOccurs=”0"/>
                        <xs:element name="XML_Selection" type="xs:string” minOccurs=”0"/>
                </xs:sequence>
        </xs:complexType>
</xs:element>
```

20.1.34 List User Selections method (Database Channel)

GET{ServiceURI}? Credentials=”Credential string” &AXRQID=”Request ID string”

Credentials contains user Credentials.
AXRQID is the unique identifier ot the Request, to be returned in following notifications from the Response Gateway

The response to the invoked method is sent via an http GET response. The response is XML coded, following the schema:

```
<xs:element name="Query_Response">
	<xs:complexType>
		<xs:sequence>
			<xs:element name="result" type="xs:boolean"/>
			<xs:element name="errormsg" type="xs:string" nillable="true" minOccurs="0"/>
			<xs:element name="errorcode" type="xs:int" minOccurs="0"/>
			<xs:element name="AXOID" type="xs:string" minOccurs="0" maxOccurs=
100"/>
			<xs:element name="SelectionID" type="xs:string" minOccurs="0"
maxOccurs="100"/>
			<xs:element name="version" type="xs:string" minOccurs="0"/>
			<xs:element name="XML_Selection" type="xs:string" minOccurs="0"/>
		</xs:sequence>
	</xs:complexType>
</xs:element>
```

20.1.35 List Entitled Selections method (Database Channel)

GET{ServiceURI}? Credentials="Credential string" &AXRQID="Request ID string"

Credentials contains user Credentials.
AXRQID is the unique identifier ot the Request, to be returned in following notifications from the Response Gateway

The response to the invoked method is sent via an http GET response. The response is XML coded, following the schema:

```
<xs:element name="Query_Response">
	<xs:complexType>
		<xs:sequence>
			<xs:element name="result" type="xs:boolean"/>
			<xs:element name="errormsg" type="xs:string" nillable="true" minOccurs="0"/>
			<xs:element name="errorcode" type="xs:int" minOccurs="0"/>
			<xs:element name="AXOID" type="xs:string" minOccurs="0" maxOccurs=
100"/>
			<xs:element name="SelectionID" type="xs:string" minOccurs="0"
maxOccurs="100"/>
			<xs:element name="version" type="xs:string" minOccurs="0"/>
			<xs:element name="XML_Selection" type="xs:string" minOccurs="0"/>
		</xs:sequence>
	</xs:complexType>
</xs:element>
```

20.1.36 Check-out sync method (Database Channel)

GET{ServiceURI}? Credentials="Credential string" &AXRQID="Request ID string" & AXOID="AXOID string of the object to be retrieved" &version="Version string of the AXOID to be retrieved"

Credentials contains user Credentials.
AXRQID is the unique identifier ot the Request, to be returned in following notifications from the Response Gateway
AXOID is the object ID to be retrieved
Version is the version of the AXOID to be retrieved

The response to the invoked method is sent via an http GET response. The response is XML coded, following the schema:

```
<xs:element name="Query_Response">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="result" type="xs:boolean"/>
                        <xs:element name="errormsg" type="xs:string" nillable="true" minOccurs="0"/>
                        <xs:element name="errorcode" type="xs:int” minOccurs=”0"/>
                        <xs:element name="AXOID" type="xs:string" minOccurs="0" maxOccurs=
100”/>
                        <xs:element name="SelectionID" type="xs:string" minOccurs="0"
maxOccurs="100"/>
                        <xs:element name="version" type="xs:string” minOccurs=”0"/>
                        <xs:element name="XML_Selection" type="xs:string” minOccurs=”0"/>
                        <xs:element name="downloadURI" type="xs:string” minOccurs=”0"/>
                </xs:sequence>
        </xs:complexType>
</xs:element>
```

20.1.37 Check-out async method (Database Channel)

GET{ServiceURI}? Credentials=”Credential string” &AXRQID=”Request ID string” & AXOID=”AXOID string of the object to be retrieved” &version=”Version string of the AXOID to be retrieved” &URI=”ListenerURI string”

Credentials contains user Credentials.
AXRQID is the unique identifier ot the Request, to be returned in following notifications from the Response Gateway
AXOID is the object ID to be retrieved
Version is the version of the AXOID to be retrieved
URI is the URI where the Response Adapter waits for notifications from the Response Gateway

The response to the invoked method is sent via an http GET response. The response is XML coded, following the schema:

```
<xs:element name="Query_Response">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="result" type="xs:boolean"/>
                        <xs:element name="errormsg" type="xs:string" nillable="true" minOccurs="0"/>
                        <xs:element name="errorcode" type="xs:int” minOccurs=”0"/>
                        <xs:element name="AXOID" type="xs:string" minOccurs="0" maxOccurs=
100”/>
                        <xs:element name="SelectionID" type="xs:string" minOccurs="0"
maxOccurs="100"/>
                        <xs:element name="version" type="xs:string” minOccurs=”0"/>
                        <xs:element name="XML_Selection" type="xs:string” minOccurs=”0"/>
                        <xs:element name="downloadURI" type="xs:string” minOccurs=”0"/>
                </xs:sequence>
        </xs:complexType>
</xs:element>
```

20.1.38 Commit sync method (Database Channel)

GET{ServiceURI}? Credentials=”Credential string” &AXRQID=”Request ID string” & AXOID=”AXOID string of the object to be retrieved” &downloadURI=”URI where file is put for uploading”

Credentials contains user Credentials.
AXRQID is the unique identifier ot the Request, to be returned in following notifications from the Response Gateway
AXOID is the object ID to be loaded
downloadURI is the URI where the AXOID to be loaded in AXMEDIS DB is put

The response to the invoked method is sent via an http GET response. The response is XML coded, following the schema:

```
<xs:element name="Query_Response">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="result" type="xs:boolean"/>
                        <xs:element name="errormsg" type="xs:string" nillable="true" minOccurs="0"/>
                        <xs:element name="errorcode" type="xs:int” minOccurs=”0"/>
                        <xs:element name="AXOID" type="xs:string" minOccurs="0" maxOccurs=
100”/>
                        <xs:element name="SelectionID" type="xs:string" minOccurs="0"
maxOccurs="100"/>
                        <xs:element name="version" type="xs:string” minOccurs=”0"/>
                        <xs:element name="XML_Selection" type="xs:string” minOccurs=”0"/>
                        <xs:element name="downloadURI" type="xs:string” minOccurs=”0"/>
                </xs:sequence>
        </xs:complexType>
</xs:element>
```

20.1.39 Commit async method (Database Channel)

GET{ServiceURI}? Credentials=”Credential string” &AXRQID=”Request ID string” & AXOID=”AXOID string of the object to be retrieved” &downloadURI=”URI where file is put for uploading” &URI=”ListenerURI string

Credentials contains user Credentials.
AXRQID is the unique identifier ot the Request, to be returned in following notifications from the Response Gateway
AXOID is the object ID to be loaded
downloadURI is the URI where the AXOID to be loaded in AXMEDIS DB is put
URI is the URI where the Response Adapter waits for notifications from the Response Gateway

The response to the invoked method is sent via an http GET response. The response is XML coded, following the schema:

```
<xs:element name="Query_Response">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="result" type="xs:boolean"/>
                        <xs:element name="errormsg" type="xs:string" nillable="true" minOccurs="0"/>
                        <xs:element name="errorcode" type="xs:int" minOccurs="0"/>
                        <xs:element   name="AXOID"   type="xs:string"   minOccurs="0"   maxOccurs=
100"/>
                        <xs:element      name="SelectionID"      type="xs:string"      minOccurs="0"
maxOccurs="100"/>
                        <xs:element name="version" type="xs:string" minOccurs="0"/>
                         <xs:element name="XML_Selection" type="xs:string" minOccurs="0"/>
                         <xs:element name="downloadURI" type="xs:string" minOccurs="0"/>
                </xs:sequence>
        </xs:complexType>
</xs:element>
```

20.1.40 Lock Object method (Database Channel)

GET{ServiceURI}? Credentials=”Credential string” &AXRQID=”Request ID string” & AXOID=”AXOID string” & version=”version string”

Credentials contains user Credentials.
AXRQID is the unique identifier ot the Request, to be returned in following notifications from the Response Gateway
AXOID is the object ID to be locked
Version is the version of the AXOID to be locked

The response to the invoked method is sent via an http GET response. The response is XML coded, following the schema:

```
<xs:element name="Query_Response">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="result" type="xs:boolean"/>
                        <xs:element name="errormsg" type="xs:string" nillable="true" minOccurs="0"/>
                        <xs:element name="errorcode" type="xs:int" minOccurs="0"/>
                        <xs:element   name="AXOID"   type="xs:string"   minOccurs="0"   maxOccurs=
100"/>
                        <xs:element      name="SelectionID"      type="xs:string"      minOccurs="0"
maxOccurs="100"/>
                        <xs:element name="version" type="xs:string" minOccurs="0"/>
                        <xs:element name="XML_Selection" type="xs:string" minOccurs="0"/>
                </xs:sequence>
        </xs:complexType>
</xs:element>
```

20.1.41 unlock Object method (Database Channel)

GET{ServiceURI}? Credentials=”Credential string” &AXRQID=”Request ID string” & AXOID=”AXOID string” & version=”version string”

Credentials contains user Credentials.

AXRQID is the unique identifier ot the Request, to be returned in following notifications from the Response Gateway
AXOID is the object ID to be unlocked
Version is the version of the AXOID to be unlocked

The response to the invoked method is sent via an http GET response. The response is XML coded, following the schema:

```
<xs:element name="Query_Response">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="result" type="xs:boolean"/>
                        <xs:element name="errormsg" type="xs:string" nillable="true" minOccurs="0"/>
                        <xs:element name="errorcode" type="xs:int” minOccurs=”0"/>
                        <xs:element name="AXOID" type="xs:string" minOccurs="0" maxOccurs=
100”/>
                        <xs:element name="SelectionID" type="xs:string" minOccurs="0"
maxOccurs="100"/>
                        <xs:element name="version" type="xs:string” minOccurs=”0"/>
                        <xs:element name="XML_Selection" type="xs:string” minOccurs=”0"/>
                </xs:sequence>
        </xs:complexType>
</xs:element>
```

20.2 xmlrpc Communication Protocol for methods from Response Gateways and Response Adapters (axWFRes)

20.2.1 Editor_Notification method (Editor Channel)

XML-RPC invocation contains:

```
<?xml version='1.0'?>
<methodCall>
        <methodName>Editor_Notification</methodName>
        <params>
                <param>
                        <value><string>result</string></value>
                </param>
                <param>
                        <value><string>errormsg</string></value>
                </param>
                <param>
                        <value><string>errorcode</string></value>
                </param>
                <param>
                        <value><string>AXRQID</string></value>
                </param>
                <param>
                        <value><string>AXOID</string></value>
                </param>
        </params>
</methodCall>
```

Result is boolean: *true* or *false*.
Note: AXOID is returned if editor created a new object, otherwise must be set to a null string

AXRQID is the unique identifier of the request previously sent by the Request Adapter whose the notification refers.

The WF Response Adapter will give the following XML response to the invoking entity, if successfully received:

```
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value><string>OK</string></value>
      </param>
    </params>
  </methodResponse>
```

Otherwise, if the Notification cannot be received:

```
<?xml version="1.0"?>
<methodResponse>
  <fault>
    <value>
      <struct>
        <member>
          <name>faultCode</name>
          <value><int>-errorcode</int></value>
          </member>
        <member>
          <name>faultString</name>
          <value><string>"Error string".</string></value>
          </member>
        </struct>
      </value>
    </fault>
  </methodResponse>
```

20.2.2 get_workflow_information method (Editor Channel)

XML-RPC containing:

```
<?xml version='1.0'?>
<methodCall>
        <methodName>get_workflow_information</methodName>
        <params>
                <param>
                        <value><string>AXOID</string></value>
                </param
        </params>
</methodCall>
```

AXOID is the Object ID whose workflow information is required.

```
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
        <value><array><data>
                <value><string>AXRQID</string></value>
```

```
                        <value><string>Title</string></value>
                      <value><string>Process_name</string></value>
                      <value><string>Activity_name</string></value>
                      <value><string>status</string></value>
                      <value><string>priority</string></value>
                      <value><string>actor</string></value>
            </data></array></value>
      </param>
    </params>
   </methodResponse>
```

Otherwise, if the Notification cannot be received:

```
<?xml version="1.0"?>
<methodResponse>
   <fault>
      <value>
         <struct>
            <member>
               <name>faultCode</name>
               <value><int>-errorcode</int></value>
               </member>
            <member>
               <name>faultString</name>
               <value><string>"Error string".</string></value>
               </member>
            </struct>
         </value>
      </fault>
   </methodResponse>
```

Where:

AXRQID:	string containing the workflow instance_ID related to the requested AXOID
Title:	the description of the process instance
Process_name:	the name of the process instanciated by the instance
Activity_name:	the name of the activity of the process being performed by the last workitem of the instance
Status:	string containing the status of the last workitem of the process instance
Priority:	string containing the priority of the last workitem of the process instance
Actor:	string containing the actor to whom the last activity (workitem) is currently assigned

20.2.3 Rule_Editor_Notification method (Rule Editor Channel)

XML-RPC containing:

```
<?xml version='1.0'?>
<methodCall>
            <methodName>Rule_editor_Notification</methodName>
            <params>
                        <param>
                                    <value><string>result</string></value>
                        </param>
                       <param>
                                    <value><string>errormsg</string></value>
                        </param>
                        <param>
                                    <value><string>errorcode</string></value>
                        </param>
                        <param>
                                    <value><string>AXRQID</string></value>
                        </param>
                        <param>
                                    <value><string>base64 encoded XML_Rule_Schema</string></value>
                        </param>
            </params>
</methodCall>
```

Result is boolean: *true* or *false*.
AXRQID is the unique identifier of the request previously sent by the Request Adapter whose the notification refers.
XML Rule Schema is the modified XML schema returned, base64 encoded

The WF Response Adapter will give the following XML response to the invoking entity, if successfully received:

```
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value><string>OK</string></value>
      </param>
    </params>
  </methodResponse>
```

Otherwise, if the Notification cannot be received:

```
<?xml version="1.0"?>
<methodResponse>
  <fault>
    <value>
      <struct>
        <member>
          <name>faultCode</name>
          <value><int>-errorcode</int></value>
          </member>
        <member>
          <name>faultString</name>
          <value><string>"Error string".</string></value>
          </member>
        </struct>
      </value>
    </fault>
  </methodResponse>
```

20.2.4 Engine_Notification method (Engine Channel)

XML-RPC containing:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
```

```
        <xs:element name="Engine_Notification">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element name="result" type="xs:boolean"/>
                                <xs:element      name="errormsg"      type="xs:string"      nillable="true"
minOccurs="0"/>
                                <xs:element name="errorcode" type="xs:int"/>
                                <xs:element name="AXRQID" type="xs:string"/>
                                <xs:element name="status" type="xs:string"/>
                        </xs:sequence>
                </xs:complexType>
        </xs:element>
</xs:schema>
```

Result is boolean: *true* or *false*.
AXRQID is the unique identifier of the request previously sent by the Request Adapter whose the notification refers
Errorcode and errormsg contains the eventual error code and message returned by the Engine
Status contains the status string returned by the Engine.

The WF Response Adapter will give the following XML response to the invoking entity, if successfully received:

```
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value><string>OK</string></value>
      </param>
    </params>
  </methodResponse>
```

Otherwise, if the Notification cannot be received:

```
<?xml version="1.0"?>
<methodResponse>
  <fault>
    <value>
      <struct>
        <member>
          <name>faultCode</name>
          <value><int>-errorcode</int></value>
          </member>
        <member>
          <name>faultString</name>
          <value><string>"Error string".</string></value>
          </member>
        </struct>
      </value>
    </fault>
  </methodResponse>
```

20.2.5 Workflow_Process_Request method (Engine Channel)

XML-RPC containing:

```
<?xml version='1.0'?>
<methodCall>
        <methodName>Workflow_Process_Request</methodName>
        <params>
                <param>
                        <value><string>AXRQID</string></value>
                </param>
                <param>
                        <value><string>processid</string></value>
                </param>
                <param>
                        <value><string>EngineLinstenerService</string></value>
                </param>
        </params>
</methodCall>
```

AXRQID is the unique identifier of the request sent by the Response Adapter, in order to identify the following workflow notifications that the Workflow engine will send back to the invoking engine.
Processed is the workflow process name to be started.
EngineListenerServices is the URI where the workflow will send the following notifications.

The WF Response Adapter will give the following XML response to the invoking entity, if successfully received:

```
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value><string>OK</string></value>
      </param>
    </params>
  </methodResponse>
```

Otherwise, if the Notification cannot be received:

```
<?xml version="1.0"?>
<methodResponse>
  <fault>
    <value>
      <struct>
        <member>
          <name>faultCode</name>
          <value><int>-errorcode</int></value>
          </member>
        <member>
          <name>faultString</name>
          <value><string>"Error string".</string></value>
          </member>
        </struct>
      </value>
    </fault>
  </methodResponse>
```

20.2.6 Query_Notification method (Database Channel)

XML-RPC invocation contains:

```
<?xml version='1.0'?>
<methodCall>
        <methodName>Query_Notification</methodName>
        <params>
                <param>
                        <value><string>result</string></value>
                </param>
                <param>
                        <value><string>errormsg</string></value>
                </param>
                <param>
                        <value><string>errorcode</string></value>
                </param>
                <param>
                        <value><string>AXRQID</string></value>
                </param>
                <param>
                        <value><string>version</string></value>
                </param>
                <param>
                        <value>
                                <array>
                                        <data>
                                                <value><string>AXOID1</string></value>
                                                <value><string>AXOID2</string></value >
                                                ……
                                        </data>
                                </array>
                        </value>
                </param>
                <param>
                        <value><string>downloadURI</string></value>
                </param>
        </params>
</methodCall>
```

Result is boolean: *true* or *false*.
AXRQID is the unique identifier of the request previously sent by the Request Adapter whose the notification refers
Errorcode and errormsg contains the eventual error code and message returned by the Engine
AXOIDs are returned as an array when the Activation Async Selection was invoked (list of selected objects)
version is returned when the Commit Async was invoked

The WF Response Adapter will give the following XML response to the invoking entity, if successfully received:

```
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value><string>OK</string></value>
      </param>
    </params>
  </methodResponse>
```

Otherwise, if the Notification cannot be received:

```
<?xml version="1.0"?>
<methodResponse>
  <fault>
    <value>
      <struct>
        <member>
          <name>faultCode</name>
          <value><int>-errorcode</int></value>
          </member>
        <member>
          <name>faultString</name>
          <value><string>"Error string".</string></value>
          </member>
        </struct>
      </value>
    </fault>
  </methodResponse>
```

21 Formal description of communication protocol Workflow Editor Request Channel (axWfEditorReq) Webservice

Call name	
Method	bool editObject();
Description	This method is invoked through a webservice call coming from Workflow Request Gateway. This method will load the specified object (AXOID) and will allow the user to edit the object in the AXMEDIS Editor.
Input parameters	char *AXOID, //The object ID, NULL if creating new object char *userCredentials, //users permissions char *executionParameters, //parameters for execution char *editingType, //type of editing char *AXRQID, //request ID to be used for notification purpose char *URI//URI to use for notification
Output parameters	bool

Request Sample Message	`<message name="EditObjectRequest">`
	`<part name="AXOID" element="s0:AXOID"/>`
	`<part name="AXRQID" element="s0:AXRQID"/>`
	`<part name="Credentials" element="s0:UserCredentials"/>`
	`<part name="EditingType" element="s0:EditingType"/>`
	`<part name="ExecutionParameters" element="s0:ExecutionParameters"/>`
	`<part name="URI" element="s0:URI"/>`
	`</message>`

