



**Automating Production of Cross Media Content
for Multi-channel Distribution**
www.AXMEDIS.org

DE3.1.2G
Framework and Tools Specifications
(Workflow)

Version: 6.6

Date: 14/03/2005

Responsible: IRC (DSI supervision)

Project Number: IST-2-511299

Project Title: AXMEDIS

Deliverable Type: report

Visible to User Groups: no

Visible to Affiliated: no

Visible to the Public: no.

Deliverable Number: DE3.1.2 Part G

Contractual Date of Delivery: January 2005

Actual Date of Delivery: 15 March 2005

Title of Deliverable: Document

Work-Package contributing to the Deliverable: WP3.1

Task contributing to the Deliverable: WP3, WP2

Nature of the Deliverable: report

Author(s): IRC, HP, XIM

Abstract: This Part G broadly comprises five parts. The first part supports the rationale for the four distinct categories of interfaces deemed required for a streamlined architecture; i.e. workflow plug-ins with AXMEDIS Editor, Engines, Rule Editor/Viewers and Query Support Interface. It describes a Neutral Exchange Format that has been devised for the plug-ins as a logical transaction format to serve as a generic standard that remains both AXMEDIS-compliant as well as technologically realisable and consistent with the capabilities of existing workflow systems given their available method invocations and transaction protocols.

The semantic analysis that is also included recommends a set of Object Status Types for consideration re design of the AXMEDIS Object Schema. The AXMEDIS workflow integration specification is then further illustrated by means of UML diagrams, tables of transactions syntax and protocol specifications as well as eight generic scenario specification diagrams that are fully described.

The second part further specifies the design of the workflow plug-ins including actual input, output and method invocations for all the workflow plug-ins based on the exchange format for the interfaces as established in the first part. The third part gives an outline of the rationale for the choice of the two selected workflow systems (i.e. the open-source workflow, *Openflow*, and the proprietary workflow system *BizTalk*). The final part includes an overview of the structure and terminology of the selected workflow systems as well as references for the relevant tools, engine downloads and further information including a Glossary and References.

Keyword List workflow, interface, plug-in, neutral exchange format, protocol, semantic types, Object status, tracking, metadata, session, bridge, workspace, open-source, Openflow, BizTalk, API, xml_rpc, dll, dcom, Zope, Python

AXMEDIS Copyright Notice

The following terms (including future possible amendments) set out the rights and obligations licensee will be requested to accept on entering into possession of any official AXMEDIS document either by downloading it from the web site or by any other means.

Any relevant AXMEDIS document includes this license. PLEASE READ THE FOLLOWING TERMS CAREFULLY AS THEY HAVE TO BE ACCEPTED PRIOR TO READING/USE OF THE DOCUMENT.

1. DEFINITIONS

- i. **"Acceptance Date"** is the date on which these terms and conditions for entering into possession of the document have been accepted.
- ii. **"Copyright"** stands for any content, document or portion of it that is covered by the copyright disclaimer in a Document.
- iii. **"Licensor"** is AXMEDIS Consortium as a de-facto consortium of the EC project and any of its derivations in terms of companies and/or associations, see www.axmedis.org
- iv. **"Document"** means the information contained in any electronic file, which has been published by the Licensor's as AXMEDIS official document and listed in the web site mentioned above or available by any other means.
- v. **"Works"** means any works created by the licensee, which reproduce a Document or any of its part.

2. LICENCE

1. The Licensor grants a non-exclusive royalty free licence to reproduce and use the Documents subject to present terms and conditions (the **Licence**) for the parts that are own and proprietary property the of AXMEDIS consortium or its members.
2. In consideration of the Licensor granting the Licence, licensee agrees to adhere to the following terms and conditions.

3. TERM AND TERMINATION

1. Granted Licence shall commence on Acceptance Date.
2. Granted Licence will terminate automatically if licensee fails to comply with any of the terms and conditions of this Licence.
3. Termination of this Licence does not affect either party's accrued rights and obligations as at the date of termination.
4. Upon termination of this Licence for whatever reason, licensee shall cease to make any use of the accessed Copyright.
5. All provisions of this Licence, which are necessary for the interpretation or enforcement of a party's rights or obligations, shall survive termination of this Licence and shall continue in full force and effect.
6. Notwithstanding License termination, confidentiality clauses related to any content, document or part of it as stated in the document itself will remain in force for a period of 5 years after license issue date or the period stated in the document whichever is the longer.

4. USE

1. Licensee shall not breach or denigrate the integrity of the Copyright Notice and in particular shall not:
 - i. remove this Copyright Notice on a Document or any of its reproduction in any form in which those may be achieved;
 - ii. change or remove the title of a Document;
 - iii. use all or any part of a Document as part of a specification or standard not emanating from the Licensor without the prior written consent of the Licensor; or
 - iv. do or permit others to do any act or omission in relation to a Document which is contrary to the rights and obligations as stated in the present license and agreed with the Licensor

5. COPYRIGHT NOTICES

1. All Works shall bear a clear notice asserting the Licensor's Copyright. The notice shall use the wording employed by the Licensor in its own copyright notice unless the Licensor otherwise instructs licensees.

6. WARRANTY

1. The Licensor warrants the licensee that the present licence is issued on the basis of full Copyright ownership or re-licensing agreements granting the Licensor full licensing and enforcement power.
2. For the avoidance of doubt the licensee should be aware that although the Copyright in the documents is given under warranty this warranty does not extend to the content of any document which may contain references or specifications or technologies that are covered by patents (also of third parties) or that refer to other standards. AXMEDIS is not responsible and does not guarantee that the information contained in the document is fully proprietary of AXMEDIS consortium and/or partners.
3. Licensee hereby undertakes to the Licensor that he will, without prejudice to any other right of action which the Licensor may have, at all times keep the Licensor fully and effectively indemnified against all and any liability (which liability shall include, without limitation, all losses, costs, claims, expenses, demands, actions, damages, legal and other professional fees and expenses on a full indemnity basis) which the Licensor may suffer or incur as a result of, or by reason of, any breach or non-fulfilment of any of his obligations in respect of this Licence.

7. INFRINGEMENT

1. Licensee undertakes to notify promptly the Licensor of any threatened or actual infringement of the Copyright which comes to licensee notice and shall, at the Licensor's request and expense, do all such things as are reasonably necessary to defend and enforce the Licensor's rights in the Copyright.

8. GOVERNING LAW AND JURISDICTION

1. This Licence shall be subject to, and construed and interpreted in accordance with Italian law.
2. The parties irrevocably submit to the exclusive jurisdiction of the Italian Courts.

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 (DSI, ALL)	7
2	WORKFLOW MANAGEMENT AREA (WP4.3.3. IRC, WP5.5.5: CRS4, XIM)	8
2.1	OUTLINE OF THE WORKFLOW MANAGEMENT SYSTEM FUNCTIONAL REQUIREMENTS	9
2.2	WORKFLOW INTEGRATION DESIGN	10
2.3	ADOPTION OF A NEUTRAL EXCHANGE MESSAGING FORMAT: THE SEMANTIC FRAMEWORK	11
2.3.1	Objects of Type Workitem	11
2.3.2	Objects of Type Workflow (Processflow)	11
2.3.3	Objects of Type Task	12
2.3.4	Objects of Type Session	12
2.3.5	Objects of Type User_Credentials	12
2.3.6	Objects of Type Workspace	12
2.3.7	Objects of Type Bridge	13
2.3.8	Objects of Type Exchange	13
2.3.9	Logical Analysis of Semantics and Syntax of the WF-Exchange_ID	15
2.3.10	Sufficiency and Necessity Criteria for the Design of Exchange Messaging Format	35
2.4	WORKFLOW TRANSACTIONS SPECIFICATION FOR ALL INTERFACES AND SELECTED SCENARIOS	36
3	WORKFLOW DATABASE	45
3.1	SEMANTIC ANALYSIS OF STATUS DATA REQUIRED FOR OBJECT TRACKING AND AXWFM CONTROL	45
3.2	SEMANTIC TYPES AND REPRESENTATION SPACES PARTITIONING	47
3.3	ESTABLISHING METADATA/DESCRIPTOR AXINFO, AXWFDB, AXDB & PMSDB DATA SCHEMA COMPLEMENTARITY	48
3.3.1	Tracking/ Messaging and Control Semantics To be Represented	49
3.3.2	Representing Projected Work-To-be-done on an (AXMEDIS)Object	50
3.3.3	Examples of Integrative Database Inference Types to be made possible	50
3.4	CHOICE OF WORKFLOW DATABASE TECHNOLOGY	51
3.5	THE SEMANTIC ELEMENTS TO BE STORED IN THE WORKFLOW DATABASE (AXWFDB)	51
3.5.1	Workflow-Embedded / Triggered Object Retrieval and Discovery Service Spaces	53
3.5.2	Unknown Objects Sought & Found Exchange Messaging Contexts	54
3.6	GENERIC SCENARIOS FOR THE WORKFLOW INTERACTION WITH AXMEDIS COMPONENTS	55
3.7	AXMEDIS WORKFLOW AREA UML DECOMPOSITION	66
3.8	WORKFLOW AND AXMEDIS INTEGRATION ARCHITECTURE	67
3.9	WORKFLOW TECHNOLOGIES	72
3.10	WORKFLOW ENGINE	72
3.11	WORKFLOW USER INTERFACES AND TOOLS (IRC, HP, XIM)	73
3.11.1	Typical Transactions between AXWF and AXMEDIS in Delivering the UseCases	74
3.11.2	WorkFlow User Interface Mock-ups (OpenFlow)	77
4	WORKFLOW REQUEST ADAPTERS (OPENFLOW)	77
4.1	WORKFLOW INPUT QUEUE ADAPTERS (OPENFLOW)	78
4.2	WORKFLOW GATEWAYS (OPENFLOW)	79
4.3	WORKFLOW GATEWAYS (MICROSOFT BIZTALK)	85
5	AXMEDIS WORKFLOW INTERFACE SPECIFICATIONS	85
5.1	THE AXMEDIS EDITOR WORKFLOW CHANNEL	85
5.1.1	The Interface between the WF Editor Request Gateway and the AXOM Command and Reporting	86
5.1.2	Interface between the WF AXOM Request Adapter and the WF Editor Request Gateway	96
5.1.3	The Interface between the WF AXOM Input Queue Adapter and the WF Editor Response Gateway	97
5.1.4	The Interface between the AXOM Command and Reporting and the WF Editor Response Gateway	98
5.2	THE WORKFLOW ENGINE CHANNEL	100
5.2.1	The Interface between the WF Engine Request Gateway and the Engine Command and Reporting	100
5.2.2	Interface between the WF Engine Request Adapter and the WF Engine Request Gateway	113
5.2.3	The Interface between the WF Engine Input Queue Adapter and the WF Engine Response Gateway	114

5.2.4	The Interface between the Engine Command and Reporting and the WF Engine Response Gateway.....	116
5.3	THE WORKFLOW RULE EDITOR CHANNEL.....	119
5.3.1	The Interface between the WF Rule Editor Request Gateway and the User Command and Reporting....	119
5.3.2	Interface between the WF Rule Editor Request Adapter and the WF Rule Editor Request Gateway	124
5.3.3	The Interface between the WF Rule Editor Input Queue Adapter and the WF Rule Editor Response Gateway	125
5.3.4	The Interface between the User Command and Reporting and the WF Rule Editor Response Gateway .	127
5.4	THE WORKFLOW QUERY AND DATABASE CHANNEL	129
5.4.1	The Interface between the WF DB Request Gateway and the Loader/Saver and Query Support WebService Interface	129
5.4.2	Interface between the WF DB Request Adapter and the WF DB Request Gateway	138
5.4.3	The Interface between the WF DB Input Queue Adapter and the WF DB Response Gateway	139
5.4.4	WebServices exposed by the WF DB Response Gateway to the AXMEDIS Object Loader/Mover and Query Support WebService Interface	140
6	AXMEDIS INTEGRATION WITH AVAILABLE WORKFLOW ENVIRONMENTS.....	142
6.1	AXMEDIS INTEGRATION AND USAGE WITH THE OPEN-SOURCE AXWF: OPENFLOW	142
6.1.1	The Rationale for Choice of Openflow as an AXWF	142
6.1.2	AXMEDIS Tools Interacting with Openflow	143
6.1.3	Openflow as the AXWF Interfacing with the AXMEDIS Components	145
6.1.4	Openflow Structures and Terminology	145
6.2	AXMEDIS INTEGRATION AND USAGE WITH THE PROPRIETARY AXWF: BIZTALK	148
6.2.1	The Rationale for the Choice of BizTalk as an AXWF	149
6.2.2	Outline Description of the BizTalk	149
6.2.3	Overview of BizTalk Architecture	151
6.3	OPEN-SOURCE LICENSING OF AXMEDIS-DEVELOPED WORKFLOW INTEGRATION SOFTWARE	152
7	GLOSSARY OF TERMS AND ABBREVIATIONS (PART G).....	153
7.1	GLOSSARY OF TERMS USED WITH WORKFLOW MANAGEMENT SYSTEMS INTEGRATION	153
7.2	TABLE OF ACRONYMS RELEVANT TO WORKFLOW MANAGEMENT SYSTEMS INTEGRATION	155
	TABLE OF ACRONYMS RELEVANT TO WORKFLOW MANAGEMENT SYSTEMS INTEGRATION.....	156
8	REFERENCES TO WFMS, OPENFLOW, ZOPE, PYTHON ETC AND DOWNLOADS.....	157
8.1	OPENFLOW WORKFLOW MANAGEMENT SYSTEM	157
8.2	AXMEDIS TECHNICAL WATCH AREA WORKFLOW FOLDER.....	157
8.2.1	Downloads.....	157
8.3	WORKFLOW MANAGEMENT COALITION(WPMC)	157
8.4	WORKFLOW AND RE-ENGINEERING INTERNATIONAL ASSOCIATION (WARIA)	158
8.5	XPDL	158

1 Executive Summary and Report Scope (DSI, all)

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

- A. general aspects up to the description of the content model
- B. Viewers and players, including plug ins, etc.
- C. Content Production tools and algorithms
- D. Fingerprint and descriptors algorithms and tools
- E. Database area, query support and Content Crawling from CMS
- F. AXEPTool area, for B2B distribution and Programme and Publication for B2C distribution
- G. Workflow aspects and tools
- H. Protection tools and support, Certification and Supervision and Accounting tools
- I. Distribution tools and AXMEDIS Portal
- J. Definitions, tables, terminology, acronyms, lists, references, links and Appendixes

This documents contains Part G only.

This Workflow Management Area (Part G) of the Frameworks and Tools Document is intended to set out the knowledge engineering basis and conclude the specification design of the workflow interface plug-ins for the relevant AXMEDIS components. This is to ensure that the AXMEDIS platform when integrated with its adopted workflow systems will be able to deliver all the functionalities expected by various user scenarios as outlined within the Requirements Document (multi-sectorial Workflow Requirements Elicitation and Domain Knowledge Analysis).

Part G broadly comprises five parts. The first part, namely the Workflow Management Area Integration Specification, builds on the earlier requirements analysis to support the rationale for the four distinct channels of interfaces deemed required for a streamlined architecture; i.e. workflow interfaces with AXMEDIS Editor, Engines, Rule Editor/Viewers and Query Support Web Services Interface. This first part describes a Neutral Exchange Format that has been devised for the plug-ins as the logical, necessary and sufficient transaction protocol to remain both AXMEDIS-compliant as well as technologically realisable and consistent with the capabilities of existing workflow systems given their available method invocations and transaction protocols. Thus the devised WF-Exchange_ID Messaging Format is expected to serve as a generic standard to be readily adoptable by most workflow systems as potential candidates for integration with the AXMEDIS Framework.

Based on domain transaction semantics, and ontological analysis, this part sets out its analysis of AXMEDIS workflow integration semantic types as well as query types; thus concluding a set of query categories that the Query Support Module and the AXDB are to be able to accommodate and also a set of candidate Object status types for the design of AXMEDIS Object Schema. The AXMEDIS workflow integration specification is then further illustrated by means of UML diagrams, tables of transactions syntax and protocol specifications as well as eight generic scenario specification diagrams that are fully described.

The second part then specifies the actual input, output and method invocations for all the workflow plug-ins based on the exchange format for the interfaces as established in the first part. In this way the plug-ins are specified such that they satisfy all the logical, necessary and sufficient criteria as stipulated earlier, and, are readily realisable with most workflow systems.

The third part gives an outline of the rationale for the choice of the two selected workflow systems (i.e. the open-source workflow, Openflow, and the proprietary workflow system BizTalk). This part describes how the specified plug-in input/output structures could be realised given the native protocol and method invocations in Openflow. The fourth part includes an overview of the structure and terminology of the selected workflow systems. The final part comprises Glossary and References; it concludes with references for relevant tools, engine downloads and further information.

2 Workflow Management Area (WP4.3.3. IRC, WP5.5.5: CRS4, XIM)

AXMEDIS Data Model Schema will define a structure by which it is possible to track the workflow of content and digital items during the production phase. For example, the schema will include standardised (at least within the AXMEDIS project and hopefully within the MPEG-21 multimedia framework) metadata related to the author, authoring tool, authoring date, etc. Such metadata may be included in each relevant sub-part of an AXMEDIS object. For example, consider a compounded item which is composed of an e-book and its corresponding movie, these two sub-parts may be created by different authors using different tools and then they can be put together by a third-party integrator using another tool. Each of these operations should be stored into the object by means of the above mentioned metadata.

In short, to enable the locating of any entities involved in the production and distribution of any digital assets, it is possible for the AXMEDIS data model to create all the *necessary* and *sufficient* status metadata to be incorporated into the object in order to allow traceability of the relevant lifecycles involved. This implies that there must be sufficient data fields to include, where necessary, data about all actors/tools which have worked on any instance of a workitem/object and the history of such work done on the said workitem/object instance.

Internally and separately the workflow management system will typically need to keep some additional lifecycle information so as to support user enquiries, re project resources management and cost accounting for line managers at various levels of the workflow/project information abstraction (workflow-native status data).

The AXMEDIS Object metadata and the internal workflow engine lifecycles status data when fully integrated have to provide all the information needed to enable the workflow management system to locate and track the progress status of any involved entity anywhere in any (sub)workspaces to enable the user to enquire about the lifecycles status of the three interacting types of entities (workflow, actors, objects) involved in any project..

Thus the AXWF has to be able *to answer the following questions* from the standpoint of any of the above four separate lifecycle viewpoints which may provide some overlapping information as follows:

Who, which actors and/or tools? (involved actors lifecycles viewpoint query)

Scope: So far (previously) and/or at this moment

Substance: At any time (including right now, and also so far since the beginning of any relevant lifecycle) who is, or has been working on which workitem/object instance in what workspace(s)?

What objects/workitems? (involved objects/workitems lifecycle viewpoint query)

Scope: So far (previously) and/or at this moment

Substance: What work is or has been done on which objects in which projects?

Wherefrom/object repositories? (involved repositories/licensor's viewpoint query)

Scope: So far (previously) and/or at this moment

Substance: Where has a particular object been or to be sourced from?

Which Project/workflow? (involved projects/workflow-instances/(sub)workspace-instances viewpoint query)

Scope: So far (previously) and/or at this moment

Substance: Which state of progress has any project reached and what process(es) are right now ongoing within it

2.1 Outline of the Workflow Management System Functional Requirements

Before we specify the design of the Workflow Plug-ins it is appropriate to give an outline of the requirements now established for the AXWF by the Consortium, as follows:

- i. Operate within the key Operating Systems (OS); for example the Windows, Linux, Mac Environments.
- ii. Interact with the AXMEDIS Object Manager to access objects and track/update their status (i.e. allow workflow metadata visualisation, editing, automated updating and storage).
- iii. Monitor the progress of assigned process activities and be capable of managing more than one workflow process instance so as to provide workflow support for multi-agency co-design and co-production of multimedia content based on open-source distributed products through LGPL, BSD or similar licences.
- iv. Provide time-and-status metadata updates that remain accessible to other Enterprise Project Management Applications, such as SAP for example (OPTIONAL).
- v. Provide a Service Interface (API) to be used for developing the plug ins for AXMEDIS Editors, Engines and Query Support.
- vi. Provide a seamless interface to AXMEDIS-native tools (e.g. tools for Content Production, Formatting, Packaging/Bundling and Distribution) for the range of operating systems selected above, i.e. specifically to provide interfaces for the following tools, engines and Query Support functions:
 - a) Editor
 - b) Rule Editor/Viewers for various tools
 - c) Composition and Formatting Engine
 - d) Programme and Publications Engine
 - e) Protection Tool Engine
 - f) P2P Active Selection Engine
 - g) Collector Engine
 - h) Publication/Loading Rules/Selections Editor
 - i) Publication Tool Engine of AXEPTool
 - j) Loading Tool Engine of AXEPTool
 - k) Administrative Information Integrator
 - l) Administrative Information Manager
 - m) Accounting Manager and Reporting Tools
 - n) User Query Support

Here we outline the main principles and scenarios for the usage of Workflow inside AXMEDIS:

- The Workflow is used for streamlining the publishing and distribution process activities performed in AXMEDIS.
- The Workflow is tightly integrated with the AXMEDIS View/Editing tools, in order to be able to automatically streamline editing/viewing activities inside publishing and distribution processes. With this kind of integration, the Workflow automatically launches AXMEDIS Editors/Viewers on the AXMEDIS Objects.
- The AXMEDIS Editors/Viewers run on the Client PC (either Microsoft Windows or MAC/OS), so typically the objects being edited must be downloaded to the PC local disk.

- The AXMEDIS View/Editor tools can be executed outside the Workflow environment (this is to accommodate those enterprises that wish to work with AXMEDIS Objects but without necessarily adopting structured processes).
- We expect that some publishing and distribution companies will not be able to adopt the fully integrated AXMEDIS framework, as:
 - They can use editing/viewing tools different from AXMEDIS ones
 - Even if they use the same editing/viewing tools, they may be unable to use the required AXMEDIS plug-ins, due to technical reasons (difficulties of distributing plug-ins, incompatibilities of AXMEDIS plug-ins with other PC tools, security reasons, heterogeneity of Client configurations, etc).
- For the latter kind of publishing/distribution companies, we shall provide a simple AXMEDIS Object check-in/check-out interface. It will be up to the user to download the AXMEDIS objects on their own PC disk, to launch the appropriate editor/viewer, to upload the modified objects and finally to inform the Workflow that the activity is terminated.
- We expect every AXMEDIS tool to update the AXMEDIS Object tracking information, while performing its actions. However, when the check-in / check-out interface is used, it will be up to the particular Workflow Object Manager to update such information. Unfortunately the Workflow cannot know what exactly the user performed between the check-in and check-out. We can provide a field to be filled in by the user during the check-out; describing what was done.

2.2 Workflow Integration Design

For now we can examine the specification of the Plug-ins from the point of view of the integration technology required for the delivery of a fully interoperable set of the Plug-ins with the APIs available from Openflow and BizTalk. This is to serve the integration with the AXMEDIS-native engines, tools and the Query Support Interface. It implies the design of the Plug-ins for full Connectivity, messaging Communication, Command and Control (C4) as well as AXMEDIS Object transfers across the relevant interfaces in particular such that the workflow will be able to provide seamless interaction with all the relevant AXMEDIS components involved in any of the 8 Workflow Scenarios that are now proposed and established.

For this we envisaged that the design specification of the required interfaces will involve the specification of:

- A. **HOW the interaction will be enabled** i.e. the integration technology to be deployed, for example, for the two adopted workflow systems, as specified above, examining their APIs, available data libraries and structures; the compatible operating systems, programming languages, dynamic and static structures and protocols e.g. HTTP, dll, Active X, COM, J2EE, XML-RPC, C/C++, Lisp, Python, Perl, Java, Frontier, PHP, MSoft.NET, WebObjects.
- B. **WHAT will be passed across the interface** i.e. parameter passing and data exchange syntax and semantics; e.g. the most efficient and expressive means of specifying the WF-EXchange_ID for each transaction between the AXWF and the relevant AXMEDIS components. This is to aid in developing a standard unique AXWF data exchange bridge to support workspaces using MPEG21 type work-items; providing a consistent mapping and interpretation of the exchange data passed across this bridge as needed. Because of the need for locating and tracking the workspaces and lifecycles involved, as described earlier, internally within workflow, the WF-Exchange_ID can be expanded to point to the workflow/work-item instance, the actors/Objects/tools_ID involved in any interaction so that the AXWF can provide for continuity, persistence, and integration semantics over dynamic scenarios of work.

The above two aspects have to be tackled for the range of *neutral bridge instances* for which we have to specify and design plug-ins as follows:

- 1) Programme and Publication Engine
- 2) AXEPTool Publishing Engine
- 3) AXEPTool Loading Engine
- 4) AXEPTool P2P Active Selection Engine
- 5) AXEPTool Objects Monitoring Tool
- 6) AXEPTool Publication and Loading Editor/Viewer
- 7) Programme and Publications Rule Editor/Viewer
- 8) Composition Engine
- 9) Compositional Commands and Reporting
- 10) Formatting Engine
- 11) Collector Engine
- 12) Protection Engine
- 13) AXOM Editor Plug-in
- 14) Compositional Rule Editor/Viewer
- 15) Protection Tool
- 16) Protection Tool Editor/Viewer
- 17) User Interface

2.3 Adoption of a Neutral Exchange Messaging Format: The Semantic Framework

At the outset it is essential to set out the domain transaction data type definitions by establishing the knowledge basis for the denotational semantics of any objects, attributes and relationships that may be relevant to any Workflow-AXMEDIS transaction content; as follows:

2.3.1 Objects of Type Workitem

This is any version of a digital object, or compound object, which may or may not be as yet a known AXMEDIS Object but which in any case is the version of any object that is undergoing an activity or sequence of activities being performed on it at any level within what can equivalently be seen here as a workspace-instance/session/workflow-instance/processflow-instance as is defined in Section 2.2.2 below.

Depending on the number of versions of the same that need to be currently relevant and thus need to be distinguished, it is possible to envisage of both a workitem_ID (AXWID) as well as a workitem – instance_ID (AXWID) to distinguish between a parent workitem and its possible renderings currently in play as work_Instance_IDs. [As will be seen, the logical class of workitem as defined here can be supplanted by an implementationally expedient sub-classing (merging) of this class (workitem) with the workflow-instance/processflow/activity class (defined in 2.1.2 below). This will enable efficient storage and re-call of each dynamic instance of an evolving process using single nodes per instance in a timeline activity graph structure with each node effectively acting as a container for the storage and re-call of the history of concomitant state changes in all participating entities. The participating entities will include the executing tool and the workitem object upon which the activity is being executed (as implemented in Openflow, see Section 4.1.4). However this is only an implementation issue and does not affect the validity of this logical class as defined above].

2.3.2 Objects of Type Workflow (Processflow)

This is an object hierarchy including various workflow management systems (WFMS). Within each WFMS various types of workflows can be defined and configured representing essentially *activity networks* encoded

as a *workflow-instance*. Each workflow-instance controls some specific set of coordinated actions. These are invoked to ensure some specific goals are achieved using some allocated business resources and processes. Each workflow-instance is set up to realise the goals of a specific new project and is associated with and shares the same lifecycle as a New Product Development (NPD). A reference to a workflow-instance can be at any level of elaboration (detail) hierarchy including at the (sub)workflow level down to the lowest level of elaboration which encodes a *processflow*.

Processflow coordinates the required processes of the workflow-instance and as such has in turn to encode coordination of the *lowest level of associated actions (tasks)* as per the activity network concerned. An activity network can be seen as a means of connecting all the required *tasks* to be co-ordinated and a processflow is a part of the workflow-instance that encodes the tasks according to business rule sets (defined company processes) as appropriate; i.e. processflows are a mapping of the tasks encoded consistent with the rules and resources of the particular NPD or company. Just as tasks, consist of sub-tasks and, at the lowest level, subtasks are accomplished through a set of actions, so also a processflow is an encoding of a set of coordinated actions.

2.3.3 Objects of Type Task

Every NPD will have to achieve some goal(s) which in turn may consist of a set of sub-goals and accordingly there exist corresponding tasks and sub-tasks that when accomplished together in an appropriately coordinated fashion lead to the fulfilment, respectively, of all the sub-goals and therefore the realisation of the NPD/project goal. At the elemental level sub-tasks consist of a set of co-ordinated actions as described above.

2.3.4 Objects of Type Session

A session is any occasion/event whereby some user or proxy (automatic actor/tool/engine) executes a single or any number of closely associated actions on any workitem(s) in the course of performing a *single (sub)task* within a processflow by logging onto and using one or a number of associated tools or resource sets (involving a single or multiple sign-ons).

A session lifecycle begins with the start of the (sub)task and the processflow level to which the (sub)task maps. A session lifecycle ends with the completion of the respective (sub)task and therefore with the ending of the processes that supported it as coordinated by the processflow level concerned. A session is the elemental level of authentication, authorisation and auditing (AAA) for the purpose of secure login, resource consumption accounting, costing and billing. Therefore session costs are to coherently map to (sub)task or process costs and are billable to the (sub)task/process/NPD owners/clients. Hence sessions will have a unique session_ID that is essential for a variety of control and audit reference purposes. A session may involve one or more workspace-instances (to be defined later).

2.3.5 Objects of Type User_Credentials

Any session owner/client will have to have a set of data to identify itself and to be used as a passport reference for access and accounting. This usually includes at least one of client/owner_ID with pointers to the password, NPD reference, session_ID and perhaps other user identity data (e.g. biometric and security data) to allow the appropriate level of secure tracking, control, costing, auditing and identity management on user activities depending on the practice within the particular company and its enterprise computing environment.

2.3.6 Objects of Type Workspace

This is the set of sessional computing states distinguished by a set of participating entities including principally a client-session/owner_ID plus the identity of any tasks, tools and objects involved and their states so that the foregoing references will constitute a necessary and sufficient set of identifiers for the said

workspace environment to be distinguished, indexed, stored and restored internally by the workflow and systemically for the said workspace to be tracked, subjected to any AAA – single sign on, single all in billing etc as may be practised in the said system.

Thus a workspace_Instance_ID is a practically realisable identifier of a client-session/owner_ID in a given context of a single task being progressed. A workspace instance can therefore take all the attributes of a workspace such as being started, ended, interrupted, suspended or awaiting certain other events before its possible resumption. It will remain traceable and restorable throughout its lifecycle.

2.3.7 Objects of Type Bridge

At the interface between an adopted AXMEDIS Workflow (AXWF) and relevant core AXMEDIS Components (tools/engines/Query Support), for the purpose of indexing, tracking, storing, as well as optionally to support AAA with single sign-on and all-in single-billing, we define an object of type bridge as a container of all the transactions required (1 to n, but typically at least 2) during the lifecycle of each single task element or notification as required for the task to be brought to a closure (success/failure).

A transaction comprises all the requests/notifications conveyed from one transactor (typically the AXWFM) to another (typically an AXMEDIS tool/engine/QSUI) i.e. essential exchanges between them to bring the (sub)task to a closure. A session may involve one or more transactions. A session bridge includes all the transactions that took place between the client/process-owner and the relevant (AXMEDIS) service providers. Thus a bridge has the same lifecycle as the session that was set-up to carry out the (sub)task concerned and as such the bridge lifecycle begins with the first request of the client/process-owner and ends with the closing response to the client's last request pertinent to the closure of the task concerned.

A bridge_ID refers to a family of exchange_IDs (as defined below) which together link all the exchanges that occurred during the lifecycle of a session so that in this way a bridge_ID can be used to store and re-call all such exchanges as well as the sessional costs, rights used, DRM and billing.

2.3.8 Objects of Type Exchange

Thus a transaction object type called a bridge itself comprises a family of co-related exchanges. An Exchange itself is the composite data structure, the whole of each particular message, that is passed across from entity A to entity B or from B to A each time that entity A or B wishes to communicate something, one to the other. At the limit an Exchange can become a multi/broad-cast where a transactor may wish to send a message, including links to objects etc, to a number of other transactors.

In principle an Exchange must satisfy the following specification criteria

- 1) must be able to identify and index itself (Self-referentiality Criterion)
- 2) must carry the user_credentials (including session_ID and any other data that may be required for AAA of the originating client-session-owner), (AAA Criterion)
- 3) must be able to bear an explicit link or implicit pointer to link it to other co-related sessional exchanges to which it refers or which might refer to it (Co-referentiality Criterion)
- 4) must be able to convey data sufficient for the goal of the exchange to be achievable such that each transactor in turn need only process elements of data sufficient and necessary for it to achieve the exchange sub-goals for which it is responsible within its own sub-system environment (Necessity and Sufficiency Criterion)

Each Exchange is an Exchange-instance and has to be indexable as an Exchange_Instance_ID. Any Exchange between the adopted AXMEDIS workflow system (AXWF) and relevant AXMEDIS service

providing components (Editors, Engines, Tools, Query Support) can thus be uniquely distinguished by means of an AXWF-Exchange-instance –ID or WF-Exchange_ID for short.

Accordingly a WF-Exchange_ID, shall in principle be a data structure as a string comprising the following data sub-fields:

Self_ID: this is the sub-field to help distinguish any particular exchange string from its nearest neighbour in the exchange-instances pattern space.

In practice a single bit to denote directionality (to or from the AXWF) should suffice in combination with the rest of the WF-Exchange_ID to constitute a uniquely indexable request/response string, given that it also contains the relevant client-session-owner's_ID, the workflow-instance_ID, and workitem/object_ID/temporary Object (AXWID/AXOID/AXTOID) with/for which the Exchange is being initiated (workspace_Instance_ID is itself an AXWF-native construct to serve as an internal pointer used by the AXWFM to track and bind data to various thus re-callable client workspaces-instances). In principle if there are going to be more responses to a request the Self_Id in its response can be incremented accordingly i.e., carry a number 1, 2, 3, ..n that is the sequential number of each of the various responses to a given request.

Workspace_instance_ID: this is the set of subfields required to specify completely and uniquely all the information about relevant client (i.e. workspace-instance) associated with the Exchange; i.e. the identity of session participants (session_ID/client_ID, running_applications_ID, workitems_ID) and their respective states including those awaiting the response to the request and which can thus be re-storable and progressed, as appropriate, upon the arrival of the response. Naturally this composite structure includes at least a session_ID.

Session_ID: this is the sub-field sufficient to identify the workspace-instance owner to which the exchange-instance relates and any other sub-fields deemed necessary by either transactors for housekeeping purposes (audit, accounting, billing, scheduling).

Addressee_ID: this is the sub-field as a reference to the transactor who is to receive and process the data conveyed in the exchange structure as a whole; typically this is either the address of a relevant AXMEDIS tool/engine/QSUI as a registered service provider with known directory ID/address e.g. a URL or alternatively it is the AXWFM itself that is being addressed by a responding/notifying AXMEDIS component (this can be an implicit/hard-wired address by virtue of the self_ID bit in the requesting Exchange_ID that is thus inverted, or incremented appropriately to denote directionality and the sequence placing of the string as responding to the requesting string i.e. the initiating Exchange instance).

Method_ID: this is the sub-field to specify the task to be carried out by the receiving tool/engine/QSUI

Parameters: this is the set of sub-fields required to enable the goals of the exchange-instance to be tackled and potentially be brought to a closure (success/failure). This should comprise the specified Method Invocation Parameters.

AXMEDIS AXOID: This could exist as a distinct sub-field or as one of the above Methods Invocation Parameters so that the identity of the AXMEDIS Object concerned could be specified for processing by the recipient of the exchange-instance.

Message: This could include any number of necessary and sufficient subfields up to a systemic limit. Typically it should include a command-line string to effect a query/request/command/notification to or from the AXWFM. It could also include references to any objects concerned by means of an AXMEDIS AXOID.

Thus in principle each such Exchange instance should be a logical, necessary, sufficient and technology-neutral data structure. Such an exchange instance, itself part of the data structure referred to as a sessional

bridge instance comprises of a string; essentially of the following sub-strings and logical structure whereby “parameters” can optionally also include Rule_ID (AXRID), as well as AXOID data types, and Workspace_instance_ID would include pointers to workitem_ID (AXWID) and Session_ID.

```
{{<Exchange-instance_self_ID>},
{workspace-instance_ID}, {<Session_ID>,<Addressee_ID>},{<Method_ID>},
{<Parameters>},{<AXOID>},{<Message>}}
```

The actual realisation of the above structure is addressed in the following sections of this document in developing the logical design of what we refer to as WF-Exchange_ID, as a neutral exchange format for accessing any transaction (and its purpose and contents as passed to and fro) between the AXWFM environment and any relevant AXMEDIS components. This is to be achieved hopefully without imposing additional metadata requirements on the AXMEDIS Object Model.

Thus the WF-Exchange-ID can be designed as a transparently decomposable ASCII string, a composite structure -itself an instance of an object type Bridge.

2.3.9 Logical Analysis of Semantics and Syntax of the WF-Exchange_ID

In essence the information exchange at the interface occurs at two levels and may require the exchange of simple but composite computing structures as follows:

2.3.9.1 The Data Exchange Semantics

The Base Level Device and Communications Bus Status:

This consists of handshake flags about device availability state and Bus Access; i.e. signals for explicit access requests, non-availability etc, to avoid contention, deadlock etc. This is expected to be universally in place as part of the low level device and communication logic.

Such handshaking status information typically includes:

```
{{request <0:1>},{ack:<0:1>},{busy:<0:1>},{waiting:<0:1>},{ready:<0:1>}}
```

The process level AXWF and AXMEDIS Exchange Semantics:

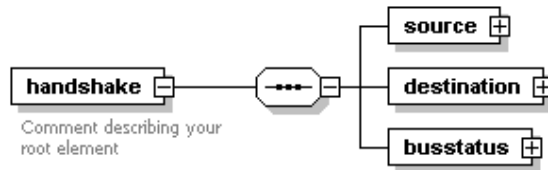
Schema **handshake.xsd**

schema location: <\\sirfs1\dfs\venus\AXMEDIS\Workflow\Schema\handshake.xsd>
 attribute form default: **unqualified**
 element form default: **qualified**

Elements
[handshake](#)

element handshake

diagram



properties content complex

children [source](#) [destination](#) [busstatus](#)

annotation documentation Comment describing your root element

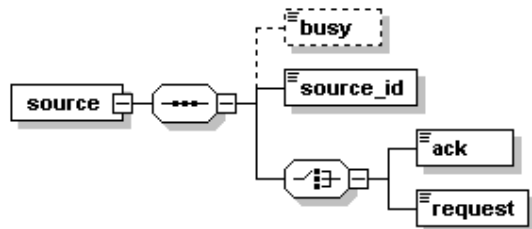
```

source <xs:element name="handshake">
  <xs:annotation>
    <xs:documentation>Comment describing your root element</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="source">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="busy" type="xs:boolean" minOccurs="0"/>
            <xs:element name="source_id" type="xs:string"/>
            <xs:choice>
              <xs:element name="ack" type="xs:boolean"/>
              <xs:element name="request" type="xs:boolean"/>
            </xs:choice>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="destination">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="busy" type="xs:boolean" minOccurs="0"/>
            <xs:element name="destination_id"/>
            <xs:choice minOccurs="0">
              <xs:element name="ack" type="xs:boolean"/>
              <xs:element name="request" type="xs:boolean"/>
            </xs:choice>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="busstatus">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="busy" type="xs:boolean"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

element handshake/source

diagram



properties isRef 0 content complex

children [busy](#) [source](#) [id](#) [ack](#) [request](#)

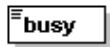
```

source <xs:element name="source">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="busy" type="xs:boolean" minOccurs="0"/>
      <xs:element name="source_id" type="xs:string"/>
      <xs:choice>
        <xs:element name="ack" type="xs:boolean"/>
        <xs:element name="request" type="xs:boolean"/>
      </xs:choice>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

element **handshake/source/busy**

diagram



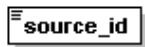
type **xs:boolean**

properties isRef 0
 content simple

source <xs:element name="busy" type="xs:boolean" minOccurs="0"/>

element **handshake/source/source_id**

diagram



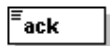
type **xs:string**

properties isRef 0
 content simple

source <xs:element name="source_id" type="xs:string"/>

element **handshake/source/ack**

diagram



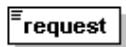
type **xs:boolean**

properties isRef 0
 content simple

source <xs:element name="ack" type="xs:boolean"/>

element **handshake/source/request**

diagram



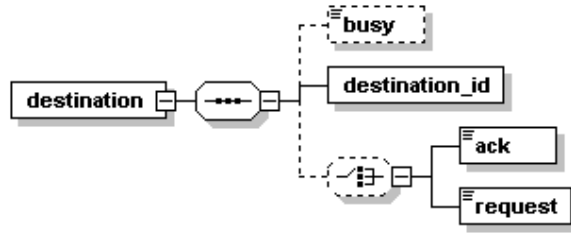
type **xs:boolean**

properties isRef 0
 content simple

source <xs:element name="request" type="xs:boolean"/>

element **handshake/destination**

diagram



properties isRef 0
 content complex

children [busy](#) [destination_id](#) [ack](#) [request](#)

source

```
<xs:element name="destination">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="busy" type="xs:boolean" minOccurs="0"/>
      <xs:element name="destination_id"/>
      <xs:choice minOccurs="0">
        <xs:element name="ack" type="xs:boolean"/>
        <xs:element name="request" type="xs:boolean"/>
      </xs:choice>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

element **handshake/destination/busy**

diagram



type **xs:boolean**

properties isRef 0
 content simple

source

```
<xs:element name="busy" type="xs:boolean" minOccurs="0"/>
```

element **handshake/destination/destination_id**

diagram



properties isRef 0

source

```
<xs:element name="destination_id"/>
```

element **handshake/destination/ack**

diagram



type **xs:boolean**

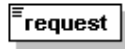
properties isRef 0
 content simple

source

```
<xs:element name="ack" type="xs:boolean"/>
```

element handshake/destination/request

diagram

type **xs:boolean**

properties	isRef	0
	content	simple

source `<xs:element name="request" type="xs:boolean"/>`**element handshake/busstatus**

diagram



properties	isRef	0
	content	complex

children [busy](#)

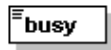
```

source <xs:element name="busstatus">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="busy" type="xs:boolean"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

element handshake/busstatus/busy

diagram

type **xs:boolean**

properties	isRef	0
	content	simple

source `<xs:element name="busy" type="xs:boolean"/>`

Typically transactions between the AXWF, on behalf of its workspace instances (or sessions), and, the AXMEDIS environment will be as follows:

- i) AXWFM Requesting/commanding work to be done by AXMEDIS Engines/Editors/Tools
- ii) AXWFM Requesting Objects from Query support
- iii) AXWFM Receiving Report back/Response/Results from any of the relevant AXMEDIS components (Engines, Editors, Tools, QS)
- iv) AXWFM Requesting single sign-on, single-sign-off, single-session-billing and/or requesting cancellation of one or all previous requests on behalf of particular client sessions.

Essentially the WF-Exchange_ID needs to allow the Workflow system *to be selected* for results/reports being returned (through Command and Reporting Module) and/or for it internally *to select* any workflow instance, workspace-instance workitem-instance, actor (including persons, tools, engines) and/or object(s) relevant to the exchange, and process the data in play accordingly at the point of transaction i.e. effect the

appropriate *tracing back and binding* of the AXMEDIS responses/reports to the relevant workspace-instances.

A (sub)workspace-instance is a transient environment of execution of any workflow-instance at any level including the leaf-node level whereby a single task is being performed on a workitem-instance involving a tool (and probably also a user) in some state of execution stored and recallable by reference to the workspace-instance-ID.

Any AXWF will have to have an appropriate mechanism for persistence and seamless continuity upon restoration of workspaces and thus will have to have some way of storing and retrieving this bundle of coupled states from its database.

At this stage the primary focus is to explore the *spaces*, *semantics* and *syntax* of the required parameters to be available for exchange at the interaction between an adopted AXWF (say Openflow) and the AXMEDIS Plug-in at the interface to an environment (AXWF) which is external to the AXMEDIS core components.

Within the AXWF we require to be able to distinguish and index the various possible instances of all workflows, NPDs, objects, workitems, actors/devices (including partners, persons, tools, engines, interfaces) as defined earlier in this document. It is clear that we also need to be able to refer to *bundles of states* of *co-implicated* objects/workitems and tools so as to be able to *deliver seamless semantic integration, continuity and dynamic persistence of the (sub) workspaces*.

Irrespective of the chosen protocol, it would appear that information within an exchange packet can be organised in pairs or tuples denoting any ID and its value pair of structures separated by some delimiter such as a colon and bounded by another set of delimiters that can allow nesting such as brackets and/or braces. It would then be possible to express the information regarding bundles of (sub)workspace states and their dynamic instances by a unique reference such as e.g. instance-ID. An Instance_ID is an internally generated and internally used AXWF vector i.e. some internally used encoding of a required bundle of pointers (composed by the AXWF, combining IDs as internal pointers to workspace entities), decomposable later within the AXWF so as to service the initiating workspace. Such a vector, as the required workspace-instance-ID can be embedded within the structure passed across the WF-AXMEDIS bridge i.e. within WF-EXchange_ID with parameters in nested braces that express the tuples related to the IDs of each of the co-implicated objects, actors and their values as follows:

{ WF-EXchange_ID }

The above composite string or packet will embody information about the following logical structures:

- 1) *Is the traffic that is moving across this bridge instance an AXWF Request/Command or a Response from some component to this or any other previous request?*

Thus the exchange needs to include:

{Request :<1>}

{Response:<0;1:2:3: n>}

As mentioned before if more responses are expected in principle we can have: 0 : Request => 1, 2 ... responses

- 2) *What is the identity of the client-session that has initiated the request?*

Or more fully but only of interest internally to the AXWF:

What is the value of the pointer generated internally by the AXWF to index the workspace-instance that is relevant to this AXWF request or any response received by the AXWF? i.e. what are the IDs of the session,

the user, tools being used and their current state, and, what is the identity of workitem-instances being worked on and their state (if of interest for session save and restoration later to enable resumption of any interrupted/pending work)? This group of identities put together as a packet of identities constitute what is referred to loosely in some workflow terminology as `session_ID` and/or `user_credentials_ID` but more accurately and consistent with our earlier lifecycles-based semantic analysis we shall refer to it coherently as the `workspace_Instance_ID`; Thus the exchange needs to include:

{AXWF-workspace_Instance_ID < >}

3) *What is the identity/address of the relevant AXMEDIS tool/engine/service-provider module that is involved/is-to-be-deployed for the workspace/task to which this request is being Targeted and /or which is hereby Responding ?* Thus the exchange needs to include:

{AXMEDIS_service_provider_registry_ID}

4) *What is the identity of any relevant AXMEDIS object(if known) and/or workitem that is the subject of this Exchange?* Thus the exchange needs to include:

{AXMEDIS object_ID:"< AXOID>"}

and/or

{workitem_ID : "<AXWID>" }

5) *What is the service being Requested/Responded to and with what qualifying parameters?*
Thus the exchange needs to include:

**{{Method_ID: < >},
{parameters_ID: <parameter1>.....< parameter n>, <AXRID>}}**

To what extent, if any, are any other message elements (including natural language input as processable by addressees) involved relating to the actors/objects, that need to be passed across and coupled to this WF-EXCHANGE_ID Packet ?

Thus the exchange needs to include:

{Message_ID: < message number/status flags/object_IDs if any, separated by delimiters--:-->}

The above is appended to the WF-Exchange_ID data packet as a `command_line` string, i.e. the last subfield within the integrated string that gives the `session_ID` i.e. the last element of the WF-Exchange_ID.

Self-referential/Reflexive AXWF workspace-instance Requests

If it is desired to have an explicit facility for cancelling/re-calling/modifying an AXWF-submitted service request that has been issued in error or needs revising (e.g. because the initiating client session/work-item has since been suspended/cancelled due to unforeseen events such as an un-expected rights withdrawal notification etc) then although a cancellation request can be included within the above message field as a command line element, it is more expressive to include an explicit “cancellations” field appended to the above. To allow for most types of cancellations, such a sub-field can be a tri-state device appended to all the requests for each session with 0 denoting that the session request remains valid, 1 denoting that the relevant (indexed by work-item/session-id) request is to be cancelled, and -1 to cancel all previous requests so far issued for the given session. A need for modification can also be accommodated in this way by a cancellation of the relevant previous request and submission of a new request.

Thus to allow explicitly for requests to cancel and/or modify an earlier request or group of requests appertaining to a particular workspace-instance (as e.g. indexable by `work-item_ID/session_ID` sub-fields

within the workspace-instance_ID), the WF-Exchange_ID can have a cancel sub-field appended to it after the message field as follows:

{Cancel_status:<0:1:-1>}

In a minimalist implementation, the interpretation of the various subfields that constitute any AXWF workspace_Instance_IDs can be said to be largely irrelevant to the AXMEDIS components interacting with the AXWF as most fields of this code can be ignored by the AXMEDIS components which could simply echo it back in their eventual response.

Thus it is possible to envisage the following overall logical structure for the WF-Exchange_ID string:

```

{{"Request/Response_ID": "<0:1:n>"},
{"Axwf_workspace_instance_ID": "<WFWINID >"},
{"Axmedis_service_provider_registry_ID": "<AXSERVPID >"},
{"Axmedis_object_ID": "<AXOID>"},
{"Method_ID": "<AXMethID>"},
{"Parameters_ID": "<AXMethID_PARA1>....."<AXPARA_n>,"<AXRID>"},
{"Message": "<message_ID, AXOID, "messages separated by delimiters">"},
{"Cancel_status": "<0:1:-1>"}}

```

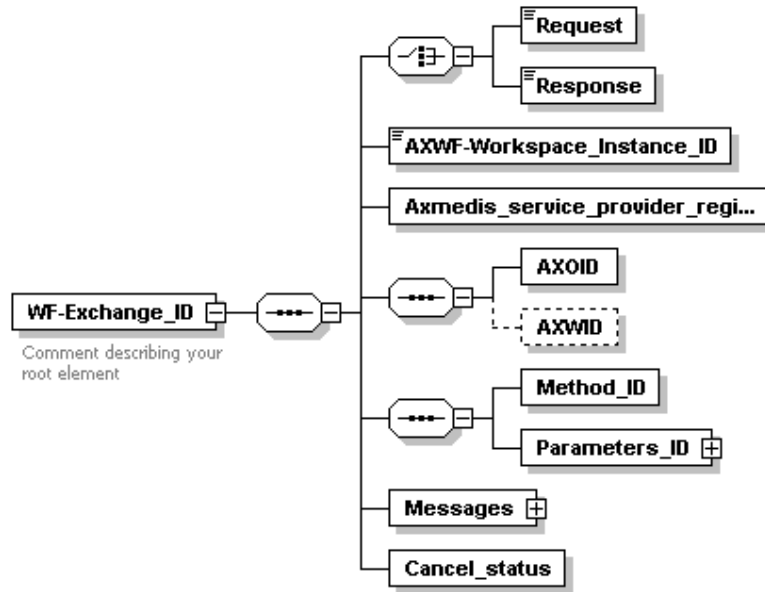
Schema **WF_Exchange_ID.xsd**

schema location: \\sirfs1\dfs\venus\AXMEDIS\AXMEDIS FW&TS\WF\Schema\WF_Exchange_ID.xsd
 attribute form default: **unqualified**
 element form default: **qualified**

Elements
[WF-Exchange_ID](#)

element **WF-Exchange_ID**

diagram



properties content complex

children [Request](#) [Response](#) [AXWF-Workspace_Instance_ID](#) [AXMEDIS service provider registry ID](#) [AXOID](#) [AXWID](#) [Method_ID](#) [Parameters_ID](#) [Messages](#) [Cancel_status](#)

annotation documentation Comment describing your root element

```

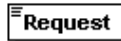
source <xs:element name="WF-Exchange_ID">
  <xs:annotation>
    <xs:documentation>Comment describing your root element</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:choice>
        <xs:element name="Request" type="xs:positiveInteger"/>
        <xs:element name="Response" type="xs:positiveInteger"/>
      </xs:choice>
      <xs:element name="AXWF-Workspace_Instance_ID" type="xs:positiveInteger"/>
      <xs:element name="AXMEDIS_service_provider_registry_ID"/>
      <xs:sequence>
        <xs:element name="AXOID"/>
        <xs:element name="AXWID" minOccurs="0"/>
      </xs:sequence>
      <xs:sequence>
        <xs:element name="Method_ID"/>
        <xs:element name="Parameters_ID">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Parameter" maxOccurs="unbounded"/>
              <xs:element name="AXRID"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:element name="Messages">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Message_ID"/>
            <xs:element name="AXOID"/>
            <xs:element name="Message" maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="Cancel_status"/>
    </xs:sequence>
  </xs:complexType>

```

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

element **WF-Exchange_ID/Request**

diagram



type **xs:positiveInteger**

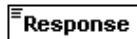
properties isRef 0
 content simple

source

```
<xs:element name="Request" type="xs:positiveInteger"/>
```

element **WF-Exchange_ID/Response**

diagram



type **xs:positiveInteger**

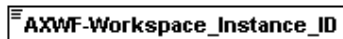
properties isRef 0
 content simple

source

```
<xs:element name="Response" type="xs:positiveInteger"/>
```

element **WF-Exchange_ID/AXWF-Workspace_Instance_ID**

diagram



type **xs:positiveInteger**

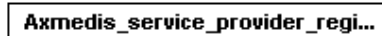
properties isRef 0
 content simple

source

```
<xs:element name="AXWF-Workspace_Instance_ID" type="xs:positiveInteger"/>
```

element **WF-Exchange_ID/AXMEDIS_service_provider_registry_ID**

diagram



properties isRef 0

source

```
<xs:element name="AXMEDIS_service_provider_registry_ID"/>
```

element **WF-Exchange_ID/AXOID**

diagram



properties isRef 0

source

```
<xs:element name="AXOID"/>
```

element **WF-Exchange_ID/AXWID**

diagram



properties isRef 0

source

```
<xs:element name="AXWID" minOccurs="0"/>
```


element **WF-Exchange_ID/Method_ID**

diagram



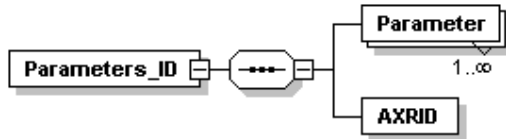
properties

isRef 0

source `<xs:element name="Method_ID"/>`

element **WF-Exchange_ID/Parameters_ID**

diagram



properties

isRef 0
content complex

children [Parameter](#) [AXRID](#)

source `<xs:element name="Parameters_ID">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="Parameter" maxOccurs="unbounded"/>
 <xs:element name="AXRID"/>
 </xs:sequence>
 </xs:complexType>
</xs:element>`

element **WF-Exchange_ID/Parameters_ID/Parameter**

diagram



properties

isRef 0

source `<xs:element name="Parameter" maxOccurs="unbounded"/>`

element **WF-Exchange_ID/Parameters_ID/AXRID**

diagram



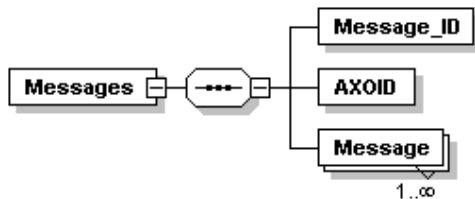
properties

isRef 0

source `<xs:element name="AXRID"/>`

element **WF-Exchange_ID/Messages**

diagram



properties

isRef 0
content complex

children [Message_ID](#) [AXOID](#) [Message](#)

```

source  <xs:element name="Messages">
         <xs:complexType>
           <xs:sequence>
             <xs:element name="Message_ID"/>
             <xs:element name="AXOID"/>
             <xs:element name="Message" maxOccurs="unbounded"/>
           </xs:sequence>
         </xs:complexType>
       </xs:element>

```

element **WF-Exchange_ID/Messages/Message_ID**

```

diagram
  
  Message_ID

properties
  isRef  0

source  <xs:element name="Message_ID"/>

```

element **WF-Exchange_ID/Messages/AXOID**

```

diagram
  
  AXOID

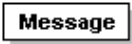
properties
  isRef  0

source  <xs:element name="AXOID"/>

```

element **WF-Exchange_ID/Messages/Message**

```

diagram
  
  Message


properties
  isRef  0

source  <xs:element name="Message" maxOccurs="unbounded"/>

```

element **WF-Exchange_ID/Cancel_status**

```

diagram
  
  Cancel_status

properties
  isRef  0

source  <xs:element name="Cancel_status"/>

```

Potentially the session or workspace-instance owner's `_ID` is the only part of the `workspace_Instance_ID` that is of interest globally, both within the AXWF as well as within the AXMEDIS components. The AXWF session identity can enable seamless service provision through session identity relay/broadcast and thus enabling single-sign on.

Typically if we designate the first element of the `WF-Exchange_ID` to encode for whether the current exchange is a AXWF request `<0>` or a response from an AXMEDIS component `<1>` then in a response to an AXWF request, the responding/reporting AXMEDIS component will be expected to start the first element of the `Exchange_ID` with a 1 i.e. turning it from zero (as it would have been for the request string) to an 1. The responding component will normally be expected to return, without any changes, the AXMEDIS tool/engine `_ID` and the `workspace_Instance_ID` to which the response relates thus appending only a response

status code or message string in response to the original AXWF-supplied WF-Exchange_ID request message string to indicate complete fulfilment, end, or failure status etc. in the response packet.

The unknown object_ID, where applicable, is the only other remaining element of the WF-Exchange_ID still to be discussed in the context of what should be the composition of the Response WF-Exchange_ID packet coming back from the responding AXMEDIS components. The object_ID is also returned in the Response string from the relevant AXMEDIS component except in cases when the object_ID in the request string is a code for an unknown object e.g. X000001 whereupon either the discovered AXMEDIS AXOID replaces this code in the Response WF-Exchange_ID packet string or the unknown object_ID is returned as it was and the discovered AXMEDIS AXOID(s) appended to the WF-Exchange_ID in the messaging field as the Result. This will be the case for fishing requests for unknown objects with a particular quality (sought_properties string) whereupon in the subsequent Response from the Query Support, the initiating query string may also be appended to the Response in its messaging field.

Note that when such a hitherto unknown object of a desirable property is being sought by a workspace instance, this has to be communicated to the Query Support through a Request WF-Exchange_ID packet with a default AXWF-native *sought-object_ID*, compliant with the AXMEDIS AXOID format (imagine this to be X000001 just for argument's sake for the moment); this number can be passed by the AXWF to the Query Support and later upon return of any discovered AXOIDs, some internal mechanism within the AXWF will be able to effect the binding between any given AXRQID for a *sought-object_ID* (e.g. X000001) and the actual *found* AXMEDIS AXOIDs returned by the Query Support as candidate objects in response to the WF-Exchange_ID which carried the AXRQID for the objects.

So the desirable properties of the unknown object that is being thus sought by such a Request can be inserted in the messaging field appended to the WF-Exchange_ID. This messaging field is to be available for full duplex messaging which thanks to the above design will always remain *relevantly and highly efficiently* context-aware as it is coupled with pointers to relevant workspace contexts (i.e. AXWF-workspace_Instance_IDs) as known to and readily restorable by the AXWF. In principle, this messaging field can be populated by the sending and receiving parties by appending or overwriting with any set of strings to constitute the desired message back and forth between the transactors involved. In practice they will have to overwrite it to avoid inefficiencies as described later.

For example in the case of a AXWFM Request for an object, the desirable object properties can be inserted in terms of keyword particulars or any other available strings that can be interpreted by the Query Support. If and when suitable object or objects are found in response, then their AXMEDIS AXOIDs can be made to overwrite the received message field or appended to it (every subset e.g. every AXOID or response messages as distinct from received messages will be separated by some delimiter such as :).

Please note that although appending such a message field does provide a running history of requests and responses between the AXWFM and the relevant AXMEDIS components, appending is actually not an option as it can lead to uncontrolled packet size for the WF-Exchange_ID. Internally within AXWF, a history of the messages back and forth relating to any workspace_Instance_IDs should be available at any time by simply popping the stack of WF_Exchange_IDs used in the context of any workspace_Instance_ID (logs may be kept, locally and/or globally by any parties), and there will have to be some form of messaging register attached with the particular workspace_Instance_IDs within the AXWF. The relevant AXMEDIS components transacting with the AXWFM can also trigger a save of a stack of their own WF-Exchange_IDs and logs to serve internal AXMEDIS Audits for Usage, Service Levels and Rights Accounting and Billing purposes.

Further, although the workspace_Instance_ID that is embedded within the WF-Exchange_IDs as a sub-field is only to serve internal AXWF bindings and as such is to be left un-processed by the receiving AXMEDIS components; it can nevertheless serve as a unique, albeit opaque, identifier for usage audit and royalties billing chain purposes because ultimately it remains traceable to a human user or partner organisation's

workflow instance and thus to a project and product that has initiated the service requests thus conveyed by the AXWF by way of its WF-Exchange_IDs.

In specially customised applications of AXMEDIS, and, AXWF for some enterprises, if the workspace_Instance_ID is not internally precompiled and encoded somehow as a single vector but itself remains as a set of distinct sub-fields as carried within the WF-EXCHANGE_ID, then its first subfield can be made to identify the human user/partner_ID or project_ID to which the workspace-instance belongs. This can make the Workspace_instance_ID itself partially transparent to the AXMEDIS platform to help *verify* for the relevant AXMEDIS modules (e.g. CAMART-DRM, with the usage/service/rights/royalties accounting and billing thus becoming readily traceable to each service atom and instance.

In practice it is not envisaged that any unsolicited messages will be sent by any AXMEDIS components to the AXWF. Therefore any report sent to the AXWF will normally be in reactive mode and relating to an earlier request/command initiated by the workflow thus a simple replacement of 0 by 1 in the response and the return of the originating exchange_ID with the message field appropriately filled by the respondent should suffice. Any associated files that might be pointed to in the message field and/or passed as attached files could have the associated WF-Exchange_ID included somewhere in a header or subject field of all related automated mail messaging.

2.3.9.2 *WF-Exchange_ID for an AXMEDIS Component Responding/Reporting*

The above Exchange Messaging Format is recommended for adoption by the relevant AXMEDIS service provider modules including the gateway module, namely the Command and Reporting module.

As discussed previously, the n-element field as an internal AXWF pointer to a (sub)workspace instance is the only part of the above string that may be ignorable by some AXMEDIS components depending on the AAA policy etc practiced within the Enterprise Computing Environment concerned.

Each AXWF will have its own way of saving and retrieving its workspace-instances as subsets of a workflow instance. Each workflow will thus generate its own workspace_instance_IDs for internal use. From a minimalist standpoint, all that is required from any AXMEDIS component responding to a AXWF request is to return the same code in the workspace_instance_ID as that received in the Request string so that the AXWF can effect the appropriate binding of the response to the relevant workspace-instance wherefrom the Request was initiated.

2.3.9.3 *Logical Semantics and Syntax for a workspace_Instance_ID*

A workspace-instance set-up can have the following characteristics:

- i) may have been scheduled for automatic execution
- ii) may have already started/ended
- iii) may be about to start/end
- iv) may be currently live – i.e. ongoing
- v) may have been interrupted for some reason
- vi) may be about to be resumed
- vii) may have been suspended or be pending some CPA/rights-critical trigger
- viii) is usually part of a workflow-instance or bundle of co-dependent workspace instances

Implicitly the workspace_Instance_ID as a complex structure will point to a set of requisite pre-defined and pre-stored state descriptors with their respective state values for each relevant entity involved in a workspace instance.

This could for example link a set of participating entities contained in co-related workspaces together with their state at the time of interest as a snapshot to be re-storable later.

This could include reference to an AXMEDIS AXOID via its being-worked-on instance, i.e. the workitem_Instance_ID in such a workspace and any tool(s) used and their relevant state descriptors and values, and, any other actors such as person(s) who might have been participating in the respective workspace and the descriptors for their state and the values for such states.

2.3.9.4 *Workspace Hierarchy Dynamic Persistence, Storage and Retrieval within the AXWF*

Although the possible structure of the workspace_instance_ID is only of interest for internal processing within the AXWF, and thus will not concern the processing of the Exchange_IDs by the relevant AXMEDIS components, it can be of interest to explore what the logical elements of the workspace-instance-ID might be:

```

{{{“AXWF_Instance_ID”: “<WFISID>”},
  “AXWF-workitem_Instance_ID”: “< WKITID/AXWID/AXTOID >”},
  “AXWF-native actor/application_instance_ID”: “<WFAPID>”},
  “AXWF_Instance_ID_start”: “<start-time>”},
  “AXWF_Instance_ID_end”: “<end-time>”},
  { “AXWF_Instance_ID_interrupt”: “<0:1>”},
  { “AXWF-instance_interrupter_ID”: “<WFINTID>”}, {interrupt_time: <WFINTT>}},
  { “AXWF_Instance_ID_resume”: <0:1>, {resume_time: “< WFIRST>”}}},
  {AXWF_Instance_ID_suspended <0:1>},
  {AXWF_Instance_ID_cpa-slack_critical: <0:1>}}

```

Schema Workspace_Instance_ID.xsd

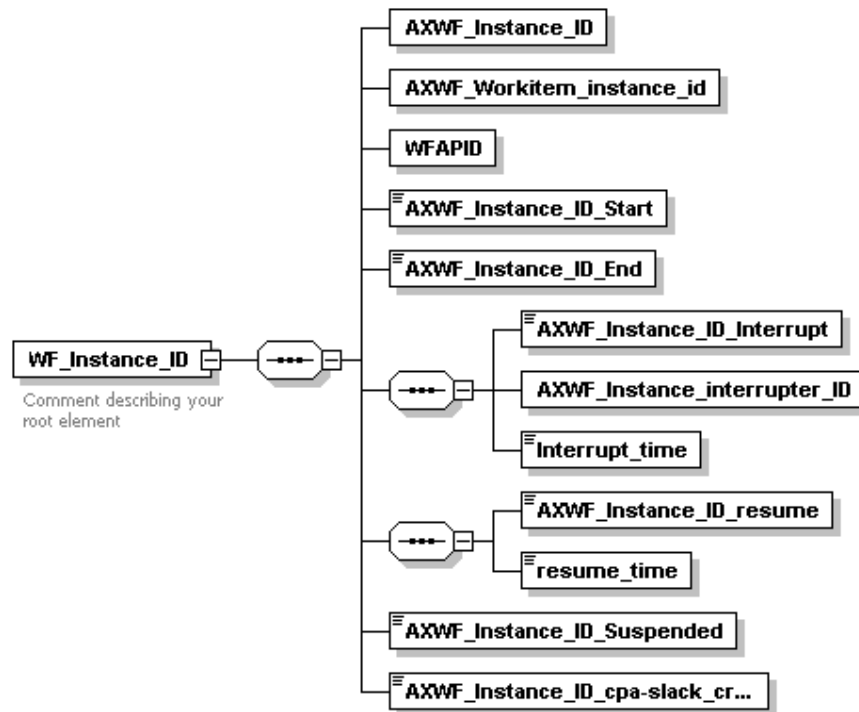
schema location: \\sirfs1dfs\venus\AXMEDIS\WorkFlow\Schema\Workspace_Instance_ID.xsd
 attribute form default: **unqualified**
 element form default: **qualified**