Response Sample Message	`<xs:element name="Result">`
	`<xs:complexType>`
	`<xs:sequence>`
	`<xs:element name="Result" type="xs:boolean"/>`
	`<xs:element name="ErrorCode" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ErrorMSG" type="xs:string" minOccurs="0"/>`
	`<xs:element name="Attributes" type="xs:string" minOccurs="0"/>`
	`</xs:sequence>`
	`</xs:complexType>`
	`</xs:element>`

Call name	
Method	bool viewObjectAttribute();
Description	This method is invoked through a webservice call coming from Workflow Request Gateway. This method will allow the workflow engine to retrive the object attributes for the object specified by the AXOID.
Input parameters	char* AXOID, //Object ID char* userCredentials, //Users permissions char* attributeList, //Comma Seperated Attribute IDs

	char* AXRQID//request ID to be used for notification purpose
Output parameters	bool

Request Sample Message	`<message name="ViewObjectAttributeRequest">`
	`<part name="AttributeList" element="s0:AttributeList"/>`
	`<part name="AXRQID" element="s0:AXRQID"/>`
	`<part name="Credentials" element="s0:UserCredentials"/>`
	`<part name="AXOID" element="s0:AXOID"/>`
	`</message>`

Response Sample Message	`<xs:element name="Result">`
	`<xs:complexType>`
	`<xs:sequence>`
	`<xs:element name="Result" type="xs:boolean"/>`
	`<xs:element name="ErrorCode" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ErrorMSG" type="xs:string" minOccurs="0"/>`
	`<xs:element name="Attributes" type="xs:string" minOccurs="0"/>`
	`</xs:sequence>`
	`</xs:complexType>`
	`</xs:element>`

Call name	
Method	bool addHistoryInfo();
Description	This method is invoked through a webservice call coming from Workflow Request Gateway. This method will allow the workflow engine to add the object history for the object specified by the AXOID.
Input parameters	char* AXOID, //Object ID char* userCredentials, //Users permissions char* historyInfo, //history log info string char* AXRQID//request ID to be used for notification purpose
Output parameters	bool

Request Sample Message	`<message name="AddHistoryInfoRequest">`
	`<part name="HistoryInfo" element="s0:HistoryInfo"/>`
	`<part name="AXRQID" element="s0:AXRQID"/>`
	`<part name="Credentials" element="s0:UserCredentials"/>`
	`<part name="AXOID" element="s0:AXOID"/>`
	`</message>`

Response Sample Message	`<xs:element name="Result">`
	`<xs:complexType>`
	`<xs:sequence>`
	`<xs:element name="Result" type="xs:boolean"/>`
	`<xs:element name="ErrorCode" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ErrorMSG" type="xs:string" minOccurs="0"/>`
	`<xs:element name="Attributes" type="xs:string" minOccurs="0"/>`
	`</xs:sequence>`
	`</xs:complexType>`
	`</xs:element>`

Workflow Editor Request Channel

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns="http://www.axmedis.org/WF/Editor"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
targetNamespace="http://www.axmedis.org/WF/Editor"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:types>
    <s:schema xmlns:s0="http://www.axmedis.org/WF/Editor"
targetNamespace="http://www.axmedis.org/WF/Editor">
      <s:element name="Result">
        <s:complexType>
          <s:sequence>
            <s:element name="Result" type="s:boolean" />
            <s:element minOccurs="0" name="ErrorCode" type="s:string" />
            <s:element minOccurs="0" name="ErrorMSG" type="s:string" />
            <s:element minOccurs="0" name="Attributes" type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="editRequest">
        <s:complexType>
          <s:sequence>
            <s:element name="AXOID" type="s:string" />
            <s:element name="AXRQID" type="s:string" />
            <s:element name="UserCredentials" type="s:string" />
            <s:element name="EditingType" type="s:string" />
            <s:element name="ExecutionParameters" type="s:string" />
            <s:element name="URI" type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="viewAttributeRequest">
        <s:complexType>
          <s:sequence>
            <s:element name="AttributeList" type="s:string" />
            <s:element name="AXRQID" type="s:string" />
            <s:element name="UserCredentials" type="s:string" />
            <s:element name="AXOID" type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="addHistoryRequest">
        <s:complexType>
          <s:sequence>
            <s:element name="HistoryInfo" type="s:string" />
            <s:element name="AXRQID" type="s:string" />
            <s:element name="UserCredentials" type="s:string" />
            <s:element name="AXOID" type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
    </s:schema>
  </wsdl:types>
  <wsdl:message name="EditObjectRequest">
    <wsdl:part name="editRequest" element="tns:editRequest" />
  </wsdl:message>
  <wsdl:message name="EditObjectResponse">
    <wsdl:part name="Result" element="tns:Result" />
  </wsdl:message>
  <wsdl:message name="ViewObjectAttributeRequest">
    <wsdl:part name="viewAttributeRequest" element="tns:viewAttributeRequest" />
  </wsdl:message>
  <wsdl:message name="ViewObjectAttributeResponse">
    <wsdl:part name="Result" element="tns:Result" />
  </wsdl:message>
  <wsdl:message name="AddHistoryInfoRequest">
    <wsdl:part name="addHistoryRequest" element="tns:addHistoryRequest" />
  </wsdl:message>
  <wsdl:message name="AddHistoryInfoResponse">
    <wsdl:part name="Result" element="tns:Result" />
  </wsdl:message>
  <wsdl:portType name="EditorSOAP">
    <wsdl:operation name="EditObject">
      <wsdl:input message="tns:EditObjectRequest" />
      <wsdl:output message="tns:EditObjectResponse" />
    </wsdl:operation>
    <wsdl:operation name="ViewObjectAttribute">
      <wsdl:input message="tns:ViewObjectAttributeRequest" />
      <wsdl:output message="tns:ViewObjectAttributeResponse" />
    </wsdl:operation>
    <wsdl:operation name="AddHistoryInfo">
      <wsdl:input message="tns:AddHistoryInfoRequest" />
      <wsdl:output message="tns:AddHistoryInfoResponse" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="EditorSOAP" type="tns:EditorSOAP">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document" />
    <wsdl:operation name="EditObject">
      <soap:operation soapAction="#EditObject" style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="ViewObjectAttribute">
      <soap:operation soapAction="#ViewObjectAttribute" style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="AddHistoryInfo">
      <soap:operation soapAction="#AddHistoryInfo" 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="WFEditorService">
    <wsdl:port name="EditorSOAP" binding="tns:EditorSOAP">
      <soap:address
location="http://www.axmedis.org/WF/Editor/EditorChannel.cgi" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

22 Formal description of communication protocol Workflow Editor Response Channel (axWfEditorRes) Webservice

Call name	
Method	bool notifyCompletion();
Description	This method is invoked by the editor to send back the notification towards workflow engine for the completion of previously issues asynchronous request.
Input parameters	char* AXRQID, //The original request ID char* URI, //URI to be used for sending notifications char* endpURL, //URI for the response gateway bool result=true, //The result of operation true = successful or false = failure char* errorCode="NULL", //Error Code in case of failure char* errorMSG="NULL", //Error MSG in case of failure char* AXOID="NULL", // Not NULL if new object created
Output parameters	bool

Request Sample Message	`<message name="EditorNotificationRequest">`
	`<part name="AXRQID" element="s0:AXRQID"/>`
	`<part name="URI" element="s0:URI"/>`
	`<part name="Result" element="s0:Result"/>`
	`<part name="ErrorMsg" element="s0:ErrorMSG"/>`
	`<part name="ErrorCode" element="s0:ErrorCode"/>`
	`<part name="AXOID" element="s0:AXOID"/>`
	`</message>`

Response Sample Message	`<xs:element name="MethodResponse">`
	`<xs:complexType>`
	`<xs:sequence>`
	`<xs:element name="Result" type="xs:boolean"/>`
	`<xs:element name="FaultCode" type="xs:string" minOccurs="0"/>`
	`<xs:element name="FaultString" type="xs:string" minOccurs="0"/>`
	`<xs:element name="AXRQID" type="xs:string" minOccurs="0"/>`
	`<xs:element name="Title" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ProcessName" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ActivityName" type="xs:string" minOccurs="0"/>`
	`<xs:element name="Status" type="xs:string" minOccurs="0"/>`
	`<xs:element name="Priority" type="xs:string" minOccurs="0"/>`
	`<xs:element name="Actor" type="xs:string" minOccurs="0"/>`
	`</xs:sequence>`

	`</xs:complexType>` `</xs:element>`

Call name	
Method	bool getWFInfo();
Description	This method is invoked by the application to request workflow information from workflow engine.
Input parameters	char* AXOID, char* endpURL,bool *Result, char *FaultCode,char *FaultString, char *AXRQID, char *Title, char *ProcessName, char *ActivityName, char *Status, char *Priority, char *Actor The only input parameter is AXOID, others are the return parameters whose values are obtained from workflow's response gateway
Output parameters	bool
Request Sample Message	`<message name="GetWFInfoRequest">` `<part name="AXOID" element="s0:AXOID"/>` `</message>`
Response Sample Message	`<xs:element name="MethodResponse">` `<xs:complexType>` `<xs:sequence>` `<xs:element name="Result" type="xs:boolean"/>` `<xs:element name="FaultCode" type="xs:string" minOccurs="0"/>` `<xs:element name="FaultString" type="xs:string" minOccurs="0"/>` `<xs:element name="AXRQID" type="xs:string" minOccurs="0"/>` `<xs:element name="Title" type="xs:string" minOccurs="0"/>` `<xs:element name="ProcessName" type="xs:string" minOccurs="0"/>` `<xs:element name="ActivityName" type="xs:string" minOccurs="0"/>` `<xs:element name="Status" type="xs:string" minOccurs="0"/>` `<xs:element name="Priority" type="xs:string" minOccurs="0"/>` `<xs:element name="Actor" type="xs:string" minOccurs="0"/>` `</xs:sequence>` `</xs:complexType>` `</xs:element>`

Workflow Editor Response Channel

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:tns="http://www.axmedis.org/WF/Editor/"
```

```
xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
targetNamespace="http://www.axmedis.org/WF/Editor/">
      <wsdl:types>
            <s:schema elementFormDefault="unqualified"
targetNamespace="http://www.axmedis.org/WF/Editor/">
                  <s:element name="GetWFInfo">
                        <s:complexType>
                              <s:sequence>
                                    <s:element minOccurs="0" maxOccurs="1"
name="AXOID" type="s:string"/>
                              </s:sequence>
                        </s:complexType>
                  </s:element>
                  <s:element name="GetWFInfoResponse">
                        <s:complexType>
                              <s:sequence>
                                    <s:element minOccurs="0" maxOccurs="1"
name="GetWFInfoResult" type="tns:MethodResponse"/>
                              </s:sequence>
                        </s:complexType>
                  </s:element>
                  <s:element name="EditorNotification">
                        <s:complexType>
                              <s:sequence>
                                    <s:element minOccurs="0" maxOccurs="1"
name="AXRQID" type="s:string"/>
                                    <s:element minOccurs="0" maxOccurs="1"
name="URI" type="s:string"/>
                                    <s:element minOccurs="1" maxOccurs="1"
name="Result" type="s:boolean"/>
                                    <s:element minOccurs="0" maxOccurs="1"
name="ErrorMSG" type="s:string"/>
                                    <s:element minOccurs="0" maxOccurs="1"
name="ErrorCode" type="s:string"/>
                                    <s:element minOccurs="0" maxOccurs="1"
name="AXOID" type="s:string"/>
                              </s:sequence>
                        </s:complexType>
                  </s:element>
                  <s:complexType name="MethodResponse">
                        <s:sequence>
                              <s:element minOccurs="1" maxOccurs="1"
form="unqualified" name="Result" type="s:boolean"/>
                              <s:element minOccurs="0" maxOccurs="1"
form="unqualified" name="FaultCode" type="s:string"/>
                              <s:element minOccurs="0" maxOccurs="1"
form="unqualified" name="FaultString" type="s:string"/>
                              <s:element minOccurs="0" maxOccurs="1"
form="unqualified" name="AXRQID" type="s:string"/>
                              <s:element minOccurs="0" maxOccurs="1"
form="unqualified" name="Title" type="s:string"/>
                              <s:element minOccurs="0" maxOccurs="1"
form="unqualified" name="ProcessName" type="s:string"/>
                              <s:element minOccurs="0" maxOccurs="1"
form="unqualified" name="ActivityName" type="s:string"/>
                              <s:element minOccurs="0" maxOccurs="1"
form="unqualified" name="Status" type="s:string"/>
```

```
                              <s:element minOccurs="0" maxOccurs="1"
form="unqualified" name="Priority" type="s:string"/>
                              <s:element minOccurs="0" maxOccurs="1"
form="unqualified" name="Actor" type="s:string"/>
                        </s:sequence>
                  </s:complexType>
                  <s:complexType name="EditorNotificationType">
                        <s:sequence>
                              <s:element minOccurs="0" maxOccurs="1"
name="AXRQID" type="s:string"/>
                              <s:element minOccurs="0" maxOccurs="1" name="URI"
type="s:string"/>
                              <s:element minOccurs="1" maxOccurs="1"
name="Result" type="s:boolean"/>
                              <s:element minOccurs="0" maxOccurs="1"
name="ErrorMSG" type="s:string"/>
                              <s:element minOccurs="0" maxOccurs="1"
name="ErrorCode" type="s:string"/>
                              <s:element minOccurs="0" maxOccurs="1"
name="AXOID" type="s:string"/>
                        </s:sequence>
                  </s:complexType>
                  <s:element name="EditorNotificationResponse">
                        <s:complexType>
                              <s:sequence>
                                    <s:element minOccurs="0" maxOccurs="1"
name="EditorNotificationResult" type="tns:MethodResponse"/>
                              </s:sequence>
                        </s:complexType>
                  </s:element>
            </s:schema>
      </wsdl:types>
      <wsdl:message name="GetWFInfoSoapIn">
            <wsdl:part name="GetWFInfo" element="tns:GetWFInfo"/>
      </wsdl:message>
      <wsdl:message name="GetWFInfoSoapOut">
            <wsdl:part name="GetWFInfoResponse"
element="tns:GetWFInfoResponse"/>
      </wsdl:message>
      <wsdl:message name="EditorNotificationSoapIn">
            <wsdl:part name="EditorNotification"
element="tns:EditorNotification"/>
      </wsdl:message>
      <wsdl:message name="EditorNotificationSoapOut">
            <wsdl:part name="EditorNotificationResponse"
element="tns:EditorNotificationResponse"/>
      </wsdl:message>
      <wsdl:portType name="editorChannelClassSoap">
            <wsdl:operation name="GetWFInfo">
                  <wsdl:intput message="tns:GetWFInfoSoapOut"/>
                  <wsdl:output message="tns:GetWFInfoSoapIn"/>
            </wsdl:operation>
            <wsdl:operation name="EditorNotification">
                  <wsdl:input message="tns:EditorNotificationSoapOut"/>
                  <wsdl:output message="tns:EditorNotificationSoapIn"/>
            </wsdl:operation>
      </wsdl:portType>
      <wsdl:binding name="editorChannelClassSoap"
type="tns:editorChannelClassSoap">
            <soap:binding transport="http://schemas.xmlsoap.org/soap/http"/>
            <wsdl:operation name="GetWFInfo">
```