Elements

[WF_Instance_ID](#)

element **WF_Instance_ID**

diagram



properties content complex

children [AXWF_Instance_ID](#) [AXWF_Workitem_instance_id](#) [WFAPID](#) [AXWF_Instance_ID_Start](#) [AXWF_Instance_ID_End](#) [AXWF_Instance_ID_Interrupt](#) [AXWF_Instance_interrupter_ID](#) [Interrupt_time](#) [AXWF_Instance_ID_resume](#) [resume_time](#) [AXWF_Instance_ID_Suspended](#) [AXWF_Instance_ID_cpa-slack_critical](#)

annotation documentation Comment describing your root element

source

```
<xs:element name="WF_Instance_ID">
  <xs:annotation>
    <xs:documentation>Comment describing your root element</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="AXWF_Instance_ID"/>
      <xs:element name="AXWF_Workitem_instance_id"/>
      <xs:element name="WFAPID"/>
      <xs:element name="AXWF_Instance_ID_Start" type="xs:time"/>
      <xs:element name="AXWF_Instance_ID_End" type="xs:time"/>
      <xs:sequence>
        <xs:element name="AXWF_Instance_ID_Interrupt" type="xs:boolean"/>
        <xs:element name="AXWF_Instance_interrupter_ID"/>
        <xs:element name="Interrupt_time" type="xs:time"/>
      </xs:sequence>
      <xs:sequence>
        <xs:element name="AXWF_Instance_ID_resume" type="xs:boolean"/>
        <xs:element name="resume_time" type="xs:time"/>
      </xs:sequence>
      <xs:element name="AXWF_Instance_ID_Suspended" type="xs:boolean"/>
      <xs:element name="AXWF_Instance_ID_cpa-slack_critical" type="xs:boolean"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```


element **WF_Instance_ID/AXWF_Instance_ID**

diagram

AXWF_Instance_ID

properties isRef 0
 source `<xs:element name="AXWF_Instance_ID"/>`

element **WF_Instance_ID/AXWF_Workitem_instance_id**

diagram 

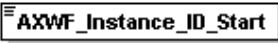
properties isRef 0
 source `<xs:element name="AXWF_Workitem_instance_id"/>`

element **WF_Instance_ID/WFAPID**

diagram 


properties isRef 0
 source `<xs:element name="WFAPID"/>`

element **WF_Instance_ID/AXWF_Instance_ID_Start**

diagram 

type **xs:time**
 properties isRef 0
 content simple
 source `<xs:element name="AXWF_Instance_ID_Start" type="xs:time"/>`

element **WF_Instance_ID/AXWF_Instance_ID_End**

diagram 

type **xs:time**
 properties isRef 0
 content simple
 source `<xs:element name="AXWF_Instance_ID_End" type="xs:time"/>`

element **WF_Instance_ID/AXWF_Instance_ID_Interrupt**

diagram 

type **xs:boolean**
 properties isRef 0
 content simple
 source `<xs:element name="AXWF_Instance_ID_Interrupt" type="xs:boolean"/>`

element **WF_Instance_ID/AXWF_Instance_interrupter_ID**

diagram 

properties isRef 0

source `<xs:element name="AXWF_Instance_interrupter_ID"/>`

element **WF_Instance_ID/Interrupt_time**



type **xs:time**

properties	isRef	0
	content	simple

source `<xs:element name="Interrupt_time" type="xs:time"/>`

element **WF_Instance_ID/AXWF_Instance_ID_resume**



type **xs:boolean**

properties	isRef	0
	content	simple

source `<xs:element name="AXWF_Instance_ID_resume" type="xs:boolean"/>`

element **WF_Instance_ID/resume_time**



type **xs:time**

properties	isRef	0
	content	simple

source `<xs:element name="resume_time" type="xs:time"/>`

element **WF_Instance_ID/AXWF_Instance_ID_Suspended**

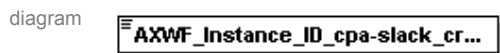


type **xs:boolean**

properties	isRef	0
	content	simple

source `<xs:element name="AXWF_Instance_ID_Suspended" type="xs:boolean"/>`

element **WF_Instance_ID/AXWF_Instance_ID_cpa-slack_critical**



type **xs:boolean**

properties	isRef	0
	content	simple

source `<xs:element name="AXWF_Instance_ID_cpa-slack_critical" type="xs:boolean"/>`

The above data structure is internally generated and managed by the AXWFM.

In the above structure, the emboldened sub-fields are the candidate sub-fields of the workspace-instance_ID that will be *necessary and sufficient* for session identification by AXMEDIS service providers (AXMEDIS-native tools, engines, QS) and these are to be included within WF-Exchange_ID.

The above workspace_Instance_ID complex structure can be nested to any level sufficient to allow the storage and retrieval of the relevant control/tracking-critical states of all participant entities in any workflow instance and workspace scenario. For computational efficiency the above workspace_Instance_ID can be partitioned so that only the relevant elements of it can be passed on and processed by each respective transactor i.e. within the AXWF *internally* versus within the environment of each AXMEDIS component interacting with AXWF and *external* to it. For example within the AXWF all the above sub-fields relevant to a workspace_Instance_ID will be used to a greater or lesser degree by all types of workflow management systems to effect various tracking control functions and other user services as required.

Thus whenever any actor or any AXWFM may wish to store away, retrieve, resume and/or update such “workspace instances” they would be able to do so whilst ensuring the *semantic and logical integrity and continuity* of the entire dynamic system. In this way there will be a consistency of representation that would allow the traversal within and indexation into any permutation of AXMEDIS Object LifeCycle spaces (LC1-LC5) and Workflow Interaction Environments (AXWF-IE, 0-2) for any scenario of (sub)workspace storage/re-call - as outlined later in this document and described in the AXMEDIS Requirement Document as described in the AXMEDIS Requirements Document.

2.3.9.5 WF-Exchange_ID Protocols and Methods

An Exchange_ID is defined as a logical neutral format. This means it indicates what should logically minimally and necessarily be passed but it rightly defines itself not in terms of some local dialect from any particular WFMS currently in the market but in terms of generic knowledge engineering-led ontology which is designed exactly to ensure efficient effective as well as complete, consistent, and coherent messaging.

So as long as all the necessary data from WF-Exchange_ID as described, are made available appropriately to various contexts of transaction, i.e. to each transactor as required, so that:

- a) Those systems that need specific Exchange data elements in order to pass the right data back and forth can efficiently find, within the WF-Exchange_ID, the data for targeting/linking/binding/tracking their Requests/Responses and any related Objects
- b) Those systems that do not need to know/process certain elements of the Exchange data are not forced to process those bits

Then the WF-Exchange_ID design will have satisfied our requirements.

2.3.9.6 Mapping the WF-Exchange_ID to WebServices XML Envelope or DCOM Call in .NET

The WF-Exchange-ID format (which is basically a text string) can be conveyed using any communication protocol as desired.

It must be noted therefore that the WF-Exchange-ID as devised above includes:

As a WebServices XML Envelope

- 0) RequestID
- 1) addresseeID
- 2) User_Credentials
- 3) The method invocation,

- 4) INPUT/OUTPUT/NOTIFICATION parameters mapped in the WS XML Envelope
- 5) ObjectID (Optional)
- 6) Any other Message

Or in *.NET remote method invocation (DCOM)*:

Same as the above but parameters mapped in the DCOM Call and WF-Exchange_ID encoded as C++ string.

Where WebServices are deployed for AXWF transactions with the AXMEDIS Service Provider Components such as tools/engines/editors etc, the WF-Exchange-ID has to be encoded as an XML string and accordingly its parameters (e.g. the invoked engine and method etc) will have to find an appropriate mapping or expression in the WebService XML envelope. On the other hand if .Net remote method invocation (DCOM), were to be used then some of the WF-Exchange_ID parameters have to be mapped (i.e. appropriately expressed) in the DCOM call and the WF-Exchange-ID would be encoded as a C++ string. Accordingly some replication of the WF-Exchange-ID parameters will occur in the invocation call or envelope.

The above data exchange requirements are protocol invariant but should be deliverable by for example the XML-RPC protocol over HTTP as deployed by some workflow engines such as OpenFlow

The available Workflow engines as candidates for integration with AXMEDIS will not be expected to use the term WF-Exchange_ID necessarily but will have to have their own equally capable WF-Exchange format and library for parameters that will thus be logically consistent with the AXMEDIS WF-Exchange_ID syntax and Semantics as presented herein. For example some systems such as Openflow use AXRQID but the content of their respective WF-Exchange format is in any case found to comply with our WF-Exchange_ID that is devised as a neutral standard. Most Workflow systems, as applications that are designed to make an inroad into the marketplace, are expected to be relatively generic and customisable; thus accommodating; particularly with respect to their applicable library and so should be either readily WF-Exchange_ID-compliant or be easily made to conform.

2.3.9.7 *Secure Session Identity Management, Single AAA-Sign-on and Single All-in-Billing*

Within the environment of the relevant AXMEDIS components transacting with the AXWF to provide services, the only sub-fields of the workspace_Instance_ID of interest to be processed shall be those that are essential for clients' secure session identity management, single sign-on and single-all-in-billing requirements for each session and each session owner so as to provide and bind services/notifications coupled to the requesting session and chargeable to its owner as well as to ensure globally complete and consistent service delivery, rights accounting and billing with traceable audit trails.

In practice for the AXMEDIS components this means that only a few sub-fields from the workspace_Instance_ID structure, as described above, will be required to satisfy their needs for binding and tracking.

Thus every usage instance has to be afforded secure triple A services with a single sign on i.e. secure Authentication, Authorisation and Auditing assured on the basis of the available enterprise computing platform. Strictly, this is outside the scope of the provisions foreseen within the AXMEDIS Framework. However such modalities of operation as above that may be deemed desirable for at least certain enterprise platforms have to be borne in mind to ensure Maximum AXMEDIS Integration with some potential client enterprise platforms.

In this context, a primary consideration is the domain for the single sign-on system, that is the applications which are expected to jointly use the single sign-on system and thus to "trust" each other. In our case the

domains are all the AXMEDIS modules (engines, editors, WF, AXOM, etc) inside a Company's AXMEDIS framework.

In principle to operate a secure session identity management, it is assumed that three infrastructural layers should be in place:

- 1) A secure relay (PKI-based) of only the relevant “session-and-owner-specific_ID” to help targeting the requested services for delivery to the right session of course but also for billing and accounting.
- 2) A central global Rights/Services –usage module to which each service-providing component within the AXMEDIS framework sends its record of services delivered for each session and Owner_ID so that each user/project-owner can get a single cumulative bill for all services (rights used up so far etc) for each of its sessions within each project
- 3) A Trusted System for secure (wireless) sign-on must be in place

It is however important to note at this stage that in practice, as a single sign-on facility falls outside the scope of this R&D work, we need to address only the taken-for-granted technological facility that must exist within the AXMEDIS User Environment for sharing the relevant Exchange_ID sub-fields between all modules of AXMEDIS, without needing to login for every module/application (this in generic terms boils down to passing some selected sub-fields from the “workspace_Instance_ID”, itself a sub-field of Exchange_ID. In Openflow “credentials_ID” plays the same representation role as envisaged for workspace_Instance_ID and as such it would be “credential_ID” as an equivalent that would be passed). We have also to consider that we assume that all the AXMEDIS components used inside a Company (e.g. Giunti) trust each other and so a simple mechanism for sharing credentials is sufficient; for example:

The first application or service to which a user connects, has to identify him, using the single sign-on system User Directory and getting the required credentials (whose content and format depends on the single sign-on system in operation) valid for this new session (and typically with an expiration time). As soon as the same user connects to a new application or service, these credentials are passed to the respective application, which can check their validity of the relayed single sign-on signature thus made available to it.

Typically the first application is expected to ask for user/password and check with the User Directory. The application maintains the user/password in session memory; these become the “credentials”.

- when a new application is called, the user/password is sent to the new application (possibly encrypted).
- The new application decrypts the credentials and checks again with the User Directory

In this context within a Microsoft environment, the Microsoft Identity framework will be deployed.

2.3.10 Sufficiency and Necessity Criteria for the Design of Exchange Messaging Format

The AXMEDIS Workflow Data Exchange Format (AXWF-DXF) is to be designed as a canonical standard such that it allows a seamless and unique mapping of transaction syntax and semantics for:

- *the AXWF and all the relevant AXMEDIS-native components likely to interact with the AXWF*
- *the workflow-native indexing of the above lifecycles internally to cater for the above questions i.e. fulfil all the requirements for object/actor/workflow locating and tracking as well as allowing seamless interrupts and resumption of (sub) workspaces i.e. workspaces save and restore at any levels.*

However the purpose of the above analysis is not to demand that the AXMEDIS Object Model or the AXMEDIS Workflow Data Exchange Format should in any way be overloaded with additional internal workflow status data but that the Exchange Format must allow a *binding* (mutual referencing) of each request/command from the workflow *with* the response to it (and vice versa) to or from any AXMEDIS-native components as well as to the involved workspace instance internally within the AXWF.

In this way a generic model for AXWF-AXMEDIS transactions is created that abstracts from the idiosyncrasies of design/processing modes within individual adopted workflow engines. At the same time the generic model provides a vehicle for sufficient and necessary status data/parameters to be exchanged between any AXWF and the relevant AXMEDIS components through a single composite string structure, here to be named WF-EXchange_ID. This WF-EXchange_ID is transparently and uniquely decodable and decomposable within the individual workflow engines to provide all the necessary lifecycles information. For some workflow engines depending on the number of users and the range of user queries and functionalities supported, some of the fields within the above canonical WF-EXchange_ID format may be unused and thus will be a null-set but this is of no concern once a neutral exchange format is established as sufficient, necessary and thus canonical for all transactions.

The design of such WF-EXchange_ID in a way that delivers all the above requirements is one of the critical challenges in achieving full AXMEDIS Framework integration whilst recognising the distinct boundaries between workflow-native and AXMEDIS-native status data and providing an elegant and generic bridge that will work seamlessly in all cases. The Conceptual, Logical and Physical design of such a bridge will be addressed later in this document.

2.4 Workflow Transactions Specification for all Interfaces and Selected Scenarios

No.	Workflow Manager Interface/Plug-in	Data Exchange Spec' Between Transactors	Description of Transaction
A	Editor/Viewer WF Plug-in		
1	AXMEDIS Editor: Plug-in Manager	<p>In the first instance, the Workflow Manager will inform the Plug-in manager of its role and then a method to invoke (API/ActiveX) the workflow editor (registration).</p> <p>Any module which wants to access a plug-in (in our case the AXMEDIS WorkFlow and, from the other side the AXMEDIS Command and Reporting) calls the Manager in order to know how to invoke it.</p> <p>Thus, the information exchanged will be the role of any plug-in associated with the plug-in manager and API/ActiveX invoker. (This can also include the path for the executable file).</p> <p>The interface will be based on the definition provided by the</p>	<p>This should allow the use of external plug-ins which can be deployed for accomplishing various tasks (e.g., workflow plug-ins).</p> <p>These plug-ins have to be certified in some manner to guarantee the safety of their environment. Thus communication through these plug-ins has to be performed in some protected manner since the content is going to be processed by them.</p> <p>AXMEDIS Editor is assumed to be as versatile and flexible as possible. In order to achieve this goal, various AXMEDIS Editor modules need to support plug-in technology.</p> <p>Hence, an AXMEDIS Editor Plug-in Manager is needed. Such a manager will be able to support installation/registration of plug-ins, to load such</p>

		<p>workflow manger.</p> <p>Through this interface the data will be exchanged between WF Manager and AXOM (Commands and Reporting)</p>	<p>plug-ins for AXMEDIS Editor modules which request it and to maintain/manage the relationships among plug-ins and related entities or actions, e.g. the AXMEDIS Editor Plug-in Manager shall maintain relations among a specific set of metadata and the corresponding production or visualisation plug-ins</p>
2	AXMEDIS Editor: Commands and Reporting	<p>The WFM will pass control (commands) for any actions needed and will receive messages (results) as defined by this component. These commands will be as defined by the AXMEDIS Editor.</p> <p>As a result of the interaction activity the WF will receive information like the kind of action, actor and the tool involved, e.g. the relevant time-stamps, etc.</p> <p>For an Editor/Viewer the typical command is “launch the Editor/Viewer passing parameters (e.g. ID of the object to be edited)”;</p> <p>(see later).</p> <p>For the AXOM typical command is add/delete/ modify an Object with parameters (see later).</p> <p>The Interface will be through AXOM_WebService_Listener module which will handle incoming Request and sends the notifications back to Workflow.</p>	<p>This plug in interface allows the control of action of the AXMEDIS Object Manager and the send messages and controls outside;</p>
3	AXMEDIS Object Manager	<p>The WF will interact with the AXOM through the Command and Reporting tool.</p> <p>So all the commands defined by the AXMEDIS Editor will be used.</p> <p>As a result of the activities, the WF will receive information like the kind of action, actor and tool involved, e.g. the relevant time-stamps, etc.</p>	<p>An AXMEDIS object model container wrapped for secure AXMEDIS object content manipulation;</p> <p>AXMEDIS Object Manager is the coordinator of all other modules used by or built in the AXMEDIS Editor</p>
4	AXMEDIS Editor Workflow plug-in	<p>This is the native editor/viewer for the WF; hence the data exchanged here will be the definition of the</p>	<p>The workflow editor and viewer is the gateway interface for creating and changing new project</p>

		WF and its components.	workspaces referred to as NPDs in the terminology adopted for the AXMEDIS Workflow and object life cycle analysis elsewhere in the AXMEDIS Requirements documents
B	WF Engine Plug-in In principle, the exchange information in terms of the parameters to be passed from the AXWF to invoke a particular engine are typically confined to essential session_ID-specific type of information; i.e. AXOID(s), AXRID, start_time, end_time, user_ID, and perhaps also AXTID (AXTID), tool_ID (AXTID), result (success/failure,...), new AXOID. However in practice, the scheduler is capable of managing the allocation of a large number of processes and thus as the allocation is run-time dependently decided by the scheduler. It is therefore unnecessary to pre-specify the AXTID or the tool_ID for this class of transactions involving AXWF on the one side, and, the Engines as the providers of registry specified services, on the other side. In what follows the session_ID is used to suffice as a reference synonymous to client/session_owner/workspace_Instance_ID.		
1	AXMEDIS Editor Plug-in Manager	Same as in Section A.	
2	Compositional/Formatting Engine: Engine Commands and Reporting	<p>A Composition/ Formatting request is sent to the compositional/ Formatting engine passing the required parameters for e.g.</p> <p>AXOID(s), AXRID, start_time, end_time, session_ID, result (success/failure,...), new object id.</p> <p>The engines are activated through the rule editor: this is used by WF only for receiving notifications.</p> <p>What has to be done is to generalise a control behaviour instead of creating several identical modalities of work.</p> <p>All the Engines are activated through invocation of given Rule that has to have been included in the activation call or can send an activation signal that is detected by the Scheduler through to the Engine.</p> <p>The activation parameters can include, for example:</p> <ul style="list-style-type: none"> • a Selection (see for details the part on database) • a rule • activation time 	<p>The Compositional/ Formatting Engine receives a composition request coming from the AXMEDIS Workflow Manager or the internal scheduler activates a rule from the Active Composition Rules</p> <p>PnP engine requires the formatting engine on quick trial will provided a response on whether the functionality to accomplish the task required is possible and return the results through WF to the PnP Engine before actual activation of the rule. On full trial or actual activation, a new AXOID will be returned or an error response, via WF to PnP Engine.</p>

		<ul style="list-style-type: none"> etc. <p>The WF can de-activate or stop a rule, etc...</p>	
3	Collector Internal Engine: Engine Commands and Reporting	Used by WF for receiving notifications from the engine.	
4	Collector Internal Engine		Collector/Transcoder Engine will migrate the content from the Crawled Results Integrated Database (created by the Crawler Collector Indexer) to the AXMEDIS database format. These tools will be capable of processing data; automatically updating the content into the AXMEDIS database when these are updated into the CMSs. In addition, the same engine tool will be capable of integrating tools (by means of plug-in) for direct estimation of fingerprint to complete the indexing by using the results of the following task
5	AXEPTool Command and Reporting	Used by WF for receiving notifications from the engines (Publication and Loading) and from Monitoring and Active Selection	
6	AXEPTool: Publishing and Monitoring Object	<p>The WF will receive the data produced by the monitors to update the history of the object.</p> <p>Based on these data the WF can trigger further tasks.</p>	<p>The main tasks of this component are the following:</p> <ol style="list-style-type: none"> 1. when an object is published or updated (by the Publishing Tool Engine) in the AXEPTool OUT Database of an AXEPTool connected to the P2P Network, a related event is generated and broadcast to the P2P Network. 2. When events of the types referred to in step 1 above are received from the P2P Network, it alerts the AXEPTool Active Selection Engine. <p>The AXEPTool Monitor performs the task of continuously giving real-time feedback about status of possible downloads and uploads of AXMEDIS objects from and to the P2P Low Level Virtual Database.</p> <p>At a first level of detail there are two main components responsible for</p>

			<p>these tasks; The Downloads Monitor for downloads and the Uploads Monitor for uploads.</p> <p>These modules give feedback about related operations status, working with information taken from the P2P Network and information taken from the IN/OUT Database Managers. Data produced by Monitors is used by the AXEPTool Monitor GUI which gives a graphical representation of status.</p>
7	AXEPTool: Publication Tool Engine	<p>A request for publishing is sent to the Publish Tool passing the required parameters for e.g.</p> <p>AXOID(s), AXRID, start_time, end_time, session_ID, result (success/failure,...), new AXOID.</p> <p>The engine is activated through the rule editor; this is used by WF only for receiving notifications</p>	<p>The Publishing Tool Engine, according to the Rules and Active Selections as edited by the user with the Publication/Loading Rules/Selections Editor, moves the elected objects to the AXEPTool OUT Database, so that they could be published and made visible in the P2P Low Level Virtual Database. After the objects are published the Publishing and Monitoring Objects component is notified and it broadcasts the event to the P2P Network.</p>
8	AXEPTool: Loading Tool Engine	<p>A request for Loading is sent to the Loading Tool passing the required parameters for e.g.</p> <p>AXOID(s), AXRID, start_time, end_time, session_ID, result (success/failure,...), new object id.</p> <p>The engine is activated through the rule editor; this is used by WF only for receiving notifications</p>	<p>The Loading Tool Engine, according to the Rules and Active Selections (as edited by the user with the Publication/Loading Rules/Selections Editor), moves the objects to the AXMEDIS Database.</p> <p>If an event arrives through the P2P Network, related to a new object publication or update, the Publishing and Monitoring Objects component is notified; it alerts the AXEPTool P2P Active Selection Engine which downloads the new or updated object and verifies whether it is in an Active Selection. If so, the Loading Tool Engine is advised and it loads the object.</p>
9	AXEPTool:P2P Active selection	<p>The WF will receive the signal (as defined) from P2P active selection engine for the loading of the new AXMEDIS object.</p> <p>The WF will record the time-stamps and update the object history.</p>	<p>This component, mainly, provides for the following tasks:</p> <ol style="list-style-type: none"> 1. It receives an event sent from the Publishing and Monitoring Objects component. This event is to notify it of a new object having been published on the

		<p>The result is also notified to the WF.</p>	<p>P2P Network or an object having been updated on the P2P Network.</p> <p>2. It verifies if the new or updated object, as referred to under point 1 above, belongs to an Active Query. If so, then it alerts the Loading Tool Engine because a new loading operation must be performed for such an object.</p> <p>It periodically expands Active Selections according to their time slices as defined.</p> <p>An Active Selection is a selection of AXMEDIS objects (i.e. as a result of a query) or Queries that the user selects to be expanded and maintained by the AXEPTool P2P Active Selection Engine. If an Active Selection is a Query, then the Selection should be expanded to include new objects because it is possible that new published objects are published in the P2P Network and they are “suitable” for that Selection.</p>
10	Protection Tool Command and Reporting	Used by WF for receiving notifications from the Protection Tool engine	
11	Protection Tool Engine		
12	PnP Command and Reporting	<p>Used by WF for receiving notifications and commands from the PnP engine</p> <p>In general PnP uses command and reporting to report its status to WF and any programme activation status.</p>	<p>Command and Reporting communication can be used between WF and PnP in order to request formatting of AXMEDIS object, i.e. for PnP to request the formatting engine to perform some operation on an AXMEDIS object via WF.</p> <p>This can be done in different level including quick-trail, full-trail and actual activation. WF also channel back the completed request from the formmating engine to the PnP engine together with the new AXMEDIS object ID resulted from the operation.</p>
13	PnP Engine	In order to activate an on-demand operation, a request is sent from WF to the PnP engine with a valid PnP programme passing the	The Programme and Publication Engine will be developed exploiting the work performed for the Publication tool in WP4.4. This will

		<p>required parameters for e.g. AXOID(s), AXRID, start_time, end_time, session ID, AXTID, result (success/ failure,...), new object id etc, as specified in the PnP schema.</p> <p>The PnP engine is activated by WF at system initialisation.</p> <p>PnP Engine require WF to communicate with the Formatting Engine. A request of formatting can be done by, for example: CallFormatEng(ProcessingFuncID, "xml-of-the-formatting-req", WhoCalled) (e.g. CallFormatEnmg(ID_of_a_scale_func, "<selection>AX_Obj_ID</selection><parameter:size_width>88</parameter:size_height>888</parameter:size_height>xyz</>...", ID_PnP_Eng);)</p> <p>This is called from the PnP Engine to WF and WF to Formatting. After that Formatting calls WF and WF calls PnP Engine via TCP/IP GSoap with the resulted new object ID. WF is responsible to send the actual rules to the Formatting Engine.</p> <p>For object distribution to a specified channel/terminal, the PnP Engine requests the object via WF with the AXMEDIS Object ID and WF return the PnP Engine an URI to obtain the object.</p>	<p>allow the reception of specific commands (requests) for creating content produced by exploiting the capabilities of the AXMEDIS formatting engine. In addition, the Programme and Publication Engine will also have the capabilities for producing the programme based on the specific rules.</p> <p>The active engine is continuously running software; accessing the system clock to process a list of programmes, which consists of "rules" to make available AXMEDIS objects to the specified destination channels at the correct time, taking into account the transfer and/or formatting time (if required). This is achieved by the input of <i>activated</i> rules to control scheduled distribution.</p> <p>For on-demand PnP processes, WF activate the PnP engine via web services interface with a valid PnP programme with AXObjID, time (immediate), channel, terminal, The time is set to immediate and PnP Engine process this immediately (formatting as necessary, delivery to the channel specified or terminal specified if this is possible. On completion, send notification (success/failure) to WF.</p> <p>Profiles of the users and Profiles of the distribution channel (including bandwidths, formats etc) are required by the PnP so that the PnP can request for the proper formatting operations depending on the profiles. PnP is to request the profiles from WF.</p>
C	WF Rule Editor Plug-in		
1	AXMEDIS Editor Plug-in Manager	Same as in Section A	
2	Compositional/Formatting Engine: User Commands and Reporting	Same as in Section B	
3	Compositional/Formatting Rule Editor UI	AXOID, start_time, end_time, session ID. AXTID, tool ID.	

		<p>result (success/failure), rights, etc.</p> <p>As a result of the activities, the WF will receive information like the kind of action, actor and tool involved, e.g. relevant time-stamps, etc.</p>	
4	PnP Editor: User Commands and Reporting	Same as in Section B	
5	PnP Editor	<p>AXOID, start_time, end_time, session_ID, terminal_ID, tool_ID, result (success/failure), rights, etc.</p> <p>As a result of the activities, the WF will receive information like the kind of action, actor and tool involved, e.g. the relevant time-stamps, etc.</p> <p>Actor/user can start the Editor manually and WF start the PnP editor.</p> <p>PnP Editor uses command and reporting to report status to WF, via TCP/IP GSoap. PnP Editor repository does not report to WF. PnP Editor send programme activation request to PnP Engine directly and send a notification message to WF.</p>	<p>A Programme Manager (user) can use the PnP Editor (a GUI) to interact with the Query Engine and to make selections from the Query Engine results in order to schedule some programmes (e.g. on a daily, weekly, monthly, yearly basis) with the following rules:</p> <ul style="list-style-type: none"> • WHAT: the AXMEDIS object of interest • WHERE: destination channel, where to publish e.g. iTV or kiosk or other, and “where” profile • WHEN: date, time, slot, duration • HOW: direct transfer, reference or the requiring formatting engine <p>One or more of the above rules make up a PnP programme which is represented using XML as specified in the PnP schema.</p>
6	AXEPTool: User Commands and Reporting	Same as in Section B	
7	Publication Rule Editor/Selector	<p>AXOID, start_time, end_time, session_ID, terminal_ID, tool_ID, result (success/failure), rights, etc.</p> <p>As a result of the activities, the WF will receive information like the kind of action, actor and tool involved, e.g. the relevant time-stamps, etc.</p>	<p>The Publication/loading Rules/Selection Editor is the AXEPTool GUI for editing Rules/Selections in order to Publish/Load AXMEDIS objects.</p> <p>Also, using the Publication/Loading Rules/Selections Editor, the user can activate a Selection submitting it to the P2P Active Selections</p>
8	Loading Rule Editor/Selector	AXOID, start_time, end_time, session_ID, terminal_ID, tool_ID,	

		<p>result (success/failure), rights, etc.</p> <p>As a result of the activities, the WF will receive information like the kind of action, actor and tool involved, e.g. the relevant time-stamps, etc.</p>	
9	Protection Tool: User Commands and Reporting	Same as in Section B	
10	Protection Tool Rule Editor UI	<p>AXOID, start_time, end_time, user_ID, terminal_ID, tool_ID, result (success/ failure), rights, etc.</p> <p>As a result of the activities, the WF will receive information like the kind of action, actor and tool involved, e.g. the relevant time-stamps, etc.</p>	
D Query Support			
1	QS Web Service UI	<p>The WF will use the QS to enable users to query the AXMEDIS DB.</p> <p>The WF will invoke the UI for the user.</p> <p>The WF will pass the query to the QS based on the syntax defined.</p> <p>The result (list) of the query will be collected by the WF and may be further filtered</p>	This interface will be based on Web-services.
2	QS UI	Same as above.	
3	AXMEDIS Object Loader/Saver	The WF will pass the AXOID which is to be loaded or saved through WF's Check-out, Check-in interfaces respectively. In addition to it WF will also pass start_time, end_time, user_ID, terminal_ID, tool_ID, result (success/ failure), rights, path for loading/saving, etc	<p>AXMEDIS object Loader/Saver is a Web service that is capable of getting an AXMEDIS object (in the MPEG21 compliant format plus the additional information stored inside) and putting it in the database, that is the loading function; and it is also capable, given an Object ID (AXOID) to return the AXMEDIS object in the MPEG21 compliant format, that is the saving function.</p> <p>The AXMEDIS Loader/Saver interact, or better is part of, the AXMEDIS database Interface</p>

3 Workflow DataBase

Before specifying the workflow database, an analysis of the status types and partitions is presented as follows:

3.1 Semantic Analysis of Status Data Required for Object Tracking and AXWFM Control

From the requirements for tracking, audit and control of objects and workflow as determined following our extensive process of multi-sector workflow knowledge elicitation, (as reported in the AXMEDIS Requirements Document), the following knowledge analysis arises pertinent to the logical design of the interface spaces, syntax and semantics as required to deliver full AXWF-AXMEDIS integration.

In particular this analysis has supported the rationale for the development of the 4-tiered design of interfaces to accommodate all types of possible interactions between the AXWF and the relevant AXMEDIS components i.e. a streamlined integration via separate classes of interfaces designed for the integration of

1. The AXWF and the AXMEDIS Workflow Editor,
2. The AXWF and the AXMEDIS Rule Editor/Viewers,
3. The AXWF and the AXMEDIS Engines
4. The AXWF and the AXMEDIS Query Support Interface

as illustrated in the 4 UML diagrams and the 8 Scenario diagrams included in this document (see Sections 2.3 and 2.8).

The Domain Knowledge Analysis based on the requirements elicitation data has allowed an initial set of Semantic Types to emerge as candidates for inclusion in various metadata partitions, based on our analysis of workflow interaction environment and AXMEDIS Object lives abstraction spaces as follows:

Here we simply list the initial, the indicative but not the actual form of the semantic types emerging as being relevant to tracking and control at this stage.

In practice Generalisation Ontologies can be deployed to allow the hierarchical decomposition and horizontal partitioning of the full domain ontology to (sub)ontologies. In this way, for any given context, the semantic integration and reasoning required for the workflow tracking of an AXMEDIS Object will remain adequately expressive, efficient, effective and above computationally manageable.

The object birth here starts when the object is first registered and deposited as an AXMEDIS Compliant Object known to, and indexable by, the AXMEDIS Object Manager.

In principle this should be a perpetual life extending into the future with object death being triggered only in the unlikely event of a full-rights object owner or an AXMEDIS Authority deciding to withdraw the object and all its versions from AXMEDIS Compliance.

This can use a set of 10 canonical Situation Assessment (SA) primitives, the 10P-SA; i.e. purpose, period (these first two status attributes are possibly to serve as a viewpoint selector i.e. filter for metadata partitioning), place, project, phase, process, partner, person, progress(-to-date), projected(-work-remaining), (using the “10P-stamped” project workflow objects, Ref Use Case Document, table 1).

We distinguish 3 distinct interaction environments for the workflow control of the AXMEDIS multimedia project processes and AXMEDIS object lives development as follows:

The Global AXWF-AXOM Interaction Environment: AXWF-IE0

This is the highest level control involving the AXWF interacting with the AXMEDIS Object Database, which is the generic level of global tracking and audit of development of object and versions of object lives.

This is associated with the following object life abstraction spaces and associated candidate triggers and states:

LifeCycle 1, (LC1): Global Reference AXMEDIS Repository Object LifeCycle:

This is that object life known to the AXMEDIS Object Manager which is to act as the universal custodian of such global reference versions of AXMEDIS Objects.

LifeCycle 1, LC1 Semantic Types:

These are candidate semantic types, including triggers, for global tracking and control of Object States at the interaction space of the AXMEDIS Workflow with the AXMEDIS Object Manager:

new-usage-instance-needed, new action, full-rights, relative-rights, rights-updated, rights-granted, rights-denied, protection-status/(un)protected, modified/rendered, new-view-created, metadata-updated, metadata-viewed, metadata missing/incomplete, ready, not-ready, interrupted-process-n, barred/stopped, history, formatted, packaged-n, edit-started, edit-completed, protection-tool-started, protection-tool-ended, license-manager-started, licence-manager-ended, wanted, deposited, owned, viewed-n, taken-n, requested-n, time-done, phase-done, process-done, waiting-on/for-process-n, awaited-by-process-n, suspended, internal, external, authorised/signed-off

The Project Manager Level AXWF Interaction Environment: AXWF-IE1

This is the next high level interaction environment between project workers and the workflow management system, whereby line managers from a particular partner or the overall project coordinator can be interacting with the workflow to enquire the overall progress of the project including the various states of progression of development of groups or individual AXMEDIS object lives, e.g. the states of all embedded sounds, videos, etc.

Thus LC3 is the overarching lifecycle which tracks the global evolutionary development path of an object as it is worked on, as the object is passed back and forth across Enterprise Boundaries in the course of its development within a project. This extends from inception to completion of a product as happens in net-economy virtual value chain constellations, for collaborative New Product Development or multi-agent contract fulfilment projects.

These are candidate semantic types, including triggers, for global tracking and control of Object Usage Instances and Rights Compliance within an Enterprise (similar to internal policing of software licences processing and compliance).

This is associated with the following object life abstraction spaces and associated candidate triggers and states:

Version-type, view-type, usage-instances-in-credit, licence-manager-request-pending, rights-last-re-newed, full-rights, relative-rights, right-last-updated, rights-granted, rights-denied, protection-status, (un)protected, metadata-updated, not-available, stopped, X-bar, Stuck-at-Y, license-suspended/rejected, wanted, owned,

viewed-n, taken-n, requested-n, internal, external, authorised/signed-off, internal/external-copyrighted, DRM-needed, DRM-added, DRM-ok, binned/discarded, running, first-pass-sealed, packaged-n, bundled-n.

The Base User Operations Level AXWF Interaction Environment: AXWF IE2

This is the environment of any project team member/user interaction with the workflow in controlling and progressing their own individual work load. Thus LC4 tracks the evolutionary path of an AXMEDIS object whilst it is being worked on in any of the distributed project cells by each single Manufacturing Cell/Work Group/Unit/Team/Person. This is associated with the following object life abstraction spaces and associated candidate triggers and states:

LifeCycle 3 and LifeCycle 4: LC3 and LC4 Semantic Types:

These are candidate semantic types, including triggers, for Project-specific local and global tracking and control of object development trajectories (Situation Assessment) in distributed New Product Development Zones; as follows:

Early, late, ready, not-ready, stuck-at-Z, pending-Y, terminated-at-N, binned/discarded, running, earliest-started, latest-finished, CPA-slack-(non)-critical, first-pass-sealed, re-drafted-n, re-sealed-n, (re)composed-n, (re)formatted-n, (re)packaged-n, (re)bundled-n, distributed, within-phase-n, starting-phase-n, at-end-of-phase-n, within-process-n, starting-process-n, at-end-of-process-n, project-part-n, time-done, phase-done, place-done, partner-done, person-done, process-n-awaited, process-n-awaiting, suspended, contingent, authorised/signed-off, handed-over-sealed, internal-copyrighted, external-copyrighted, DRM-needed, DRM-added, DRM-ok.

3.2 Semantic Types and Representation Spaces Partitioning

The total available capability for AXOID/AXWID/workspace-instance status tracking, control and inferencing is dependent upon the integrated representations of objects across four distinct but mutually supportive sub-spaces for objects and workitem states recording and indexing namely the Metadata (AXinfo), Descriptors (AXdb), PMS database serving the AXMEDIS Certifier and Supervisor (PMSdb-AXCS) all of which are within the AXMEDIS Framework, and, the database of the adopted Workflow (AXWFDB). In what follows the suggested semantic types broadly include two categories of status namely *descriptors* as distinguished from *states*. Descriptors are the relatively more stable attributes of an Object or Actor (e.g. the origin of an Object) and include optionally specified sub-fields whereas the states refer to the relatively more transient and evolving conditions of Objects, workitems and/or actors and generally include mandatory sub-field that are crucial for the direct and/or indirect command and control of the AXMEDIS service-oriented environment through the AXMEDIS Workflow Manager. A set of candidate semantic types are suggested together with their expected logical home for representation. This is the subject of ongoing discussion with colleagues including those responsible for the design of the AXMEDIS Object Schema and AXdb and has been found to be consistent with, and complementary to, their own independent findings. Thus the list below represents a set of candidate semantic types and their assignment to control-inference-supportive repositories as appropriate (i.e. AXinfo, AXdb, PMSdb (FUPF) or AXWFDB as the case may be).

This is another attempt to seek the views of all concerned who are thus being re-invited to comment in writing with respect to each state/descriptor listed below and clearly articulate and justify their views. As the list below is presented as a discussion list, colleagues are encouraged to add and/or delete entries in the list accompanied by a line or two in justification. In this way we can hopefully rapidly conclude this matter.

3.3 Establishing Metadata/Descriptor AXinfo, AXWFDB, AXdb & PMSdb Data Schema Complementarity

This category is to make available key information about Object's origin, genre/purpose, history, version number, views and rights. In the list below, the first field i.e. period/purpose/genre/version_type can be used as a viewpoint selector of metadata partitions (i.e. a metadata filter for indexing/viewing/visualisation/processing) for processing/querying only those AXMEDIS Objects or metadata partitions relevant to the current focus of the client's work.

States and Descriptors Semantic Types Candidates and Partitioning

Examples to consider include:

period/purpose/genre/version_type

Status type: descriptors

Example values: 1960s, impressionist, post-modern, medieval, 2002 etc

Assigned storage spaces: AXinfo, AXdb

rights

Status type: state

Example values: (un)protected, available, full, partial, exclusions/exclusives, suspended, withdrawn (bar_zones, X-bar, Y-bar), discarded etc

Assigned storage spaces: PMSdb (FUPF)

authorisation

Status type: state:

Example values: signed-off/sealed-status for Objects/sets-of- Objects/bundles etc

Assigned storage spaces: AXWFDB

view-type

Status type: descriptor

Example values: 3D-formats, new_view/usage/action/Object-instance- (e.g. may be created, added, awaited, requested etc)

Assigned storage spaces: AXinfo, AXdb

version_number & date (for Objects/sets-of-Objects/bundles)

Status type: state

Example values: "AXOID-6.3", 05-02-202, etc

Assigned storage spaces: AXdb, AXinfo

processing_history

Other fields: e.g edit_status, formatting_status etc.

Example values: distributed, loaded, moved, published, queried, sought, found, batched (start-time, end-time), 0, 1, 15.38

Assigned storage spaces: AXWFDB modifications/edits/formatting/rendering etc performed on the Object, etc

Assigned storage spaces: AXWFDB

origin

Status type: descriptor

Example values: creator/owner, place, partner-no, project-no, session_ID, etc

Assigned storage spaces: AXCS, AXdb

3.3.1 Tracking/ Messaging and Control Semantics To be Represented

locked, and similar sub-fields

Other sub-fields: e.g. committed, started, interrupted, ended, suspended, resumed, completed etc

Status type: state

Example value: 0, 1

Assigned storage spaces: AXinfo, AXdb, AXWFDB (this would require some form of *sync* with AXWFDB)

metadata_status

Status type: state

Example values: e.g. metadata missing/incomplete, metadata-viewed/updated, audited, assured

Assigned storage spaces: AXinfo, AXdb

drm_status

Status type: state

Example values: e.g. discarded, added, needed, ok

Assigned storage spaces: AXdb

Licence

Status type: state

Example values: e.g. valid, expired, violated, in-compliance, limited, unlimited, perpetual, temporary, ended/suspended/restricted, not-available, not-ready, requested, ordered by, awarded/ credited-added/deducted)

Assigned storage spaces: PMS-AXCS area database, here referred to as PMS-db (FUPF)

Licence are recorded and dealt with through the PMS-db; AXWFM is to generate action-logs that are needed to be made available also to the AXMEDIS Certifier Supervisor (AXCS) which has to process licenses usage auditing-control over on/offline AXWF client sessions with variable granted rights and rights consumptions.

queue-position

Status type: state

Example values: 0, 1, 2, n, 65, 43, awaited-by-process_ID/session_ID

Assigned storage spaces: AXdb or other (e.g. PnP scheduler-db). This is to serve the scheduler in the AXMEDIS Programmes & Publications Engine for control of publications stack by-pass/over-rides as may be required).

stuck-at, or similar sub-fields

Other sub-fields: e.g. pending, waiting

Status type: state

Example values: at-process-point-z, waiting-for-process-n,

Assigned storage spaces: AXWFDB from the workflow action log. Axdb or other (PnP Scheduler-db). This is to serve control decisions by workflow and/or the scheduler in the AXMEDIS Programmes & Publications Engine e.g. for control of publications stack by-passes/over-rides as may be required).

urgency *Other Sub-fields/alternatives:* CPA-(non)-critical,

Other Sub-fields/alternatives: CPA-(non)-critical,

Status type: State

Example values: 0:1, CPA-slack-(non)-criticality(n minutes/hrs/days/weeks)

Assigned storage spaces: AXWDB, possibly also AXdb or PnP scheduler-db

Please Note: there can rightly be semantic asymmetries in the way this state is interpreted and deployed for and within different AXMEDIS sub-systems where relevant.

progress

Status type: state

Example values: earliest-started, latest-finished, pending-Y, early, late, ready, not-ready

Assigned storage spaces: AXWFDB and/or AXdb

Tracking information re work-in-progress on any Objects and their locked status etc (logs) shall be stored inside the Object so that this can be readily made available to all system actors who may at any time require to access such information.

3.3.2 Representing Projected Work-To-be-done on an (AXMEDIS)Object

This need not be included inside the Object as it could more efficiently and context-specifically be supplied on-demand using the available information maintained by the AXWFM on the workflow-instances/workitems concerned. This can be implemented as some sort of a "reminder field", that is a descriptive field (string) that the user gets when accessing the Object. This is to remind users about what must be done and it is for the workflow to update this field when terminating an activity.

3.3.3 Examples of Integrative Database Inference Types to be made possible

From the above it is concluded that the states that may be stored, or also stored, in the AXWFDB as a logical space for them broadly include *three categories* of status types all of which could be subsumed as sub-fields under *history*; namely: *history progress and sign-off* status types as follows:

- i. **processing_history**
- ii. **progress** (locked ,stuck-at, urgency, queue-position)
- iii. **authorisation_status**

All other states of relevance to various AXMEDIS components shall be stored either with the object (i.e. in metadata Axinfo) or in AXdb or PMS-db. Sync functions should be provided to ensure global states integrity for those states that are instantiated, stored and mostly used in inferencing within one AXMEDIS sub-space but may occasionally be needed for local control within another sub-space (e.g. scheduler

Examples of integrative inference types commonly expected by the *Workflow Participants* (including relevant AXMEDIS components) can be as follows:

Inferences about, for example:

price, rights_tariff

rights_last_updated/renewed/awarded/added/reduced/restrictions/

distribution_zones/ (exclusives/exclusions)

To be inferred from: PMSdb

Inferences about, for example:

trailer_Object

To be inferred from: AXinfo, AXdb (if mapped)

Inferences or query support about a client's on/of line consumption of some granted right(s), for example:

rights_usage, taken-n/viewed-n, /rights-in-credit_per_client_ID
To be inferred from: PMS-db communicating with AXCS

Inferences about, for example:

applicable-rights-jurisdiction(-origin) for given AXOID

Example values: internal/external/foreign-copyrighted, native, worldwide, X-law-only, Y-law-excluded

To be inferred from: AXinfo, PMSdb

3.4 Choice of Workflow Database Technology

The WorkFlow Manager for the Demonstrators will be based upon either Openflow or BizTalk. For Openflow the WorkFlow Manager will be based upon Zope technology and can use different choices of Database technologies including:

Oracle
IBM DB2
Microsoft SQLServer
Sybase
SAP DB
PostgresSQL
MySQL
Interbase
Gadfly

In congruence with the AXMEDIS Database Module, PostgreSQL Database is recommended to be deployed for the Workflow Database.

3.5 The semantic elements to be stored in the Workflow Database (AXWFDB)

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

Openflow
Process
Activity
Transition
Instance
Workitem
Employee
Role
History

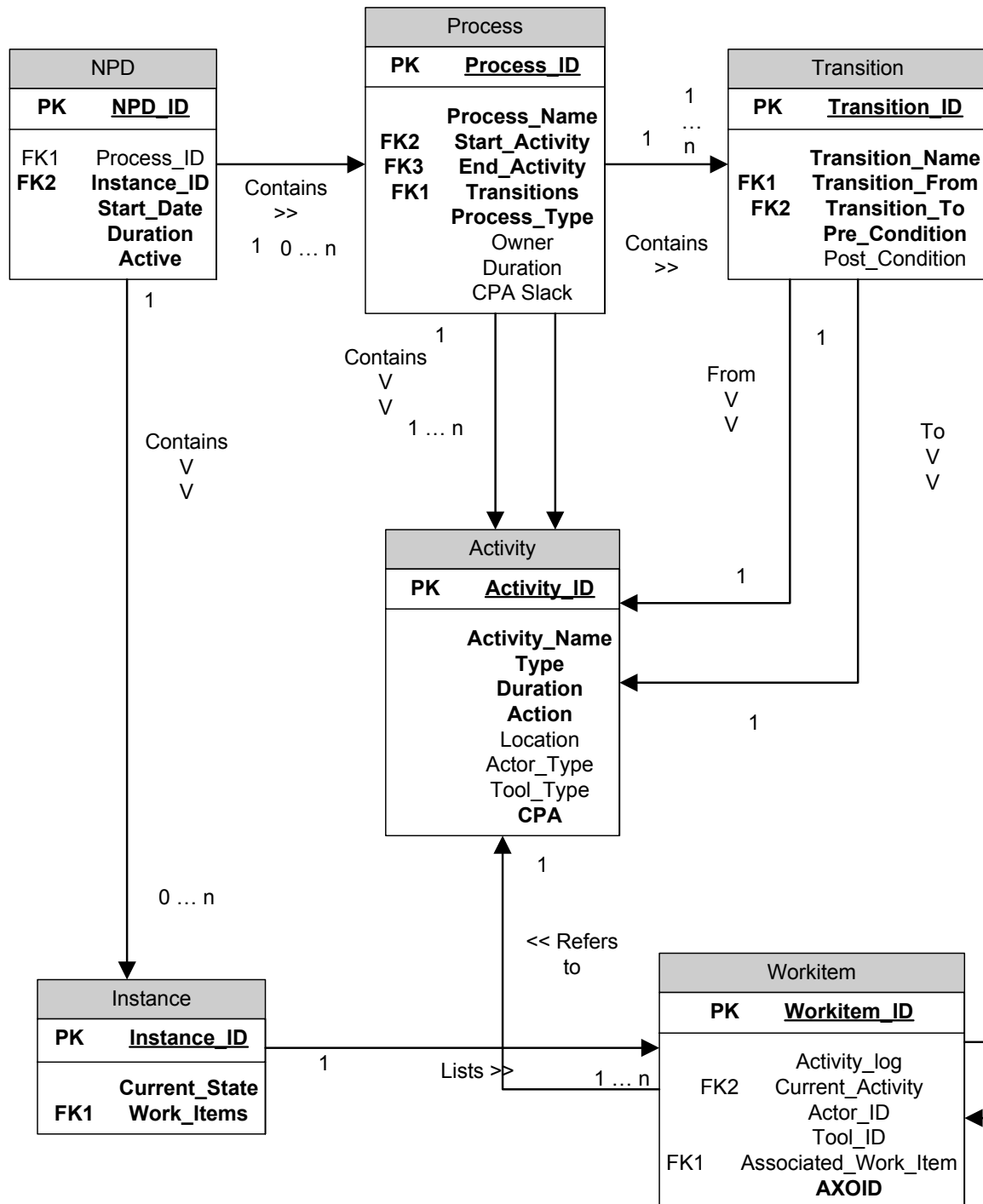
Where in the context of Openflow deployed as an AXWF, the field *History* from the above list could include the following sub-fields:

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 ActionLog 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)



3.5.1 Workflow-Embedded / Triggered Object Retrieval and Discovery Service Spaces

It is important for the AXMEDIS Modules providing object services to the Workflow Management to accommodate the various modalities that could be expected in terms of delivery of object search and discovery services to client-session owners who may or may not be online and/or directly involved in submitting Requests and in receiving the Responses/Results from such service providers. Some requests can be pre-scripted/planned as automated i.e. embedded workflow actions (batch process) to be submitted and pre-fetched as services provided to particular client-sessions per particular workflow instances etc. Other Requests may not have been planned as embedded workflow actions but may have in any case been triggered as a contingent request during a session or at-session-closure.

Thus in principle one or more of the following modalities may be in operation and distinguishable re Search Requests from the viewpoint of the *roots of and the routes to* Request Submission and/or Discovery and Delivery of Results. It must be emphasised that from the standpoint of workflow integration, the prevailing consideration here will be any implications of the above modalities on the content of the Exchange Data to be passed when Requesting Search and Discovery of various types of unknowns i.e. new and/or yet-to-be discovered objects.

A Search Request for an Object could involve the following possible features:

- *Asynchronously Proxy-Requested* and/or *Asynchronously-Proxy-Responded*, Received and Stored by AXWFM in its Database (**ASYN Requests**)

This type of request and (pre)fetch processing may be explicitly planned as an automated single/batch request on behalf of one or more workspace-instances (workflow client-sessions).

- *Synchronous Request-Response Reception* by the workspace client/session-owner and Service Provider—both online.

This type of Request is submitted and its Result Received by the workflow or workers whilst the session being thus served is online and live (**SYN Requests**).

It must be noted that in principle, in the case of Sync Requests, as the plug-in is a stateless machine, the same request may be made several times. As an example; the workflow client may issue a request, the plug-in receives it and forwards the request to an engine, the engine answers back to the plug-in which may crash in these circumstances. So the notification is never received by the workflow which owing to a time-out trigger resends the request.

- *Known Knowns (KK-Requests)*

Requesting a known object whose AXOID is precisely, unmistakeably and uniquely known to both the requester and service provider.

- *Known Semi-Unknowns (KSU-Requests)*

Requesting a known object whose AXOID is uncertain or partially unknown at least as far as the requester is concerned

- *Known Unknowns (KU-Requests)*

Requesting a known object whose AXOID is completely unknown at least as far as the requester is concerned

- *Unknown Unknowns (UU-Requests)*

Requesting any objects hitherto unknowable by the requester (and possibly also the service provider) which may satisfy some requester-stipulated interesting-ness criteria.

The UU-Request could be further specified to be at any one of two levels:

- **UU-Request-Level-1 Object Discovery Requests: Request for *Chance Discoveries* in any Pools**

This is the type of request which submits an interesting-ness or other evaluation function parameters with a request for an object fishing exercise (exploration) from unspecified (don't care), pools or domains of exploration.

- **UU-Request-Level-2 Object Discovery Requests: Request for *Discovering Chances*:**

i.e. identifying the Serendipity Index of Query Forms likely to be fruitful for Specific Pools/Events encountered (and/or opportunistically acted upon) during the search path travelled.

This is the type of request that may or may not be expecting any specific digital asset(s) as a found object(s) in response. The UU-Request-Level-2 Object Discovery Request will essentially be requesting the identity of any Pools/Repositories and associated Query_ID which resulted in the return of certain objects with a particular property or level of interesting-ness. Thus this will return Pool_IDs, AXRQIDs with associated ranking/probability of finding certain category of objects satisfying certain interesting-ness or other evaluation function (optionally it might be expected to return a sample of the respective objects).

Found Object Delivery Routes

Pre-fetching: This a scenario comprising of specific Object Getting Acts (OGA), Object Reporting Acts (ORA) and Object Fetching Acts (OFA) as planned and enacted automatically within a workflow-instance to serve forthcoming workspaces that are either about to be triggered into starting or will be next in the queue to start.

Fetching: This is a scenario comprising of specific Object Getting Acts (OGA) and Object Fetching Acts (OFA) as processed responsively at the *live* request of workflow-instance(-owner) that is presently online.

3.5.2 Unknown Objects Sought & Found Exchange Messaging Contexts

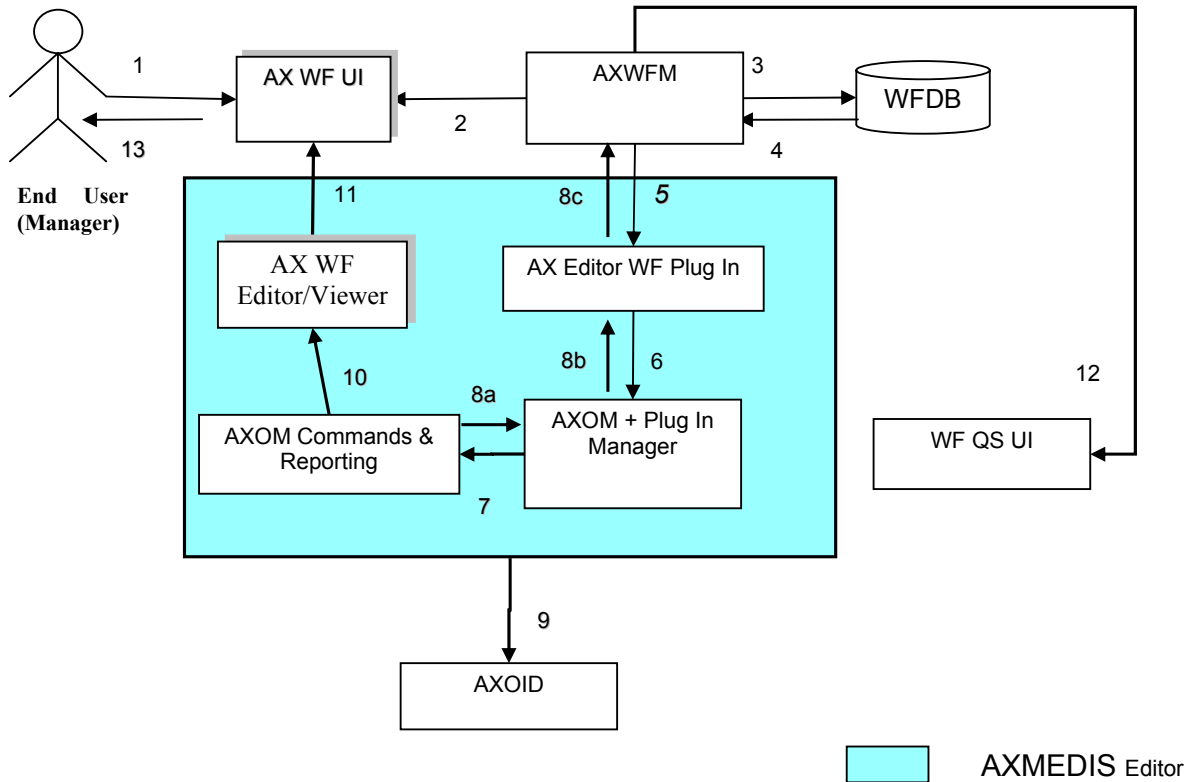
It is important to note the types of data that need to be exchanged between the Object Requester and AXMEDIS Object Service Provider Components under the various possible contexts that can arise; all of the above types may, within limits, be expected to be accommodated.

Our focus at this stage is on the knowledge analysis of the relevant possible spaces, syntax and semantics to be logically incorporated in the Transaction Exchange data as necessary and sufficient for traceability, audit and cross-linking of various contexts (e.g. linking any search request(s) to any related response(s) as well as to groups of inter-related requests and/or responses and their clients and respondents).

It is clear that for the first type of search request above, i.e. the request for known knowns, *KK-Search Request*, the fetch can be simply accommodated by either a Check-in (AXWF) or Add_new_component-AXOM functionality. For all other types of Search Request as outlined above, at the very least a pseudo/temporary/dummy/placeholder object_ID (to be referred to as Sought_object_ID) must be included in the Request string to allow the binding of the eventually fetched object, if any, to the request that originated it as well as to provide the basis for various other necessary self/cross-referencing needed for tracking/audit etc. This is developed later in this document in the section dealing with the rationalisation of the logical design for the Neutral Exchange Transaction Data Format developed for the AXWF-AXMEDIS integration.

3.6 Generic Scenarios for the Workflow Interaction with AXMEDIS Components

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.

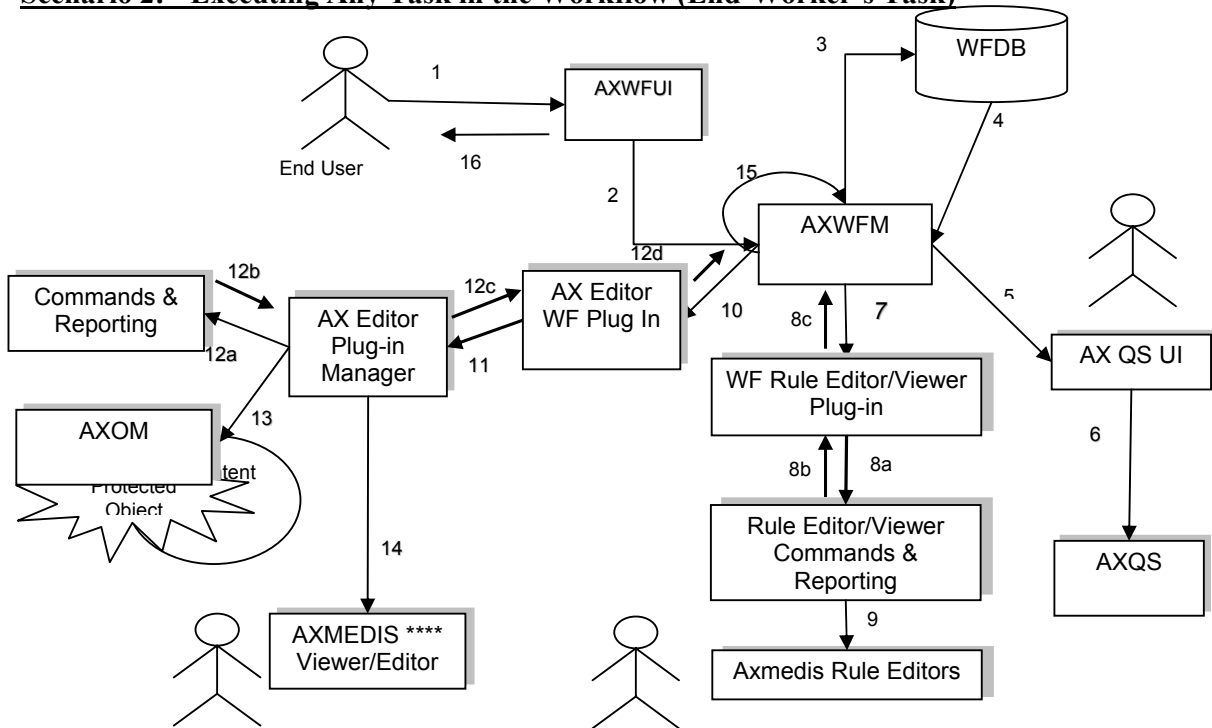
13) NPD set-up phase: The user can now select and add components to define the new project. These components can be tasks, people, project, products, objects, places, links, etc. This functionality of the workflow editor is similar to a drawing utility provided by the Microsoft Word editor, which allows 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. XPD); 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.

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

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).

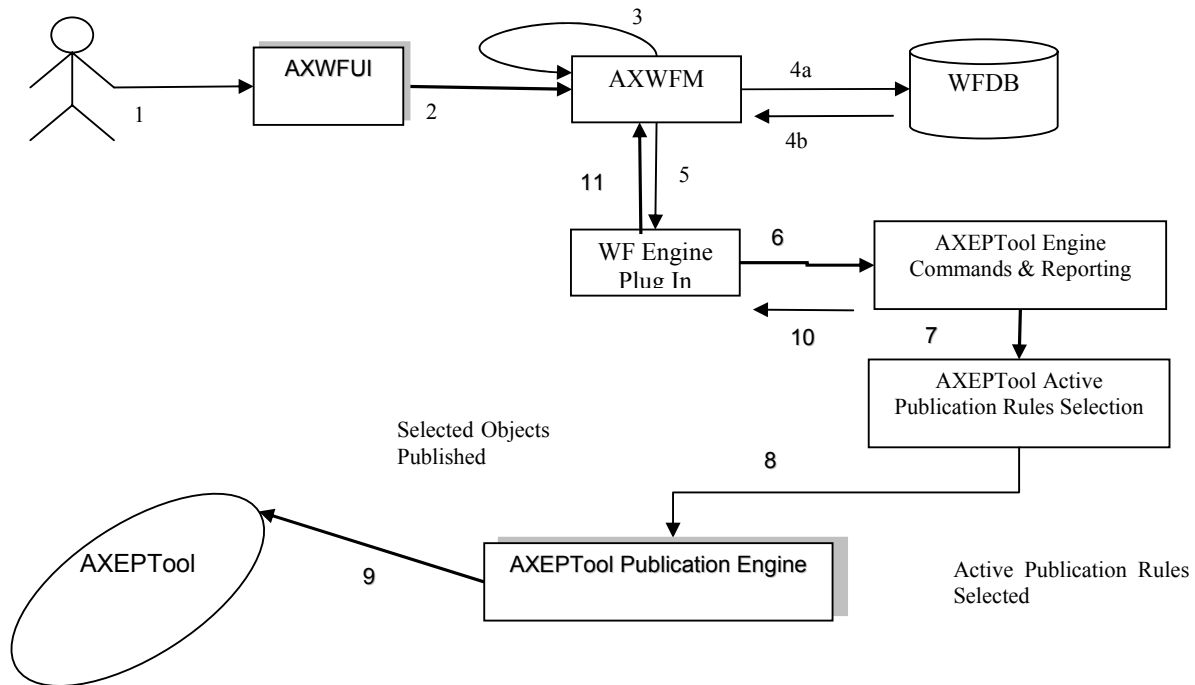
Based on the selected Action the workflow system launches the required tool using the appropriate Interface (e.g. Web-services) or plug-ins associated with that tool. If the tool is in the exclusive access area of the user, the “Check-in” and “Check-out” interfaces will be invoked.

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.

Scenario 3: Invoking the AXEPTool (Publish)**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.

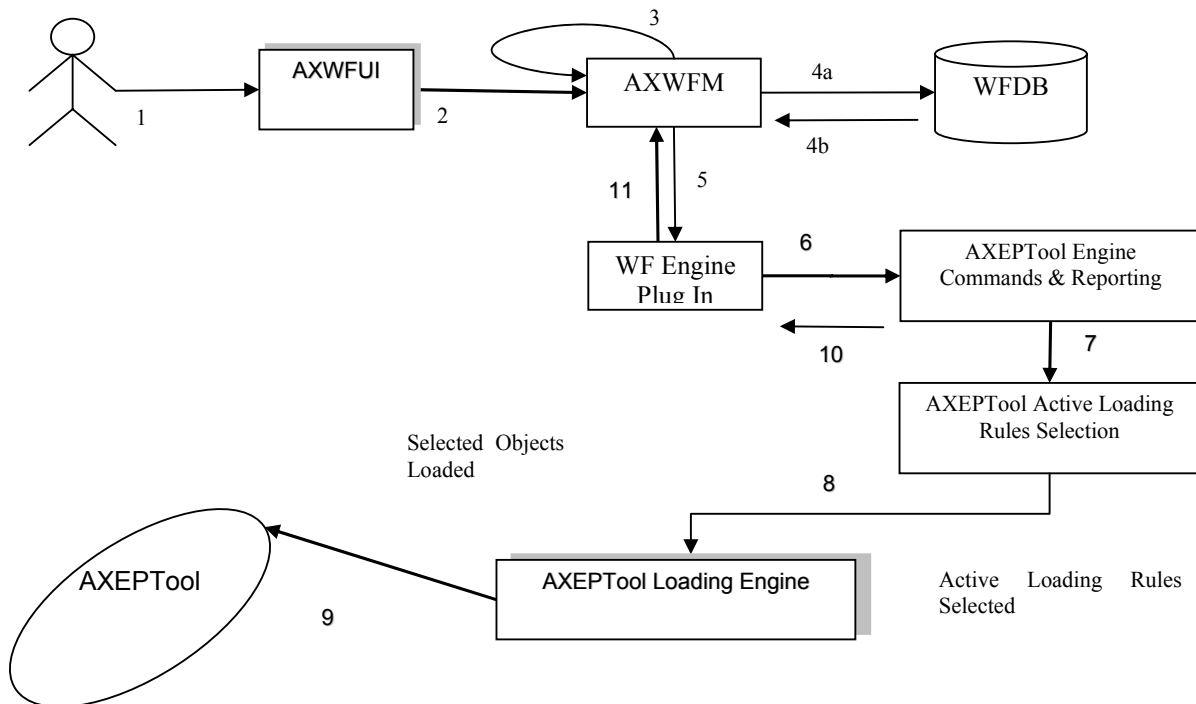
5, 6, 7, 8) The workflow system passes the so-called Active Publication Request via the workflow plug-in to the AXEPTool Command and Reporting module to trigger the AXEPTool Active Publications Rule Selection Module which enables the selection and submission of appropriate objects for publication. The 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.

Scenario 4: Invoking the AXEPTool (Load)



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.

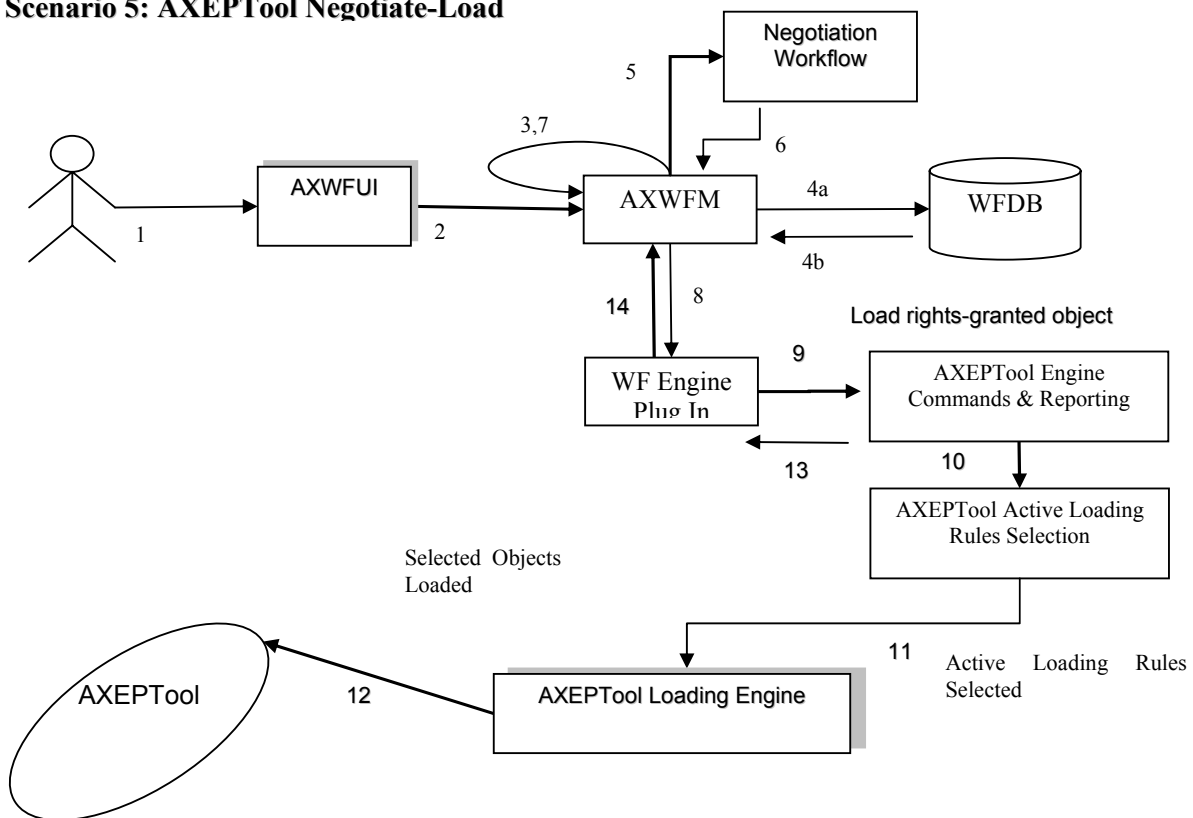
2, 3, 4) We can also assume that the client's (i.e. session-owner's) credentials_ID has been made available to be authenticated for sign-on to initiate the required exchanges information for authentication and billing purposes.

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.

Scenario 5: AXEPTool Negotiate-Load



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 Project

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.

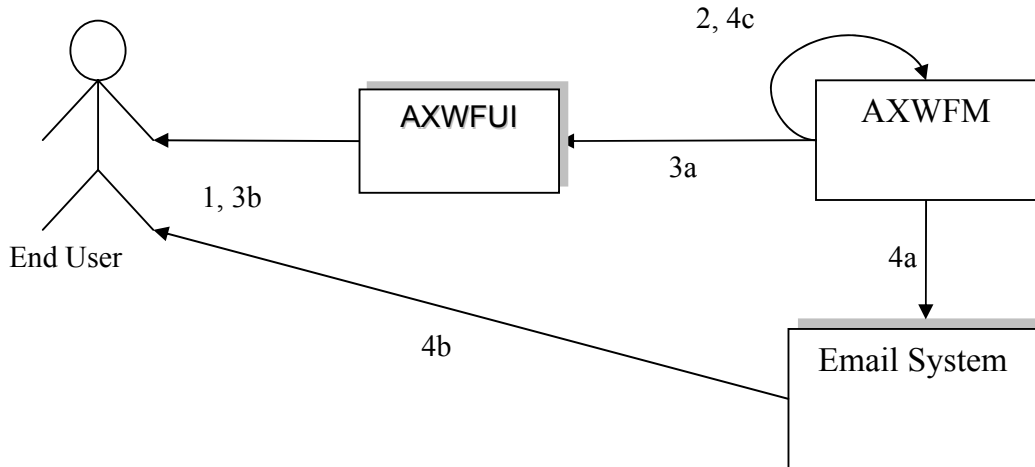
2, 3, 4) Thus the user (Worker) is already authenticated and logged onto the system and the workflow system is up and running. It is assumed that the user is in interaction with the Workflow System and that the AXWF contains the AXMEDIS Object Licence Procurement Negotiation Workflow.

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.

Scenario 6: Sending out Notifications to People**Scenario 6: Sending out Notifications to People**

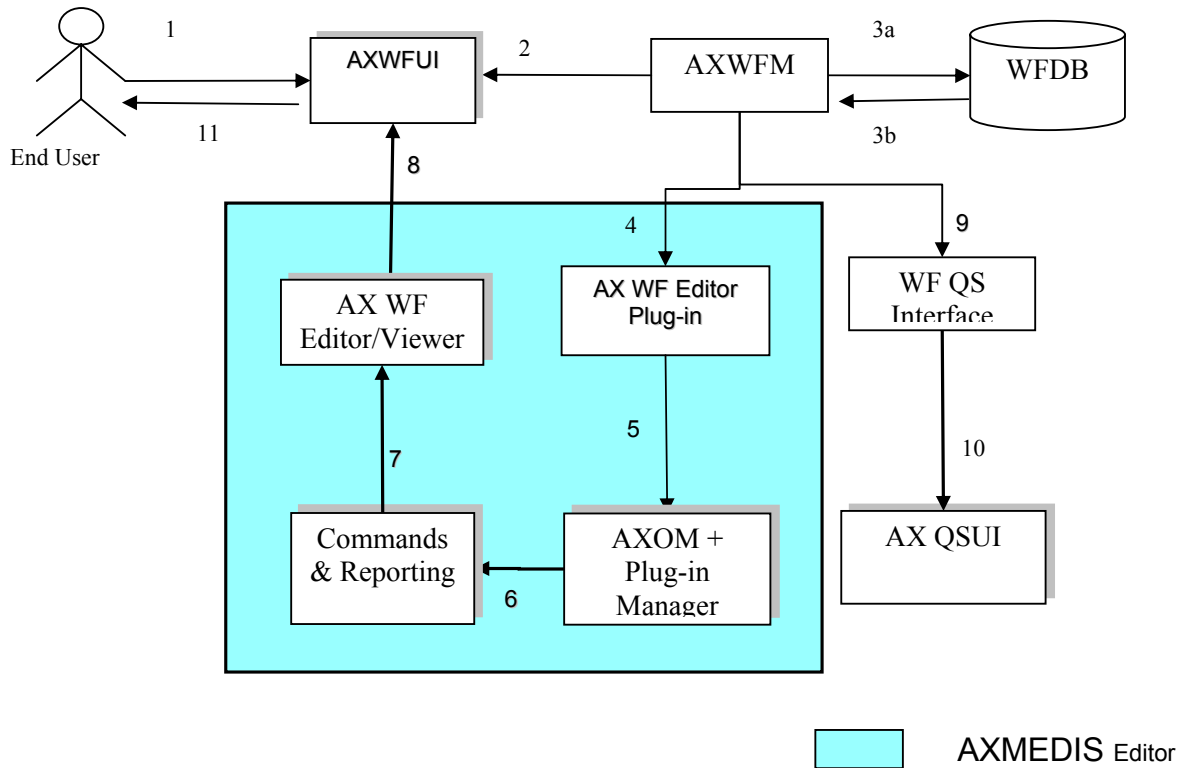
This scenario describes the process of sending out Notifications initiated by the workflow system or by the people within the workflow environment. At the end of this scenario, the user can expect that notifications are generated and sent to appropriate target(s). The scenario proceeds as follows:

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 AXEPTTool, 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.

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.

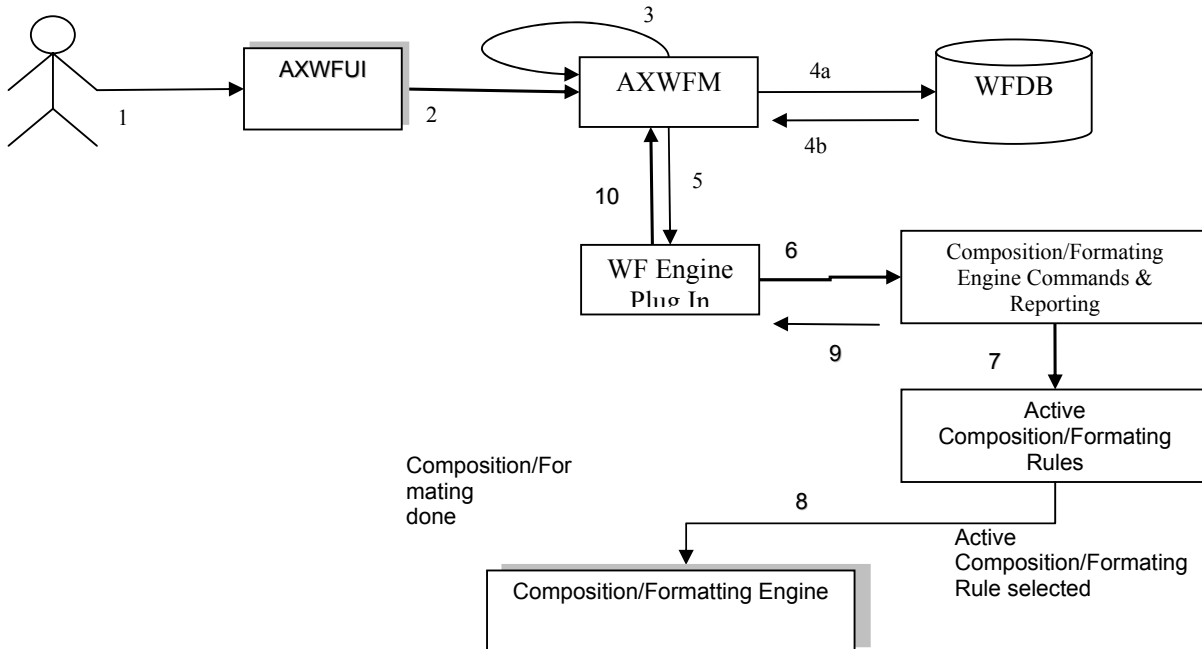
4, 5, 6, 7, 8) The workflow systems can then launch an Interactive GUI (Workflow viewer) to show the overall status of the NPD workflow along with its Critical Path Tasks (CPA), based on the results received for the above queries.

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.

Scenario 8: Invoking the 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.

3.7 AXMEDIS WorkFlow Area UML Decomposition

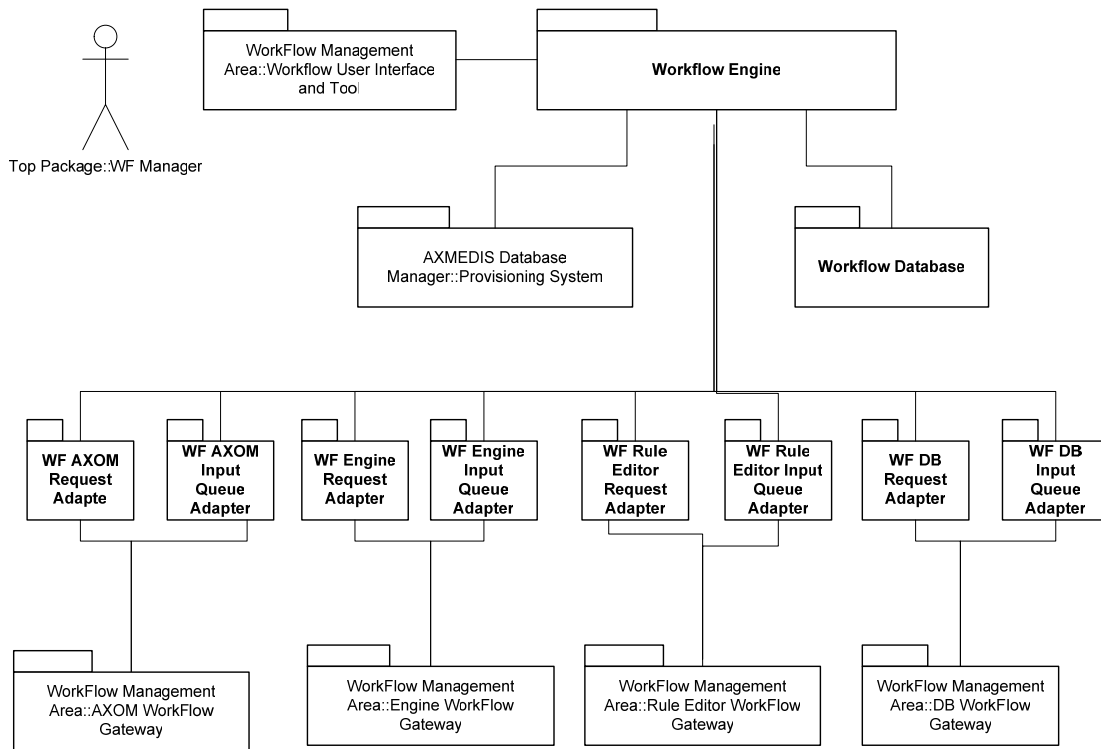
The AXMEDIS WorkFlow Area includes:

- 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
- Engine 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



3.8 WorkFlow and AXMEDIS Integration Architecture

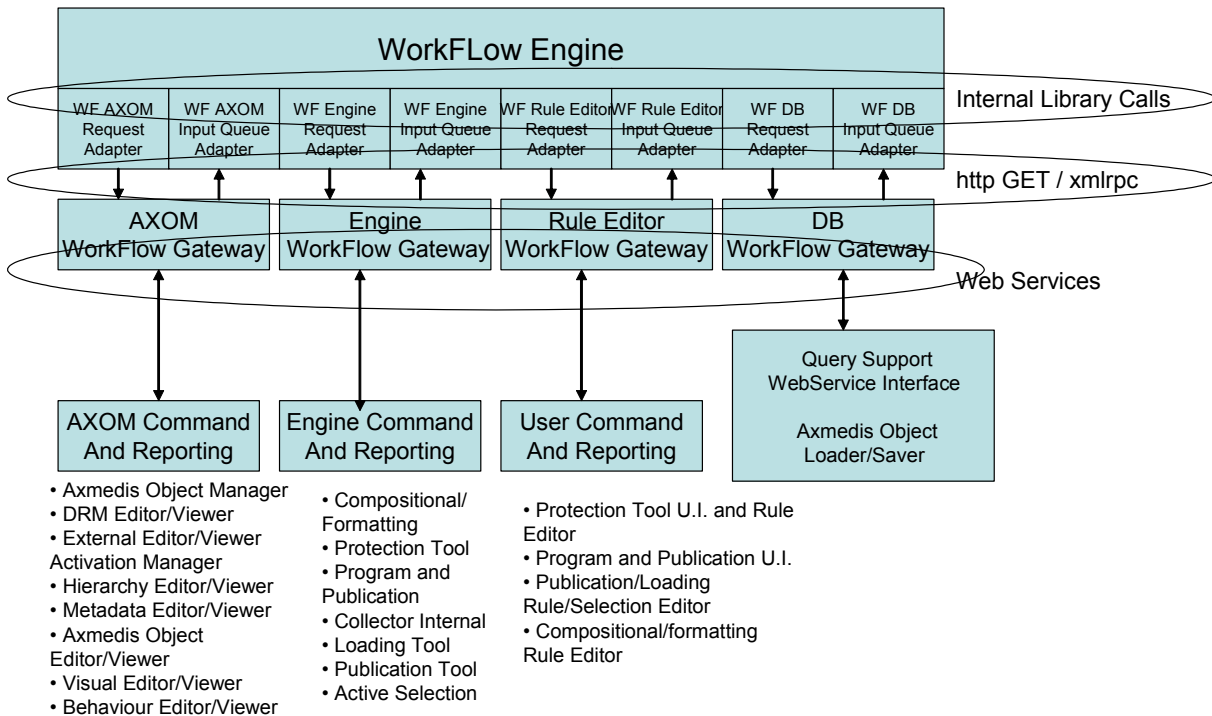
Here we outline the key patterns of deployment in scenarios for the integration of any AXMEDIS-adopted workflow management system (AXWF) with the AXMEDIS Framework; as follows:

- The AXWF is used for streamlining the publishing and distribution process activities that can be supported by the AXMEDIS Framework
- The AXWF should be capable of launching the execution of:
 - AXMEDIS Object Editor/Viewer via the AXOM WorkFlow Gateway
 - AXMEDIS Object Manager via the AXOM WorkFlow Gateway
 - AXMEDIS DRM Editor/Viewer via the AXOM WorkFlow Gateway
 - AXMEDIS External Editor/Viewer Activation Manager via the AXOM WorkFlow Gateway
 - AXMEDIS Hierarchy Editor/Viewer via the AXOM WorkFlow Gateway
 - AXMEDIS Metadata Editor/Viewer via the AXOM WorkFlow Gateway
 - AXMEDIS Visual Editor/Viewer via the AXOM WorkFlow Gateway
 - AXMEDIS Behaviour Editor/Viewer via the AXOM WorkFlow Gateway
 - AXMEDIS Compositional/Formatting Engine via the Engine WorkFlow Gateway
 - AXMEDIS Collector Internal Engine via the Engine WorkFlow Gateway
 - AXMEDIS Program and Publication Engine via the Engine WorkFlow Gateway
 - AXEPTool Loading Tool Engine via the Engine WorkFlow Gateway
 - AXEPTool Publication Tool Engine via the Engine WorkFlow Gateway
 - The Protection Tool Engine via the AXMEDIS Engine WorkFlow Gateway
 - AXMEDIS Loader/Saver via the AXMEDIS DB WorkFlow Gateway
 - AXMEDIS Query Support via the DB WorkFlow Gateway
 - AXMEDIS Compositional/Formatting Rule Editor via the Rule Editor WorkFlow Gateway
 - AXMEDIS Program and Publication User Interface via the Rule Editor WorkFlow Gateway

- AXEPTool Publication/Loading Rule/Selection Editor via the Rule Editor WorkFlow Gateway
- AXMEDIS Protection User Interface and Rule Editor via the Rule Editor WorkFlow Gateway
- Accordingly the AXWFM Workflow Manager will be capable of interacting with the AXMEDIS Object Manager via the AXOM WorkFlow Gateway
- It is possible that some publishing and distribution companies will not be able to adopt the fully integrated AXMEDIS framework, as:
 - They may wish to keep to their established client applications thus using editing/viewing tools other than the AXMEDIS-native ones.
 - There may be another group of companies who will actually be using the same editing/viewing tools as those for which AXMEDIS Plug-ins are provided; however for technical reasons they may still be unable to use the respective AXMEDIS plug-ins, due to difficulties of distributing plug-ins, incompatibilities of AXMEDIS plug-ins with other PC tools, security reasons, heterogeneity of Client configurations, etc)
- For the above type of companies given that their staff will routinely need to have access to AXMEDIS Objects within their own exclusive application areas to work on such object in the normal course of processing them for publication/distribution, the provision of a simple *AXMEDIS Object check-in/check-out interface* should provide a means of access to the AXMEDIS Object Manager's services. These check-in/check-out functions will be operated via the integration with the AXMEDIS Object Loader/Saver.
- Within such partial integration of the AXMEDIS Framework as operated by the above companies, the users shall have to take responsibility for manually fetching i.e. downloading the respective AXMEDIS Objects onto their own PC hard disk; manually invoking the appropriate Editors/Viewers as well as informing the workflow when a particular activity is terminated and of the work carried out on the respective object. This is to ensure Object Metadata update and that workflow processes progress consistently and coherently.
- It is expected that every AXMEDIS tool updates the AXMEDIS Object tracking information, while performing its actions. However, when the check-in/check-out interface is used, it is up to the Workflow Object Manager to update such information. Unfortunately the Workflow cannot know exactly what action the user performed between the Check-in and Check-out stage. Accordingly we can provide a field to be filled by users to describe the work that was done. This is to support the users in ensuring that they will remember to give full information about such the work carried out on the respective objects.

The following schematic depicts the Integration Architecture of WorkFlow OpenFlow Engine with the other AXMEDIS components:

Axmedis WorkFlow Integration Architecture (OpenFlow)



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_Listener
- AXOM Command and Reporting

The Response Chain:

- AXOM Command and Reporting
- AXOM_WebService_Listener
- AXOM WorkFlow gateway
- AXOM Input Queue Adapter

2. The WorkFlow Engine Channel encompassing:

The Request Chain:

- WF Engine Request Adapter

- Engine WorkFlow Gateway
- Engine Command and Reporting

The Response Chain:

- Engine Command and Reporting
- 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
- User Command and Reporting

The Response Chain:

- User Command and Reporting
- 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 Gateway
- AXMEDIS Object Loader/Saver and Query Support WebService Interface

The Response Chain:

- AXMEDIS Object Loader/Saver and Query Support WebService Interface
- DB WorkFlow Gateway
- 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 communication methods and protocols.

The Gateways communicates towards WF Input Queue Adapters via WorkFlow available communication 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 WF Plug-ins and WF Query and DataBase Interface. These further modules will have the same interface with the Plug-in Manager and Query Support WebService Interface / AXMEDIS Object Loader/Saver, while their interface with the WorkFlow Adapters will be different.

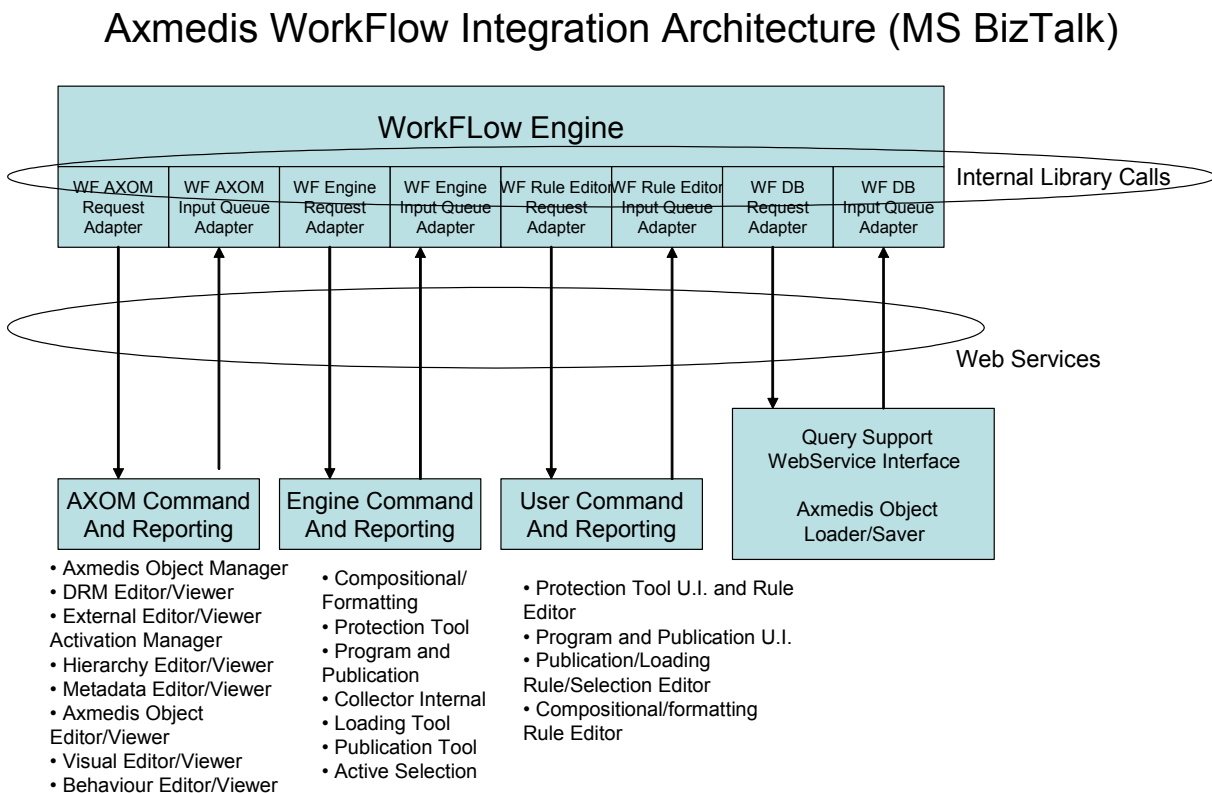
The WorkFlow Engine communicates with the WF Adapters via library calls internal to the Zope Application Server. This interface will be replaced by the Microsoft BizTalk WebServices adapter internal interface when using BizTalk as the adopted AXMEDIS workflow system.

The WF Request Adapters communicates with Gateways via GET calls in http protocol.

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:



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

3.9 WorkFlow Technologies

Two distinct workflow technologies are to be deployed for the AXMEDIS Integration Demonstrators for Phase I and Phase II of the project, respectively as follows:

1. an open-source WFMS, namely OpenFlow
2. a proprietary commercial WFMS, namely Microsoft BizTalk

The rationale for the choice of the above as the adopted AXMEDIS WFMSs (i.e. AXWFs), appears in sub-sections 4.1.1 and 4.2.1 respectively.

The present document is to specify the foundational design architecture applicable generically for the integration of the AXMEDIS Framework with most of the WFMSs likely to find a significant presence in the target user sectors; including both Openflow and BizTalk. Further, the present document is also to elaborate the full design specification for the Phase I AXMEDIS Demonstrator implementation of an AXWF-AXMEDIS integration which is to be with Openflow. This will thus allow the full elaboration of the design specification for the second AXMEDIS Demonstrator, with BizTalk, as planned for the Phase II of the projects, to benefit from the planned continuous requirements refinement and validation to be conducted in Phase I of the project.

3.10 WorkFlow Engine

Sub-sections 4.1.2 through to 4.1.4 provide the description of the Phase I AXWF which is the Openflow.

Module Profile		
WorkFlow Engine		
Executable or Library(Support)	Executable OpenFlow Package, based upon Zope Application Server	
Single Thread or Multi-thread	Multi-thread	
Language of Development	User interface: Zope DTML (a superset of html) Application logic: Python or DTML	
Responsible Name		
Responsible Partner		
Status (proposed/approved)		
Platforms supported	Microsoft Windows, Linux	
Interfaces with other tools	Name of the communicating tools	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
File Formats Used	Shared with	File format name or reference to a section

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	Licence 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

3.11 Workflow User Interfaces and Tools (IRC, HP, XIM)

Module Profile		
WorkFlow User Interface		
Executable or Library(Support)	Executable OpenFlow User Interface based upon Zope User Interface	
Single Thread or Multithread	Multithread	
Language of Development	User interface: Zope DTML (a superset of html) Application logic: Python or DTML	
Responsible Name		
Responsible Partner		
Status (proposed/approved)		
Platforms supported	Microsoft Windows, Linux	
Interfaces with other tools:	Name of the communicating tools	Communication model and format (protected or not, etc.)
WorkFlow Engine		Via Zope Web Server
File Formats Used	Shared with	File format name or reference to a section
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	Licence 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

3.11.1 Typical Transactions between AXWF and AXMEDIS in Delivering the UseCases

All the possible Workflow engines in the Open Source arena that are suitable for AXMEDIS offer a Web application based User Interface.

The User Interface is then executed via a normal browser, while the User Interface logics reside on the Application Server of the WorkFlow Engine.

The WorkFlow Manager will be a customisation of one of the suitable Open-Source Workflow Management Systems. Some functions will be added, if not already present (e.g. searching engine).

The Workflow User Interface is included in the customisation to serve the integration of the selected WorkFlow product with the AXMEDIS Framework. The Use Interface will use the WorkFlow Manager class methods via the selected Workflow product internal interface.

The Workflow Manager supports the User Interface with the following class methods, once the user has been identified as required (see requirements document and earlier sections relating to session AAA):

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

W) Management functions for:

- Starting/stopping Workflows
- Viewing WorkFlow exceptions log
- Viewing Users' Work-lists
- Viewing/Modifying/Deleting Process Instances
- Viewing/Modifying/Adding/Deleting Process definitions
- Viewing Process histories
- Viewing/Adding/Modifying/Deleting Users and permissions
- Viewing Adding/Modifying/Deleting Roles

3.11.2 WorkFlow User Interface Mock-ups (OpenFlow)

Description for the User Interface is reported in Part B Section 2.8 Workflow Editor and Viewer.

4 WorkFlow Request Adapters (OpenFlow)

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

1. WF AXOM Request Adapter 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.

Module Profile	
WorkFlow Request Adapters	
Executable or Library(Support)	Library
Single Thread or Multithread	Multithread

Language of Development	Python or DTML	
Responsible Name		
Responsible Partner		
Status (proposed/approved)		
Platforms supported	Microsoft Windows, Linux	
Interfaces with other tools:	Name of the communicating tools	Communication model and format (protected or not, etc.)
WorkFlow Engine		Internal library calls
Plug-ins		GET calls in http
WF Query and DB Interface		GET calls in http
File Formats Used	Shared with	File format name or reference to a section
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
Used Libraries	Name of the library and version	Licence 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

4.1 WorkFlow Input Queue Adapters (OpenFlow)

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

5. WF AXOM Input Queue Adapter for integrating the AXMEDIS Object Manager and all AXOM Editors (WorkFlow Editor Channel)
6. WF Rule Editor Input Queue Adapter for integrating all AXMEDIS Rule Editors (WorkFlow Rule Editor Channel)
7. WF Engine Input Queue Adapter for integrating all AXMEDIS Engines (WorkFlow Engine Channel)
8. 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.

Module Profile		
WorkFlow Input Queue Adapters		
Executable or Library(Support)	Library	
Single Thread or Multithread	Multithread	
Language of Development	Python or DTML	
Responsible Name		
Responsible Partner		
Status (proposed/approved)		
Platforms supported	Microsoft Windows, Linux	
Interfaces with other tools:	Name of the communicating tools	Communication model and format (protected or not, etc.)
WorkFlow Engine		Internal librariy calls
Plug-ins		Xmlrpc calls in http
WF Query and DB Interface		Xmlrpc calls in http
File Formats Used	Shared with	File format name or reference to a section
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
Used Libraries	Name of the library and version	Licence 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

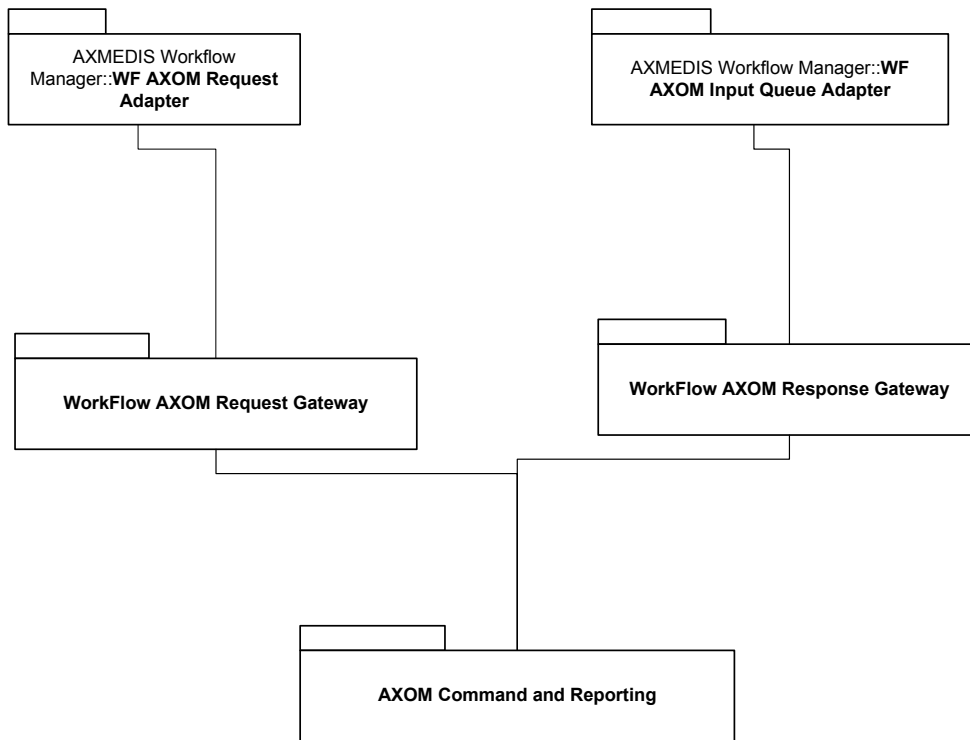
4.2 WorkFlow Gateways (OpenFlow)

There are four Gateways:

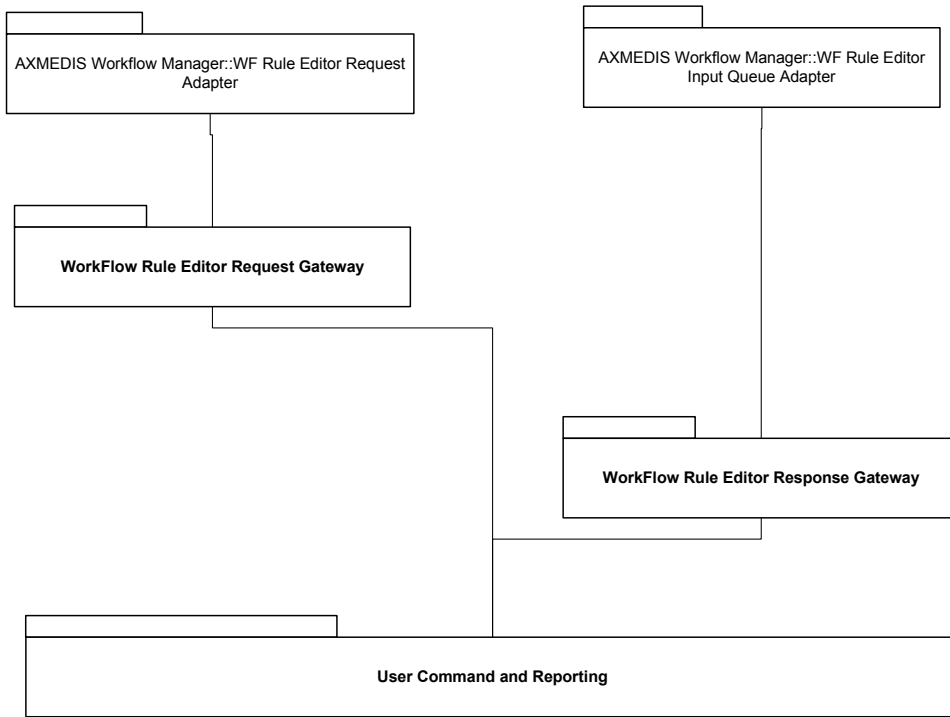
- AXOM WorkFlow Gateway (for the WorkFlow Editor Channel)
- Rule Editor WorkFlow Gateway (for the WorkFlow Rule Editor Channel)
- Engine WorkFlow Gateway (for the WorkFlow Engine Channel)

- DB WorkFlow Gateay (for Query and DataBase Channel)

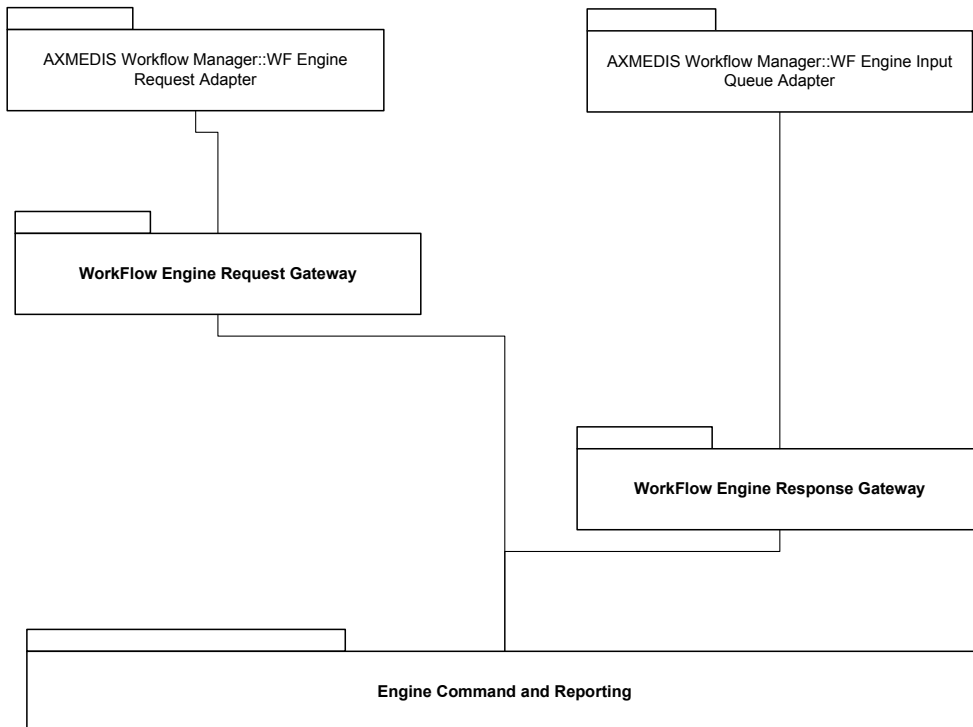
The following UML schema describes the AXOM WorkFlow Gateway:



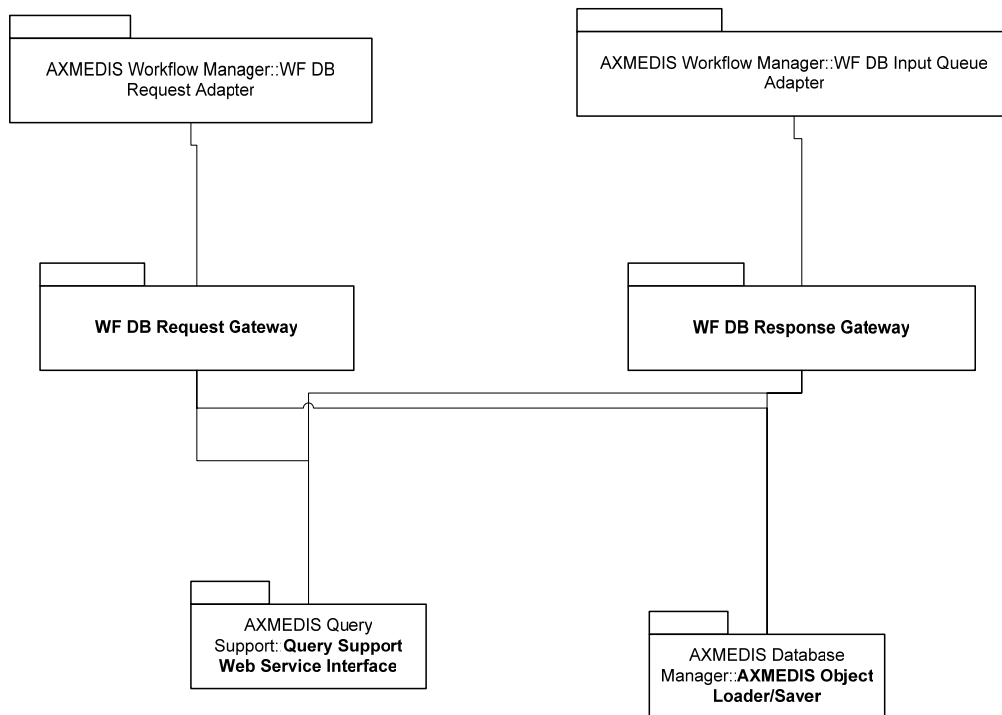
The following UML schema describes the Rule Editor WorkFlow Gateway:



The following UML schema describes the Engine WorkFlow Gateway:

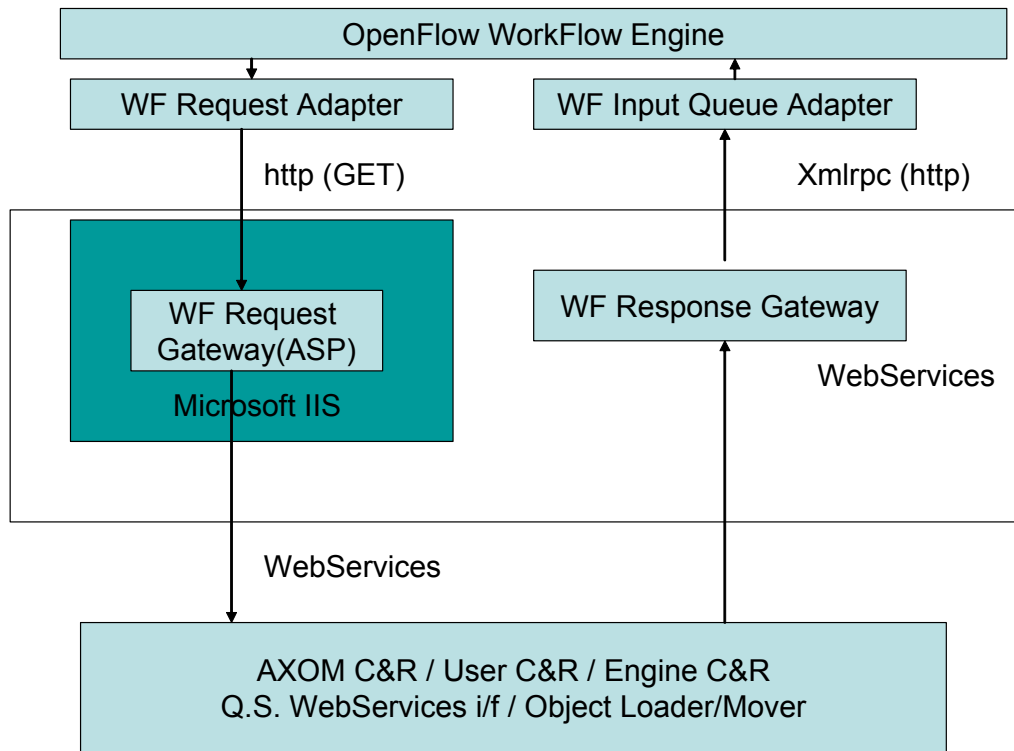


The following UML schema describes the DB WorkFlow Gateway:



All the four Gateways share the same architecture as outlined below:

OpenFlow WorkFlow Plug-in Architecture



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 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 module.

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.

Module Profile		
WF Request Gateway		
Executable or Library(Support)	ASP Process	
Single Thread or Multithread	Multithread	
Language of Development	C++	
Responsible Name		
Responsible Partner		
Status (proposed/approved)		
Platforms supported	Microsoft Windows	
Interfaces with other tools:	Name of the communicating tools	Communication model and format (protected or not, etc.)
WF Request Adapters		GET calls in http

AXMEDIS modules		WebServices
File Formats Used	Shared with	File format name or reference to a section
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
None		
Used Libraries	Name of the library and version	Licence status: GPL. LGPL. PEK, proprietary, authorized or not
Microsoft ASP	TBD	Microsoft .license terms

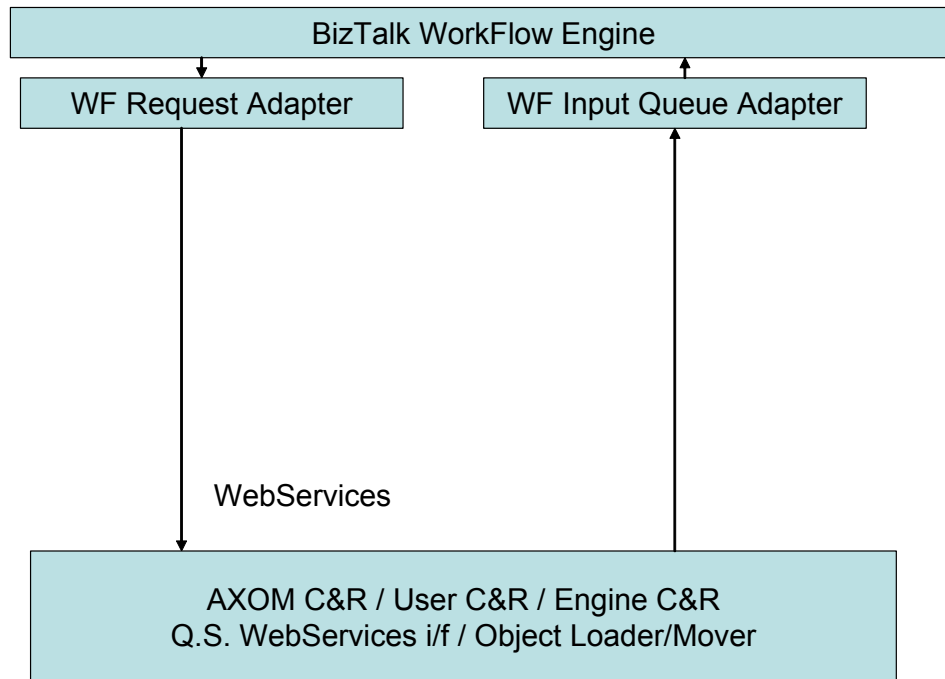
Module Profile		
WF Response Gateway		
Executable or Library(Support)	Executable process	
Single Thread or Multithread	Multithread	
Language of Development	C++	
Responsible Name		
Responsible Partner		
Status (proposed/approved)		
Platforms supported	Microsoft Windows	
Interfaces with other tools:	Name of the communicating tools	Communication model and format (protected or not, etc.)
WF Input Queue Adapters		Xmlrpc calls
AXMEDIS modules		WebServices
File Formats Used	Shared with	File format name or reference to a section
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
None		
Used Libraries	Name of the library and version	Licence status: GPL. LGPL. PEK, proprietary, authorized or not
Microsoft .net	TBD	Microsoft .license terms
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 License Version 1.1

--	--	--

4.3 WorkFlow Gateways (Microsoft BizTalk)

When using Microsoft Biztalk, the architecture of the Gateways will be as shown below:

MS BizTalk WorkFlow Plug-in Architecture



The Gateways will be really void, as they are no more needed, because MS BizTalk is able to natively communicate through WebServices.

5 AXMEDIS Workflow Interface Specifications

5.1 The AXMEDIS Editor WorkFlow Channel

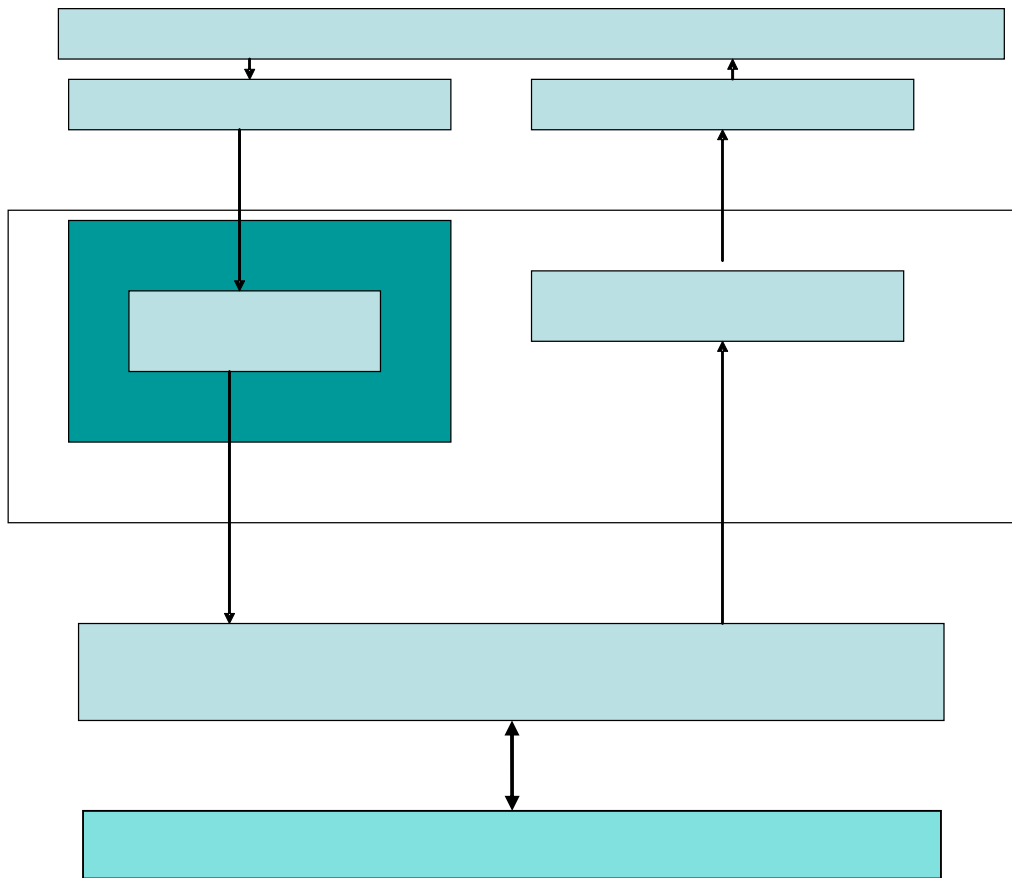
In the following Paragraph we will specify the interface between the WorkFlow and the AXMEDIS Object Manager and the various AXMEDIS Editor/Viewer:

- AXMEDIS Object Editor/Viewer
- AXMEDIS DRM Editor/Viewer
- AXMEDIS External Editor/Viewer

- AXMEDIS Hierarchy Editor/Viewer
- AXMEDIS Metadata Editor/Viewer
- AXMEDIS Visual Editor/Viewer
- AXMEDIS Behaviour Editor/Viewer

5.1.1 The Interface between the WF Editor Request Gateway and the AXOM Command and Reporting

The AXMEDIS Workflow Manager will communicate to AXOM's Command and Reporting through WF Editor 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 module, AXMEDIS Editor in this case. As AXOM, along with Command and Reporting, is a static library, a listener service is required to listen to incoming Request from Workflow and invoke AXMEDIS Editor accordingly. We call this listener service as AXOM_WebServices_Listener which will be resident on client's machine. AXOM WebServices Listener is a multithreading process written in C++ which exposes methods through WebServices, listens to them and forwards the requests to the AXOM Command and Reporting module which is a C++ library. So the interface between AXOM WebServices Listener and AXOM Command and Reporting are C++ library calls.



As the AXOM and Commands and Reporting are a set of libraries, we define a new function `Workflow_Applic_Launcher` within the `AXOM_WebService_Listener` Module to launch the Editor. Upon receipt of a request to launch the AXMEDIS Editor, the `Workflow_Applic_Launcher` will launch the editor using a system call. The completion of the activity will be given directly by the Editor to the workflow manager. It is probable that the `Workflow_Applic_Launcher` will be realised as independent process capable of receiving commands from the Workflow manager and launching the specific Applications if they are not currently under execution on that computer.

As described before, the AXMEDIS Editor Workflow channel passes through the WF Editor Request Gateway where the `AXOM_WebService_Listener` will expose the following methods, via WebServices:

- `Edit_Object`, for launching the AXMEDIS Object Editor and all its Plug-ins used for editing and viewing AXMEDIS Objects, Object Behaviours, DRMs, Hierarchies and Metadata
- `Add_Object` to request the AXMEDIS Object Manager to have a new object created
- `Compose_Object` for creating a subobject inside an AXMEDIS Object
- `Delete_Object` to request the AXMEDIS Object Manager to have a specific object deleted
- `Modify_Object` to request the AXMEDIS Object Manager to have certain object attributes modified
- `View_Object_Attribute` to request the AXMEDIS Object Manager to allow the viewing of the object attributes

- Add_History_Info to request the AXMEDIS Object Manager to have a history information added to an object AXINFO
- View_History_Info to request the AXMEDIS Object Manager to retrieve history information of a given object AXINFO

The method invocation is performed using a WebService request where the following parameters are sent (to AXOM Command and Reporting) and received back (in WebService result) from the AXOM Command and Reporting via AXOM_Web_Service_Listener:

- Edit_Object
 - INPUT: AXOID, User_Credentials, AXRQID, Editing_type, Execution_Parameters, EditorListenerService
 - OUTPUT: Operation_Result (OK, EXCEPTION)
 - Editing_type specifies the type of Editor function or Plug-in to invoke (E.g. Metadata, DRM, Behaviour, etc)
 - Execution_parameters is a string containing the execution string for the Editor
 - EditorListernerService is the Uri used by the WF Response Gateway for receiving the following Notification about the termination of the Editor
- Add_Object
 - INPUT: List of attributes_value (attributeID, attributevalue), User_Credentials, AXRQID
 - OUTPUT: Operation_Result (AXOID, EXCEPTION)
 - List of attributes_value is a list of duplets attribute_ID:attribute_value specifying the initial attribute values for the Object: see AXOM for list of possible attributes
 - AXOID is the ID of the newly created Axmedi Object
- Compose_Object
 - INPUT: AXOID, User_Credentials, AXRQID, Component_specifications
 - OUTPUT: Operation_Result (OK, EXCEPTION)
 - Component_specifications contains XML specifications for the new created component
- Delete_Object
 - INPUT: AXOID, User_Credentials, AXRQID
 - OUTPUT: Operation_Result (OK, EXCEPTION)
- Modify_Object
 - INPUT: AXOID, User_Credentials, list of attributes_value(attributeID:attributevalue),, AXRQID
 - OUTPUT: Operation_Result (OK, EXCEPTION)
 - List of attributes_value is a list of duplets attribute_ID:attribute_value specifying the attributes to modify and their new value: see AXOM for list of possible attributes
- View_Object_Attribute
 - INPUT: AXOID, User_Credentials, list of attributes(attributeID)
 - OUTPUT: Operation_Result (list of attributes_value(attributeID:attributevalue), EXCEPTION)
 - List_of_attribute is a list of attribute_ID, specifying the list of attribute values to be retrieved.
 - List of attributes_value is a list of duplets attribute_ID:attribute_value specifying the attributes with their current value.
- Add_History_Info
 - INPUT: AXOID, User_Credentials, log_info, AXRQID

- OUTPUT: Operation_Result (OK, EXCEPTION)
Log_info: contains log information to add: see AXINFO for specification.
- View_History_Info
 - INPUT: AXOID, User_Credentials
 - OUTPUT: Operation_Result (List of log_info, EXCEPTION)
Listo_of_log_info is a list of log_info: see AXINFO for specification.

User_Credentials contain information for:

- User Identification
- User Session Identification

AXOID is the identification of the AXMEDIS Object to be viewed/edited, etc

AXRQID uniquely identifies the update/create/delete request made to the AXOM Command and Reporting.
This is needed in order to:

- Identify the subsequent Notification from the AXOM Command and Reporting to the WF Response Gateway via AXOM_WebService_Listener
- Not to have to duplicate the same request as previously put to the AXOM Command and Reporting Module. We assume that, if the AXOM Command and Reporting receives a request with the same AXRQID as a previous one it will send back the same Notification

The AXRQID “encapsulates” the WorkFlow Manager generated parameters (openflow_ID, process_ID, activity_ID, instance_ID, workitem_ID).

The WSDL specification for the edit_Object method invocation is:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="Editor"
targetNamespace="http://www.AXMEDIS.org/editor.wsdl"
xmlns:tns="http://www.AXMEDIS.org/editor.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="unqualified"
attributeFormDefault="unqualified">
<import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
<complexType name="attribute">
<sequence>
    <element name="attributeid" type="xsd:string" minOccurs="1" maxOccurs="1" />
    <element name="attributevalue" type="xsd:string" minOccurs="1"
maxOccurs="1" />
</sequence>
</complexType>
<complexType name="attributelist">
<sequence>
    <element name="attribute" type="attribute" minOccurs="1" maxOccurs="20" />
</sequence>
</complexType>
<complexType name="attribute_request_list">
<sequence>
    <element name="attributeid" type="xsd:string" minOccurs="1" maxOccurs="20" />
</sequence>
</complexType>
<complexType name="HistoryInfo">
<sequence>
    <element name="HistoryLog" type="xsd:string" minOccurs="1"
maxOccurs="100" />
</sequence>
</complexType>
<complexType name="edit_Object-result">
<sequence>
    <element name="result" type="xsd:boolean" minOccurs="1" maxOccurs="1" />
    <element name="errmsg" type="xsd:string" minOccurs="0" maxOccurs="1" nillable="true" />
    <element name="errorcode" type="xsd:int" minOccurs="1" maxOccurs="1" />
</sequence>
</complexType>
<!-- operation request element -->
<element name="AXOID" type="xsd:string" />
<!-- operation request element -->
<element name="User_Credeintials" type="xsd:string" />
<!-- operation request element -->
<element name="execution_Parameters" type="xsd:string" />
<!-- operation request element -->
<element name="editing_type" type="xsd:string" />
<!-- operation response element -->
<element name="return" type="ax:edit_Object-result" />
<!-- operation request element -->