```
				<soap:operation
soapAction="http://www.axmedis.org/WF/Editor/GetWFInfo" style="document"/>
				<wsdl:input>
					<soap:body use="literal"/>
				</wsdl:input>
				<wsdl:output>
					<soap:body use="literal"/>
				</wsdl:output>
			</wsdl:operation>
			<wsdl:operation name="EditorNotification">
				<soap:operation
soapAction="http://www.axmedis.org/WF/Editor/EditorNotification"
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="editorChannelClass">
			<wsdl:port name="editorChannelClassSoap"
binding="tns:editorChannelClassSoap">
				<soap:address
location="http://localhost/responseGateway/editorChannel/editorChannel.asmx"/>
			</wsdl:port>
		</wsdl:service>
</wsdl:definitions>
```

23 Formal description of communication protocol Workflow Engine Request Channel (axWfEngineReq) Webservice

Call name	
Method	InstallAndActivate();
Description	This method is invoked through a webservice call coming from Workflow Request Gateway. This method will allow the workflow engine to install a new rule in the AXCP engine. The ruleID of the newly installed rule will be returned as ruleID.
Input parameters	char *AXRQID,char *userCredentials, char *xmlRuleSchema, char *URI,std::string &ruleID
Output parameters	bool

Request Sample Message	`<message name="InstallAndActivateRequest">`
	`<part name="UserCredentials" element="s0:UserCredentials"/>`
	`<part name="XMLRuleSchema" element="s0:XMLRuleSchema"/>`
	`<part name="AXRQID" element="s0:AXRQID"/>`
	`<part name="EngineListenerService" element="s0:EngineListenerService"/>`
	`</message>`

Response Sample Message	`<xs:element name="EngineResult">`
	`<xs:complexType>`
	`<xs:sequence>`
	`<xs:element name="Result" type="xs:boolean"/>`
	`<xs:element name="ErrorMSG"`

	`type="xs:string" minOccurs="0"/> <xs:element name="ErrorCode" type="xs:string" minOccurs="0"/> <xs:element name="status" type="xs:string" minOccurs="0"/> <xs:element name="ruleIDs" type="xs:string" minOccurs="0"/> <xs:element name="ruleLogs" type="xs:string" minOccurs="0"/> <xs:element name="ruleSchema" type="xs:string" minOccurs="0"/> </xs:sequence> </xs:complexType> </xs:element>`

Call name	
Method	RunRule();
Description	This method is invoked through a webservice call coming from Workflow Request Gateway. This method will allow the workflow engine to run a rule in the AXCP engine as per the supplied parameters.
Input parameters	char *AXRQID,char *userCredentials, char *ruleID, char *ruleType, char *URI, char *time, char *arguments char*
Output parameters	bool
Request Sample Message	`<message name="RunRuleRequest"> <part name="Time" element="s0:Time"/> <part name="AXRQID" element="s0:AXRQID"/> <part name="RuleID" element="s0:RuleID"/> <part name="RuleType" element="s0:RuleType"/> <part name="EngineListenerService" element="s0:EngineListenerService"/> <part name="UserCredentials" element="s0:UserCredentials"/> <part name="Arguments" element="s0:Arguments"/> </message>`
Response Sample Message	`<xs:element name="EngineResult"> <xs:complexType> <xs:sequence> <xs:element name="Result" type="xs:boolean"/> <xs:element name="ErrorMSG" type="xs:string" minOccurs="0"/> <xs:element name="ErrorCode" type="xs:string" minOccurs="0"/> <xs:element name="status" type="xs:string" minOccurs="0"/> <xs:element name="ruleIDs" type="xs:string" minOccurs="0"/> <xs:element name="ruleLogs" type="xs:string" minOccurs="0"/> <xs:element name="ruleSchema" type="xs:string" minOccurs="0"/> </xs:sequence> </xs:complexType> </xs:element>`

Call name	
Method	DeactivateRule();
Description	This method is invoked through a webservice call coming from Workflow Request Gateway. This method will allow the workflow engine to deactivate a rule in the AXCP engine as per the supplied parameters.
Input parameters	char *AXRQID, char *userCredentials, char *ruleID, char *ruleType
Output parameters	bool

Request Sample Message	`<message name="DeactivateRuleRequest">`
	`<part name="RuleID" element="s0:RuleID"/>`
	`<part name="AXRQID" element="s0:AXRQID"/>`
	`<part name="UserCredentials" element="s0:UserCredentials"/>`
	`<part name="RuleType" element="s0:RuleType"/>`
	`</message>`

Response Sample Message	`<xs:element name="EngineResult">`
	`<xs:complexType>`
	`<xs:sequence>`
	`<xs:element name="Result" type="xs:boolean"/>`
	`<xs:element name="ErrorMSG" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ErrorCode" type="xs:string" minOccurs="0"/>`
	`<xs:element name="status" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ruleIDs" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ruleLogs" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ruleSchema" type="xs:string" minOccurs="0"/>`
	`</xs:sequence>`
	`</xs:complexType>`
	`</xs:element>`

Call name	
Method	bool SuspendRule()
Description	This method is invoked through a webservice call coming from Workflow Request Gateway. This method will allow the workflow engine to suspend a rule in the AXCP engine as per the supplied parameters.
Input parameters	char *AXRQID,char *userCredentials, char *ruleID, char *ruleType, char *suspendTime
Output parameters	bool

Request Sample Message	`<message name="SuspendRuleRequest">`
	`<part name="SuspendTime" element="s0:SuspendTime"/>`
	`<part name="UserCredentials" element="s0:UserCredentials"/>`
	`<part name="RuleID" element="s0:RuleID"/>`
	`<part name="RuleType" element="s0:RuleType"/>`
	`<part name="AXRQID" element="s0:AXRQID"/>`
	`</message>`

Response	`<xs:element name="EngineResult">`

Sample Message	`<xs:complexType>`
	`<xs:sequence>`
	`<xs:element name="Result" type="xs:boolean"/>`
	`<xs:element name="ErrorMSG" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ErrorCode" type="xs:string" minOccurs="0"/>`
	`<xs:element name="status" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ruleIDs" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ruleLogs" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ruleSchema" type="xs:string" minOccurs="0"/>`
	`</xs:sequence>`
	`</xs:complexType>`
	`</xs:element>`

Call name	
Method	bool PauseRule(;
Description	This method is invoked through a webservice call coming from Workflow Request Gateway. This method will allow the workflow engine to pause a rule in the AXCP engine as per the supplied parameters.
Input parameters	char *AXRQID, char *userCredentials, char *ruleID,char *ruleType
Output parameters	bool

Request Sample Message	`<message name="PauseRuleRequest">`
	`<part name="RuleType" element="s0:RuleType"/>`
	`<part name="AXRQID" element="s0:AXRQID"/>`
	`<part name="UserCredentials" element="s0:UserCredentials"/>`
	`<part name="RuleID" element="s0:RuleID"/>`
	`</message>`

Response Sample Message	`<xs:element name="EngineResult">`
	`<xs:complexType>`
	`<xs:sequence>`
	`<xs:element name="Result" type="xs:boolean"/>`
	`<xs:element name="ErrorMSG" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ErrorCode" type="xs:string" minOccurs="0"/>`
	`<xs:element name="status" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ruleIDs" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ruleLogs" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ruleSchema" type="xs:string" minOccurs="0"/>`
	`</xs:sequence>`
	`</xs:complexType>`
	`</xs:element>`

Call name	
Method	bool KillRule( );
Description	This method is invoked through a webservice call coming from Workflow Request Gateway. This method will allow the workflow engine to kill a rule in the AXCP engine as per the supplied parameters.
Input parameters	char *AXRQID, char *userCredentials, char *ruleID, char *ruleType
Output parameters	bool
Request Sample Message	`<message name="KillRuleRequest">` `<part name="AXRQID" element="s0:AXRQID"/>` `<part name="UserCredentials" element="s0:UserCredentials"/>` `<part name="RuleID" element="s0:RuleID"/>` `<part name="RuleType" element="s0:RuleType"/>` `</message>`
Response Sample Message	`<xs:element name="EngineResult">` `<xs:complexType>` `<xs:sequence>` `<xs:element name="Result" type="xs:boolean"/>` `<xs:element name="ErrorMSG" type="xs:string" minOccurs="0"/>` `<xs:element name="ErrorCode" type="xs:string" minOccurs="0"/>` `<xs:element name="status" type="xs:string" minOccurs="0"/>` `<xs:element name="ruleIDs" type="xs:string" minOccurs="0"/>` `<xs:element name="ruleLogs" type="xs:string" minOccurs="0"/>` `<xs:element name="ruleSchema" type="xs:string" minOccurs="0"/>` `</xs:sequence>` `</xs:complexType>` `</xs:element>`

Call name	
Method	bool RemoveRule();
Description	This method is invoked through a webservice call coming from Workflow Request Gateway. This method will allow the workflow engine to remove a rule from the AXCP engine as per the supplied parameters.
Input parameters	char *AXRQID, char *userCredentials, char *ruleID, char *ruleType
Output parameters	bool
Request Sample Message	`<message name="RemoveRuleRequest">` `<part name="Dummy" element="s0:Dummy"/>` `<part name="UserCredentials" element="s0:UserCredentials"/>` `<part name="AXRQID" element="s0:AXRQID"/>` `<part name="RuleID" element="s0:RuleID"/>` `<part name="RuleType" element="s0:RuleType"/>` `</message>`

Response Sample Message	`<xs:element name="EngineResult">`
	`<xs:complexType>`
	`<xs:sequence>`
	`<xs:element name="Result" type="xs:boolean"/>`
	`<xs:element name="ErrorMSG" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ErrorCode" type="xs:string" minOccurs="0"/>`
	`<xs:element name="status" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ruleIDs" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ruleLogs" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ruleSchema" type="xs:string" minOccurs="0"/>`
	`</xs:sequence>`
	`</xs:complexType>`
	`</xs:element>`

Call name	
Method	bool ResumeRule();
Description	This method is invoked through a webservice call coming from Workflow Request Gateway. This method will allow the workflow engine to resume a rule in the AXCP engine as per the supplied parameters.
Input parameters	char *AXRQID, char *userCredentials, char *ruleID, char *ruleType, char *URI,char *arguments
Output parameters	bool

Request Sample Message	`<message name="ResumeRuleRequest">`
	`<part name="Dummy" element="s0:Arguments"/>`
	`<part name="UserCredentials" element="s0:UserCredentials"/>`
	`<part name="AXRQID" element="s0:AXRQID"/>`
	`<part name="RuleID" element="s0:RuleID"/>`
	`<part name="RuleType" element="s0:RuleType"/>`
	`<part name="EngineListenerService" element="s0:EngineListenerService"/>`
	`</message>`

Response Sample Message	`<xs:element name="EngineResult">`
	`<xs:complexType>`
	`<xs:sequence>`
	`<xs:element name="Result" type="xs:boolean"/>`
	`<xs:element name="ErrorMSG" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ErrorCode" type="xs:string" minOccurs="0"/>`
	`<xs:element name="status" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ruleIDs" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ruleLogs" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ruleSchema" type="xs:string" minOccurs="0"/>`

	`</xs:sequence>` `</xs:complexType>` `</xs:element>`

Call name	
Method	bool GetRuleStatus();
Description	This method is invoked through a webservice call coming from Workflow Request Gateway. This method will allow the workflow engine to know the status of the a rule in the AXCP engine as per the supplied parameters.
Input parameters	char *AXRQID, char *userCredentials, char *ruleID, char *ruleType,std::string &status
Output parameters	bool
Request Sample Message	`<message name="GetRuleStatusRequest">` `<part name="Dummy" element="s0:CommitListenerID"/>` `<part name="UserCredentials" element="s0:UserCredentials"/>` `<part name="AXRQID" element="s0:AXRQID"/>` `<part name="RuleID" element="s0:RuleID"/>` `<part name="RuleType" element="s0:RuleType"/>` `</message>`
Response Sample Message	`<xs:element name="EngineResult">` `<xs:complexType>` `<xs:sequence>` `<xs:element name="Result" type="xs:boolean"/>` `<xs:element name="ErrorMSG" type="xs:string" minOccurs="0"/>` `<xs:element name="ErrorCode" type="xs:string" minOccurs="0"/>` `<xs:element name="status" type="xs:string" minOccurs="0"/>` `<xs:element name="ruleIDs" type="xs:string" minOccurs="0"/>` `<xs:element name="ruleLogs" type="xs:string" minOccurs="0"/>` `<xs:element name="ruleSchema" type="xs:string" minOccurs="0"/>` `</xs:sequence>` `</xs:complexType>` `</xs:element>`

Call name	
Method	bool GetRuleLog();
Description	This method is invoked through a webservice call coming from Workflow Request Gateway. This method will allow the workflow engine to know the run log for a rule in the AXCP engine as per the supplied parameters.
Input parameters	char *AXRQID,char *userCredentials, char *ruleID, char *ruleType, std::string &ruleLogs
Output parameters	bool
Request Sample	`<message name="GetRuleLogRequest">` `<part name="Dummy" element="s0:EngineListenerService"/>` `<part name="UserCredentials"`

Message	`element="s0:UserCredentials"/>`
	`            <part name="AXRQID" element="s0:AXRQID"/>`
	`            <part name="RuleID" element="s0:RuleID"/>`
	`            <part name="RuleType" element="s0:RuleType"/>`
	`      </message>`

Response Sample Message	`                  <xs:element name="EngineResult">`
	`                        <xs:complexType>`
	`                              <xs:sequence>`
	`                                    <xs:element name="Result"`
	`type="xs:boolean"/>`
	`                                    <xs:element name="ErrorMSG"`
	`type="xs:string" minOccurs="0"/>`
	`                                    <xs:element name="ErrorCode"`
	`type="xs:string" minOccurs="0"/>`
	`                                    <xs:element name="status"`
	`type="xs:string" minOccurs="0"/>`
	`                                    <xs:element name="ruleIDs"`
	`type="xs:string" minOccurs="0"/>`
	`                                    <xs:element name="ruleLogs"`
	`type="xs:string" minOccurs="0"/>`
	`                                    <xs:element name="ruleSchema"`
	`type="xs:string" minOccurs="0"/>`
	`                              </xs:sequence>`
	`                        </xs:complexType>`
	`                  </xs:element>`

Call name	
Method	bool GetListofRules();
Description	This method is invoked through a webservice call coming from Workflow Request Gateway. This method will allow the workflow engine to retrieve the list of currently installed rules in the AXCP engine as per the supplied parameters.
Input parameters	char *AXRQID,char *userCredentials, char *ruleType, std::string &ruleIDList
Output parameters	bool

Request Sample Message	`      <message name="GetListofRulesRequest">`
	`            <part name="Dummy" element="s0:SuspendTime"/>`
	`            <part name="UserCredentials"`
	`element="s0:UserCredentials"/>`
	`            <part name="AXRQID" element="s0:AXRQID"/>`
	`            <part name="RuleType" element="s0:RuleType"/>`
	`      </message>`

Response Sample Message	`                  <xs:element name="EngineResult">`
	`                        <xs:complexType>`
	`                              <xs:sequence>`
	`                                    <xs:element name="Result"`
	`type="xs:boolean"/>`
	`                                    <xs:element name="ErrorMSG"`
	`type="xs:string" minOccurs="0"/>`
	`                                    <xs:element name="ErrorCode"`
	`type="xs:string" minOccurs="0"/>`
	`                                    <xs:element name="status"`
	`type="xs:string" minOccurs="0"/>`
	`                                    <xs:element name="ruleIDs"`
	`type="xs:string" minOccurs="0"/>`

	`<xs:element name="ruleLogs" type="xs:string" minOccurs="0"/>` `<xs:element name="ruleSchema" type="xs:string" minOccurs="0"/>` `</xs:sequence>` `</xs:complexType>` `</xs:element>`

Call name	
Method	bool GetRule();
Description	This method is invoked through a webservice call coming from Workflow Request Gateway. This method will allow the workflow engine to retrieve the rule schema from the AXCP engine as per the supplied parameters.
Input parameters	char *AXRQID,char *userCredentials, char *ruleID, char *ruleType,std::string &ruleSchema
Output parameters	bool
Request Sample Message	`<message name="GetRuleRequest">` `<part name="Dummy" element="s0:ErrorCode"/>` `<part name="UserCredentials" element="s0:UserCredentials"/>` `<part name="AXRQID" element="s0:AXRQID"/>` `<part name="RuleID" element="s0:RuleID"/>` `<part name="RuleType" element="s0:RuleType"/>` `</message>`
Response Sample Message	`<xs:element name="EngineResult">` `<xs:complexType>` `<xs:sequence>` `<xs:element name="Result" type="xs:boolean"/>` `<xs:element name="ErrorMSG" type="xs:string" minOccurs="0"/>` `<xs:element name="ErrorCode" type="xs:string" minOccurs="0"/>` `<xs:element name="status" type="xs:string" minOccurs="0"/>` `<xs:element name="ruleIDs" type="xs:string" minOccurs="0"/>` `<xs:element name="ruleLogs" type="xs:string" minOccurs="0"/>` `<xs:element name="ruleSchema" type="xs:string" minOccurs="0"/>` `</xs:sequence>` `</xs:complexType>` `</xs:element>`

Call name	
Method	bool StatusRequestPnP();
Description	This method is invoked through a webservice call coming from Workflow Request Gateway. This method will allow the workflow engine to retrieve the status of the PnP engine.
Input parameters	char *AXRQID, char *userCredentials, char *ProgramID, std::string &status
Output parameters	bool

Request Sample Message	`<message name="StatusRequestPnPRequest">`
	`<part name="Dummy" element="s0:ErrorMSG"/>`
	`<part name="UserCredentials" element="s0:UserCredentials"/>`
	`<part name="AXRQID" element="s0:AXRQID"/>`
	`<part name="ProgramID" element="s0:ProgramID"/>`
	`</message>`

Response Sample Message	`<xs:element name="EngineResult">`
	`<xs:complexType>`
	`<xs:sequence>`
	`<xs:element name="Result" type="xs:boolean"/>`
	`<xs:element name="ErrorMSG" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ErrorCode" type="xs:string" minOccurs="0"/>`
	`<xs:element name="status" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ruleIDs" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ruleLogs" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ruleSchema" type="xs:string" minOccurs="0"/>`
	`</xs:sequence>`
	`</xs:complexType>`
	`</xs:element>`

Call name	
Method	bool SuspendPnPProgram();
Description	This method is invoked through a webservice call coming from Workflow Request Gateway. This method will allow the workflow engine to suspend a program in the PnP engine.
Input parameters	char *AXRQID, char *userCredentials,char *programID
Output parameters	bool

Request Sample Message	`<message name="SuspendPnPProgramRequest">`
	`<part name="ProgramID" element="s0:ProgramID"/>`
	`<part name="AXRQID" element="s0:AXRQID"/>`
	`<part name="UserCredentials" element="s0:UserCredentials"/>`
	`</message>`

Response Sample Message	`<xs:element name="EngineResult">`
	`<xs:complexType>`
	`<xs:sequence>`
	`<xs:element name="Result" type="xs:boolean"/>`
	`<xs:element name="ErrorMSG" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ErrorCode" type="xs:string" minOccurs="0"/>`
	`<xs:element name="status" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ruleIDs" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ruleLogs"`

	`type="xs:string" minOccurs="0"/>`
	`                                    <xs:element name="ruleSchema"`
	`type="xs:string" minOccurs="0"/>`
	`                              </xs:sequence>`
	`                        </xs:complexType>`
	`                  </xs:element>`

Call name	
Method	bool AbortPnPProgram();
Description	This method is invoked through a webservice call coming from Workflow Request Gateway. This method will allow the workflow engine to abort a program in the PnP engine.
Input parameters	char *AXRQID, char *userCredentials, char *ProgramID
Output parameters	bool

Request Sample Message	`      <message name="AbortPnPProgramRequest">`
	`            <part name="Dummy" element="s0:HistoryInfo"/>`
	`            <part name="UserCredentials"`
	`element="s0:UserCredentials"/>`
	`            <part name="AXRQID" element="s0:AXRQID"/>`
	`            <part name="ProgramID" element="s0:ProgramID"/>`
	`      </message>`

Response Sample Message	`                  <xs:element name="EngineResult">`
	`                        <xs:complexType>`
	`                              <xs:sequence>`
	`                                    <xs:element name="Result"`
	`type="xs:boolean"/>`
	`                                    <xs:element name="ErrorMSG"`
	`type="xs:string" minOccurs="0"/>`
	`                                    <xs:element name="ErrorCode"`
	`type="xs:string" minOccurs="0"/>`
	`                                    <xs:element name="status"`
	`type="xs:string" minOccurs="0"/>`
	`                                    <xs:element name="ruleIDs"`
	`type="xs:string" minOccurs="0"/>`
	`                                    <xs:element name="ruleLogs"`
	`type="xs:string" minOccurs="0"/>`
	`                                    <xs:element name="ruleSchema"`
	`type="xs:string" minOccurs="0"/>`
	`                              </xs:sequence>`
	`                        </xs:complexType>`
	`                  </xs:element>`

Call name	
Method	bool ResumePnPProgram( );
Description	This method is invoked through a webservice call coming from Workflow Request Gateway. This method will allow the workflow engine to resume a program in the PnP engine.
Input parameters	char *AXRQID, char *userCredentials, char *ProgramID, char *URI
Output parameters	bool

Request Sample Message	`      <message name="ResumePnPProgramRequest">`
	`            <part name="Dummy" element="s0:RuleHistory"/>`
	`            <part name="UserCredentials"`
	`element="s0:UserCredentials"/>`

	`<part name="AXRQID" element="s0:AXRQID"/>`
	`<part name="ProgramID" element="s0:ProgramID"/>`
	`<part name="EngineListenerService" element="s0:EngineListenerService"/>`
	`</message>`

Response Sample Message	`<xs:element name="EngineResult">`
	`<xs:complexType>`
	`<xs:sequence>`
	`<xs:element name="Result" type="xs:boolean"/>`
	`<xs:element name="ErrorMSG" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ErrorCode" type="xs:string" minOccurs="0"/>`
	`<xs:element name="status" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ruleIDs" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ruleLogs" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ruleSchema" type="xs:string" minOccurs="0"/>`
	`</xs:sequence>`
	`</xs:complexType>`
	`</xs:element>`

Call name	
Method	bool ActivatePnPProgram();
Description	This method is invoked through a webservice call coming from Workflow Request Gateway. This method will allow the workflow engine to activate a program in the PnP engine.
Input parameters	char *AXRQID,char *userCredentials, char *ProgramID, char *URI
Output parameters	bool

Request Sample Message	`<message name="ActivatePnPProgramRequest">`
	`<part name="Dummy" element="s0:Version"/>`
	`<part name="UserCredentials" element="s0:UserCredentials"/>`
	`<part name="AXRQID" element="s0:AXRQID"/>`
	`<part name="ProgramID" element="s0:ProgramID"/>`
	`<part name="EngineListenerService" element="s0:EngineListenerService"/>`
	`</message>`

Response Sample Message	`<xs:element name="EngineResult">`
	`<xs:complexType>`
	`<xs:sequence>`
	`<xs:element name="Result" type="xs:boolean"/>`
	`<xs:element name="ErrorMSG" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ErrorCode" type="xs:string" minOccurs="0"/>`
	`<xs:element name="status" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ruleIDs" type="xs:string" minOccurs="0"/>`

	`<xs:element name="ruleLogs" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ruleSchema" type="xs:string" minOccurs="0"/>`
	`</xs:sequence>`
	`</xs:complexType>`
	`</xs:element>`

Call name	
Method	bool WFNotification();
Description	This method is invoked through a webservice call coming from Workflow Request Gateway. This method will allow the workflow engine to send the notification to the PnP engine for the previously issues request to activate a process.
Input parameters	char *AXRQID, char *userCredentials,char *status
Output parameters	bool

Request Sample Message	`<message name="WFNotificationRequest">`
	`<part name="Status" element="s0:Status"/>`
	`<part name="AXRQID" element="s0:AXRQID"/>`
	`<part name="UserCredentials" element="s0:UserCredentials"/>`
	`</message>`

Response Sample Message	`<xs:element name="EngineResult">`
	`<xs:complexType>`
	`<xs:sequence>`
	`<xs:element name="Result" type="xs:boolean"/>`
	`<xs:element name="ErrorMSG" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ErrorCode" type="xs:string" minOccurs="0"/>`
	`<xs:element name="status" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ruleIDs" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ruleLogs" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ruleSchema" type="xs:string" minOccurs="0"/>`
	`</xs:sequence>`
	`</xs:complexType>`
	`</xs:element>`

Workflow Engine Request Channel

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:s0="http://www.axmedis.org/WF/Engine/"
targetNamespace="http://www.axmedis.org/WF/Engine/">
```

```
	<types>
		<xs:schema targetNamespace="http://www.axmedis.org/WF/Engine/">
			<xs:element name="EngineResult">
				<xs:complexType>
					<xs:sequence>
						<xs:element name="Result"
type="xs:boolean"/>
						<xs:element name="ErrorMSG" type="xs:string"
minOccurs="0"/>
						<xs:element name="ErrorCode"
type="xs:string" minOccurs="0"/>
						<xs:element name="status" type="xs:string"
minOccurs="0"/>
						<xs:element name="ruleIDs" type="xs:string"
minOccurs="0"/>
						<xs:element name="ruleLogs" type="xs:string"
minOccurs="0"/>
						<xs:element name="ruleSchema"
type="xs:string" minOccurs="0"/>
						<xs:element name="ProgramIDs"
type="xs:string" minOccurs="0"/>
					</xs:sequence>
				</xs:complexType>
			</xs:element>
			<xs:element name="installRequest">
				<xs:complexType>
					<xs:sequence>
						<xs:element name="UserCredentials"
type="xs:string"/>
						<xs:element name="XMLRuleSchema"
type="xs:string"/>
						<xs:element name="AXRQID" type="xs:string"/>
						<xs:element name="EngineListenerService"
type="xs:anyURI"/>
					</xs:sequence>
				</xs:complexType>
			</xs:element>
			<xs:element name="runRequest">
				<xs:complexType>
					<xs:sequence>
						<xs:element name="Time" type="xs:string"/>
						<xs:element name="AXRQID" type="xs:string"/>
						<xs:element name="RuleID" type="xs:string"/>
						<xs:element name="RuleType"
type="xs:string"/>
						<xs:element name="EngineListenerService"
type="xs:anyURI"/>
						<xs:element name="UserCredentials"
type="xs:string"/>
						<xs:element name="Arguments"
type="xs:string"/>
					</xs:sequence>
				</xs:complexType>
			</xs:element>
			<xs:element name="deactivateRequest">
				<xs:complexType>
					<xs:sequence>
						<xs:element name="RuleID" type="xs:string"/>
						<xs:element name="RuleType"
type="xs:string"/>
						<xs:element name="AXRQID" type="xs:string"/>
```

```
                                    <xs:element name="UserCredentials"
type="xs:string"/>
                              </xs:sequence>
                        </xs:complexType>
                  </xs:element>
                  <xs:element name="suspendRequest">
                        <xs:complexType>
                              <xs:sequence>
                                    <xs:element name="RuleID" type="xs:string"/>
                                    <xs:element name="RuleType"
type="xs:string"/>
                                    <xs:element name="AXRQID" type="xs:string"/>
                                    <xs:element name="UserCredentials"
type="xs:string"/>
                                    <xs:element name="SuspendTime"
type="xs:string"/>
                              </xs:sequence>
                        </xs:complexType>
                  </xs:element>
                  <xs:element name="pauseRequest">
                        <xs:complexType>
                              <xs:sequence>
                                    <xs:element name="RuleID" type="xs:string"/>
                                    <xs:element name="RuleType"
type="xs:string"/>
                                    <xs:element name="AXRQID" type="xs:string"/>
                                    <xs:element name="UserCredentials"
type="xs:string"/>
                              </xs:sequence>
                        </xs:complexType>
                  </xs:element>
                  <xs:element name="killRequest">
                        <xs:complexType>
                              <xs:sequence>
                                    <xs:element name="RuleID" type="xs:string"/>
                                    <xs:element name="RuleType"
type="xs:string"/>
                                    <xs:element name="AXRQID" type="xs:string"/>
                                    <xs:element name="UserCredentials"
type="xs:string"/>
                              </xs:sequence>
                        </xs:complexType>
                  </xs:element>
                  <xs:element name="removeRequest">
                        <xs:complexType>
                              <xs:sequence>
                                    <xs:element name="RuleID" type="xs:string"/>
                                    <xs:element name="RuleType"
type="xs:string"/>
                                    <xs:element name="AXRQID" type="xs:string"/>
                                    <xs:element name="UserCredentials"
type="xs:string"/>
                              </xs:sequence>
                        </xs:complexType>
                  </xs:element>
                  <xs:element name="resumeRequest">
                        <xs:complexType>
                              <xs:sequence>
                                    <xs:element name="RuleID" type="xs:string"/>
                                    <xs:element name="RuleType"
type="xs:string"/>
```

```
                                    <xs:element name="AXRQID" type="xs:string"/>
                                    <xs:element name="UserCredentials"
type="xs:string"/>
                                    <xs:element name="EngineListenerService"
type="xs:anyURI"/>
                              </xs:sequence>
                        </xs:complexType>
                  </xs:element>
                  <xs:element name="getStatusRequest">
                        <xs:complexType>
                              <xs:sequence>
                                    <xs:element name="RuleID" type="xs:string"/>
                                    <xs:element name="RuleType"
type="xs:string"/>
                                    <xs:element name="AXRQID" type="xs:string"/>
                                    <xs:element name="UserCredentials"
type="xs:string"/>
                              </xs:sequence>
                        </xs:complexType>
                  </xs:element>
                  <xs:element name="getLogRequest">
                        <xs:complexType>
                              <xs:sequence>
                                    <xs:element name="RuleID" type="xs:string"/>
                                    <xs:element name="RuleType"
type="xs:string"/>
                                    <xs:element name="AXRQID" type="xs:string"/>
                                    <xs:element name="UserCredentials"
type="xs:string"/>
                              </xs:sequence>
                        </xs:complexType>
                  </xs:element>
                  <xs:element name="getListRequest">
                        <xs:complexType>
                              <xs:sequence>
                                    <xs:element name="RuleType"
type="xs:string"/>
                                    <xs:element name="AXRQID" type="xs:string"/>
                                    <xs:element name="UserCredentials"
type="xs:string"/>
                              </xs:sequence>
                        </xs:complexType>
                  </xs:element>
                  <xs:element name="getRuleRequest">
                        <xs:complexType>
                              <xs:sequence>
                                    <xs:element name="RuleID" type="xs:string"/>
                                    <xs:element name="RuleType"
type="xs:string"/>
                                    <xs:element name="AXRQID" type="xs:string"/>
                                    <xs:element name="UserCredentials"
type="xs:string"/>
                              </xs:sequence>
                        </xs:complexType>
                  </xs:element>
                  <xs:element name="pnpStatusRequest">
                        <xs:complexType>
                              <xs:sequence>
                                    <xs:element name="ProgramID"
type="xs:string"/>
                                    <xs:element name="AXRQID" type="xs:string"/>
                                    <xs:element name="UserCredentials"
type="xs:string"/>
                              </xs:sequence>
                        </xs:complexType>
                  </xs:element>
                  <xs:element name="pnpSuspendRequest">
                        <xs:complexType>
                              <xs:sequence>
                                    <xs:element name="ProgramID"
type="xs:string"/>
                                    <xs:element name="AXRQID" type="xs:string"/>
                                    <xs:element name="UserCredentials"
type="xs:string"/>
                              </xs:sequence>
                        </xs:complexType>
                  </xs:element>
                  <xs:element name="pnpAbortRequest">
                        <xs:complexType>
                              <xs:sequence>
                                    <xs:element name="ProgramID"
type="xs:string"/>
                                    <xs:element name="AXRQID" type="xs:string"/>
                                    <xs:element name="UserCredentials"
type="xs:string"/>
                              </xs:sequence>
                        </xs:complexType>
                  </xs:element>
                  <xs:element name="pnpResumeRequest">
                        <xs:complexType>
                              <xs:sequence>
                                    <xs:element name="ProgramID"
type="xs:string"/>
                                    <xs:element name="AXRQID" type="xs:string"/>
                                    <xs:element name="UserCredentials"
type="xs:string"/>
                                    <xs:element name="EngineListenerService"
type="xs:anyURI"/>
                              </xs:sequence>
                        </xs:complexType>
                  </xs:element>
                  <xs:element name="pnpActivateRequest">
                        <xs:complexType>
                              <xs:sequence>
                                    <xs:element name="Programme"
type="xs:string"/>
                                    <xs:element name="AXRQID" type="xs:string"/>
                                    <xs:element name="UserCredentials"
type="xs:string"/>
                                    <xs:element name="EngineListenerService"
type="xs:anyURI"/>
                              </xs:sequence>
                        </xs:complexType>
                  </xs:element>
                  <xs:element name="pnpGetListRequest">
                        <xs:complexType>
                              <xs:sequence>
                                    <xs:element name="AXRQID" type="xs:string"/>
                                    <xs:element name="UserCredentials"
type="xs:string"/>
                              </xs:sequence>
                        </xs:complexType>
                  </xs:element>
                  <xs:element name="wfNotificationRequest">
                        <xs:complexType>
                              <xs:sequence>
                                    <xs:element name="Status" type="xs:string"/>
                                    <xs:element name="AXRQID" type="xs:string"/>
                                    <xs:element name="UserCredentials"
type="xs:string"/>
                              </xs:sequence>
                        </xs:complexType>
                  </xs:element>
            </xs:schema>
      </types>
      <message name="InstallAndActivateRequest">
            <part name="installRequest" element="s0:installRequest"/>
      </message>
      <message name="InstallAndActivateResponse">
            <part name="EngineResult" element="s0:EngineResult"/>
      </message>
      <message name="RunRuleRequest">
            <part name="runRequest" element="s0:runRequest"/>
      </message>
      <message name="RunRuleResponse">
            <part name="EngineResult" element="s0:EngineResult"/>
      </message>
      <message name="DeactivateRuleRequest">
            <part name="deactivateRequest" element="s0:deactivateRequest"/>
      </message>
      <message name="DeactivateRuleResponse">
            <part name="EngineResult" element="s0:EngineResult"/>
      </message>
      <message name="SuspendRuleRequest">
            <part name="suspendRequest" element="s0:suspendRequest"/>
      </message>
      <message name="SuspendRuleResponse">
            <part name="EngineResult" element="s0:EngineResult"/>
      </message>
      <message name="PauseRuleRequest">
            <part name="pauseRequest" element="s0:pauseRequest"/>
      </message>
      <message name="PauseRuleResponse">
            <part name="EngineResult" element="s0:EngineResult"/>
      </message>
      <message name="KillRuleRequest">
            <part name="killRequest" element="s0:killRequest"/>
      </message>
      <message name="KillRuleResponse">
            <part name="EngineResult" element="s0:EngineResult"/>
      </message>
      <message name="RemoveRuleRequest">
            <part name="removeRequest" element="s0:removeRequest"/>
      </message>
      <message name="RemoveRuleResponse">
            <part name="EngineResult" element="s0:EngineResult"/>
      </message>
      <message name="ResumeRuleRequest">
            <part name="resumeRequest" element="s0:resumeRequest"/>
      </message>
      <message name="ResumeRuleResponse">
            <part name="EngineResult" element="s0:EngineResult"/>
      </message>
```

```
	<message name="GetRuleStatusRequest">
		<part name="getStatusRequest" element="s0:getStatusRequest"/>
	</message>
	<message name="GetRuleStatusResponse">
		<part name="EngineResult" element="s0:EngineResult"/>
	</message>
	<message name="GetRuleLogRequest">
		<part name="getLogRequest" element="s0:getLogRequest"/>
	</message>
	<message name="GetRuleLogResponse">
		<part name="EngineResult" element="s0:EngineResult"/>
	</message>
	<message name="GetListofRulesResponse">
		<part name="EngineResult" element="s0:EngineResult"/>
	</message>
	<message name="GetRuleRequest">
		<part name="getRuleRequest" element="s0:getRuleRequest"/>
	</message>
	<message name="GetRuleResponse">
		<part name="EngineResult" element="s0:EngineResult"/>
	</message>
	<message name="StatusRequestPnPRequest">
		<part name="pnpStatusRequest" element="s0:pnpStatusRequest"/>
	</message>
	<message name="StatusRequestPnPResponse">
		<part name="EngineResult" element="s0:EngineResult"/>
	</message>
	<message name="AbortPnPProgramRequest">
		<part name="pnpAbortRequest" element="s0:pnpAbortRequest"/>
	</message>
	<message name="AbortPnPProgramResponse">
		<part name="EngineResult" element="s0:EngineResult"/>
	</message>
	<message name="ActivatePnPProgramRequest">
		<part name="pnpActivateRequest" element="s0:pnpActivateRequest"/>
	</message>
	<message name="ActivatePnPProgramResponse">
		<part name="EngineResult" element="s0:EngineResult"/>
	</message>
	<message name="WFNotificationRequest">
		<part name="wfNotificationRequest"
element="s0:wfNotificationRequest"/>
	</message>
	<message name="WFNotificationResponse">
		<part name="EngineResult" element="s0:EngineResult"/>
	</message>
	<message name="GetListofRulesRequest">
		<part name="getListRequest" element="s0:getListRequest"/>
	</message>
	<message name="GetListofProgramsRequest">
		<part name="pnpGetListRequest" element="s0:pnpGetListRequest"/>
	</message>
	<message name="GetListofProgramsResponse">
		<part name="EngineResult" element="s0:EngineResult"/>
	</message>
	<portType name="EngineSOAP">
		<operation name="InstallAndActivate">
			<input message="s0:InstallAndActivateRequest"/>
			<output message="s0:InstallAndActivateResponse"/>
		</operation>
		<operation name="RunRule">
```

```
			<input message="s0:RunRuleRequest"/>
			<output message="s0:RunRuleResponse"/>
		</operation>
		<operation name="DeactivateRule">
			<input message="s0:DeactivateRuleRequest"/>
			<output message="s0:DeactivateRuleResponse"/>
		</operation>
		<operation name="SuspendRule">
			<input message="s0:SuspendRuleRequest"/>
			<output message="s0:SuspendRuleResponse"/>
		</operation>
		<operation name="PauseRule">
			<input message="s0:PauseRuleRequest"/>
			<output message="s0:PauseRuleResponse"/>
		</operation>
		<operation name="KillRule">
			<input message="s0:KillRuleRequest"/>
			<output message="s0:KillRuleResponse"/>
		</operation>
		<operation name="RemoveRule">
			<input message="s0:RemoveRuleRequest"/>
			<output message="s0:RemoveRuleResponse"/>
		</operation>
		<operation name="ResumeRule">
			<input message="s0:ResumeRuleRequest"/>
			<output message="s0:ResumeRuleResponse"/>
		</operation>
		<operation name="GetRuleStatus">
			<input message="s0:GetRuleStatusRequest"/>
			<output message="s0:GetRuleStatusResponse"/>
		</operation>
		<operation name="GetRuleLog">
			<input message="s0:GetRuleLogRequest"/>
			<output message="s0:GetRuleLogResponse"/>
		</operation>
		<operation name="GetListofRules">
			<input message="s0:GetListofRulesRequest"/>
			<output message="s0:GetListofRulesResponse"/>
		</operation>
		<operation name="GetRule">
			<input message="s0:GetRuleRequest"/>
			<output message="s0:GetRuleResponse"/>
		</operation>
		<operation name="StatusRequestPnP">
			<input message="s0:StatusRequestPnPRequest"/>
			<output message="s0:StatusRequestPnPResponse"/>
		</operation>
		<operation name="AbortPnPProgram">
			<input message="s0:AbortPnPProgramRequest"/>
			<output message="s0:AbortPnPProgramResponse"/>
		</operation>
		<operation name="ActivatePnPProgram">
			<input message="s0:ActivatePnPProgramRequest"/>
			<output message="s0:ActivatePnPProgramResponse"/>
		</operation>
		<operation name="WFNotification">
			<input message="s0:WFNotificationRequest"/>
			<output message="s0:WFNotificationResponse"/>
		</operation>
		<operation name="GetListofPrograms">
			<input message="s0:GetListofProgramsRequest"/>
```

```
                  <output message="s0:GetListofProgramsResponse"/>
            </operation>
      </portType>
      <binding name="EngineSOAP" type="s0:EngineSOAP">
            <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
            <operation name="InstallAndActivate">
                  <soap:operation
soapAction="http://www.axmedis.org/WF/Engine/InstallAndActivateRule"
style="document"/>
                  <input>
                        <soap:body use="literal"/>
                  </input>
                  <output>
                        <soap:body use="literal"/>
                  </output>
            </operation>
            <operation name="RunRule">
                  <soap:operation
soapAction="http://www.axmedis.org/WF/Engine/RunRule" style="document"/>
                  <input>
                        <soap:body use="literal"/>
                  </input>
                  <output>
                        <soap:body use="literal"/>
                  </output>
            </operation>
            <operation name="DeactivateRule">
                  <soap:operation
soapAction="http://www.axmedis.org/WF/Engine/DeactivateRule" style="document"/>
                  <input>
                        <soap:body use="literal"/>
                  </input>
                  <output>
                        <soap:body use="literal"/>
                  </output>
            </operation>
            <operation name="SuspendRule">
                  <soap:operation
soapAction="http://www.axmedis.org/WF/Engine/SuspendRule" style="document"/>
                  <input>
                        <soap:body use="literal"/>
                  </input>
                  <output>
                        <soap:body use="literal"/>
                  </output>
            </operation>
            <operation name="PauseRule">
                  <soap:operation
soapAction="http://www.axmedis.org/WF/Engine/PauseRule" style="document"/>
                  <input>
                        <soap:body use="literal"/>
                  </input>
                  <output>
                        <soap:body use="literal"/>
                  </output>
            </operation>
            <operation name="KillRule">
                  <soap:operation
soapAction="http://www.axmedis.org/WF/Engine/KillRule" style="document"/>
                  <input>
```

```
                        <soap:body use="literal"/>
                  </input>
                  <output>
                        <soap:body use="literal"/>
                  </output>
            </operation>
            <operation name="RemoveRule">
                  <soap:operation
soapAction="http://www.axmedis.org/WF/Engine/RemoveRule" style="document"/>
                  <input>
                        <soap:body use="literal"/>
                  </input>
                  <output>
                        <soap:body use="literal"/>
                  </output>
            </operation>
            <operation name="ResumeRule">
                  <soap:operation
soapAction="http://www.axmedis.org/WF/Engine/ResumeRule" style="document"/>
                  <input>
                        <soap:body use="literal"/>
                  </input>
                  <output>
                        <soap:body use="literal"/>
                  </output>
            </operation>
            <operation name="GetRuleStatus">
                  <soap:operation
soapAction="http://www.axmedis.org/WF/Engine/GetRuleStatus" style="document"/>
                  <input>
                        <soap:body use="literal"/>
                  </input>
                  <output>
                        <soap:body use="literal"/>
                  </output>
            </operation>
            <operation name="GetRuleLog">
                  <soap:operation
soapAction="http://www.axmedis.org/WF/Engine/GetRuleLog" style="document"/>
                  <input>
                        <soap:body use="literal"/>
                  </input>
                  <output>
                        <soap:body use="literal"/>
                  </output>
            </operation>
            <operation name="GetListofRules">
                  <soap:operation
soapAction="http://www.axmedis.org/WF/Engine/GetListofRules" style="document"/>
                  <input>
                        <soap:body use="literal"/>
                  </input>
                  <output>
                        <soap:body use="literal"/>
                  </output>
            </operation>
            <operation name="GetRule">
                  <soap:operation
soapAction="http://www.axmedis.org/WF/Engine/GetRule" style="document"/>
                  <input>
                        <soap:body use="literal"/>
                  </input>
                  <output>
                        <soap:body use="literal"/>
                  </output>
            </operation>
            <operation name="StatusRequestPnP">
                  <soap:operation
soapAction="http://www.axmedis.org/WF/Engine/StatusRequestPnP"
style="document"/>
                  <input>
                        <soap:body use="literal"/>
                  </input>
                  <output>
                        <soap:body use="literal"/>
                  </output>
            </operation>
            <operation name="AbortPnPProgram">
                  <soap:operation
soapAction="http://www.axmedis.org/WF/Engine/AbortPnPProgram" style="document"/>
                  <input>
                        <soap:body use="literal"/>
                  </input>
                  <output>
                        <soap:body use="literal"/>
                  </output>
            </operation>
            <operation name="ActivatePnPProgram">
                  <soap:operation
soapAction="http://www.axmedis.org/WF/Engine/ActivatePnPProgram"
style="document"/>
                  <input>
                        <soap:body use="literal"/>
                  </input>
                  <output>
                        <soap:body use="literal"/>
                  </output>
            </operation>
            <operation name="WFNotification">
                  <soap:operation
soapAction="http://www.axmedis.org/WF/Engine/WFNotification" style="document"/>
                  <input>
                        <soap:body use="literal"/>
                  </input>
                  <output>
                        <soap:body use="literal"/>
                  </output>
            </operation>
            <operation name="GetListofPrograms">
                  <soap:operation soapAction="urn:#GetListofPrograms"
style="document"/>
                  <input>
                        <soap:body use="literal"/>
                  </input>
                  <output>
                        <soap:body use="literal"/>
                  </output>
            </operation>
      </binding>
      <service name="EngineService">
            <port name="EngineSOAP" binding="s0:EngineSOAP">
```

```
                <soap:address
location="http://www.axmedis.org/WF/Engine/Engine.cgi"/>
            </port>
      </service>
</definitions>
```