```

```

<element name="EditorListenerService" type="xsd:anyURI"/>
<!-- operation request element -->
<element name="AXRQID" type="xsd:string"/>
<!-- operation response element -->
<element name="result" type="xsd:boolean"/>
<element name="attributelist" type="attributelist"/>
<element name="attribute_request_list" type="attribute_request_list" />
<element name="HistoryInfo" type="HistoryInfo"/>
</schema>

</types>

<message name="edit_ObjectRequest">
<part name="AXOID" element="ax:AXOID"/>
<part name="User_Credentials" element="ax:User_Credentials"/>
<part name="execution_Parameters" element="ax:execution_Parameters"/>
<part name="editing_type" element="ax:editing_type"/>
<part name="AXRQID" element="ax:AXRQID"/>
<part name="EditorListenerService" element="ax:EditorListenerService"/>
</message>

<message name="geteditObjectResult">
<part name="return" element="ax:return"/>
</message>

<message name="Add_ObjectRequest">
<part name="User_Credentials" element="ax:User_Credentials"/>
<part name="AXRQID" element="ax:AXRQID"/>
<part name="attributeList" element="ax:attributeList" />
</message>

<message name="getAddObjectResult">
<part name="return" element="ax:return"/>
<part name="AXOID" element="ax:AXOID"/>
</message>

<message name="Compose_ObjectRequest">
<part name="AXOID" element="ax:AXOID"/>
<part name="User_Credentials" element="ax:User_Credentials"/>
<part name="AXRQID" element="ax:AXRQID"/>
<part name="attributeList" element="ax:attributeList" />
</message>

<message name="getComposeObjectResult">
<part name="return" element="ax:return"/>
</message>

<message name="Delete_ObjectRequest">
<part name="AXOID" element="ax:AXOID"/>