24 Formal description of communication protocol Workflow Engine Response Channel (axWfEngineRes) Webservice

Call name	
Method	notifyCompletion()
Description	This method is invoked by the engine to send back the notification towards workflow engine for the completion of previously issues asynchronous request.
Input parameters	char* AXRQID,char* URI, char* endpURL,bool result=true, char* errorCode="NULL",char* errorMSG="NULL",char* Status="NULL"
Output parameters	bool
Request Sample Message	`<message name="EngineNotificationRequest">` `<part name="AXRQID" element="s0:AXRQID"/>` `<part name="URI" element="s0:URI"/>` `<part name="Status" element="s0:Status"/>` `<part name="Result" element="s0:Result"/>` `<part name="ErrorCode" element="s0:ErrorCode"/>` `<part name="ErrorMSG" element="s0:ErrorMSG"/>` `</message>`
Response Sample Message	`<xs:element name="MethodResponse">` `<xs:complexType>` `<xs:sequence>` `<xs:element name="Result" type="xs:boolean"/>` `<xs:element name="FaultCode" type="xs:string" minOccurs="0"/>` `<xs:element name="FaultString" type="xs:string" minOccurs="0"/>` `</xs:sequence>` `</xs:complexType>` `</xs:element>`

Call name	
Method	bool wfProcessRequest();
Description	This method is invoked by the engine to send request for activation of a workflow process identified the supplied processed.
Input parameters	char* AXRQID,char* URI, char* endpURL, char* processID
Output parameters	bool
Request Sample Message	`<message name="WFProcessRequest">` `<part name="AXRQID" element="s0:AXRQID"/>` `<part name="ProcessID" element="s0:ProcessID"/>` `<part name="EngineListenerService" element="s0:URI"/>` `</message>`

Response Sample Message	`<xs:element name="MethodResponse"> <xs:complexType> <xs:sequence> <xs:element name="Result" type="xs:boolean"/> <xs:element name="FaultCode" type="xs:string" minOccurs="0"/> <xs:element name="FaultString" type="xs:string" minOccurs="0"/> </xs:sequence> </xs:complexType> </xs:element>`

Workflow Engine Response Channel

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:tns="http://www.axmedis.org/WF/Engine/"
xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
targetNamespace="http://www.axmedis.org/WF/Engine/">
      <wsdl:types>
            <s:schema elementFormDefault="unqualified"
targetNamespace="http://www.axmedis.org/WF/Engine/">
                  <s:element name="EngineNotification">
                        <s:complexType>
                              <s:sequence>
                                    <s:element minOccurs="0" maxOccurs="1"
form="unqualified" name="AXRQID" type="s:string"/>
                                    <s:element minOccurs="0" maxOccurs="1"
form="unqualified" name="URI" type="s:string"/>
                                    <s:element minOccurs="0" maxOccurs="1"
form="unqualified" name="Status" type="s:string"/>
                                    <s:element minOccurs="1" maxOccurs="1"
form="unqualified" name="Result" type="s:boolean"/>
                                    <s:element minOccurs="0" maxOccurs="1"
form="unqualified" name="ErrorCode" type="s:string"/>
                                    <s:element minOccurs="0" maxOccurs="1"
form="unqualified" name="ErrorMSG" type="s:string"/>
                              </s:sequence>
                        </s:complexType>
                  </s:element>
                  <s:element name="EngineNotificationResponse">
                        <s:complexType>
                              <s:sequence>
                                    <s:element minOccurs="0" maxOccurs="1"
form="unqualified" name="EngineNotificationResult" type="tns:MethodResponse"/>
                              </s:sequence>
                        </s:complexType>
                  </s:element>
                  <s:complexType name="MethodResponse">
                        <s:sequence>
```

```
                              <s:element minOccurs="1" maxOccurs="1"
form="unqualified" name="Result" type="s:boolean"/>
                              <s:element minOccurs="0" maxOccurs="1"
form="unqualified" name="FaultCode" type="s:string"/>
                              <s:element minOccurs="0" maxOccurs="1"
form="unqualified" name="FaultString" type="s:string"/>
                        </s:sequence>
                  </s:complexType>
                  <s:element name="WFProcess">
                        <s:complexType>
                              <s:sequence>
                                    <s:element minOccurs="0" maxOccurs="1"
form="unqualified" name="AXRQID" type="s:string"/>
                                    <s:element minOccurs="0" maxOccurs="1"
form="unqualified" name="ProcessID" type="s:string"/>
                                    <s:element minOccurs="0" maxOccurs="1"
form="unqualified" name="URI" type="s:string"/>
                              </s:sequence>
                        </s:complexType>
                  </s:element>
                  <s:element name="WFProcessResponse">
                        <s:complexType>
                              <s:sequence>
                                    <s:element minOccurs="0" maxOccurs="1"
form="unqualified" name="WFProcessResult" type="tns:MethodResponse"/>
                              </s:sequence>
                        </s:complexType>
                  </s:element>
            </s:schema>
      </wsdl:types>
      <wsdl:message name="EngineNotificationSoapIn">
            <wsdl:part name="EngineNotification"
element="tns:EngineNotification"/>
      </wsdl:message>
      <wsdl:message name="EngineNotificationSoapOut">
            <wsdl:part name="EngineNotificationResponse"
element="tns:EngineNotificationResponse"/>
      </wsdl:message>
      <wsdl:message name="WFProcessSoapIn">
            <wsdl:part name="WFProcess" element="tns:WFProcess"/>
      </wsdl:message>
      <wsdl:message name="WFProcessSoapOut">
            <wsdl:part name="WFProcessResponse"
element="tns:WFProcessResponse"/>
      </wsdl:message>
      <wsdl:portType name="engineChannelClassSoap">
            <wsdl:operation name="EngineNotification">
                  <wsdl:input message="tns:EngineNotificationSoapIn"/>
                  <wsdl:output message="tns:EngineNotificationSoapOut"/>
            </wsdl:operation>
            <wsdl:operation name="WFProcess">
                  <wsdl:input message="tns:WFProcessSoapIn"/>
                  <wsdl:output message="tns:WFProcessSoapOut"/>
            </wsdl:operation>
      </wsdl:portType>
      <wsdl:binding name="engineChannelClassSoap"
type="tns:engineChannelClassSoap">
            <soap:binding transport="http://schemas.xmlsoap.org/soap/http"/>
            <wsdl:operation name="EngineNotification">
```

```
                  <soap:operation
soapAction="http://www.axmedis.org/WF/Engine/EngineNotification"
style="document"/>
                  <wsdl:input>
                        <soap:body use="literal"/>
                  </wsdl:input>
                  <wsdl:output>
                        <soap:body use="literal"/>
                  </wsdl:output>
            </wsdl:operation>
            <wsdl:operation name="WFProcess">
                  <soap:operation
soapAction="http://www.axmedis.org/WF/Engine/WFProcess" 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="engineChannelClass">
            <wsdl:port name="engineChannelClassSoap"
binding="tns:engineChannelClassSoap">
                  <soap:address
location="http://134.225.4.14:8000/responseGateway/engineChannel/engineChannel.a
smx"/>
            </wsdl:port>
      </wsdl:service>
</wsdl:definitions>
```

25 Formal description of communication protocol Workflow Rule Editor Request Channel (axWfReReq) Webservice

Call name	
Method	bool editRule();
Description	This method is invoked through a webservice call coming from Workflow Request Gateway. This method will allow the workflow engine to invoke the rule editor and edit the rule. The rule header will be supplied by the workflow engine.
Input parameters	char* AXRQID, char* userCredentials, char *xmlRuleHeader,char *URI, char* editingType
Output parameters	bool
Request Sample Message	`<message name="EditCompositionFormatingRuleRequest">` `<part name="UserCredentials" element="s0:UserCredentials"/>` `<part name="XMLRuleSchema" element="s0:XMLRuleSchema"/>` `<part name="AXRQID" element="s0:AXRQID"/>` `<part name="REListenerService" element="s0:REListenerService"/>` `</message>`
Response	`<xs:element name="REResult">`

Sample Message	`<xs:complexType>` `<xs:sequence>` `<xs:element name="Result" type="xs:boolean"/>` `<xs:element name="ErrorMSG" type="xs:string" minOccurs="0"/>` `<xs:element name="ErrorCode" type="xs:string"/>` `<xs:element name="ProgramList" type="xs:string" minOccurs="0"/>` `</xs:sequence>` `</xs:complexType>` `</xs:element>`

Call name	
Method	bool PnPUI();
Description	This method is invoked through a webservice call coming from Workflow Request Gateway. This method will allow the workflow engine to invoke the PnP Program Editor to edit a program as specified by its name.
Input parameters	char *AXRQID, char *userCredentials,char *programName, char *URI
Output parameters	bool
Request Sample Message	`<message name="PnPUIRequest">` `<part name="ProgramName" element="s0:ProgramName"/>` `<part name="UserCredentials" element="s0:UserCredentials"/>` `<part name="AXRQID" element="s0:AXRQID"/>` `<part name="REListenerService" element="s0:REListenerService"/>` `</message>`
Response Sample Message	`<xs:element name="REResult">` `<xs:complexType>` `<xs:sequence>` `<xs:element name="Result" type="xs:boolean"/>` `<xs:element name="ErrorMSG" type="xs:string" minOccurs="0"/>` `<xs:element name="ErrorCode" type="xs:string"/>` `<xs:element name="ProgramList" type="xs:string" minOccurs="0"/>` `</xs:sequence>` `</xs:complexType>` `</xs:element>`

Workflow Rule Editor Request Channel

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:s0="http://www.axmedis.org/WF/RE/"
targetNamespace="http://www.axmedis.org/WF/RE/">
      <types>
            <xs:schema targetNamespace="http://www.axmedis.org/WF/RE/">
                  <xs:element name="REResult">
                        <xs:complexType>
                              <xs:sequence>
                                    <xs:element name="Result"
type="xs:boolean"/>
                                    <xs:element name="ErrorMSG" type="xs:string"
minOccurs="0"/>
                                    <xs:element name="ErrorCode"
type="xs:string" minOccurs="0"/>
                              </xs:sequence>
                        </xs:complexType>
                  </xs:element>
                  <xs:element name="editRequest">
                        <xs:complexType>
                              <xs:sequence>
                                    <xs:element name="UserCredentials"
type="xs:string"/>
                                    <xs:element name="XMLRuleSchema"
type="xs:string"/>
                                    <xs:element name="AXRQID" type="xs:string"/>
                                    <xs:element name="REListenerService"
type="xs:string"/>
                              </xs:sequence>
                        </xs:complexType>
                  </xs:element>
                  <xs:element name="editProgrammeRequest">
                        <xs:complexType>
                              <xs:sequence>
                                    <xs:element name="UserCredentials"
type="xs:string"/>
                                    <xs:element name="Programme"
type="xs:string"/>
                                    <xs:element name="AXRQID" type="xs:string"/>
                                    <xs:element name="REListenerService"
type="xs:string"/>
                              </xs:sequence>
                        </xs:complexType>
                  </xs:element>
            </xs:schema>
      </types>
      <message name="EditRuleRequest">
            <part name="editRequest" element="s0:editRequest"/>
      </message>
      <message name="EditRuleResponse">
            <part name="REResult" element="s0:REResult"/>
      </message>
      <message name="PnPUIRequest">
            <part name="editProgrammeRequest"
element="s0:editProgrammeRequest"/>
      </message>
      <message name="PnPUIResponse">
```

```
            <part name="REResult" element="s0:REResult"/>
      </message>
      <portType name="RuleEditorSOAP">
            <operation name="EditRule">
                  <input message="s0:EditRuleRequest"/>
                  <output message="s0:EditRuleResponse"/>
            </operation>
            <operation name="PnPUI">
                  <input message="s0:PnPUIRequest"/>
                  <output message="s0:PnPUIResponse"/>
            </operation>
      </portType>
      <binding name="RuleEditorSOAP" type="s0:RuleEditorSOAP">
            <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
            <operation name="EditRule">
                  <soap:operation
soapAction="http://www.axmedis.org/WF/RE/EditCompositionFormatingRule"
style="document"/>
                  <input>
                        <soap:body use="literal"/>
                  </input>
                  <output>
                        <soap:body use="literal"/>
                  </output>
            </operation>
            <operation name="PnPUI">
                  <soap:operation
soapAction="http://www.axmedis.org/WF/RE/PnPUI" style="document"/>
                  <input>
                        <soap:body use="literal"/>
                  </input>
                  <output>
                        <soap:body use="literal"/>
                  </output>
            </operation>
      </binding>
      <service name="RuleEditorService">
            <port name="RuleEditorSOAP" binding="s0:RuleEditorSOAP">
                  <soap:address
location="http://www.axmedis.org/WF/RE/RuleEditor.cgi"/>
            </port>
      </service>
</definitions>
```

26 Formal description of communication protocol Workflow Rule Editor Response Channel (axWfReRes) Webservice

Call name	
Method	bool notifyCompletion();
Description	This method is invoked by the rule editor to send back the notification towards workflow engine for the completion of previously issues asynchronous request.
Input parameters	char* AXRQID,char* URI, char* endpURL,bool result=true, char* errorCode="NULL",char* errorMSG="NULL", char* xmlRuleSchema="NULL"
Output parameters	bool
Request	`<message name="RENotificationRequest">`

Sample Message	`<part name="AXRQID" element="s0:AXRQID"/>`
	`<part name="URI" element="s0:URI"/>`
	`<part name="Result" element="s0:Result"/>`
	`<part name="ErrorCode" element="s0:ErrorCode"/>`
	`<part name="ErrorMSG" element="s0:ErrorMSG"/>`
	`<part name="RuleSchema" element="s0:RuleSchema"/>`
	`</message>`

Response Sample Message	`<xs:element name="MethodResponse">`
	`<xs:complexType>`
	`<xs:sequence>`
	`<xs:element name="Result"`
	`type="xs:boolean"/>`
	`<xs:element name="ErrorCode"`
	`type="xs:string" minOccurs="0"/>`
	`<xs:element name="ErrorMSG"`
	`type="xs:string" minOccurs="0"/>`
	`</xs:sequence>`
	`</xs:complexType>`
	`</xs:element>`

Workflow Rule Editor Response Channel

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:tns="http://www.axmedis.org/WF/RE/"
xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
targetNamespace="http://www.axmedis.org/WF/RE/">
      <wsdl:types>
            <s:schema elementFormDefault="unqualified"
targetNamespace="http://www.axmedis.org/WF/RE/">
                  <s:element name="RENotification">
                        <s:complexType>
                              <s:sequence>
                                    <s:element minOccurs="0" maxOccurs="1"
form="unqualified" name="AXRQID" type="s:string"/>
                                    <s:element minOccurs="0" maxOccurs="1"
form="unqualified" name="URI" type="s:string"/>
                                    <s:element minOccurs="1" maxOccurs="1"
form="unqualified" name="Result" type="s:boolean"/>
                                    <s:element minOccurs="0" maxOccurs="1"
form="unqualified" name="ErrorCode" type="s:string"/>
                                    <s:element minOccurs="0" maxOccurs="1"
form="unqualified" name="ErrorMSG" type="s:string"/>
                                    <s:element minOccurs="0" maxOccurs="1"
form="unqualified" name="ProgramID" type="s:string"/>
                                    <s:element minOccurs="0" maxOccurs="1"
form="unqualified" name="RuleSchema" type="s:string"/>
                              </s:sequence>
                        </s:complexType>
                  </s:element>
                  <s:element name="RENotificationResponse">
                        <s:complexType>
```

```
                              <s:sequence>
                                    <s:element minOccurs="0" maxOccurs="1"
form="unqualified" name="RENotificationResult" type="tns:MethodResponse"/>
                              </s:sequence>
                        </s:complexType>
                  </s:element>
                  <s:complexType name="MethodResponse">
                        <s:sequence>
                              <s:element minOccurs="1" maxOccurs="1"
form="unqualified" name="Result" type="s:boolean"/>
                              <s:element minOccurs="0" maxOccurs="1"
form="unqualified" name="ErrorCode" type="s:string"/>
                              <s:element minOccurs="0" maxOccurs="1"
form="unqualified" name="ErrorMSG" type="s:string"/>
                        </s:sequence>
                  </s:complexType>
            </s:schema>
      </wsdl:types>
      <wsdl:message name="RENotificationSoapIn">
            <wsdl:part name="RENotification" element="tns:RENotification"/>
      </wsdl:message>
      <wsdl:message name="RENotificationSoapOut">
            <wsdl:part name="RENotificationResponse"
element="tns:RENotificationResponse"/>
      </wsdl:message>
      <wsdl:portType name="reChannelClassSoap">
            <wsdl:operation name="RENotification">
                  <wsdl:input message="tns:RENotificationSoapIn"/>
                  <wsdl:output message="tns:RENotificationSoapOut"/>
            </wsdl:operation>
      </wsdl:portType>
      <wsdl:binding name="reChannelClassSoap" type="tns:reChannelClassSoap">
            <soap:binding transport="http://schemas.xmlsoap.org/soap/http"/>
            <wsdl:operation name="RENotification">
                  <soap:operation
soapAction="http://www.axmedis.org/WF/RE/RENotification" 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="reChannelClass">
            <wsdl:port name="reChannelClassSoap"
binding="tns:reChannelClassSoap">
                  <soap:address
location="http://localhost:8000/responseGateway/rechannel/rechannel.asmx"/>
            </wsdl:port>
      </wsdl:service>
</wsdl:definitions>
```

27 Formal description of communication protocol Workflow Database Request Channel (axWfDbReq) Webservice

Call name	
Method	editQuery
Description	Used to editQuery for the AXDBQS
Input	SelectionID, Credentials, AXRQID, URI

parameters	
Output parameters	bool

Request Sample Message	`<message name="EditQueryRequest">`
	`<part name="SelectionID" element="s0:SelectionID"/>`
	`<part name="UserCredentials" element="s0:UserCredentials"/>`
	`<part name="AXRQID" element="s0:AXRQID"/>`
	`<part name="CommitListenerID" element="s0:CommitListenerID"/>`
	`</message>`

Response Sample Message	`<xs:element name="DBResult">`
	`<xs:complexType>`
	`<xs:sequence>`
	`<xs:element name="Result" type="xs:boolean"/>`
	`<xs:element name="XMLSelection" type="xs:string" minOccurs="0"/>`
	`<xs:element name="SelectionID" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ListofAXOID" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ListofSelectionID" type="xs:string" minOccurs="0"/>`
	`<xs:element name="Version" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ErrorCode" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ErrorMsg" type="xs:string" minOccurs="0"/>`
	`<xs:element name="DownloadURI" type="xs:string" minOccurs="0"/>`
	`</xs:sequence>`
	`</xs:complexType>`
	`</xs:element>`

Call name	
Method	Load Selection
Description	To load the selection as specified by its ID
Input parameters	SelectionID, Credentials, AXRQID
Output parameters	bool

Request Sample Message	`<message name="LoadSelectionRequest">`
	`<part name="SelectionID" element="s0:SelectionID"/>`
	`<part name="UserCredentials" element="s0:UserCredentials"/>`
	`<part name="AXRQID" element="s0:AXRQID"/>`
	`</message>`

Response Sample Message	`<xs:element name="DBResult">`
	`<xs:complexType>`
	`<xs:sequence>`
	`<xs:element name="Result" type="xs:boolean"/>`
	`<xs:element name="XMLSelection"`

```
type="xs:string" minOccurs="0"/>
                                    <xs:element name="SelectionID"
type="xs:string" minOccurs="0"/>
                                    <xs:element name="ListofAXOID"
type="xs:string" minOccurs="0"/>
                                    <xs:element
name="ListofSelectionID" type="xs:string" minOccurs="0"/>
                                    <xs:element name="Version"
type="xs:string" minOccurs="0"/>
                                    <xs:element name="ErrorCode"
type="xs:string" minOccurs="0"/>
                                    <xs:element name="ErrorMsg"
type="xs:string" minOccurs="0"/>
                                    <xs:element name="DownloadURI"
type="xs:string" minOccurs="0"/>
                              </xs:sequence>
                        </xs:complexType>
                  </xs:element>
```

Call name	
Method	saveSelection
Description	To save the selection for the user
Input parameters	selectionID, credentials, AXRQID, XMLSelection
Output parameters	bool

Request Sample Message

```
      <message name="SaveSelectionReqeust">
            <part name="SelectionID" element="s0:SelectionID"/>
            <part name="UserCredentials"
element="s0:UserCredentials"/>
            <part name="AXRQID" element="s0:AXRQID"/>
            <part name="XMLSelection" element="s0:XMLSelection"/>
      </message>
```

Response Sample Message

```
                  <xs:element name="DBResult">
                        <xs:complexType>
                              <xs:sequence>
                                    <xs:element name="Result"
type="xs:boolean"/>
                                    <xs:element name="XMLSelection"
type="xs:string" minOccurs="0"/>
                                    <xs:element name="SelectionID"
type="xs:string" minOccurs="0"/>
                                    <xs:element name="ListofAXOID"
type="xs:string" minOccurs="0"/>
                                    <xs:element
name="ListofSelectionID" type="xs:string" minOccurs="0"/>
                                    <xs:element name="Version"
type="xs:string" minOccurs="0"/>
                                    <xs:element name="ErrorCode"
type="xs:string" minOccurs="0"/>
                                    <xs:element name="ErrorMsg"
type="xs:string" minOccurs="0"/>
                                    <xs:element name="DownloadURI"
type="xs:string" minOccurs="0"/>
                              </xs:sequence>
                        </xs:complexType>
                  </xs:element>
```


Call name	
Method	ListUserSelection
Description	Retrieves all the selections for the current user
Input parameters	Credentials, AXRQID
Output parameters	SelectionID list

Request Sample Message	`<message name="ListUserSelectionRequest">`
	`<part name="UserCredential" element="s0:UserCredentials"/>`
	`<part name="AXRQID" element="s0:AXRQID"/>`
	`</message>`

Response Sample Message	`<xs:element name="DBResult">`
	`<xs:complexType>`
	`<xs:sequence>`
	`<xs:element name="Result" type="xs:boolean"/>`
	`<xs:element name="XMLSelection" type="xs:string" minOccurs="0"/>`
	`<xs:element name="SelectionID" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ListofAXOID" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ListofSelectionID" type="xs:string" minOccurs="0"/>`
	`<xs:element name="Version" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ErrorCode" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ErrorMsg" type="xs:string" minOccurs="0"/>`
	`<xs:element name="DownloadURI" type="xs:string" minOccurs="0"/>`
	`</xs:sequence>`
	`</xs:complexType>`
	`</xs:element>`

Call name	
Method	listEntitledSelection
Description	Retrieves all the selections that a user is entitled for
Input parameters	Credentials, AXRQID
Output parameters	selectionID list

Request Sample Message	`<message name="ListEntitledSelectionReqeuest">`
	`<part name="UserCredentials" element="s0:UserCredentials"/>`
	`<part name="AXRQID" element="s0:AXRQID"/>`
	`</message>`

Response Sample	`<xs:element name="DBResult">`
	`<xs:complexType>`

Message	`<xs:sequence>`
	`<xs:element name="Result" type="xs:boolean"/>`
	`<xs:element name="XMLSelection" type="xs:string" minOccurs="0"/>`
	`<xs:element name="SelectionID" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ListofAXOID" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ListofSelectionID" type="xs:string" minOccurs="0"/>`
	`<xs:element name="Version" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ErrorCode" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ErrorMsg" type="xs:string" minOccurs="0"/>`
	`<xs:element name="DownloadURI" type="xs:string" minOccurs="0"/>`
	`</xs:sequence>`
	`</xs:complexType>`
	`</xs:element>`

Call name	
Method	deleteSelection
Description	To delete a particular selection
Input parameters	SelectionID, credentials, AXRQID
Output parameters	Bool

Request Sample Message	`<message name="DeleteSelectionRequest">`
	`<part name="SelectionID" element="s0:SelectionID"/>`
	`<part name="UserCredentials" element="s0:UserCredentials"/>`
	`<part name="AXRQID" element="s0:AXRQID"/>`
	`</message>`

Response Sample Message	`<xs:element name="DBResult">`
	`<xs:complexType>`
	`<xs:sequence>`
	`<xs:element name="Result" type="xs:boolean"/>`
	`<xs:element name="XMLSelection" type="xs:string" minOccurs="0"/>`
	`<xs:element name="SelectionID" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ListofAXOID" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ListofSelectionID" type="xs:string" minOccurs="0"/>`
	`<xs:element name="Version" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ErrorCode" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ErrorMsg" type="xs:string" minOccurs="0"/>`
	`<xs:element name="DownloadURI" type="xs:string" minOccurs="0"/>`

	</xs:sequence> </xs:complexType> </xs:element>

Call name	
Method	ActivateSelectionSync
Description	To activate any selection synchronously
Input parameters	selectionID, credentials, axrqid
Output parameters	Bool
Request Sample Message	<message name="ActivateSelectionSyncRequest"> <part name="SelectionID" element="s0:SelectionID"/> <part name="UserCredentials" element="s0:UserCredentials"/> <part name="AXRQID" element="s0:AXRQID"/> </message>
Response Sample Message	<xs:element name="DBResult"> <xs:complexType> <xs:sequence> <xs:element name="Result" type="xs:boolean"/> <xs:element name="XMLSelection" type="xs:string" minOccurs="0"/> <xs:element name="SelectionID" type="xs:string" minOccurs="0"/> <xs:element name="ListofAXOID" type="xs:string" minOccurs="0"/> <xs:element name="ListofSelectionID" type="xs:string" minOccurs="0"/> <xs:element name="Version" type="xs:string" minOccurs="0"/> <xs:element name="ErrorCode" type="xs:string" minOccurs="0"/> <xs:element name="ErrorMsg" type="xs:string" minOccurs="0"/> <xs:element name="DownloadURI" type="xs:string" minOccurs="0"/> </xs:sequence> </xs:complexType> </xs:element>

Call name	
Method	ActivateSelectionAsync
Description	To activate a selection asynchronously
Input parameters	selectionID, credentials, AXRQID, URI
Output parameters	bool
Request Sample Message	<message name="ActivateSelectionAsyncRequest"> <part name="SelectionID" element="s0:SelectionID"/> <part name="UserCredentials" element="s0:UserCredentials"/> <part name="AXRQID" element="s0:AXRQID"/> <part name="CommitListenerID" element="s0:CommitListenerID"/>

	`</message>`

Response Sample Message	`<xs:element name="DBResult">`
	`<xs:complexType>`
	`<xs:sequence>`
	`<xs:element name="Result"`
	`type="xs:boolean"/>`
	`<xs:element name="XMLSelection"`
	`type="xs:string" minOccurs="0"/>`
	`<xs:element name="SelectionID"`
	`type="xs:string" minOccurs="0"/>`
	`<xs:element name="ListofAXOID"`
	`type="xs:string" minOccurs="0"/>`
	`<xs:element`
	`name="ListofSelectionID" type="xs:string" minOccurs="0"/>`
	`<xs:element name="Version"`
	`type="xs:string" minOccurs="0"/>`
	`<xs:element name="ErrorCode"`
	`type="xs:string" minOccurs="0"/>`
	`<xs:element name="ErrorMsg"`
	`type="xs:string" minOccurs="0"/>`
	`<xs:element name="DownloadURI"`
	`type="xs:string" minOccurs="0"/>`
	`</xs:sequence>`
	`</xs:complexType>`
	`</xs:element>`

Call name	
Method	checkOutsync
Description	To check-out any object from the AXDB synchronously
Input parameters	AXOID, version, credentials, AXRQID
Output parameters	bool

Request Sample Message	`<message name="CheckOutSyncRequest">`
	`<part name="AXOID" element="s0:AXOID"/>`
	`<part name="Version" element="s0:Version"/>`
	`<part name="UserCredentials"`
	`element="s0:UserCredentials"/>`
	`<part name="AXRQID" element="s0:AXRQID"/>`
	`</message>`

Response Sample Message	`<xs:element name="DBResult">`
	`<xs:complexType>`
	`<xs:sequence>`
	`<xs:element name="Result"`
	`type="xs:boolean"/>`
	`<xs:element name="XMLSelection"`
	`type="xs:string" minOccurs="0"/>`
	`<xs:element name="SelectionID"`
	`type="xs:string" minOccurs="0"/>`
	`<xs:element name="ListofAXOID"`
	`type="xs:string" minOccurs="0"/>`
	`<xs:element`
	`name="ListofSelectionID" type="xs:string" minOccurs="0"/>`
	`<xs:element name="Version"`
	`type="xs:string" minOccurs="0"/>`

	`<xs:element name="ErrorCode" type="xs:string" minOccurs="0"/> <xs:element name="ErrorMsg" type="xs:string" minOccurs="0"/> <xs:element name="DownloadURI" type="xs:string" minOccurs="0"/> </xs:sequence> </xs:complexType> </xs:element>`
Call name	
Method	CheckOutAsync
Description	To checkout any object from the AXDB asynchronously
Input parameters	AXOID, version, credentials, AXRQID, URI
Output parameters	bool
Request Sample Message	`<message name="CheckOutAsyncRequest"> <part name="AXOID" element="s0:AXOID"/> <part name="Version" element="s0:Version"/> <part name="UserCredentials" element="s0:UserCredentials"/> <part name="AXRQID" element="s0:AXRQID"/> <part name="CommitListenerID" element="s0:CommitListenerID"/> </message>`
Response Sample Message	`<xs:element name="DBResult"> <xs:complexType> <xs:sequence> <xs:element name="Result" type="xs:boolean"/> <xs:element name="XMLSelection" type="xs:string" minOccurs="0"/> <xs:element name="SelectionID" type="xs:string" minOccurs="0"/> <xs:element name="ListofAXOID" type="xs:string" minOccurs="0"/> <xs:element name="ListofSelectionID" type="xs:string" minOccurs="0"/> <xs:element name="Version" type="xs:string" minOccurs="0"/> <xs:element name="ErrorCode" type="xs:string" minOccurs="0"/> <xs:element name="ErrorMsg" type="xs:string" minOccurs="0"/> <xs:element name="DownloadURI" type="xs:string" minOccurs="0"/> </xs:sequence> </xs:complexType> </xs:element>`
Call name	
Method	CommitSync
Description	To commit any changes for an object synchronously
Input parameters	AXOID, credentials, AXRQID, DownloadURI

Output parameters	bool
Request Sample Message	`<message name="CommitSyncRequest"> <part name="AXOID" element="s0:AXOID"/> <part name="UserCredentials" element="s0:UserCredentials"/> <part name="AXRQID" element="s0:AXRQID"/> <part name="DownloadURI" element="s0:DownloadURI"/> </message>`
Response Sample Message	`<xs:element name="DBResult"> <xs:complexType> <xs:sequence> <xs:element name="Result" type="xs:boolean"/> <xs:element name="XMLSelection" type="xs:string" minOccurs="0"/> <xs:element name="SelectionID" type="xs:string" minOccurs="0"/> <xs:element name="ListofAXOID" type="xs:string" minOccurs="0"/> <xs:element name="ListofSelectionID" type="xs:string" minOccurs="0"/> <xs:element name="Version" type="xs:string" minOccurs="0"/> <xs:element name="ErrorCode" type="xs:string" minOccurs="0"/> <xs:element name="ErrorMsg" type="xs:string" minOccurs="0"/> <xs:element name="DownloadURI" type="xs:string" minOccurs="0"/> </xs:sequence> </xs:complexType> </xs:element>`

Call name	
Method	CommitAsync
Description	To commit any changes for an object asynchronously
Input parameters	AXOID, credentials, AXRQID, DownloadURI, URI
Output parameters	bool
Request Sample Message	`<message name="CommitAsyncRequest"> <part name="AXOID" element="s0:AXOID"/> <part name="UserCredentials" element="s0:UserCredentials"/> <part name="AXRQID" element="s0:AXRQID"/> <part name="DownloadURI" element="s0:DownloadURI"/> <part name="CommitListenerID" element="s0:CommitListenerID"/> </message>`
Response Sample Message	`<xs:element name="DBResult"> <xs:complexType> <xs:sequence> <xs:element name="Result" type="xs:boolean"/>`

	`<xs:element name="XMLSelection"`
	`type="xs:string" minOccurs="0"/>`
	`<xs:element name="SelectionID"`
	`type="xs:string" minOccurs="0"/>`
	`<xs:element name="ListofAXOID"`
	`type="xs:string" minOccurs="0"/>`
	`<xs:element`
	`name="ListofSelectionID" type="xs:string" minOccurs="0"/>`
	`<xs:element name="Version"`
	`type="xs:string" minOccurs="0"/>`
	`<xs:element name="ErrorCode"`
	`type="xs:string" minOccurs="0"/>`
	`<xs:element name="ErrorMsg"`
	`type="xs:string" minOccurs="0"/>`
	`<xs:element name="DownloadURI"`
	`type="xs:string" minOccurs="0"/>`
	`</xs:sequence>`
	`</xs:complexType>`
	`</xs:element>`

Call name	
Method	LockObject
Description	To acquire lock for any object in AXDB
Input parameters	UserID, PWD, AXOID, Version, AXRQID, Credentials
Output parameters	bool

Request Sample Message	`<message name="LockObjectRequest">`
	`<part name="UserID" element="s0:UserID"/>`
	`<part name="PWD" element="s0:PWD"/>`
	`<part name="AXOID" element="s0:AXOID"/>`
	`<part name="Version" element="s0:Version"/>`
	`<part name="AXRQID" element="s0:AXRQID"/>`
	`<part name="UserCredentials"`
	`element="s0:UserCredentials"/>`
	`</message>`

Response Sample Message	`<xs:element name="DBResult">`
	`<xs:complexType>`
	`<xs:sequence>`
	`<xs:element name="Result"`
	`type="xs:boolean"/>`
	`<xs:element name="XMLSelection"`
	`type="xs:string" minOccurs="0"/>`
	`<xs:element name="SelectionID"`
	`type="xs:string" minOccurs="0"/>`
	`<xs:element name="ListofAXOID"`
	`type="xs:string" minOccurs="0"/>`
	`<xs:element`
	`name="ListofSelectionID" type="xs:string" minOccurs="0"/>`
	`<xs:element name="Version"`
	`type="xs:string" minOccurs="0"/>`
	`<xs:element name="ErrorCode"`
	`type="xs:string" minOccurs="0"/>`
	`<xs:element name="ErrorMsg"`
	`type="xs:string" minOccurs="0"/>`
	`<xs:element name="DownloadURI"`
	`type="xs:string" minOccurs="0"/>`

	`</xs:sequence>`
	`</xs:complexType>`
	`</xs:element>`

Call name	
Method	UnLockObject
Description	To release lock for any object in AXDB
Input parameters	UserID, PWD, AXOID, Version, AXRQID, Credentials
Output parameters	bool

Request Sample Message	`<message name="UnlockObjectRequest">`
	`<part name="UserID" element="s0:UserID"/>`
	`<part name="PWD" element="s0:PWD"/>`
	`<part name="AXOID" element="s0:AXOID"/>`
	`<part name="Version" element="s0:Version"/>`
	`<part name="AXRQID" element="s0:AXRQID"/>`
	`<part name="UserCredentials" element="s0:UserCredentials"/>`
	`</message>`

Response Sample Message	`<xs:element name="DBResult">`
	`<xs:complexType>`
	`<xs:sequence>`
	`<xs:element name="Result" type="xs:boolean"/>`
	`<xs:element name="XMLSelection" type="xs:string" minOccurs="0"/>`
	`<xs:element name="SelectionID" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ListofAXOID" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ListofSelectionID" type="xs:string" minOccurs="0"/>`
	`<xs:element name="Version" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ErrorCode" type="xs:string" minOccurs="0"/>`
	`<xs:element name="ErrorMsg" type="xs:string" minOccurs="0"/>`
	`<xs:element name="DownloadURI" type="xs:string" minOccurs="0"/>`
	`</xs:sequence>`
	`</xs:complexType>`
	`</xs:element>`

Workflow Database Request Channel

Lock Unlock:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2006 sp1 U (http://www.altova.com) by Maulik Sailor
(UNIVERSITY OF READING) -->
<WSDL:definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="http://www.exitech.it/axmedis/LockUnlockWS" xmlns:SOAP-
```

```
ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ax="http://www.exitech.it/axmedis/LockUnlock"
xmlns:SOAP="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:MIME="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:DIME="http://schemas.xmlsoap.org/ws/2002/04/dime/wsdl/"
xmlns:WSDL="http://schemas.xmlsoap.org/wsdl/"
targetNamespace="http://www.exitech.it/axmedis/LockUnlockWS" name="LockUnlock">
      <WSDL:types>
            <schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.exitech.it/axmedis/LockUnlock"
elementFormDefault="qualified" attributeFormDefault="qualified">
                  <import
namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
                  <!-- operation request element -->
                  <element name="lockObject">
                        <complexType>
                              <sequence>
                                    <element name="userId" type="xsd:string"
minOccurs="1" maxOccurs="1"/>
                                    <element name="pwd" type="xsd:string"
minOccurs="1" maxOccurs="1"/>
                                    <element name="axoid" type="xsd:string"
minOccurs="1" maxOccurs="1"/>
                                    <element name="version" type="xsd:int"
minOccurs="1" maxOccurs="1"/>
                              </sequence>
                        </complexType>
                  </element>
                  <!-- operation response element -->
                  <element name="lockObjectResponse">
                        <complexType>
                              <sequence>
                                    <element name="Result" type="xsd:int"
minOccurs="1" maxOccurs="1"/>
                              </sequence>
                        </complexType>
                  </element>
                  <!-- operation request element -->
                  <element name="unlockObject">
                        <complexType>
                              <sequence>
                                    <element name="userId" type="xsd:string"
minOccurs="1" maxOccurs="1"/>
                                    <element name="pwd" type="xsd:string"
minOccurs="1" maxOccurs="1"/>
                                    <element name="axoid" type="xsd:string"
minOccurs="1" maxOccurs="1"/>
                                    <element name="version" type="xsd:int"
minOccurs="1" maxOccurs="1"/>
                              </sequence>
                        </complexType>
                  </element>
                  <!-- operation response element -->
                  <element name="unlockObjectResponse">
                        <complexType>
                              <sequence>
                                    <element name="Result" type="xsd:int"
minOccurs="1" maxOccurs="1"/>
```

```
                              </sequence>
                        </complexType>
                  </element>
            </schema>
      </WSDL:types>
      <message name="lockObjectRequest">
            <part name="parameters" element="ax:lockObject"/>
      </message>
      <message name="lockObjectResponse">
            <part name="parameters" element="ax:lockObjectResponse"/>
      </message>
      <message name="unlockObjectRequest">
            <part name="parameters" element="ax:unlockObject"/>
      </message>
      <message name="unlockObjectResponse">
            <part name="parameters" element="ax:unlockObjectResponse"/>
      </message>
      <portType name="LockUnlockPortType">
            <operation name="lockObject">
                  <documentation>Service definition of function
ax__lockObject</documentation>
                  <input message="tns:lockObjectRequest"/>
                  <output message="tns:lockObjectResponse"/>
            </operation>
            <operation name="unlockObject">
                  <documentation>Service definition of function
ax__unlockObject</documentation>
                  <input message="tns:unlockObjectRequest"/>
                  <output message="tns:unlockObjectResponse"/>
            </operation>
      </portType>
      <binding name="LockUnlock" type="tns:LockUnlockPortType">
            <SOAP:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
            <operation name="lockObject">
                  <SOAP:operation/>
                  <input>
                        <SOAP:body use="literal"/>
                  </input>
                  <output>
                        <SOAP:body use="literal"/>
                  </output>
            </operation>
            <operation name="unlockObject">
                  <SOAP:operation/>
                  <input>
                        <SOAP:body use="literal"/>
                  </input>
                  <output>
                        <SOAP:body use="literal"/>
                  </output>
            </operation>
      </binding>
      <wsdl:service name="LockUnlockWS"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
            <wsdl:port name="LockUnlockPortTypePort" binding="tns:LockUnlock">
                  <SOAP:address
location="http://81.73.104.125:8080/LockUnlockWS/lockunlock"/>
```

```
            </wsdl:port>
      </wsdl:service>
</WSDL:definitions>
```