```

```

<part name="User_Credentials" element="ax:User_Credentials"/>
<part name="AXRQID" element="ax:AXRQID"/>
</message>

<message name="getDeleteObjectResult">
<part name="return" element="ax:return"/>
</message>

<message name="Modify_ObjectRequest">
<part name="AXOID" element="ax:AXOID"/>
<part name="User_Credentials" element="ax:User_Credentials"/>
<part name="AXRQID" element="ax:AXRQID"/>
<part name="attributeList" element="ax:attributeList" />
</message>

<message name="getModifyObjectResult">
<part name="return" element="ax:return"/>
</message>

<message name="View_Object_AttributeRequest">
<part name="AXOID" element="ax:AXOID"/>
<part name="User_Credentials" element="ax:User_Credentials"/>
<part name="AXRQID" element="ax:AXRQID"/>
<part name="attribute_Request_List" element="ax:attribute_Request_List" />
</message>

<message name="getViewObjectAttributeResult">
<part name="return" element="ax:return"/>
<part name="attributeList" element="ax:attributeList" />
</message>

<message name="Add_History_InfoRequest">
<part name="AXOID" element="ax:AXOID"/>
<part name="User_Credentials" element="ax:User_Credentials"/>
<part name="AXRQID" element="ax:AXRQID"/>
<part name="HistoryInfo" element="ax:HistoryInfo" />
</message>

<message name="getAddHistoryInfoResult">
<part name="return" element="ax:return"/>
</message>

<message name="View_History_InfoRequest">
<part name="AXOID" element="ax:AXOID"/>
<part name="User_Credentials" element="ax:User_Credentials"/>
<part name="AXRQID" element="ax:AXRQID"/>
</message>

<message name="getViewHistoryInfoResult">

```

```

<part name="return" element="ax:return"/>
<part name="HistoryInfo" element="ax:HistoryInfo" />
</message>

<portType name="EditorPortType">
  <operation name="edit_Object">
    <documentation>Service definition of function ax:edit_Object</documentation>
    <input message="tns:edit_ObjectRequest"/>
    <output message="tns:getEditObjectResult"/>
  </operation>
  <operation name="Add_Object">
    <documentation>Service definition of function ax:Add_Object</documentation>
    <input message="tns:Add_ObjectRequest"/>
    <output message="tns:getAddObjectResult"/>
  </operation>
  <operation name="Compose_Object">
    <documentation>Service definition of function ax:Compose_Object</documentation>
    <input message="tns:Compose_ObjectRequest"/>
    <output message="tns:getComposeObjectResult"/>
  </operation>
  <operation name="Delete_Object">
    <documentation>Service definition of function ax>Delete_Object</documentation>
    <input message="tns>Delete_ObjectRequest"/>
    <output message="tns:getDeleteObjectResult"/>
  </operation>
  <operation name="Modify_Object">
    <documentation>Service definition of function ax:Modify_Object</documentation>
    <input message="tns:Modify_ObjectRequest"/>
    <output message="tns:getModifyObjectResult"/>
  </operation>
  <operation name="View_Object_Attribute">
    <documentation>Service definition of function ax:View_Object_attribute</documentation>
    <input message="tns:View_Object_attributeRequest"/>
    <output message="tns:getviewobjectattributeResult"/>
  </operation>
  <operation name="Add_History_Info">
    <documentation>Service definition of function ax:Add_History_Info</documentation>
    <input message="tns:Add_History_InfoRequest"/>
    <output message="tns:getAddHistoryInfoResult"/>
  </operation>
  <operation name="View_History_Info">
    <documentation>Service definition of function ax:View_History_Info</documentation>
    <input message="tns:View_History_InfoRequest"/>
    <output message="tns:getViewHistoryInfoResult"/>
  </operation>
</portType>

```

```

<binding name="Editor" type="tns:EditorPortType">
<SOAP:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
<operation name="edit_Object">
<SOAP:operation style="rpc" soapAction=""/>
<input>
<SOAP:body use="literal" namespace="urn:ax"/>
</input>
<output>
<SOAP:body use="literal" namespace="urn:ax"/>
</output>
</operation>
<operation name="Add_Object">
<SOAP:operation style="rpc" soapAction=""/>
<input>
<SOAP:body use="literal" namespace="urn:ax"/>
</input>
<output>
<SOAP:body use="literal" namespace="urn:ax"/>
</output>
</operation>
<operation name="Compose_Object">
<SOAP:operation style="rpc" soapAction=""/>
<input>
<SOAP:body use="literal" namespace="urn:ax"/>
</input>
<output>
<SOAP:body use="literal" namespace="urn:ax"/>
</output>
</operation>
<operation name="Delete_Object">
<SOAP:operation style="rpc" soapAction=""/>
<input>
<SOAP:body use="literal" namespace="urn:ax"/>
</input>
<output>
<SOAP:body use="literal" namespace="urn:ax"/>
</output>
</operation>
<operation name="Modify_Object">
<SOAP:operation style="rpc" soapAction=""/>
<input>
<SOAP:body use="literal" namespace="urn:ax"/>
</input>
<output>
<SOAP:body use="literal" namespace="urn:ax"/>
</output>
</operation>
<operation name="View_Object_Attribute">
<SOAP:operation style="rpc" soapAction=""/>

```

```

<input>
  <SOAP:body use="literal" namespace="urn:ax"/>
</input>
<output>
  <SOAP:body use="literal" namespace="urn:ax"/>
</output>
</operation>
<operation name="Add_History_Info">
  <SOAP:operation style="rpc" soapAction=""/>
  <input>
    <SOAP:body use="literal" namespace="urn:ax"/>
  </input>
  <output>
    <SOAP:body use="literal" namespace="urn:ax"/>
  </output>
</operation>
<operation name="View_History_Info">
  <SOAP:operation style="rpc" soapAction=""/>
  <input>
    <SOAP:body use="literal" namespace="urn:ax"/>
  </input>
  <output>
    <SOAP:body use="literal" namespace="urn:ax"/>
  </output>
</operation>

</binding>

<service name="Editor">
  <documentation>gSOAP 2.7.0e generated service definition</documentation>
  <port name="Editor" binding="tns:Editor">
    <SOAP:address location="http://www.AXMEDIS.org/Editor.cgi"/>
  </port>
</service>

</definitions>

```

Operation_Result takes on different meaning/contents. Depending on the context of the invoked method, the Operation_Result, if the result is satisfactory, it is to contain and/or confirm the following:

1. For View_Object_attribute it contains the retrieved attribute values
2. For View_History_Info it contains the retrieved tracking information
3. For Edit_Object it means that the Editor was successfully launched

4. For Add/Delete/Modify_Object and Add_History_Info it means that the modification is correctly completed

In cases 3 above, a NOTIFICATION is needed in order to inform the WorkFlow engine of the completion status of the committed operation.

The Notification is an asynchronous notification containing the original AXRQID and the Result. The Notification is sent back through a WebService towards the WF Response Gateway, at the URI specified in the EditorListenerService parameter.

5.1.2 Interface between the WF AXOM Request Adapter and the WF Editor Request Gateway

The methods invoked and the parameters sent by WF AXOM Request Adapter to the WF Editor Request Gateway are the same described in the preceeding Paragraph. There encoding, however, is different. The request in fact sent through an http GET call where the parameters are invoked as follows:

```
GET/Control_Panel/Products/OpenFlow/AXWF/editor_name_request_editor?AXOID="Object ID
string"&Credentials="Credential string"&AXRQID="Request ID
string"&execution_parameters="execution parameter
string"&attribute_values="attribute_name_1:attribute_value1,attribute_name2:attribute_value2;,etc
"&log_info="log_info string" HTTP/1.1" 200 368
"http://15.156.120.195:8080/Control_Panel/Products/OpenFlow/leave/leave_startform"
"Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR 1.0.3705; .NET CLR 1.1.4322)"
```

Where **editor_name** is to be replaced by appropriate editor identifier “Object ID string” is a string containing the AXOID, “Credential string” is a string containing the credentials and “Request ID string” is a string containing the Request ID.

The response to the invoked method has the same contents listed in the preceeding Paragraph and 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="errmsg" 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>
```



```

    </xs:complexType>
</xs:element>

```

5.1.3 The Interface between the WF AXOM Input Queue Adapter and the WF Editor Response Gateway

This interface is used for returning back Notification to WorkFlow Engine, related to previously issued requests.

The Notification is used, as mentioned, to return to the WorkFlow the results of the requested operation in a XMLRPC call. Specifically, it will contain:

- Edit_Object
 - NOTIFICATION: AXRQID, Completion_Result (OK, EXCEPTION)

Where AXRQID is the AXRQID in the original request from the WorkFlow and Completion_result can be either positive (OK or returned parameters) or negative (EXCEPTION is an error code returned for diagnostic purposes and useful for troubleshooting).

The NOTIFICATION is sent via an XMLRPC call whose method name is “EditorNotification” and whose parameter section is XML encoded as specified by the following schema:

```

<xs:element name="Editor_Notification">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="result" type="xs:boolean"/>
      <xs:element name="errmsg" type="xs:string" nillable="true" minOccurs="0"/>
      <xs:element name="errorcode" type="xs:int"/>
      <xs:element name="AXRQID" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

On receiving the Notification, the WF Response Adapter will give the following XML response, 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>

```

5.1.4 The Interface between the AXOM Command and Reporting and the WF Editor Response Gateway

As described before, the AXOM Command and Reporting will send its notifications to the WorkFlow Engine via AXOM_WebService_Listener which will call a WebServices exposed by the WF Editor Response Gateway .

The URI of the WebService is indicated in the previous request made by the WF Request Gateway in the EditorListenerService parameter.

The Notification shall also contain the original Request ID issued in the request (AXRQID).

The following WSDL defines the Editor Notification WebService:

```

<?xml version="1.0" encoding="UTF-8"?>
<WSDL:definitions xmlns:tns="http://www.AXMEDIS.org/editor.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/"
targetNamespace="http://www.AXMEDIS.org/editor.wsdl" name="Editor">
  <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="unqualified"
attributeFormDefault="unqualified">
      <import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
      <complexType name="Editor-result">
        <sequence>

```

```

maxOccurs="1"/>
<element name="result" type="xsd:boolean" minOccurs="1"
maxOccurs="1" nillable="true"/>
<element name="errmsg" type="xsd:string" minOccurs="0"
maxOccurs="1"/>
<element name="errorcode" type="xsd:int" minOccurs="1"
maxOccurs="1"/>
</sequence>
</complexType>
<!-- operation response element -->
<element name="return" type="ax:Editor-result"/>
<!-- operation request element -->
<element name="AXRQID" type="xsd:string"/>
<!-- operation response element -->
<element name="result" type="xsd:boolean"/>
</schema>
</WSDL:types>
<message name="Editor_Notification">
  <part name="AXRQID" element="ax:AXRQID"/>
  <part name="return" element="ax:return"/>
</message>
<message name="getNotificationResult">
  <part name="return" element="ax:return"/>
</message>
<portType name="EditorNotificationPortType">
  <operation name="Editor_Notification">
    <documentation>Service definition of function
ax:Editor_Notification</documentation>
    <input message="tns:Editor_Notification"/>
    <output message="tns:getNotificationResult"/>
  </operation>
</portType>
<binding name="EditorNotification" type="tns:EditorNotificationPortType">
  <SOAP:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="Editor_Notification">
    <SOAP:operation style="rpc">
      <input>
        <SOAP:body use="literal" namespace="urn:ax"/>
      </input>
      <output>
        <SOAP:body use="literal" namespace="urn:ax"/>
      </output>
    </SOAP:operation>
  </operation>
</binding>
<service name="EditorNotification">
  <documentation>gSOAP 2.7.0e generated service definition</documentation>
  <port name="EditorNotification" binding="tns:EditorNotification">
    <SOAP:address
location="http://www.AXMEDIS.org/EditorNotification.cgi"/>
  </port>

```

```

    </service>
  </WSDL:definitions>

```

5.2 The WorkFlow Engine Channel

In the following Paragraph we shall specify the interface between the WorkFlow and the AXMEDIS Engines:

- AXMEDIS Compositional/Formatting Engine
- AXMEDIS Program and Publication Engine
- AXEPTool Loading Tool Engine
- AXEPTool Publication Tool Engine
- The Protection Tool Engine

5.2.1 The Interface between the WF Engine Request Gateway and the Engine Command and Reporting

As described before, the AXMEDIS Engine WorkFlow channel passes through the WF Engine Request Gateway where the Engine Command and Reporting will expose the following methods, via WebServices:

- `Install_and_activate` for installing a XML rule in the scheduler and activate it. This method is valid for the Compositional/Formatting engine, for the AxepTool Loading and Publication Engine and for the Protection Engine.
- `Run_rule` for immediately run a rule already loaded inside the scheduler. This method is valid for the Compositional/Formatting engine, for the AxepTool Loading and Publication Engine and for the Protection Engine.
- `Activate_rule` for activating a rule already loaded inside the scheduler. This method is valid for the Compositional/Formatting engine, for the AxepTool Loading and Publication Engine and for the Protection Engine.
- `deactivate_rule` for disabling a not-running rule. This method is valid for the Compositional/Formatting engine, for the AxepTool Loading and Publication Engine and for the Protection Engine.
- `Suspend_rule` for suspending a rule for a specified time interval. This method is valid for the Compositional/Formatting engine, for the AxepTool Loading and Publication Engine and for the Protection Engine.
- `Pause_rule`, for suspending a rule until it will be restarted. This method is valid for the Compositional/Formatting engine, for the AxepTool Loading and Publication Engine and for the Protection Engine.
- `Kill_rule` for stopping the execution of a rule. This method is valid for the Compositional/Formatting engine, for the AxepTool Loading and Publication Engine and for the Protection Engine.
- `Remove_rule` for removing a rule from the scheduler. This method is valid for the Compositional/Formatting engine, for the AxepTool Loading and Publication Engine and for the Protection Engine.
- `Resume_rule` for resuming a paused rule. This method is valid for the Compositional/Formatting engine, for the AxepTool Loading and Publication Engine and for the Protection Engine.
- `Get_rule_status` for getting the status of a rule inside the scheduler. This method is valid for the Compositional/Formatting engine, for the AxepTool Loading and Publication Engine and for the Protection Engine.

- `Get_rule_logs` for getting history log of a rule. This method is valid for the Compositional/Formatting engine, for the AxepTool Loading and Publication Engine and for the Protection Engine.
- `Get_list_of_rules` for getting the list of the rules of a certain user inside the scheduler. This method is valid for the Compositional/Formatting engine, for the AxepTool Loading and Publication Engine and for the Protection Engine.
- `Get_rule` for getting the XML definition of a rule inside the scheduler. This method is valid for the Compositional/Formatting engine, for the AxepTool Loading and Publication Engine and for the Protection Engine.
- `Status_request_to_PnP` for getting the status of a Program of the Program and Publication Engine
- `Suspend_PnP_Program` for suspending a Program of the Program and Publication Engine
- `Abort_PnP_Program` for aborting a Program of the Program and Publication Engine
- `Resume_PnP_Program` for resuming a suspended Program of the Program and Publication Engine
- `Activate_PnP_Program` for activating a Program of the Program and Publication Engine
- `WorFlow_Notification` is used to return back to the requesting engine (basically PnP) the status about the requested execution of a WorFlow process

The method invocation is performed via a WebService request where the following parameters are sent (to Engine Command and Reporting) and received back (in WebService result) from the Engine Command and Reporting:

- `install_and_activate`:
 - INPUT: `User_Credentials`, `AXRQID`, `XML_rule_schema`, `EngineListenerService`
 - OUTPUT: `Operation_Result` (OK, EXCEPTION)
 - XML_Rule_schema specifies the XML schema of the rule to install in the scheduler
- `run_rule`:
 - INPUT: `User_Credentials`, `AXRQID`, `RuleID`, `rule_type`, `EnginelistenerService`
 - OUTPUT: `Operation_Result` (OK, EXCEPTION)
- `Activate_rule`:
 - INPUT: `User_Credentials`, `AXRQID`, `RuleID`, `rule_type`, `EnginelistenerService`
 - OUTPUT: `Operation_Result` (OK, EXCEPTION)
- `deactivate_rule`:
 - INPUT: `User_Credentials`, `AXRQID`, `RuleID`, `rule_type`
 - OUTPUT: `Operation_Result` (OK, EXCEPTION)
- `Suspend_rule`:
 - INPUT: `User_Credentials`, `AXRQID`, `RuleID`, `rule_type`, `Max_time`
 - OUTPUT: `Operation_Result` (OK, EXCEPTION)
 - Max_time specifies the maximum amount of time for the rule to remain suspended
- `Pause_rule`:
 - INPUT: `User_Credentials`, `AXRQID`, `RuleID`, `rule_type`
 - OUTPUT: `Operation_Result` (OK, EXCEPTION)

- Kill_rule:
 - INPUT: User_Credentials, AXRQID, RuleID, rule_type
 - OUTPUT: Operation_Result (OK, EXCEPTION)
- Remove_rule:
 - INPUT: User_Credentials, AXRQID, RuleID, rule_type
 - OUTPUT: Operation_Result (OK, EXCEPTION)
- Resume_rule:
 - INPUT: User_Credentials, AXRQID, RuleID, rule_type, EnginelistenerService
 - OUTPUT: Operation_Result (OK, EXCEPTION)
- Get_rule_status:
 - INPUT: User_Credentials, AXRQID, RuleID, rule_type
 - OUTPUT: Operation_Result (STATUS, EXCEPTION)
STATUS is the current status of the rule in the scheduler
- Get_rule_logs:
 - INPUT: User_Credentials, AXRQID, RuleID, rule_type
 - OUTPUT: Operation_Result (List of log_info, EXCEPTION)
List of log_info is the list containint the lof of the rule
- Get_list_of_rules:
 - INPUT: User_Credentials, AXRQID, rule_type
 - OUTPUT: Operation_Result (List of RuleID, EXCEPTION)
- Get_rule:
 - INPUT: User_Credentials, AXRQID, RuleID, rule_type
 - OUTPUT: Operation_Result (XML_Rule_schema, EXCEPTION)
XML_Rule_schema specifies the XML schema of the rule in the scheduler
- Status_request_to_PnP:
 - INPUT: User_Credentials, AXRQID, ProgramID
 - OUTPUT: Operation_Result (STATUS, EXCEPTION)
ProgramID is the unique identifier of the program.
STATUS is the status of the Program
- Suspend_PnP_Program:
 - INPUT: User_Credentials, AXRQID, ProgramID
 - OUTPUT: Operation_Result (OK, EXCEPTION)
ProgramID is the unique identifier of the program.
- abort_PnP_Program:
 - INPUT: User_Credentials, AXRQID, ProgramID
 - OUTPUT: Operation_Result (OK, EXCEPTION)

ProgramID is the unique identifier of the program.

- resume_PnP_Program:
 - INPUT: User_Credentials, AXRQID, ProgramID, EnginelistenerService
 - OUTPUT: Operation_Result (OK, EXCEPTION)
 ProgramID is the unique identifier of the program.
- activate_PnP_Program:
 - INPUT: User_Credentials, AXRQID, ProgramID, EnginelistenerService
 - OUTPUT: Operation_Result (OK, EXCEPTION)
 ProgramID is the unique identifier of the program.
- WorkFlow_Notification:
 - INPUT: AXRQID, STATUS
 - OUTPUT: Operation_Result (OK, EXCEPTION)
 Where, in this case, AXRQID is the original RequestID sent from the engine to the WorkFlow and STATUS is the completion status of the requested process.

User_Credentials contain information for:

- User Identification
- User Session Identification

AXRQID uniquely identifies the previous request sent to the engine through the WF Rule Editor Channel.

RuleID specifies the unique identification of the rule to be managed.

Rule_type specifies the type of rule, i.e. the Engine that will run it.

EnginelistenerService is the URI to be used by the engine to return back the status of the evolution of the rule in the WebService to the WF Response Gateway.

The WSDL specification for the method invocations is:

```
<?xml version="1.0" encoding="UTF-8"?>
<WSDL:definitions xmlns:tns="http://www.AXMEDIS.org/engine.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/" targetNamespace="http://www.AXMEDIS.org/engine.wsdl"
name="Engine">
  <WSDL:types>
    <xsi:schema 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"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:ax="urn:ax"
xmlns="http://www.w3.org/2001/XMLSchema" targetNamespace="urn:ax" elementFormDefault="unqualified"
attributeFormDefault="unqualified">
      <xsi:import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
      <xsi:complexType name="rule_list">
        <xsi:sequence>
          <xsi:element name="ruleid" type="xsd:string" maxOccurs="20"/>
        </xsi:sequence>
      </xsi:complexType>
      <xsi:complexType name="rulehistory">
        <xsi:sequence>
          <xsi:element name="ruleLog" type="xsd:string" maxOccurs="100"/>
        </xsi:sequence>
      </xsi:complexType>
    </xsi:schema>
  </WSDL:types>
</WSDL:definitions>
```

```

        </xsi:complexType>
        <xsi:complexType name="engine-result">
            <xsi:sequence>
                <xsi:element name="result" type="xsd:boolean"/>
                <xsi:element name="errmsg" type="xsd:string" nillable="true"
minOccurs="0"/>
                <xsi:element name="errorcode" type="xsd:int"/>
            </xsi:sequence>
        </xsi:complexType>
        <!-- operation request element -->
        <xsi:element name="User_Credentials" type="xsd:string"/>
        <!-- operation request element -->
        <xsi:element name="rule_type" type="xsd:string"/>
        <!-- operation response element -->
        <xsi:element name="return" type="ax:engine-result"/>
        <!-- operation request element -->
        <xsi:element name="EngineListenerService" type="xsd:anyURI"/>
        <!-- operation request element -->
        <xsi:element name="AXRQID" type="xsd:string"/>
        <!-- operation response element -->
        <xsi:element name="result" type="xsd:boolean"/>
        <xsi:element name="rulehistory" type="ax:rulehistory"/>
        <xsi:element name="xml_rule_schema" type="xsd:string"/>
        <xsi:element name="ruleid" type="xsd:string"/>
        <xsi:element name="programid" type="xsd:string"/>
        <xsi:element name="suspend_time" type="xsd:int"/>
        <xsi:element name="status" type="xsd:string"/>
        <xsi:element name="rule_list" type="ax:rule_list"/>
    </xsi:schema>
</WSDL:types>
<message name="install_and_activateRequest">
    <part name="User_Credentials" element="ax:User_Credentials"/>
    <part name="xml_rule_schema" element="ax:xml_rule_schema"/>
    <part name="AXRQID" element="ax:AXRQID"/>
    <part name="EngineListenerService" element="ax:EngineListenerService"/>
</message>
<message name="getinstallandactivateResult">
    <part name="return" element="ax:return"/>
</message>
<message name="run_ruleRequest">
    <part name="User_Credentials" element="ax:User_Credentials"/>
    <part name="AXRQID" element="ax:AXRQID"/>
    <part name="ruleid" element="ax:ruleid"/>
    <part name="rule_type" element="ax:rule_type"/>
    <part name="EngineListenerService" element="ax:EngineListenerService"/>
</message>
<message name="getrunruleResult">
    <part name="return" element="ax:return"/>
</message>
<message name="activate_ruleRequest">
    <part name="User_Credentials" element="ax:User_Credentials"/>
    <part name="AXRQID" element="ax:AXRQID"/>
    <part name="ruleid" element="ax:ruleid"/>
    <part name="rule_type" element="ax:rule_type"/>
    <part name="EngineListenerService" element="ax:EngineListenerService"/>
</message>
<message name="getactivateruleResult">
    <part name="return" element="ax:return"/>
</message>

```



```

<message name="deactivate_ruleRequest">
  <part name="User_Credentials" element="ax:User_Credentials"/>
  <part name="AXRQID" element="ax:AXRQID"/>
  <part name="ruleid" element="ax:ruleid"/>
  <part name="rule_type" element="ax:rule_type"/>
</message>
<message name="getdeactivateruleResult">
  <part name="return" element="ax:return"/>
</message>
<message name="suspend_ruleRequest">
  <part name="User_Credentials" element="ax:User_Credentials"/>
  <part name="AXRQID" element="ax:AXRQID"/>
  <part name="ruleid" element="ax:ruleid"/>
  <part name="rule_type" element="ax:rule_type"/>
  <part name="suspent_time" element="ax:suspend_time"/>
</message>
<message name="getsuspendruleResult">
  <part name="return" element="ax:return"/>
</message>
<message name="pause_ruleRequest">
  <part name="User_Credentials" element="ax:User_Credentials"/>
  <part name="AXRQID" element="ax:AXRQID"/>
  <part name="ruleid" element="ax:ruleid"/>
  <part name="rule_type" element="ax:rule_type"/>
</message>
<message name="getpauseruleResult">
  <part name="return" element="ax:return"/>
</message>
<message name="kill_ruleRequest">
  <part name="User_Credentials" element="ax:User_Credentials"/>
  <part name="AXRQID" element="ax:AXRQID"/>
  <part name="ruleid" element="ax:ruleid"/>
  <part name="rule_type" element="ax:rule_type"/>
</message>
<message name="getkillruleResult">
  <part name="return" element="ax:return"/>
</message>
<message name="remove_ruleRequest">
  <part name="User_Credentials" element="ax:User_Credentials"/>
  <part name="AXRQID" element="ax:AXRQID"/>
  <part name="ruleid" element="ax:ruleid"/>
  <part name="rule_type" element="ax:rule_type"/>
</message>
<message name="getremoveruleResult">
  <part name="return" element="ax:return"/>
</message>
<message name="resume_ruleRequest">
  <part name="User_Credentials" element="ax:User_Credentials"/>
  <part name="AXRQID" element="ax:AXRQID"/>
  <part name="ruleid" element="ax:ruleid"/>
  <part name="rule_type" element="ax:rule_type"/>
  <part name="EngineListenerService" element="ax:EngineListenerService"/>
</message>
<message name="getresumeruleResult">
  <part name="return" element="ax:return"/>
</message>
<message name="get_rule_statusRequest">
  <part name="User_Credentials" element="ax:User_Credentials"/>
  <part name="AXRQID" element="ax:AXRQID"/>

```

```

        <part name="ruleid" element="ax:ruleid"/>
        <part name="rule_type" element="ax:rule_type"/>
    </message>
    <message name="getgetrulestatusResult">
        <part name="return" element="ax:return"/>
        <part name="status" element="ax:status"/>
    </message>
    <message name="get_rule_logsRequest">
        <part name="User_Credentials" element="ax:User_Credentials"/>
        <part name="AXRQID" element="ax:AXRQID"/>
        <part name="ruleid" element="ax:ruleid"/>
        <part name="rule_type" element="ax:rule_type"/>
    </message>
    <message name="getgetrulelogsResult">
        <part name="return" element="ax:return"/>
        <part name="rulehistory" element="ax:rulehistory"/>
    </message>
    <message name="get_list_of_rulesRequest">
        <part name="User_Credentials" element="ax:User_Credentials"/>
        <part name="AXRQID" element="ax:AXRQID"/>
        <part name="rule_type" element="ax:rule_type"/>
    </message>
    <message name="getgetlistofrulesResult">
        <part name="return" element="ax:return"/>
        <part name="rule_list" element="ax:rule_list"/>
    </message>
    <message name="get_ruleRequest">
        <part name="User_Credentials" element="ax:User_Credentials"/>
        <part name="AXRQID" element="ax:AXRQID"/>
        <part name="ruleid" element="ax:ruleid"/>
        <part name="rule_type" element="ax:rule_type"/>
    </message>
    <message name="getgetruleResult">
        <part name="return" element="ax:return"/>
        <part name="xml_rule_schema" element="ax:xml_rule_schema"/>
    </message>
    <message name="status_request_to_PnPRequest">
        <part name="User_Credentials" element="ax:User_Credentials"/>
        <part name="AXRQID" element="ax:AXRQID"/>
        <part name="programid" element="ax:programid"/>
    </message>
    <message name="getstatusrequesttoPnPResult">
        <part name="return" element="ax:return"/>
        <part name="status" element="ax:status"/>
    </message>
    <message name="suspend_PnP_programRequest">
        <part name="User_Credentials" element="ax:User_Credentials"/>
        <part name="AXRQID" element="ax:AXRQID"/>
        <part name="programid" element="ax:programid"/>
    </message>
    <message name="getsuspendPnPprogramResult">
        <part name="return" element="ax:return"/>
    </message>
    <message name="abort_PnP_programRequest">
        <part name="User_Credentials" element="ax:User_Credentials"/>
        <part name="AXRQID" element="ax:AXRQID"/>
        <part name="programid" element="ax:programid"/>
    </message>
    <message name="getabortPnPprogramResult">

```

```

    <part name="return" element="ax:return"/>
</message>
<message name="resume_PnP_programRequest">
    <part name="User_Credentials" element="ax:User_Credentials"/>
    <part name="AXRQID" element="ax:AXRQID"/>
    <part name="programid" element="ax:programid"/>
</message>
<message name="getresumePnPprogramResult">
    <part name="return" element="ax:return"/>
</message>
<message name="activate_PnP_programRequest">
    <part name="User_Credentials" element="ax:User_Credentials"/>
    <part name="AXRQID" element="ax:AXRQID"/>
    <part name="ruleid" element="ax:ruleid"/>
    <part name="EngineListenerService" element="ax:EngineListenerService"/>
</message>
<message name="getactivatePnPprogramResult">
    <part name="return" element="ax:return"/>
</message>
<message name="workflow_NotificationRequest">
    <part name="AXRQID" element="ax:AXRQID"/>
    <part name="status" element="ax:status"/>
</message>
<message name="getworkflownotificationResult">
    <part name="return" element="ax:return"/>
</message>
<portType name="EnginePortType">
    <operation name="install_and_activate">
        <documentation>Service definition of function ax:install_and_activate</documentation>
        <input message="tns:install_and_activateRequest"/>
        <output message="tns:getinstallandactivateResult"/>
    </operation>
    <operation name="run_rule">
        <documentation>Service definition of function ax:run_rule</documentation>
        <input message="tns:run_ruleRequest"/>
        <output message="tns:getrunruleResult"/>
    </operation>
    <operation name="activate_rule">
        <documentation>Service definition of function ax:activate_rule</documentation>
        <input message="tns:activate_ruleRequest"/>
        <output message="tns:getactivateruleResult"/>
    </operation>
    <operation name="deactivate_rule">
        <documentation>Service definition of function ax:deactivate_rule</documentation>
        <input message="tns:deactivate_ruleRequest"/>
        <output message="tns:getdeactivateruleResult"/>
    </operation>
    <operation name="suspend_rule">
        <documentation>Service definition of function ax:suspend_rule</documentation>
        <input message="tns:suspend_ruleRequest"/>
        <output message="tns:getsuspendruleResult"/>
    </operation>
    <operation name="pause_rule">
        <documentation>Service definition of function ax:pause_rule</documentation>
        <input message="tns:pause_ruleRequest"/>
        <output message="tns:getpauseruleResult"/>
    </operation>
    <operation name="kill_rule">
        <documentation>Service definition of function ax:kill_rule</documentation>

```

```

        <input message="tns:kill_ruleRequest"/>
        <output message="tns:getkillruleResult"/>
    </operation>
    <operation name="remove_rule">
        <documentation>Service definition of function ax:remove_rule</documentation>
        <input message="tns:remove_ruleRequest"/>
        <output message="tns:getremoveruleResult"/>
    </operation>
    <operation name="resume_rule">
        <documentation>Service definition of function ax:resume_rule</documentation>
        <input message="tns:resume_ruleRequest"/>
        <output message="tns:getresumeruleResult"/>
    </operation>
    <operation name="get_rule_status">
        <documentation>Service definition of function ax:get_rule_status</documentation>
        <input message="tns:get_rule_statusRequest"/>
        <output message="tns:getgetrulestatusResult"/>
    </operation>
    <operation name="get_rule_logs">
        <documentation>Service definition of function ax:get_rule_logs</documentation>
        <input message="tns:get_rule_logsRequest"/>
        <output message="tns:getgetrulelogsResult"/>
    </operation>
    <operation name="get_list_of_rules">
        <documentation>Service definition of function ax:get_list_of_rules</documentation>
        <input message="tns:get_list_of_rulesRequest"/>
        <output message="tns:getgetlistofrulesResult"/>
    </operation>
    <operation name="get_rule">
        <documentation>Service definition of function ax:get_rule</documentation>
        <input message="tns:get_ruleRequest"/>
        <output message="tns:getgetruleResult"/>
    </operation>
    <operation name="status_request_to_PnP">
        <documentation>Service definition of function ax:status_request_to_PnP</documentation>
        <input message="tns:status_request_to_PnPRequest"/>
        <output message="tns:getstatusrequesttoPnPResult"/>
    </operation>
    <operation name="suspend_PnP_program">
        <documentation>Service definition of function ax:suspend_PnP_program</documentation>
        <input message="tns:suspend_PnP_programRequest"/>
        <output message="tns:getsuspendPnPprogramResult"/>
    </operation>
    <operation name="abort_PnP_program">
        <documentation>Service definition of function ax:abort_PnP_program</documentation>
        <input message="tns:abort_PnP_programRequest"/>
        <output message="tns:getabortPnPprogramResult"/>
    </operation>
    <operation name="resume_PnP_program">
        <documentation>Service definition of function ax:resume_PnP_program</documentation>
        <input message="tns:resume_PnP_programRequest"/>
        <output message="tns:getresumePnPprogramResult"/>
    </operation>
    <operation name="activate_PnP_program">
        <documentation>Service definition of function ax:activate_PnP_program</documentation>
        <input message="tns:activate_PnP_programRequest"/>
        <output message="tns:getactivatePnPprogramResult"/>
    </operation>
    <operation name="workflow_notification">

```

```

        <documentation>Service definition of function ax:workflow_notification</documentation>
        <input message="tns:workflow_NotificationRequest"/>
        <output message="tns:getworkflownotificationResult"/>
    </operation>
</portType>
<WSDL:binding name="Engine" type="tns:EnginePortType">
    <SOAP:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <WSDL:operation name="install_and_activate">
        <SOAP:operation soapAction="urn:#install_and_activate"/>
        <input>
            <SOAP:body use="literal"/>
        </input>
        <input>
            <SOAP:body use="literal"/>
        </input>
        <output>
            <SOAP:body use="literal"/>
        </output>
    </WSDL:operation>
    <WSDL:operation name="run_rule">
        <SOAP:operation soapAction="urn:#run_rule"/>
        <input>
            <SOAP:body use="literal"/>
        </input>
        <input>
            <SOAP:body use="literal"/>
        </input>
        <output>
            <SOAP:body use="literal"/>
        </output>
    </WSDL:operation>
    <WSDL:operation name="activate_rule">
        <SOAP:operation soapAction="urn:#activate_rule"/>
        <input>
            <SOAP:body use="literal"/>
        </input>
        <input>
            <SOAP:body use="literal"/>
        </input>
        <output>
            <SOAP:body use="literal"/>
        </output>
    </WSDL:operation>
    <WSDL:operation name="deactivate_rule">
        <SOAP:operation soapAction="urn:#deactivate_rule"/>
        <input>
            <SOAP:body use="literal"/>
        </input>
        <input>
            <SOAP:body use="literal"/>
        </input>
        <output>
            <SOAP:body use="literal"/>
        </output>
    </WSDL:operation>
    <WSDL:operation name="suspend_rule">
        <SOAP:operation soapAction="urn:#suspend_rule"/>
        <input>
            <SOAP:body use="literal"/>

```

```

        </input>
        <input>
            <SOAP:body use="literal"/>
        </input>
        <output>
            <SOAP:body use="literal"/>
        </output>
    </WSDL:operation>
    <WSDL:operation name="pause_rule">
        <SOAP:operation soapAction="urn:#pause_rule"/>
        <input>
            <SOAP:body use="literal"/>
        </input>
        <input>
            <SOAP:body use="literal"/>
        </input>
        <output>
            <SOAP:body use="literal"/>
        </output>
    </WSDL:operation>
    <WSDL:operation name="kill_rule">
        <SOAP:operation soapAction="urn:#kill_rule"/>
        <input>
            <SOAP:body use="literal"/>
        </input>
        <input>
            <SOAP:body use="literal"/>
        </input>
        <output>
            <SOAP:body use="literal"/>
        </output>
    </WSDL:operation>
    <WSDL:operation name="remove_rule">
        <SOAP:operation soapAction="urn:#remove_rule"/>
        <input>
            <SOAP:body use="literal"/>
        </input>
        <input>
            <SOAP:body use="literal"/>
        </input>
        <output>
            <SOAP:body use="literal"/>
        </output>
    </WSDL:operation>
    <WSDL:operation name="resume_rule">
        <SOAP:operation soapAction="urn:#resume_rule"/>
        <input>
            <SOAP:body use="literal"/>
        </input>
        <input>
            <SOAP:body use="literal"/>
        </input>
        <output>
            <SOAP:body use="literal"/>
        </output>
    </WSDL:operation>
    <WSDL:operation name="get_rule_status">
        <SOAP:operation soapAction="urn:#get_rule_status"/>
        <input>

```

```

        <SOAP:body use="literal"/>
    </input>
    <input>
        <SOAP:body use="literal"/>
    </input>
    <output>
        <SOAP:body use="literal"/>
    </output>
</WSDL:operation>
<WSDL:operation name="get_rule_logs">
    <SOAP:operation soapAction="urn:#get_rule_logs"/>
    <input>
        <SOAP:body use="literal"/>
    </input>
    <input>
        <SOAP:body use="literal"/>
    </input>
    <output>
        <SOAP:body use="literal"/>
    </output>
</WSDL:operation>
<WSDL:operation name="get_list_of_rules">
    <SOAP:operation soapAction="urn:#get_list_of_rules"/>
    <input>
        <SOAP:body use="literal"/>
    </input>
    <input>
        <SOAP:body use="literal"/>
    </input>
    <output>
        <SOAP:body use="literal"/>
    </output>
</WSDL:operation>
<WSDL:operation name="get_rule">
    <SOAP:operation soapAction="urn:#get_rule"/>
    <input>
        <SOAP:body use="literal"/>
    </input>
    <input>
        <SOAP:body use="literal"/>
    </input>
    <output>
        <SOAP:body use="literal"/>
    </output>
</WSDL:operation>
<WSDL:operation name="status_request_to_PnP">
    <SOAP:operation soapAction="urn:#status_request_to_PnP"/>
    <input>
        <SOAP:body use="literal"/>
    </input>
    <input>
        <SOAP:body use="literal"/>
    </input>
    <output>
        <SOAP:body use="literal"/>
    </output>
</WSDL:operation>
<WSDL:operation name="suspend_PnP_program">
    <SOAP:operation soapAction="urn:#suspend_PnP_program"/>

```

```

        <input>
        <SOAP:body use="literal"/>
    </input>
    <input>
        <SOAP:body use="literal"/>
    </input>
    <output>
        <SOAP:body use="literal"/>
    </output>
</WSDL:operation>
<WSDL:operation name="abort_PnP_program">
    <SOAP:operation soapAction="urn:#abort_PnP_program"/>
    <input>
        <SOAP:body use="literal"/>
    </input>
    <input>
        <SOAP:body use="literal"/>
    </input>
    <input>
        <SOAP:body use="literal"/>
    </input>
    <output>
        <SOAP:body use="literal"/>
    </output>
</WSDL:operation>
<WSDL:operation name="resume_PnP_program">
    <SOAP:operation soapAction="urn:#resume_PnP_program"/>
    <input>
        <SOAP:body use="literal"/>
    </input>
    <input>
        <SOAP:body use="literal"/>
    </input>
    <input>
        <SOAP:body use="literal"/>
    </input>
    <output>
        <SOAP:body use="literal"/>
    </output>
</WSDL:operation>
<WSDL:operation name="activate_PnP_program">
    <SOAP:operation soapAction="urn:#activate_PnP_program"/>
    <input>
        <SOAP:body use="literal"/>
    </input>
    <input>
        <SOAP:body use="literal"/>
    </input>
    <input>
        <SOAP:body use="literal"/>
    </input>
    <output>
        <SOAP:body use="literal"/>
    </output>
</WSDL:operation>
<WSDL:operation name="workflow_notification">
    <SOAP:operation soapAction="urn:#workflow_notification"/>
    <input>
        <SOAP:body use="literal"/>
    </input>
    <input>
        <SOAP:body use="literal"/>
    </input>
    <input>
        <SOAP:body use="literal"/>
    </input>
    <output>
        <SOAP:body use="literal"/>
    </output>
</WSDL:operation>
</WSDL:binding>

```



```

<service name="Engine">
  <documentation>gSOAP 2.7.0e generated service definition</documentation>
  <port name="Engine" binding="tns:Engine">
    <SOAP:address location="http://www.AXMEDIS.org/Engine.cgi"/>
  </port>
</service>
</WSDL:definitions>

```

Operation_result a Boolean “true” if the request is OK, or a Boolean “false” if the request gets an error: in the latter case a error code and an error message is returned as well.

The Notifications about the Rule evolution are asynchronous notification containing the original AXRQID and the Status. The Notification is sent back through a WebService towards the WF Response Gateway, at the URI specified in the EnginelistenerService parameter.

5.2.2 Interface between the WF Engine Request Adapter and the WF Engine Request Gateway

The methods invoked and the parameters invoked by WF Engine Request Adapter to the WF Engine Request Gateway are the same described in the preceeding Paragraph. Their encoding, however, is different. The request in fact sent through an http GET call where the paramteres are invoked as follows:

As an example to understand the response, consider the compositional/formatting engine response to a request.

GET/Control_Panel/Products/OpenFlow/AXWF/comp_format_request_status?Credentials=”Credential string”&AXRQID=”Request ID string” HTTP/1.1" 200 368
["http://15.156.120.195:8080/Control_Panel/Products/OpenFlow/leave/leave_startform"](http://15.156.120.195:8080/Control_Panel/Products/OpenFlow/leave/leave_startform)
 "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR 1.0.3705; .NET CLR 1.1.4322)"

Where “Credential string” is a string containing the credentials and “Request ID string” is a string containing the Request ID.

The response to the invoked method has the same contents listed in the preceeding Paragraph and 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="errmsg" 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>

```

5.2.3 The Interface between the WF Engine Input Queue Adapter and the WF Engine Response Gateway

The WF Engine Response Gateway gets, through WebServices:

- Notifications about the status of the requested RuleID and ProgramID
- Requests for executing a Workflow process

The Notification is used to return to the Workflow the status of the evolution of the rules requested to the AXMEDIS Engine via the WF Rule Editor Channel. Specifically, it will contain:

From the Compositional / Formatting, AxepTool Loading and Publication and Protection:

- NOTIFICATION Input: AXRQID, rule_type, Completion_Result (STATUS, EXCEPTION)

STATUS = Status of the Request identified by AXRQID

Rule_type = type of rule identifying the Engine

From the Program and Publication Engine:

- NOTIFICATION Input: AXRQID, Completion_Result (STATUS, EXCEPTION)

STATUS = Status of the Request identified by AXRQID

Where AXRQID is the AXRQID in the original request from the Workflow and Completion_result can be either positive (OK or result) or negative (EXCEPTION is an error code returned for diagnostic purposes and useful for troubleshooting).

The NOTIFICATION is sent via an XMLRPC call whose method name is “EngineNotification” and whose parameter section is XML encoded as specified by the following 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_Notification">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="result" type="xs:boolean"/>
        <xs:element name="errmsg" 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:element name="rule_type" type="xs:string" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

On receiving the Notification, the WF Response Adapter will give the following XML response, 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>

```

Where AXRQID is a string containing the the original request sent to the Engine.

Result is a positive integer when the request was successfully completed, or negative integer otherwise.

Status is the Status of the Request.

The request for executing a WorFlow process is sent with the following parameters:

WORKFLOW_PROCESS_REQUEST Input: ProcessID, AXRQID, EnginelistenerService

ProcessID is the identification of the process definition that has to be instantiated, AXRQID is the Request identification which will be later used to send back notification to the requesting entity, EnginelistenerService is the URI to be used to send back such notifications.

The WORKFLOW_PROCESS_REQUEST is sent via an XMLRPC call whose XML encoding is specified:
The WORKFLOW_PROCESS_REQUEST is sent via an XMLRPC call whose method name is "WorkFlow_Process_Request" and whose parameter section is XML encoded as specified by the following 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="WorkFlow_Process_Request">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="AXRQID" type="xs:string"/>
      <xs:element name="processid" type="xs:string"/>
      <xs:element name="EngineListenerService" type="xs:anyURI"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>

```

On receiving the Request, the WF Response Adapter will give the following XML response, 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>

```

5.2.4 The Interface between the Engine Command and Reporting and the WF Engine Response Gateway

As described before, the Engine Command and Reporting will send its notifications to the WorkFlow Engine by calling a WebServices exposed by the WF Engine Response Gateway.

The URI of the WebService is indicated in the previous request made by the WF Request Gateway in the EngineListenerService parameter.

The Notification shall also contain the original Request ID issued in the request (AXRQID).

The following WSDL defines the Engine Notification WebService:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="Engine"
  targetNamespace="http://www.AXMEDIS.org/engine.wsdl"
  xmlns:tns="http://www.AXMEDIS.org/engine.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="unqualified"
      attributeFormDefault="unqualified">
      <import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
      <complexType name="Engine-result">
        <sequence>
          <element name="result" type="xsd:boolean" minOccurs="1" maxOccurs="1"/>
          <element name="errmsg" type="xsd:string" minOccurs="0" maxOccurs="1" nillable="true"/>
          <element name="errorcode" type="xsd:int" minOccurs="1" maxOccurs="1"/>
        </sequence>
      </complexType>
      <!-- operation response element -->
      <element name="return" type="ax:Engine-result"/>
      <!-- operation request element -->
      <element name="AXRQID" type="xsd:string"/>
      <!-- operation response element -->
      <element name="result" type="xsd:boolean"/>
      <element name="status" type="xsd:string"/>
      <element name="rule_type" type="xsd:string"/>
      <element name="processid" type="xsd:string"/>
      <element name="EngineListenerService" type="xsd:anyURI"/>
    </schema>

  </types>

  <message name="Engine_Notification">
    <part name="AXRQID" element="ax:AXRQID"/>
    <part name="status" element="ax:status"/>
    <part name="return" element="ax:return"/>
    <part name="rule_type" element="ax:rule_type"/>
  </message>
```

```

<message name="getNotificationResult">
<part name="return" element="ax:return"/>
</message>

<message name="WorFlow_ProcessRequest">
<part name="AXRQID" element="ax:AXRQID"/>
<part name="return" element="ax:return"/>
<part name="processid" element="ax:processid"/>
<part name="EngineListenerService" element="ax:EngineListenerService"/>
</message>

<message name="getworkflowprocessResult">
<part name="return" element="ax:return"/>
</message>

<portType name="EdnginNotificationPortType">
<operation name="Engine_Notification">
<documentation>Service definition of function ax:Engine_Notification</documentation>
<input message="tns:Engine_Notification"/>
<output message="tns:getNotificationResult"/>
</operation>
<operation name="WorkFlow_Process">
<documentation>Service definition of function ax:WorkFlow_Process</documentation>
<input message="tns:WorkFlow_ProcessRequest"/>
<output message="tns:getworkflowprocessResult"/>
</operation>
</portType>

<binding name="EngineNotification" type="tns:EngineNotificationPortType">
<SOAP:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
<operation name="Engine_Notification">
<SOAP:operation style="rpc" soapAction=""/>
<input>
<SOAP:body use="literal" namespace="urn:ax"/>
</input>
<output>
<SOAP:body use="literal" namespace="urn:ax"/>
</output>
</operation>
<operation name="WorkFlow_Process">
<SOAP:operation style="rpc" soapAction=""/>
<input>
<SOAP:body use="literal" namespace="urn:ax"/>
</input>
<output>
<SOAP:body use="literal" namespace="urn:ax"/>
</output>
</operation>
</binding>

<service name="EngineNotification">
<documentation>gSOAP 2.7.0e generated service definition</documentation>
<port name="EditorNotification" binding="tns:EngineNotification">
<SOAP:address location="http://www.AXMEDIS.org/EngineNotification.cgi"/>

```

```

</port>
</service>

</definitions>

```

5.3 The WorkFlow Rule Editor Channel

In the following Paragraph we will specify the interface between the WorkFlow and the AXMEDIS Rule Editors:

- AXMEDIS Compositional/Formatting Rule Editor
- AXMEDIS Program and Publication User Interface
- AXEPTool Publication/Loading Rule Editor
- AXMEDIS Protection User Interface and Rule Editor

5.3.1 The Interface between the WF Rule Editor Request Gateway and the User Command and Reporting

As described before, the AXMEDIS Rule Editor WorkFlow channel passes through the WF Rule Editor Request Gateway where the User Command and Reporting will expose the following methods, via WebServices:

- Edit_Composition_Formatting_Rule for launching the Composition/Formatting Rule Editor
- Program_Publication_User_Interface for launching the Program/Publication User Interface
- Activate_Program_Publication for requesting the activation of a Program
- List_of_Programs for requesting the list of current programs in PnP
- Edit_AXEPTool_Rule for launching the AXEPTool Rule Editor
- Edit_Protection_Rule for launching the Protection Rule Editor

The method invocation is performed via a WebService request where the following parameters are sent (to User Command and Reporting) and received back (in WebService result) from the User Command and Reporting:

- Edit_Composition_Formatting_Rule:
 - INPUT: AXRQID, User_Credentials, XML_Rule_header, UserListenerService
 - OUTPUT: Operation_Result (OK, EXCEPTION)
 - XML_Rule_header is an XML header containing an uncomplete rule to edit
 - UserListenerService is the URI to send back notification when the editor is terminated
- Program_Publication_User_Interface:
 - INPUT: User_Credentials, AXRQID, UserListenerService, ProgramName
 - OUTPUT: Operation_Result (OK, EXCEPTION)
 - UserListenerService is the URI to send back notification when the editor is terminated
 - ProgramName is the name of the Program to edit
- Activate_Program_Publication:
 - INPUT: User_Credentials, AXRQID, ProgramID, UserListenerService
 - OUTPUT: Operation_Result (OK, EXCEPTION)
 - ProgramID is the indentionification of the Program to activate

UserListenerService is the URI to send back notifications about the status of the evolution of the activated program

- List_of_Programs:
 - INPUT: User_Credentials, AXRQID
 - OUTPUT: Operation_Result (List of ProgramName, EXCEPTION)
List of ProgramName is the current list of the programs
- Edit_AXEPTool_Rule:
 - INPUT: AXRID, User_Credentials, AXRQID, UserListenerService, RuleName
 - OUTPUT: Operation_Result (OK, EXCEPTION)
UserListenerService is the URI to send back notification when the editor is terminated
RuleName is the name of the Rule to edit
- Edit_Protection_Rule:
 - INPUT: AXRID, User_Credentials, AXRQID, UserListenerService, RuleName
 - OUTPUT: Operation_Result (OK, EXCEPTION)
UserListenerService is the URI to send back notification when the editor is terminated
RuleName is the name of the Rule to edit

User_Credentials contain information for:

- User Identification
- User Session Identification

AXRQID uniquely identifies the update/create/delete request made to the User Command and Reporting. This is needed in order to:

- Identify the subsequent Notification from the Engine or User Command and Reporting to the WorkFlow Rule Editor Response Gateway. \
- Not to have to duplicate the same request as previously put to the User Command and Reporting Module. We assume that, if the User Command and Reporting receives a request with the same AXRQID as a previous one it will send back the same Notification

The AXRQID “encapsulates” the WorkFlow Manager generated parameters (openflow_ID, process_ID, activity_ID, instance_ID, AXWID).