Selection Archive:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="Selection_Archive"
 targetNamespace="http://www.axmedis.org/selection_archive.wsdl"
 xmlns:tns="http://www.axmedis.org/selection_archive.wsdl"
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:ax="urn:ax"
 xmlns:SOAP="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:MIME="http://schemas.xmlsoap.org/wsdl/mime/"
 xmlns:DIME="http://schemas.xmlsoap.org/ws/2002/04/dime/wsdl/"
 xmlns:WSDL="http://schemas.xmlsoap.org/wsdl/"
 xmlns="http://schemas.xmlsoap.org/wsdl/">

<types>

 <schema targetNamespace="urn:ax"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ax="urn:ax"
  xmlns="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  attributeFormDefault="qualified">
  <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
  <complexType name="SelectionDetail">
   <sequence>
     <element name="Id" type="xsd:string" minOccurs="1" maxOccurs="1"/>
     <element name="Name" type="xsd:string" minOccurs="1" maxOccurs="1"/>
     <element name="Timestamp" type="xsd:string" minOccurs="1" maxOccurs="1"/>
   </sequence>
  </complexType>
  <complexType name="SL">
   <sequence>
     <element name="RetCode" type="xsd:int" minOccurs="1" maxOccurs="1"/>
     <element name="NumberOfSelection" type="xsd:int" minOccurs="1"
maxOccurs="1"/>
     <element name="Selection" type="ax:SelectionDetail" minOccurs="0"
maxOccurs="unbounded"/>
   </sequence>
  </complexType>
  <!-- operation request element -->
  <element name="ListUserSelection">
   <complexType>
    <sequence>
     <element name="user" type="xsd:string" minOccurs="1" maxOccurs="1"/>
     <element name="pwd" type="xsd:string" minOccurs="1" maxOccurs="1"/>
    </sequence>
   </complexType>
  </element>
  <!-- operation response element -->
  <element name="SelectionList">
   <complexType>
```

```
    <sequence>
     <element name="selectionresult" type="ax:SL" minOccurs="1" maxOccurs="1"/>
    </sequence>
   </complexType>
  </element>
  <!-- operation request element -->
  <element name="ListEntitledSelection">
   <complexType>
    <sequence>
     <element name="user" type="xsd:string" minOccurs="1" maxOccurs="1"/>
     <element name="pwd" type="xsd:string" minOccurs="1" maxOccurs="1"/>
    </sequence>
   </complexType>
  </element>
  <!-- operation request element -->
  <element name="LoadSelection">
   <complexType>
    <sequence>
     <element name="user" type="xsd:string" minOccurs="1" maxOccurs="1"/>
     <element name="pwd" type="xsd:string" minOccurs="1" maxOccurs="1"/>
     <element name="SelectionID" type="xsd:string" minOccurs="1" maxOccurs="1"/>
    </sequence>
   </complexType>
  </element>
  <!-- operation response element -->
  <element name="LoadSelectionResponse">
   <complexType>
    <sequence>
     <element name="Selection" type="xsd:string" minOccurs="1" maxOccurs="1"/>
    </sequence>
   </complexType>
  </element>
  <!-- operation request element -->
  <element name="SaveSelection">
   <complexType>
    <sequence>
     <element name="user" type="xsd:string" minOccurs="1" maxOccurs="1"/>
     <element name="pwd" type="xsd:string" minOccurs="1" maxOccurs="1"/>
     <element name="Selection" type="xsd:string" minOccurs="1" maxOccurs="1"/>
    </sequence>
   </complexType>
  </element>
  <!-- operation response element -->
  <element name="SaveSelectionResponse">
   <complexType>
    <sequence>
     <element name="result" type="xsd:boolean" minOccurs="1" maxOccurs="1"/>
    </sequence>
   </complexType>
  </element>
  <!-- operation request element -->
  <element name="DeleteSelection">
   <complexType>
    <sequence>
     <element name="user" type="xsd:string" minOccurs="1" maxOccurs="1"/>
     <element name="pwd" type="xsd:string" minOccurs="1" maxOccurs="1"/>
     <element name="SelectionID" type="xsd:string" minOccurs="1" maxOccurs="1"/>
    </sequence>
   </complexType>
  </element>
  <!-- operation response element -->
  <element name="DeleteSelectionResponse">
   <complexType>
    <sequence>
     <element name="result" type="xsd:boolean" minOccurs="1" maxOccurs="1"/>
    </sequence>
   </complexType>
  </element>
  <!-- operation request element -->
  <element name="ActualizeSelection-sync">
   <complexType>
    <sequence>
     <element name="user" type="xsd:string" minOccurs="1" maxOccurs="1"/>
     <element name="pwd" type="xsd:string" minOccurs="1" maxOccurs="1"/>
     <element name="selection" type="xsd:string" minOccurs="1" maxOccurs="1"/>
    </sequence>
   </complexType>
  </element>
  <!-- operation response element -->
  <element name="ActualizeSelection-syncResponse">
   <complexType>
    <sequence>
     <element name="actualizedSelection" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
    </sequence>
   </complexType>
  </element>
  <!-- operation request element -->
  <element name="ActualizeSelection-async">
   <complexType>
    <sequence>
     <element name="User" type="xsd:string" minOccurs="1" maxOccurs="1"/>
     <element name="Password" type="xsd:string" minOccurs="1" maxOccurs="1"/>
     <element name="selection" type="xsd:string" minOccurs="1" maxOccurs="1"/>
     <element name="ActualizeListenerService" type="xsd:anyURI" minOccurs="1"
maxOccurs="1"/>
     <element name="ListenerID" type="xsd:string" minOccurs="1" maxOccurs="1"/>
    </sequence>
   </complexType>
  </element>
  <!-- operation response element -->
  <element name="ActualizeSelection-asyncResponse">
   <complexType>
    <sequence>
     <element name="result" type="xsd:boolean" minOccurs="1" maxOccurs="1"/>
    </sequence>
   </complexType>
  </element>
 </schema>

</types>

<message name="ListUserSelectionRequest">
 <part name="parameters" element="ax:ListUserSelection"/>
</message>

<message name="SelectionList">
 <part name="parameters" element="ax:SelectionList"/>
</message>

<message name="ListEntitledSelectionRequest">
 <part name="parameters" element="ax:ListEntitledSelection"/>
```

```
</message>

<message name="LoadSelectionRequest">
 <part name="parameters" element="ax:LoadSelection"/>
</message>

<message name="LoadSelectionResponse">
 <part name="parameters" element="ax:LoadSelectionResponse"/>
</message>

<message name="SaveSelectionRequest">
 <part name="parameters" element="ax:SaveSelection"/>
</message>

<message name="SaveSelectionResponse">
 <part name="parameters" element="ax:SaveSelectionResponse"/>
</message>

<message name="DeleteSelectionRequest">
 <part name="parameters" element="ax:DeleteSelection"/>
</message>

<message name="DeleteSelectionResponse">
 <part name="parameters" element="ax:DeleteSelectionResponse"/>
</message>

<message name="ActualizeSelection-syncRequest">
 <part name="parameters" element="ax:ActualizeSelection-sync"/>
</message>

<message name="ActualizeSelection-syncResponse">
 <part name="parameters" element="ax:ActualizeSelection-syncResponse"/>
</message>

<message name="ActualizeSelection-asyncRequest">
 <part name="parameters" element="ax:ActualizeSelection-async"/>
</message>

<message name="ActualizeSelection-asyncResponse">
 <part name="parameters" element="ax:ActualizeSelection-asyncResponse"/>
</message>

<portType name="Selection_ArchivePortType">
 <operation name="ListUserSelection">
  <documentation>Service definition of function
ax__ListUserSelection</documentation>
  <input message="tns:ListUserSelectionRequest"/>
  <output message="tns:SelectionList"/>
 </operation>
 <operation name="ListEntitledSelection">
  <documentation>Service definition of function
ax__ListEntitledSelection</documentation>
  <input message="tns:ListEntitledSelectionRequest"/>
  <output message="tns:SelectionList"/>
 </operation>
 <operation name="LoadSelection">
  <documentation>Service definition of function
ax__LoadSelection</documentation>
  <input message="tns:LoadSelectionRequest"/>
  <output message="tns:LoadSelectionResponse"/>
 </operation>
 <operation name="SaveSelection">
  <documentation>Service definition of function
ax__SaveSelection</documentation>
  <input message="tns:SaveSelectionRequest"/>
  <output message="tns:SaveSelectionResponse"/>
 </operation>
 <operation name="DeleteSelection">
  <documentation>Service definition of function
ax__DeleteSelection</documentation>
  <input message="tns:DeleteSelectionRequest"/>
  <output message="tns:DeleteSelectionResponse"/>
 </operation>
 <operation name="ActualizeSelection-sync">
  <documentation>Service definition of function
ax__ActualizeSelection_sync</documentation>
  <input message="tns:ActualizeSelection-syncRequest"/>
  <output message="tns:ActualizeSelection-syncResponse"/>
 </operation>
 <operation name="ActualizeSelection-async">
  <documentation>Service definition of function
ax__ActualizeSelection_async</documentation>
  <input message="tns:ActualizeSelection-asyncRequest"/>
  <output message="tns:ActualizeSelection-asyncResponse"/>
 </operation>
</portType>

<binding name="Selection_Archive" type="tns:Selection_ArchivePortType">
 <SOAP:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
 <operation name="ListUserSelection">
  <SOAP:operation soapAction=""/>
  <input>
   <SOAP:body use="literal"/>
  </input>
  <output>
   <SOAP:body use="literal"/>
  </output>
 </operation>
 <operation name="ListEntitledSelection">
  <SOAP:operation soapAction=""/>
  <input>
   <SOAP:body use="literal"/>
  </input>
  <output>
   <SOAP:body use="literal"/>
  </output>
 </operation>
 <operation name="LoadSelection">
  <SOAP:operation soapAction=""/>
  <input>
   <SOAP:body use="literal"/>
  </input>
  <output>
   <SOAP:body use="literal"/>
  </output>
 </operation>
 <operation name="SaveSelection">
  <SOAP:operation soapAction=""/>
  <input>
   <SOAP:body use="literal"/>
  </input>
```

```
   <output>
    <SOAP:body use="literal"/>
   </output>
  </operation>
  <operation name="DeleteSelection">
   <SOAP:operation soapAction=""/>
   <input>
    <SOAP:body use="literal"/>
   </input>
   <output>
    <SOAP:body use="literal"/>
   </output>
  </operation>
  <operation name="ActualizeSelection-sync">
   <SOAP:operation soapAction=""/>
   <input>
    <SOAP:body use="literal"/>
   </input>
   <output>
    <SOAP:body use="literal"/>
   </output>
  </operation>
  <operation name="ActualizeSelection-async">
   <SOAP:operation soapAction=""/>
   <input>
    <SOAP:body use="literal"/>
   </input>
   <output>
    <SOAP:body use="literal"/>
   </output>
  </operation>
 </binding>

<service name="Selection_Archive">
  <documentation>gSOAP 2.7.0e generated service definition</documentation>
  <port name="Selection_Archive" binding="tns:Selection_Archive">
   <SOAP:address location="http://www.axmedis.org/selection_archive.cgi"/>
  </port>
</service>

</definitions>
```

Loader:

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:s="http://www.w3.org/2001/XMLSchema" xmlns:s0="urn:ax"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns="http://tempuri.org/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
targetNamespace="http://tempuri.org/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:types>
    <s:schema elementFormDefault="qualified" targetNamespace="urn:ax">
      <s:element name="checkout-asyncElement" type="s0:checkoutasyncElement" />
      <s:complexType name="checkoutasyncElement">
        <s:sequence>
          <s:element minOccurs="1" maxOccurs="1" name="AXOID" nillable="true"
type="s:string" />
```

```
          <s:element minOccurs="1" maxOccurs="1" name="version" type="s:int" />
          <s:element minOccurs="1" maxOccurs="1" name="User" nillable="true"
type="s:string" />
          <s:element minOccurs="1" maxOccurs="1" name="Password" nillable="true"
type="s:string" />
          <s:element minOccurs="1" maxOccurs="1" name="CheckoutListenerService"
nillable="true" type="s:anyURI" />
          <s:element minOccurs="1" maxOccurs="1" name="ListenerID"
nillable="true" type="s:string" />
        </s:sequence>
      </s:complexType>
      <s:element name="checkout-asyncResponse" type="s0:checkoutasyncResponse"
/>
      <s:complexType name="checkoutasyncResponse">
        <s:sequence>
          <s:element minOccurs="1" maxOccurs="1" name="result" type="s:boolean"
/>
        </s:sequence>
      </s:complexType>
    </s:schema>
  </wsdl:types>
  <wsdl:message name="checkoutasyncSoapIn">
    <wsdl:part name="checkoutasyncElement" element="s0:checkout-asyncElement" />
  </wsdl:message>
  <wsdl:message name="checkoutasyncSoapOut">
    <wsdl:part name="checkoutasyncResult" element="s0:checkout-asyncResponse" />
  </wsdl:message>
  <wsdl:portType name="LoaderServiceSoap">
    <wsdl:operation name="checkoutasync">
      <wsdl:input message="tns:checkoutasyncSoapIn" />
      <wsdl:output message="tns:checkoutasyncSoapOut" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="LoaderServiceSoap" type="tns:LoaderServiceSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="checkoutasync">
      <soap:operation soapAction="" style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="LoaderServiceSoap12" type="tns:LoaderServiceSoap">
    <soap12:binding transport="http://schemas.xmlsoap.org/soap/http"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/" />
    <wsdl:operation name="checkoutasync">
      <soap12:operation soapAction="" style="document"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/" />
      <wsdl:input>
        <soap12:body use="literal"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/" />
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="LoaderService">
    <wsdl:port name="LoaderServiceSoap" binding="tns:LoaderServiceSoap">
      <soap:address location="http://localhost/Loader/Loader.asmx" />
    </wsdl:port>
    <wsdl:port name="LoaderServiceSoap12" binding="tns:LoaderServiceSoap12">
      <soap12:address location="http://localhost/Loader/Loader.asmx"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Saver:

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:s="http://www.w3.org/2001/XMLSchema" xmlns:s0="urn:ax"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns="http://tempuri.org/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
targetNamespace="http://tempuri.org/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:types>
    <s:schema elementFormDefault="qualified" targetNamespace="urn:ax">
      <s:element name="commit-asyncElement" type="s0:commitasyncElement" />
      <s:complexType name="commitasyncElement">
        <s:sequence>
          <s:element minOccurs="1" maxOccurs="1" name="URI" nillable="true"
type="s:anyURI" />
          <s:element minOccurs="1" maxOccurs="1" name="User" nillable="true"
type="s:string" />
          <s:element minOccurs="1" maxOccurs="1" name="Password" nillable="true"
type="s:string" />
          <s:element minOccurs="1" maxOccurs="1" name="CommitListenerService"
nillable="true" type="s:anyURI" />
          <s:element minOccurs="1" maxOccurs="1" name="ListenerID"
nillable="true" type="s:string" />
        </s:sequence>
      </s:complexType>
      <s:element name="commit-asyncResponse" type="s0:commitasyncResponse" />
      <s:complexType name="commitasyncResponse">
        <s:sequence>
          <s:element minOccurs="1" maxOccurs="1" name="result" type="s:boolean"
/>
        </s:sequence>
      </s:complexType>
    </s:schema>
  </wsdl:types>
  <wsdl:message name="commitasyncSoapIn">
    <wsdl:part name="commitasyncElement" element="s0:commit-asyncElement" />
  </wsdl:message>
  <wsdl:message name="commitasyncSoapOut">
    <wsdl:part name="commitasyncResult" element="s0:commit-asyncResponse" />
  </wsdl:message>
  <wsdl:portType name="SaverServiceSoap">
    <wsdl:operation name="commitasync">
      <wsdl:input message="tns:commitasyncSoapIn" />
      <wsdl:output message="tns:commitasyncSoapOut" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="SaverServiceSoap" type="tns:SaverServiceSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="commitasync">
      <soap:operation soapAction="" style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="SaverServiceSoap12" type="tns:SaverServiceSoap">
    <soap12:binding transport="http://schemas.xmlsoap.org/soap/http"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap120u34 ? />
    <wsdl:operation name="commitasync">
      <soap12:operation soapAction="" style="document"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/" />
      <wsdl:input>
        <soap12:body use="literal"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/" />
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="SaverService">
    <wsdl:port name="SaverServiceSoap" binding="tns:SaverServiceSoap">
      <soap:address location="http://localhost/Saver/Saver.asmx" />
    </wsdl:port>
    <wsdl:port name="SaverServiceSoap12" binding="tns:SaverServiceSoap12">
      <soap12:address location="http://localhost/Saver/Saver.asmx"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

28 Formal description of communication protocol Workflow Database Response Channel (axWfDbRes) Webservice

Call name	
Method	DBNotification
Description	Used to send notifications back to workflow for asynchronous methods
Input parameters	AXRQID, Result, ErrorMsg, ErrorCode, AXOID, Version, DownloadURI, URI
Output parameters	Result, FaultCode, FaultMSG

Request Sample Message	`<message name="DBNotificationRequest">`
	`<part name="AXRQID" element="s0:AXRQID"/>`
	`<part name="Result" element="s0:Result"/>`
	`<part name="ErrorMsg" element="s0:ErrorMsg"/>`
	`<part name="ErrorCode" element="s0:ErrorCode"/>`
	`<part name="AXOID" element="s0:AXOID"/>`
	`<part name="Version" element="s0:Version"/>`

	`<part name="DownloadURI" element="s0:URI"/>`
	`<part name="URI" element="s0:URI"/>`
	`</message>`

Response Sample Message	`<xs:element name="NotificationResult">`
	`<xs:complexType>`
	`<xs:sequence>`
	`<xs:element name="Result" type="xs:boolean"/>`
	`<xs:element name="FaultCode" type="xs:string" minOccurs="0"/>`
	`<xs:element name="FaultString" type="xs:string" minOccurs="0"/>`
	`</xs:sequence>`
	`</xs:complexType>`
	`</xs:element>`

Workflow Database Response Channel

Load Listener:

```
<?xml version="1.0" encoding="UTF-8"?><definitions
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="http://www.someone.org/listener.wsdl" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:ax="urn:ns/ax.xsd"
xmlns:ns="urn:ns" xmlns:SOAP="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:MIME="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:DIME="http://schemas.xmlsoap.org/ws/2002/04/dime/wsdl/"
xmlns:WSDL="http://schemas.xmlsoap.org/wsdl/" name="CkeckoutListener"
targetNamespace="http://www.someone.org/listener.wsdl">
<types>
<schema xmlns="http://www.w3.org/2001/XMLSchema" targetNamespace="urn:ns/ax.xsd"
elementFormDefault="qualified" attributeFormDefault="qualified">
<import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
<complexType name="checkout-result">
<sequence>
<element name="result" type="xsd:boolean" minOccurs="1" maxOccurs="1"/>
<element name="downloadURI" type="xsd:anyURI" minOccurs="0" maxOccurs="1"
nillable="true"/>
<element name="errormsg" type="xsd:string" minOccurs="0" maxOccurs="1"
nillable="true"/>
<element name="errorcode" type="xsd:int" minOccurs="1" maxOccurs="1"/>
</sequence>
</complexType>
<complexType name="co">
<sequence>
<element name="return" type="ax:checkout-result" minOccurs="1" maxOccurs="1"/>
</sequence>
</complexType>
</schema>
<schema xmlns="http://www.w3.org/2001/XMLSchema" targetNamespace="urn:ns"
elementFormDefault="qualified" attributeFormDefault="qualified">
<import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
<!-- operation request element -->
<element name="checkout-Listener">
```

```
<complexType>
<sequence>
<element name="result" type="ax:co" minOccurs="1" maxOccurs="1"/>
<element name="ListenerID" type="xsd:string" minOccurs="0" maxOccurs="1"
nillable="true"/>
</sequence>
</complexType>
</element>
<!-- operation response element -->
<element name="checkout-ListenerResponse">
<complexType>
<sequence>
<element name="r" type="xsd:boolean" minOccurs="1" maxOccurs="1"/>
</sequence>
</complexType>
</element>
</schema>
</types>
<message name="checkout-ListenerRequest">
<part name="parameters" element="ns:checkout-Listener"/>
</message>
<message name="checkout-ListenerResponse">
<part name="parameters" element="ns:checkout-ListenerResponse"/>
</message>
<portType name="CkeckoutListenerPortType">
<operation name="checkout-Listener">
<documentation>Service definition of function
ns__checkout_Listener</documentation>
<input message="tns:checkout-ListenerRequest"/>
<output message="tns:checkout-ListenerResponse"/>
</operation>
</portType>
<binding name="CkeckoutListener" type="tns:CkeckoutListenerPortType">
<SOAP:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
<operation name="checkout-Listener">
<SOAP:operation soapAction=""/>
<input>
<SOAP:body use="literal"/>
</input>
<output>
<SOAP:body use="literal"/>
</output>
</operation>
</binding>
<wsdl:service xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" name="loaderListener">
<wsdl:port name="CkeckoutListenerPortTypePort" binding="tns:CkeckoutListener">
<soap:address location="http://87.24.151.2:8080/LoaderSaver/ws/CheckuotPort"/>
</wsdl:port>
</wsdl:service>
</definitions>
```

Save Listener:

```
<?xml version="1.0" encoding="UTF-8"?><definitions
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="http://www.someone.org/listener.wsdl" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-
```

```
ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:ax="urn:ns/ax.xsd"
xmlns:ns="urn:ns" xmlns:SOAP="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:MIME="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:DIME="http://schemas.xmlsoap.org/ws/2002/04/dime/wsdl/"
xmlns:WSDL="http://schemas.xmlsoap.org/wsdl/" name="CommitListener"
targetNamespace="http://www.someone.org/listener.wsdl">

<types>

 <schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="urn:ns/ax.xsd" elementFormDefault="qualified"
attributeFormDefault="qualified">
  <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
  <complexType name="sync-result">
   <sequence>
     <element name="result" type="xsd:boolean" minOccurs="1" maxOccurs="1"/>
     <element name="version" type="xsd:int" minOccurs="1" maxOccurs="1"/>
     <element name="errormsg" type="xsd:string" minOccurs="0" maxOccurs="1"
nillable="true"/>
     <element name="errorcode" type="xsd:int" minOccurs="1" maxOccurs="1"/>
   </sequence>
  </complexType>
  <complexType name="getSyncResult">
   <sequence>
     <element name="return" type="ax:sync-result" minOccurs="1" maxOccurs="1"/>
   </sequence>
  </complexType>
 </schema>

 <schema xmlns="http://www.w3.org/2001/XMLSchema" targetNamespace="urn:ns"
elementFormDefault="qualified" attributeFormDefault="qualified">
  <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
  <!-- operation request element -->
  <element name="commit-Listener">
   <complexType>
    <sequence>
     <element name="result" type="ax:getSyncResult" minOccurs="1"
maxOccurs="1"/>
     <element name="ListenerID" type="xsd:string" minOccurs="0" maxOccurs="1"
nillable="true"/>
    </sequence>
   </complexType>
  </element>
  <!-- operation response element -->
  <element name="commit-ListenerResponse">
   <complexType>
    <sequence>
     <element name="r" type="xsd:boolean" minOccurs="1" maxOccurs="1"/>
    </sequence>
   </complexType>
  </element>
 </schema>

</types>

<message name="commit-ListenerRequest">
 <part name="parameters" element="ns:commit-Listener"/>
</message>
```

```
<message name="commit-ListenerResponse">
 <part name="parameters" element="ns:commit-ListenerResponse"/>
</message>

<portType name="CommitListenerPortType">
 <operation name="commit-Listener">
  <documentation>Service definition of function
ns__commit_Listener</documentation>
  <input message="tns:commit-ListenerRequest"/>
  <output message="tns:commit-ListenerResponse"/>
 </operation>
</portType>

<binding name="CommitListener" type="tns:CommitListenerPortType">
 <SOAP:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
 <operation name="commit-Listener">
  <SOAP:operation soapAction=""/>
  <input>
   <SOAP:body use="literal"/>
  </input>
  <output>
   <SOAP:body use="literal"/>
  </output>
 </operation>
</binding>

<service name="CommitListener">
 <documentation>gSOAP 2.7.0e generated service definition</documentation>
 <port name="CommitListener" binding="tns:CommitListener">
  <soap:address location="http://87.24.151.2:8080/LoaderSaver/commitlist"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"/>
 </port>
</service>

</definitions>
```

Selection Archive Listener:

```
<?xml version="1.0" encoding="UTF-8"?><definitions
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="http://www.someone.org/actualizelistener.wsdl" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:ns="urn:ns"
xmlns:SOAP="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:MIME="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:DIME="http://schemas.xmlsoap.org/ws/2002/04/dime/wsdl/"
xmlns:WSDL="http://schemas.xmlsoap.org/wsdl/" name="ActualizeListener"
targetNamespace="http://www.someone.org/actualizelistener.wsdl">
      <types>
            <schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="urn:ns" elementFormDefault="qualified"
attributeFormDefault="qualified">
                  <import
namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
                  <!-- operation request element -->
                  <element name="Actualize-Listener">
                        <complexType>
```

```
                              <sequence>
                                    <element name="actualizedSelection"
type="xsd:string" minOccurs="1" maxOccurs="1"/>
                                    <element name="ListenerID" type="xsd:string"
minOccurs="1" maxOccurs="1"/>
                              </sequence>
                        </complexType>
                  </element>
                  <!-- operation response element -->
                  <element name="Actualize-ListenerResponse">
                        <complexType>
                              <sequence>
                                    <element name="r" type="xsd:boolean"
minOccurs="1" maxOccurs="1"/>
                              </sequence>
                        </complexType>
                  </element>
            </schema>
      </types>
      <message name="Actualize-ListenerRequest">
            <part name="parameters" element="ns:Actualize-Listener"/>
      </message>
      <message name="Actualize-ListenerResponse">
            <part name="parameters" element="ns:Actualize-ListenerResponse"/>
      </message>
      <portType name="ActualizeListenerPortType">
            <operation name="Actualize-Listener">
                  <documentation>Service definition of function
ns__Actualize_Listener</documentation>
                  <input message="tns:Actualize-ListenerRequest"/>
                  <output message="tns:Actualize-ListenerResponse"/>
            </operation>
      </portType>
      <binding name="ActualizeListener" type="tns:ActualizeListenerPortType">
            <SOAP:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
            <operation name="Actualize-Listener">
                  <SOAP:operation soapAction=""/>
                  <input>
                        <SOAP:body use="literal"/>
                  </input>
                  <output>
                        <SOAP:body use="literal"/>
                  </output>
            </operation>
      </binding>
      <service name="ActualizeListener">
            <documentation>gSOAP 2.7.0e generated service
definition</documentation>
            <port name="ActualizeListener" binding="tns:ActualizeListener">
                  <soap:address
location="http://87.24.151.2:8080/UserSelectionArchive/listener"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"/>
            </port>
      </service>
</definitions>
```

29 Bibliography (mandatory)

--- this section is mandatory in all deliverables ---

29.1 Openflow Workflow Management System

- http://www.openflow.it/EN/Documentation/EN/Documentation
- http://www.openflow.it/EN/index_html

29.2 AXMEDIS Technical Watch Area Workflow Folder

- http://www.AXMEDIS.org/attivita/documenti/download.php?area_id=4&attivita_id=8&l_s=struttura&gruppo=75&order_by=data&asc_desc=desc

29.2.1 Downloads

- Downloads from made available in the above folder:
- Openflow (introduction, Examples);
- Zope (Reference, Package, Code, Developer, PostgreSQL);
- Python (source code, Library, Python for Windows);
- XMLRPC C++ Library

As follows:

1. openflow.1.2.0.tgz
2. zope-2.7.3-0-win32.exe
3. xmirpclib-1.0.1.zip
4. xmirpc++0.7.zip
5. leave.zexp
6. zopebook-2_6.pdf
7. psycopgda-1.0.0.tgz
8. introduzione_a_openflow_doc[1].doc
9. devguide-2_4.pdf
10. python-2.3.4.exe
11. zope-2.7.3-0.gz
12. python-2.3.4.tar
13. html-2.3.4.zip

29.3 Workflow Management Coalition(WfMC)

- http://www.wfmc.org/standards/model.htm
- http://www.wfmc.org/standards/wfxml_demo.htm
- http://www.wfmc.org/standards/XPDL.htm
- http://www.wfmc.org/information/awards.htm
- http://www.wfmc.org/standards/conformance.htm

29.4 Workflow and Re-engineering International Association (waria)

- http://www.waria.com/
- http://www.waria.com/books/study-volume3.htm

29.5 XPDL

XML Process Definition Language (XPDL) of the Workflow Management Coalition (WfMC)

http://www.wfmc.org/standards/docs.htm#Interface%201%20-%20Process%20Definition%20Interchange

29.6 Microsoft Biztalk Server 2004 Documentation

Biztalk Server 2004, introduction document

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/BTS_2004WP/html/c245a8af-9f01-410f-b1bc-c43e725bfc27.asp

Biztalk Server 2004, Human Workflow Service Description

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/sdk/htm/ebiz_prog_hws_frea.asp

Biztalk Server 2004, Web Service Support Description

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/BTS_2004WP/html/c245a8af-9f01-410f-b1bc-c43e725bfc27.asp

30 Glossary (mandatory)

30.1 Glossary of Terms used with Workflow Management Systems Integration

Term	**Meaning**
Object	AXMEDIS (MPEG21) object
Object_ID (AXOID)	A unique identifier for reference to a particular AXMEDIS object
New Product Development (NPD)	Any project or new workflow instance set up to add value by way of creating afresh, and/or modifying existing, digital assets; usually referred to as NPD
Workflow (WF)	A workflow consists of process logic and routing rules. The process logic defines the sequence of tasks and the routing rules that must be followed, as well as deadlines plus business rules implemented by the workflow engine.
Workflow Management System (WFMS)	A software application that stores process definitions and runs jobs based on those process definitions via its workflow engine component. The workflow engine is the runtime execution module.
Instance (IN)	A particular realisation of an Activity in the context of a particular workitem, workflow-instance or workspace-instance.
Instance_ID (INID)	A unique Identifier for reference to a particular Instance.
Workitem	Represents the actual work to be done on an object by a participant for an activity in a process instance.
Workitem_Object	A version/derivative of an AXMEDIS Object or newly created and currently

Term	Meaning
	still being worked-on/modified in the course of an NPD within a process instance or workspace-instance. This is before the resulting new object can exit development and be possibly registerable as a new AXMEDIS Object.
Workitem_ID (AXWID)	A unique identifier for reference to a particular workitem Object
Workitem_Instance	A reference to a particular version or genre of a workitem amongst several derived from the a single workitem that are undergoing various activities within the same workflow instance
Workitem_Instance_ID	A unique identifier for reference to a particular workitem-instance
Workflow_Instance	A particular set of coordinated processes set up and encoded to model an activity network required to deliver the objectives of any project or NPD according to the business rules and resources as applicable to that NPD in a given business environment.
Workflow_Instance_ID	A unique identifier for reference to a particular workflow-instance
Workspace	This is a normal practice template or an abstract reference to a set of participating entities typically required to be involved in the execution of a particular activity per rules and resources available to a particular business/sector/community of practice.
Workspace_Instance	Or equivalently a workitem (as in Openflow) is referred to as a dynamic data structure recording and indexing the set of participants (actors, tools, objects, resources) and their states involved in any given workflow instance/process flow instance/workitem. Typically this is a node in a graph oriented data structure acting as a pointer to such entities and states.
Workspace_Instance_ID	A unique identifier for reference to a particular workspace instance.
Process definition	A graphical process definition, or process map, represents the process logic elements of a workflow and their relationships.
Process_Instance	A process instance, commonly called a job, is a running instance of a process definition.
Process_Instance_ID	A unique identifier for reference to a particular process instance.
Processflow	This is a part of the workflow that represents a set of coordinated activities encoded to map a subspace of the activity network consistently according to the available rules and resources of the particular NPD. In this way a workflow can be equivalent or a superset of a processflow.
Processflow_Instance	This is a particular realisation of a processflow that actually belongs to a workflow instance i.e. a particularised version of the workflow subspace as instantiated in the context of a given work item/workspace instance.
Processflow_Instance_ID	A unique identifier for reference to a particular processflow instance.
Session	A session is any event marked by a finite period with at least a start time whereby some user or proxy undertakes the execution of an activity or a set of activities in the context of a given workitem.
Session_ID	A unique identifier for reference to a particular a session.
User_Credentials_ID	An identifier that includes at least the user's identity and their current session identity often in the context of single sign on and or session traceability facilitation.
AXRQID	A composite parameter as a subtype of WF-Exchange_ID that can be passed between the AXWF and relevant AXMEDIS components to help identify a particular WF-Exchange_ID.
Response_ID	A composite parameter as a subtype of WF-Exchange_ID that can be passed from any relevant AXMEDIS components to help identify a particular WF-Exchange_ID as a response to a particular AXRQID.
Exchange_Instance	A particular exchange messaging act or communication between any relevant AXMEDIS components and AXWF within a workflow-instance.
Exchange_Instance_ID	A unique identifier that encompasses all required Exchange data parameters and messaging as required for a single communication to or from any relevant

Term	Meaning
	AXMEDIS components and the AXWF.
Bridge	An abstract container of all the transactions required between the AXWF and relevant AXMEDIS components during the lifecycle of each workitem/processflow-instance/workflow-instance/session.
Bridge_Instance	This is a specific bridge belonging to a real session/workflow instance/processflow-instance such that the bridge instance includes all the exchanges i.e. WF-Exchange_IDs that occurred between the particular workspace instance owner and the relevant AXMEDIS components.
Bridge_Instance_ID	This is a unique identifier for reference to a particular bridge-instance.
Adopted AXMEDIS Workflow System (AXWF)	Third party workflow management systems to be adopted for integration with the AXMEDIS Framework and referred to as AXWF.
Process Definition Tool	A software tool used to create and change process definitions. The tool may be a component of business process management software, a stand-alone application, or a component of a workflow management system. Process definition tools that provide the ability to re-use stored workflow elements, and even entire subprocesses, make workflow application developers more productive since they avoid reinventing the wheel when building workflows and integrating them with other applications.
Participant	One of the following types: a resource set, a specific resource, an organisational unit, a role (a function of a human within an organisation), a human, or a system (an automatic agent). Answers the question "Who?" in a business process.
Activity	A task that forms one logical step in a process definition. Can be automated or manual. Automation refers to the ability to define scripts and triggers during process operation. Specific activities in the process definition can run as unattended tasks, and automation can enforce business rules in manual, or human-driven, tasks. A common type of automated activity is deadline handling, which can automatically send a reminder message or trigger an escalation procedure if a workitem fails to be completed by a prescribed deadline.
Activity Owner	An activity owner is the participant that has the authority to declare an activity complete, thus forwarding the work to the next activity in the process.
Job Owner	A job owner is a participant that has overall control of the execution of one process instance

30.2 Table of Acronyms Relevant to Workflow Management Systems Integration

Term/Acronym	Meaning
WFMS	Any Workflow Management System
AXWFM	AXMEDIS Workflow Manager.
AXWF-DXF	AXMEDIS Workflow Data Exchange Format
QSWSI	AXMEDIS Query Support Web Services Interface
XPDL:	XML Process Definition Language (XPDL) of the Workflow Management Coalition (WfMC)
WF-XML	Wf-XML and Workflow Reference Model from the Workflow Management Coalition (WfMC): Wf-XML is an XML-based encoding of workflow interoperability messages. The Workflow Reference Model is a description of the underlying workflow system architecture. Wf-XML has no binding to SOAP and WSDL at this time.

WSFL	IBM Web Services Flow Language: Specifies two types of Web services composition 1) an executable business process known as a flowModel, and 2) a business collaboration known as a globalModel. Compatible with SOAP, UDDI, and WSDL.
XLANG	Microsoft's XLANG: Business modeling language for BizTalk, which is a component of .NET that enables EAI. BizTalk Orchestration is the workflow engine and BizTalk Orchestration Designer is the visual business process modelling tool based on XLANG.
BPEL4WS	Business Process Execution Language for Web Services is the cooperative merging of WSFL and XLANG for Web services orchestration, workflow, and composition. It has not yet been submitted to an IT standards organisation.
ebXML BPSS	The eBusiness Transition Working Group carries forward the definition of workflow conversation and orchestration in the Business Process Specification Schema (BPSS) layer of ebXML, which defines many protocols and layers for XML-based e-business.
WSCI	Sun/BEA/Intalio/SAP consortium's Web Services Choreography Interface "is an XML-based interface description language that describes the flow of messages exchanged by a Web Service participating in choreographed interactions with other services."
WSCL	W3C's Web Services Conversation Language: A submission by Hewlett-Packard to the W3C, it allows defining the abstract interfaces of Web services (that is, the business level conversations or public processes supported by a Web service), the XML documents being exchanged, and the sequencing of those documents.
PIPs	RosettaNet's Partner Interface Process: defines business processes between trading partners via specialized system-to-system XML-based dialogs. Many PIPs have been defined for various partner scenarios.
JDF	CIP4's Job Definition Format is an upcoming workflow industry standard for the Graphics Arts industry designed to simplify information exchange among different applications and systems.