The WSDL specification for the method invocations is:

```
<?xml version="1.0" encoding="UTF-8"?>
<WSDL:definitions xmlns:tns="http://www.AXMEDIS.org/Rule_Editor.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/" targetNamespace="http://www.AXMEDIS.org/Rule_Editor.wsdl"
name="Rule_Editor">
  <WSDL:types>
    <xsi:schema 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"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:ax="urn:ax"
xmlns="http://www.w3.org/2001/XMLSchema" targetNamespace="urn:ax" elementFormDefault="unqualified"
attributeFormDefault="unqualified">
      <xsi:import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
      <xsi:complexType name="program_list">
        <xsi:sequence>
          <xsi:element name="programid" type="xsd:string" maxOccurs="20"/>
        </xsi:sequence>
      </xsi:complexType>
      <xsi:complexType name="rule_editor-result">
        <xsi:sequence>
          <xsi:element name="result" type="xsd:boolean"/>
          <xsi:element name="errmsg" type="xsd:string" nillable="true"
minOccurs="0"/>
          <xsi:element name="errorcode" type="xsd:int"/>
        </xsi:sequence>
      </xsi:complexType>
      <!-- operation request element -->
      <xsi:element name="User_Credentials" type="xsd:string"/>
      <!-- operation response element -->
      <xsi:element name="return" type="ax:rule_editor-result"/>
      <!-- operation request element -->
      <xsi:element name="RuleListenerService" type="xsd:anyURI"/>
      <!-- operation request element -->
      <xsi:element name="AXRQID" type="xsd:string"/>
      <!-- operation response element -->
      <xsi:element name="result" type="xsd:boolean"/>
      <xsi:element name="xml_rule_schema" type="xsd:string"/>
      <xsi:element name="ruleid" type="xsd:string"/>
      <xsi:element name="programid" type="xsd:string"/>
      <xsi:element name="status" type="xsd:string"/>
      <xsi:element name="program_list" type="ax:program_list"/>
      <xsi:element name="program_name" type="xsd:string"/>
      <xsi:element name="rule_name" type="xsd:string"/>
    </xsi:schema>
  </WSDL:types>
  <message name="edit_composition_formatting_rule">
    <part name="User_Credentials" element="ax:User_Credentials"/>
    <part name="xml_rule_schema" element="ax:xml_rule_schema"/>
    <part name="AXRQID" element="ax:AXRQID"/>
    <part name="RuleListenerService" element="ax:RuleListenerService"/>
  </message>
  <message name="geteditcompositionformattingruleResult">
    <part name="return" element="ax:return"/>
  </message>
  <message name="program_publication_user_interface">
    <part name="User_Credentials" element="ax:User_Credentials"/>
    <part name="program_name" element="ax:program_name"/>
    <part name="AXRQID" element="ax:AXRQID"/>
    <part name="RuleListenerService" element="ax:RuleListenerService"/>
  </message>
  <message name="getprogrampublicationuserinterfaceResult">
    <part name="return" element="ax:return"/>
  </message>

```

```

<message name="activate_Program_PublicationRequest">
  <part name="User_Credentials" element="ax:User_Credentials"/>
  <part name="AXRQID" element="ax:AXRQID"/>
  <part name="programid" element="ax:programid"/>
  <part name="RuleListenerService" element="ax:RuleListenerService"/>
</message>
<message name="getactivateprogrampublicationResult">
  <part name="return" element="ax:return"/>
</message>
<message name="get_list_of_programsRequest">
  <part name="User_Credentials" element="ax:User_Credentials"/>
  <part name="AXRQID" element="ax:AXRQID"/>
</message>
<message name="getgetlistofprogramsResult">
  <part name="return" element="ax:return"/>
  <part name="program_list" element="ax:program_list"/>
</message>
<message name="edit_axeptool_rule">
  <part name="User_Credentials" element="ax:User_Credentials"/>
  <part name="rule_name" element="ax:rule_name"/>
  <part name="AXRQID" element="ax:AXRQID"/>
  <part name="RuleListenerService" element="ax:RuleListenerService"/>
</message>
<message name="geteditaxeptoolruleResult">
  <part name="return" element="ax:return"/>
</message>
<message name="edit_protection_rule">
  <part name="User_Credentials" element="ax:User_Credentials"/>
  <part name="rule_name" element="ax:rule_name"/>
  <part name="AXRQID" element="ax:AXRQID"/>
  <part name="RuleListenerService" element="ax:RuleListenerService"/>
</message>
<message name="geteditprotectionruleResult">
  <part name="return" element="ax:return"/>
</message>
<portType name="Rule_EditorPortType">
  <operation name="edit_composition_formatting_rule">
    <documentation>Service definition of function
ax:edit_composition_formatting_rule</documentation>
    <input message="tns:edit_composition_formatting_rule"/>
    <output message="tns:geteditcompositionformattingruleResult"/>
  </operation>
  <operation name="Program_Publication_User_Interface">
    <documentation>Service definition of function
ax:Program_Publication_User_Interface</documentation>
    <input message="tns:program_publication_user_interface"/>
    <output message="tns:getprogrampublicationuserinterfaceResult"/>
  </operation>
  <operation name="activate_program_publication">
    <documentation>Service definition of function
ax:activate_program_publication</documentation>
    <input message="tns:activate_Program_PublicationRequest"/>
    <output message="tns:getactivateprogrampublicationResult"/>
  </operation>
  <operation name="get_list_of_programs">
    <documentation>Service definition of function ax:get_list_of_programs</documentation>
    <input message="tns:get_list_of_programsRequest"/>
    <output message="tns:getgetlistofprogramsResult"/>
  </operation>

```

```

    <operation name="edit_axeptool_rule">
      <documentation>Service definition of function ax:edit_axeptool_rule</documentation>
      <input message="tns:edit_axeptool_rule"/>
      <output message="tns:geteditaxeptoolruleResult"/>
    </operation>
    <operation name="edit_protection_rule">
      <documentation>Service definition of function ax:edit_protection_rule</documentation>
      <input message="tns:edit_protection_rule"/>
      <output message="tns:geteditprotectionruleResult"/>
    </operation>
  </portType>
  <binding name="Rule_Editor" type="tns:Rule_EditorPortType">
    <SOAP:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <WSDL:operation name="edit_composition_formatting_rule">
      <SOAP:operation soapAction="urn:#edit_composition_formatting_rule"/>
      <input>
        <SOAP:body use="literal"/>
      </input>
      <input>
        <SOAP:body use="literal"/>
      </input>
      <output>
        <SOAP:body use="literal"/>
      </output>
    </WSDL:operation>
    <WSDL:operation name="Program_Publication_User_Interface">
      <SOAP:operation soapAction="urn:#Program_Publication_User_Interface"/>
      <input>
        <SOAP:body use="literal"/>
      </input>
      <input>
        <SOAP:body use="literal"/>
      </input>
      <output>
        <SOAP:body use="literal"/>
      </output>
    </WSDL:operation>
    <WSDL:operation name="activate_program_publication">
      <SOAP:operation soapAction="urn:#activate_program_publication"/>
      <input>
        <SOAP:body use="literal"/>
      </input>
      <input>
        <SOAP:body use="literal"/>
      </input>
      <output>
        <SOAP:body use="literal"/>
      </output>
    </WSDL:operation>
    <WSDL:operation name="get_list_of_programs">
      <SOAP:operation soapAction="urn:#get_list_of_programs"/>
      <input>
        <SOAP:body use="literal"/>
      </input>
      <input>
        <SOAP:body use="literal"/>
      </input>
      <output>
        <SOAP:body use="literal"/>
      </output>
    </WSDL:operation>
  </binding>

```

```

        </output>
      </WSDL:operation>
      <WSDL:operation name="edit_axeptool_rule">
        <SOAP:operation soapAction="urn:#edit_axeptool_rule"/>
        <input>
          <SOAP:body use="literal"/>
        </input>
        <input>
          <SOAP:body use="literal"/>
        </input>
        <output>
          <SOAP:body use="literal"/>
        </output>
      </WSDL:operation>
      <WSDL:operation name="edit_protection_rule">
        <SOAP:operation soapAction="urn:#edit_protection_rule"/>
        <input>
          <SOAP:body use="literal"/>
        </input>
        <input>
          <SOAP:body use="literal"/>
        </input>
        <output>
          <SOAP:body use="literal"/>
        </output>
      </WSDL:operation>
    </binding>
    <service name="Rule_Editor">
      <documentation>gSOAP 2.7.0e generated service definition</documentation>
      <port name="Rule_Editor" binding="tns:Rule_Editor">
        <SOAP:address location="http://www.AXMEDIS.org/Rule_Editor.cgi"/>
      </port>
    </service>
  </WSDL:definitions>

```

Operation_result a Boolean “true” if the request is OK, or a Boolean “false” if the request gets an error: in the latter case a error code and an error message is returned as well.

The Notifications about the Rule evolution are asynchronous notification containing the original AXRQID and the Status. The Notification is sent back through a WebService towards the WF Response Gateway, at the URI specified in the UserlistenerService parameter.

5.3.2 Interface between the WF Rule Editor Request Adapter and the WF Rule Editor Request Gateway

The methods invoked and the parameters invoked by WF Rule Editor Request Adapter to the WF Rule Editor Request Gateway are the same described in the preceding Paragraph. Thier encoding, however, is different. The request is in fact sent through an http GET call where the parameters are invoked as follows:

```

GET/Control_Panel/Products/OpenFlow/AXWF/rule_editor_name_request_status?AXRID="Rule
ID string"&Credentials="Credential string"&AXRQID="Request ID string" HTTP/1.1" 200 368
"http://15.156.120.195:8080/Control_Panel/Products/OpenFlow/leave/leave_startform"
"Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR 1.0.3705; .NET CLR 1.1.4322)"

```

Where **rule_ditor_name** is to be replaced by appropriate editor identifier “Rule ID string” is a string containing the AXRID, “Credential string” is a string containing the credentials and “Request ID string” is a string containing the Request ID.

The response to the invoked method has the same contents listed in the preceeding Paragraph and 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="errmsg" 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>
```

5.3.3 The Interface between the WF Rule Editor Input Queue Adapter and the WF Rule Editor Response Gateway

The WF Engine Response Gateway gets, through WebServices Notifications about the termination of the requested editor session.

Specifically, Notification it will contain:

- Edit_Composition_Formatting_Rule
 - NOTIFICATION: AXRQID, Completion_Result (OK, EXCEPTION)
- Edit_AXEPTool_Rule
 - NOTIFICATION: AXRQID, Completion_Result (OK, EXCEPTION)
- Edit_Protection_Rule
 - NOTIFICATION: AXRQID, Completion_Result (OK, EXCEPTION)
- Program_Publication_User_Interface:
 - NOTIFICATION: AXRQID, Completion_Result (OK, EXCEPTION)

Where AXRQID is the AXRQID in the original request from the WorkFlow and Completion_result can be either positive (OK) or negative (EXCEPTION is an error code returned for diagnostic purposes and useful for troubleshooting).

The NOTIFICATION is sent via an XMLRPC call whose method name is “RuleEditorNotification” and whose parameter section is XML encoded as specified by the following 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_Notification">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="result" type="xs:boolean"/>
        <xs:element name="errmsg" type="xs:string" nillable="true"
minOccurs="0"/>
        <xs:element name="errorCode" type="xs:int"/>
        <xs:element name="AXRQID" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

On receiving the Notification, the WF Response Adapter will give the following XML response, 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>
```

Where Rule_Editor_Name is the appropriate Rule Editor Identifier and AXRQID is a string containing the the original request sent to the Engine.

Result is a positive integer when the request was successfully completed, or negative integer otherwise.

Status is a string containing the newly created Object_ID if result is positive, otherwise a string containing the returned error.

5.3.4 The Interface between the User Command and Reporting and the WF Rule Editor Response Gateway

As described before, the User Command and Reporting will send its notifications to the WorkFlow Engine by calling a WebServices exposed by the WF Rule Editor Response Gateway.

The URI of the WebService is indicated in the previous request made by the WF Request Gateway in the UserListenerService parameter.

The Notification shall also contain the original Request ID issued in the request (AXRQID).

The following WSDL defines the User Notification WebService:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="Rule_Editor"
targetNamespace="http://www.AXMEDIS.org/Rule_Editor.wsdl"
xmlns:tns="http://www.AXMEDIS.org/Rule_Editor.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="unqualified"
attributeFormDefault="unqualified">
```

```

<import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
<complexType name="RuleEditor-result">
  <sequence>
    <element name="result" type="xsd:boolean" minOccurs="1" maxOccurs="1" />
    <element name="errmsg" type="xsd:string" minOccurs="0" maxOccurs="1" nillable="true" />
    <element name="errorcode" type="xsd:int" minOccurs="1" maxOccurs="1" />
  </sequence>
</complexType>
<!-- operation response element -->
<element name="return" type="ax:RuleEditor-result" />
<!-- operation request element -->
<element name="AXRQID" type="xsd:string" />
<!-- operation response element -->
<element name="result" type="xsd:boolean" />
</schema>

</types>

<message name="Rule_Editor_Notification">
<part name="AXRQID" element="ax:AXRQID" />
<part name="return" element="ax:return" />
</message>

<message name="geteNotificationResult">
<part name="return" element="ax:return" />
</message>

<portType name="RuleEditorNotificationPortType">
<operation name="Rule_Editor_Notification">
<documentation>Service definition of function ax:Rule_Editor_Notification</documentation>
<input message="tns:Rule_Editor_Notification" />
<output message="tns:getNotificationResult" />
</operation>

</portType>

<binding name="RuleEditorNotification" type="tns:RuleEditorNotificationPortType">
<SOAP:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
<operation name="Rule_Editor_Notification">
<SOAP:operation style="rpc" soapAction="" />
<input>
<SOAP:body use="literal" namespace="urn:ax" />
</input>
<output>
<SOAP:body use="literal" namespace="urn:ax" />
</output>
</operation>

```



```

</binding>

<service name="RuleEditorNotification">
<documentation>gSOAP 2.7.0e generated service definition</documentation>
<port name="RuleEditorNotification" binding="tns:RuleEditorNotification">
  <SOAP:address location="http://www.AXMEDIS.org/RuleEditorNotification.cgi"/>
</port>
</service>

</definitions>

```

5.4 The WorkFlow Query and DataBase Channel

In the following Paragraph we will specify the interface between the WorkFlow and the AXMEDIS Object Loader/Saver and the Query Support WebService Interface.

5.4.1 The Interface between the WF DB Request Gateway and the Loader/Saver and Query Support WebService Interface

As described before, the AXMEDIS DB WorkFlow channel passes through the WF DB Request Gateway where the Loader/Saver and the Query Support WebServices Interface modules will expose the following methods, via WebServices:

- Edit_Query for launching the Query Support User Interface
- Delete_selection for removing a selection from selection DB
- Load_selection for getting a selection from selection DB
- Save_selection for storing a selection in selection DB
- List_user_selection for listing the current selections in DB associated to the user
- List_entitled_selections for listing the current selections in DB that the user is entitled to execute
- Activate_selection_sync for activating a selection and waiting its completion
- Activate_selection_async for activating a selection and getting completion notification later
- Check_out_sync for checking-out an Object (MPEG-21 file) from AXMEDIS DB and waiting the completion of the operation
- Check_out_async for checking-out an Object (MPEG-21 file) from AXMEDIS DB and getting completion notification later
- commit_sync for checking-in an Object (MPEG-21 file) to AXMEDIS DB
- commit_async for checking-in an Object (MPEG-21 file) to AXMEDIS DB and getting completion notification later

The method invocation is encoded in an http GET request that contains both the method and the INPUT parameters. The related GET response will encode the OUTPUT parameters:

- Edit_Query:

- INPUT: Selection_ID, User_Credentials, AXRQID, commitListenerID
 - OUTPUT: Operation_Result (OK, EXCEPTION)

Selection_ID is the unique identifier of the Selection Rule to be edited
commitListenerID is the URI where to send back Notification about the termination of the Editor.
- Activate_Selection_sync:
 - INPUT: Selection_ID, User_Credentials, AXRQID
 - OUTPUT: Operation_Result (list of AXOID, EXCEPTION)

Selection_ID is the unique identifier of the Selection Rule to be activated
List of AXOID is the list of the selected AXMEDIS Objects
- Activate_Selection_async:
 - INPUT: Selection_ID, User_Credentials, AXRQID, commitListenerID
 - OUTPUT: Operation_Result (OK, EXCEPTION)

Selection_ID is the unique identifier of the Selection Rule to be activated
commitListenerID is the URI where to send back Notification about the evolution of the selection
- Delete_Selection:
 - INPUT: Selection_ID, User_Credentials, AXRQID
 - OUTPUT: Operation_Result (OK, EXCEPTION)

Selection_ID is the unique identifier of the Selection Rule to be deleted
- Load_Selection:
 - INPUT: Selection_ID, User_Credentials, AXRQID
 - OUTPUT: Operation_Result (XML_Selection, EXCEPTION)

Selection_ID is the unique identifier of the Selection Rule to be loaded.
XML_selection is the XML definition of the retrieved selection
- Save_Selection:
 - INPUT: Selection_ID, User_Credentials, AXRQID, XML_Selection
 - OUTPUT: Operation_Result (SelectionID, EXCEPTION)

Selection_ID is the unique identifier of the Selection Rule saved in DB
XML_selection is the XML definition of the saved selection
- List_user_Selection:
 - INPUT: User_Credentials, AXRQID
 - OUTPUT: Operation_Result (List of SelectionID, EXCEPTION)

List of Selection_ID is the list of the selections of the User in DB
- List_entitled_Selection:
 - INPUT: User_Credentials, AXRQID
 - OUTPUT: Operation_Result (List of SelectionID, EXCEPTION)

List of Selection_ID is the list of the selections in DB that the user is entitled to execute
- Check_out_sync:

- INPUT: AXOID, version, User_Credentials, AXRQID, downloadURI
 - OUTPUT: Operation_Result (OK, EXCEPTION)

DownloadURI is the pathname where the extracted MPEG-21 file will be put
Version is the version of the AXOID to retrieve
- Check_out_async:
 - INPUT: AXOID, version, User_Credentials, AXRQID, downloadURI, CommitListenerService
 - OUTPUT: Operation_Result (OK, EXCEPTION)

DownloadURI is the pathname where the extracted MPEG-21 file will be put
Version is the version of the AXOID to retrieve
CommitListenerService is the URI where to call the WebService for notifying the completion of the operation
- commit_sync:
 - INPUT: AXOID, User_Credentials, AXRQID, downloadURI,
 - OUTPUT: Operation_Result (version, EXCEPTION)

Path is the pathname where to get the MPEG-21 file to be put into the DB
downloadURI is the pathname where to get the MPEG-21 file
version is the version of the loaded object
- commit_async:
 - INPUT: AXOID, User_Credentials, AXRQID, downloadURI, CommitListenerService
 - OUTPUT: Operation_Result (OK, EXCEPTION)

Path is the pathname where to get the MPEG-21 file to be put into the DB
downloadURI is the pathname where to get the MPEG-21 file
CommitListenerService is the URI where to call the WebService for notifying the completion of the operation

User_Credentials contain information for:

- User Identification
- User Session Identification

AXRQID uniquely identifies the update/create/delete made request made. This is needed in order to:

- Identify the subsequent Notification to the WF Response Gateway
- Not to have to duplicate the same. We assume that, if the AXMEDIS Object Loader/Saver or Query Support WebService Interface receives a request with the same AXRQID as a previous one it will send back the same Notification

The AXRQID “encapsulates” the WorkFlow Manager generated parameters (openflow_ID, process_ID, activity_ID, instance_ID, AXWID).

commitListenerID is the URI where to send back Notification about the evolution of the Request.

The INPUT and OUTPUT parameters are the same specified in the previous paragraph, with the following encoding:

```

<?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:ax="urn:ax"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:tns="http://www.AXMEDIS.org/engine.wsdl" xmlns:ns=""
  targetNamespace="http://www.AXMEDIS.org/engine.wsdl">
  <types>
    <xs:schema xmlns:xsi="http://www.w3.org/2001/XMLSchema">
      <xs:element name="result" type="xs:boolean"/>
      <xs:element name="return" type="xs:string"/>
      <xs:element name="Selection_ID" type="xs:string"/>
      <xs:element name="User_Credentials" type="xs:string"/>
      <xs:element name="AXRQID" type="xs:string"/>
      <xs:element name="CommitListenerID" type="xs:anyURI"/>
      <xs:complexType name="ListofAXIOD">
        <xs:sequence>
          <xs:element name="AXOID" maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:complexType>
      <xs:complexType name="XMLSelection"/>
      <xs:complexType name="ListofSelectionID">
        <xs:sequence>
          <xs:element ref="Selection_ID"/>
        </xs:sequence>
      </xs:complexType>
      <xs:element name="AXOID" type="xs:string"/>
      <xs:element name="Version" type="xs:string"/>
      <xs:element name="DownloadURI" type="xs:anyURI"/>
    </xs:schema>
  </types>
  <message name="messageName"/>
  <message name="Edit_Query_Request">
    <part name="Selection_ID" element="ns:Selection_ID"/>
    <part name="User_Credentials" element="ns:User_Credentials"/>
    <part name="AXRQID" element="ns:AXRQID"/>
    <part name="CommitListenerID" element="ns:CommitListenerID"/>
  </message>
  <message name="Load_Selection_Request">
    <part name="Selection_ID" element="ns:Selection_ID"/>
    <part name="User_Credentials" element="ns:User_Credentials"/>
    <part name="AXRQID" element="ns:AXRQID"/>
  </message>
  <message name="Save_Selection_Request">
    <part name="Selection_ID" element="ns:Selection_ID"/>
  </message>

```

```

    <part name="User_Credentials" element="ns:Selection_ID"/>
    <part name="AXRQID" element="ns:AXRQID"/>
    <part name="XMLSelection" type="ns:XMLSelection"/>
  </message>
  <message name="List_User_Selection_Request">
    <part name="User_Credentials" element="ns:Selection_ID"/>
    <part name="AXRQID" element="ns:AXRQID"/>
  </message>
  <message name="List_Entitled_Selection_Request">
    <part name="User_Credentials" element="ns:Selection_ID"/>
    <part name="AXRQID" element="ns:AXRQID"/>
  </message>
  <message name="Activate_Selection_Sync_Request">
    <part name="Selection_ID" element="ns:Selection_ID"/>
    <part name="User_Credentials" element="ns:User_Credentials"/>
    <part name="AXRQID" element="ns:AXRQID"/>
  </message>
  <message name="Activate_Selection_ASync_Request">
    <part name="Selection_ID" element="ns:Selection_ID"/>
    <part name="User_Credentials" element="ns:Selection_ID"/>
    <part name="AXRQID" element="ns:AXRQID"/>
    <part name="CommitListenerID" element="ns:CommitListenerID"/>
  </message>
  <message name="Check_Out_Sync_Request">
    <part name="AXOID" element="ns:AXOID"/>
    <part name="Version" element="ns:Version"/>
    <part name="User_Credentials" element="ns:Selection_ID"/>
    <part name="AXRQID" element="ns:AXRQID"/>
    <part name="DownloadURI" element="ns:DownloadURI"/>
  </message>
  <message name="Check_Out_Async_Request">
    <part name="AXOID" element="ns:AXOID"/>
    <part name="Version" element="ns:Version"/>
    <part name="User_Credentials" element="ns:Selection_ID"/>
    <part name="AXRQID" element="ns:AXRQID"/>
    <part name="DownloadURI" element="ns:DownloadURI"/>
    <part name="CommitListenerID" element="ns:CommitListenerID"/>
  </message>
  <message name="Commit_Sync_Request">
    <part name="AXOID" element="ns:AXOID"/>
    <part name="User_Credentials" element="ns:Selection_ID"/>
    <part name="AXRQID" element="ns:AXRQID"/>
    <part name="DownloadURI" element="ns:DownloadURI"/>
  </message>
  <message name="Commit_Async_Request">
    <part name="AXOID" element="ns:AXOID"/>
    <part name="User_Credentials" element="ns:Selection_ID"/>
    <part name="AXRQID" element="ns:AXRQID"/>
    <part name="DownloadURI" element="ns:DownloadURI"/>
  </message>

```

```

    <part name="CommitListenerID" element="ns:CommitListenerID"/>
  </message>
  <message name="Delete_Selection_Request">
    <part name="Selection_ID" element="ns:Selection_ID"/>
    <part name="User_Credentials" element="ns:User_Credentials"/>
    <part name="AXRQID" element="ns:AXRQID"/>
  </message>
  <message name="Edit_Query_Result">
    <part name="Result" element="ns:result"/>
  </message>
  <message name="Load_Selection_Result">
    <part name="Result" type="ns:XMLSelection"/>
  </message>
  <message name="Activate_Selection_Sync">
    <part name="Result" type="ns:ListofAXIOD"/>
  </message>
  <message name="Activate_Selection_Async_Result">
    <part name="Result" element="ns:result"/>
  </message>
  <message name="Delete_Selection_Result">
    <part name="Result" element="ns:result"/>
  </message>
  <message name="Save_Selection_Result">
    <part name="Result" element="ns:Selection_ID"/>
  </message>
  <message name="List_User_Selection_Result">
    <part name="Result" type="ns:ListofSelectionID"/>
  </message>
  <message name="List_Entitled_Selection_Result">
    <part name="Result" type="ns:ListofSelectionID"/>
  </message>
  <message name="Check_Out_Sync_Result">
    <part name="Result" element="ns:result"/>
  </message>
  <message name="Check_Out_Async_Result">
    <part name="Result" element="ns:result"/>
  </message>
  <message name="Commit_Sync_Result">
    <part name="Result" element="ns:Version"/>
  </message>
  <message name="Commit_Aync_Result">
    <part name="Result" element="ns:result"/>
  </message>
  <portType name="DatabasePortType">
    <operation name="Edit_Query">
      <input message="tns:Edit_Query_Request"/>
      <output message="tns:Edit_Query_Result"/>
    </operation>
    <operation name="Load_Selection">

```

```

        <input message="tns:Load_Selection_Request"/>
        <output message="tns:Load_Selection_Result"/>
    </operation>
    <operation name="Save_Selection">
        <input message="tns:Save_Selection_Request"/>
        <output message="tns:Save_Selection_Result"/>
    </operation>
    <operation name="List_User_Selection">
        <input message="tns:List_User_Selection_Request"/>
        <output message="tns:List_User_Selection_Result"/>
    </operation>
    <operation name="List_Entitled_Selection">
        <input message="tns:List_Entitled_Selection_Request"/>
        <output message="tns:List_Entitled_Selection_Result"/>
    </operation>
    <operation name="Activate_Selection_Sync">
        <input message="tns:Activate_Selection_Sync_Request"/>
        <output message="tns:Activate_Selection_Sync"/>
    </operation>
    <operation name="Activate_Selection_Async">
        <input message="tns:Activate_Selection_ASync_Request"/>
        <output message="tns:Activate_Selection_Async_Result"/>
    </operation>
    <operation name="Check_Out_Sync">
        <input message="tns:Check_Out_Sync_Request"/>
        <output message="tns:Check_Out_Sync_Result"/>
    </operation>
    <operation name="Check_Out_Async">
        <input message="tns:Check_Out_Async_Request"/>
        <output message="tns:Check_Out_Async_Result"/>
    </operation>
    <operation name="Commit_Sync">
        <input message="tns:Commit_Sync_Request"/>
        <output message="tns:Commit_Sync_Result"/>
    </operation>
    <operation name="Commit_Async">
        <input message="tns:Commit_Async_Request"/>
        <output message="tns:Commit_Aync_Result"/>
    </operation>
    <operation name="Delete_Selection">
        <input message="tns:Delete_Selection_Request"/>
        <output message="tns:Delete_Selection_Result"/>
    </operation>
</portType>
<binding name="Database" type="tns:DatabasePortType">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="Edit_Query">
        <soap:operation soapAction="urn:#Edit_Query" style="rpc"/>
        <input>

```

```

        <soap:body use="literal"/>
    </input>
    <output>
        <soap:body use="literal"/>
    </output>
</operation>
<operation name="Load_Selection">
    <soap:operation soapAction="urn:#Load_Selection" style="rpc"/>
    <input>
        <soap:body use="literal"/>
    </input>
    <output>
        <soap:body use="literal"/>
    </output>
</operation>
<operation name="Save_Selection">
    <soap:operation soapAction="urn:#Save_Selection" style="rpc"/>
    <input>
        <soap:body use="literal"/>
    </input>
    <output>
        <soap:body use="literal"/>
    </output>
</operation>
<operation name="List_User_Selection">
    <soap:operation soapAction="urn:#List_User_Selection" style="rpc"/>
    <input>
        <soap:body use="literal"/>
    </input>
    <output>
        <soap:body use="literal"/>
    </output>
</operation>
<operation name="List_Entitled_Selection">
    <soap:operation soapAction="urn:#List_Entitled_Selection" style="rpc"/>
    <input>
        <soap:body use="literal"/>
    </input>
    <output>
        <soap:body use="literal"/>
    </output>
</operation>
<operation name="Activate_Selection_Sync">
    <soap:operation soapAction="urn:#Activate_Selection_Sync" style="rpc"/>
    <input>
        <soap:body use="literal"/>
    </input>
    <output>
        <soap:body use="literal"/>
    </output>

```



```

        </output>
    </operation>
    <operation name="Activate_Selection_Async">
        <soap:operation soapAction="urn:#Activate_Selection_Async" style="rpc"/>
        <input>
            <soap:body use="literal"/>
        </input>
        <output>
            <soap:body use="literal"/>
        </output>
    </operation>
    <operation name="Check_Out_Sync">
        <soap:operation soapAction="urn:#Check_Out_Sync" style="rpc"/>
        <input>
            <soap:body use="literal"/>
        </input>
        <output>
            <soap:body use="literal"/>
        </output>
    </operation>
    <operation name="Check_Out_Async">
        <soap:operation soapAction="urn:#Check_Out_Async" style="rpc"/>
        <input>
            <soap:body use="literal"/>
        </input>
        <output>
            <soap:body use="literal"/>
        </output>
    </operation>
    <operation name="Commit_Sync">
        <soap:operation soapAction="urn:#Commit_Sync" style="rpc"/>
        <input>
            <soap:body use="literal"/>
        </input>
        <output>
            <soap:body use="literal"/>
        </output>
    </operation>
    <operation name="Commit_Async">
        <soap:operation soapAction="urn:#Commit_Async" style="rpc"/>
        <input>
            <soap:body use="literal"/>
        </input>
        <output>
            <soap:body use="literal"/>
        </output>
    </operation>
    <operation name="Delete_Selection">
        <soap:operation soapAction="urn:#Delete_Selection"/>

```

```

        <input>
          <soap:body use="literal"/>
        </input>
        <output>
          <soap:body use="literal"/>
        </output>
      </operation>
    </binding>
    <service name="Database">
      <port name="Database" binding="tns:Database">
        <soap:address location="http://www.AXMEDIS.org/Database.cgi"/>
      </port>
    </service>
  </definitions>

```

5.4.2 Interface between the WF DB Request Adapter and the WF DB Request Gateway

The methods invoked and the parameters invoked by WF DB Request Adapter to the WF DB Request Gateway are the same described in the preceding Paragraph. Their encoding, however, is different. The request in fact sent through an http GET call where the parameters are invoked as follows:

As an example to understand the response, consider the compositional/formatting engine response to a request.

GET/Control_Panel/Products/OpenFlow/AXWF/WFDB_request_status?Selection_ID="Selection ID string"&Credentials="Credential string"&AXRQID="Request ID string"&Path="path string" HTTP/1.1" 200 368

"http://15.156.120.195:8080/Control_Panel/Products/OpenFlow/leave/leave_startform"

"Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR 1.0.3705; .NET CLR 1.1.4322)"

Where WFDB is the workflow database, "Selection ID string" is a string containing the AXOID, "Credential string" is a string containing the credentials and "Request ID string" is a string containing the Request ID and "path string" contains the designated pathname to get the MPEG-21 file.

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="Database_Response">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="result" type="xs:boolean"/>
        <xs:element name="errmsg" type="xs:string" nillable="true" minOccurs="0"/>
        <xs:element name="errorcode" type="xs:int"/>
        <xs:element name="AXOID" type="xs:string" minOccurs="0"
maxOccurs="unbounded"/>
        <xs:element name="XML_Selection" minOccurs="0">
          <xs:complexType>

```

```

        <xs:simpleContent>
            <xs:extension base="xs:string"/>
        </xs:simpleContent>
    </xs:complexType>
</xs:element>
<xs:element name="Selection_ID" type="xs:string" minOccurs="0"
maxOccurs="unbounded"/>
    <xs:element name="Version" type="xs:string" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>

```

5.4.3 The Interface between the WF DB Input Queue Adapter and the WF DB Response Gateway

The Notification is used, as mentioned, to return to the WorkFlow the results of the requested operation. Specifically, it will contain:

- Edit_Query for notifying the termination of the editor:
 - NOTIFICATION: AXRQID, Completion_Result (OK, EXCEPTION)
- Activate_Selection_async for notifying the completion of the search:
 - NOTIFICATION: AXRQID, Completion Result (list of selected objects, EXCEPTION)
List_of_selected_objects is the list of AXOIDs selected in the Query
- Check_out_async:
 - NOTIFICATION: AXRQID, Completion_Result (OK, EXCEPTION)
- commit_async
 - NOTIFICATION: AXRQID, Completion_Result (version, EXCEPTION)
Version is the version of the loaded object

Where the AXRQID is the AXRQID in the original request from the WorkFlow and Completion_result it can be either positive (OK or returned parameters) or negative (EXCEPTION is an error code returned for diagnostic purposes and useful for troubleshooting).

The NOTIFICATION is sent via an XMLRPC call whose XML encoding is specified:

```

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

```

        <xs:element name="errorCode" type="xs:int"/>
        <xs:element name="AXRQID" type="xs:string"/>
        <xs:element name="AXOID" type="xs:string" minOccurs="0"
maxOccurs="unbounded"/>
        <xs:element name="Version" type="xs:string" minOccurs="0"/>
    </xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>

```

Where WFDB is the workflow database and AXRQID is a string containing the the original request sent to the Engine.

Result is a positive integer when the request was successfully completed, or negative integer otherwise.

Status is a string containing the newly created AXOID if result is positive, otherwise a string containing the returned error.

5.4.4 WebServices exposed by the WF DB Response Gateway to the AXMEDIS Object Loader/Mover and Query Support WebService Interface

As described before, the Loader/Saver and the Query Support WebService Interface will send its notifications to the WorkFlow Engine by calling a WebServices exposed by the WF DB Response Gateway.

The URI of the WebService is indicated in the previous request made by the WF Request Gateway in the commitListenerService parameter.

The Notification shall also contain the original Request ID issued in the request (AXRQID).

The following WSDL defines the User Notification WebService:

```

<?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:y="http://new.webservice.namespace" xmlns:ns="urn:ax"
targetNamespace="http://new.webservice.namespace">
    <types>
        <xsi:schema 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"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:ax="urn:ax"
xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:DIME="http://schemas.xmlsoap.org/ws/2002/04/dime/wsdl/"

```

```

xmlns:MIME="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:SOAP="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:WSDL="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="http://www.AXMEDIS.org/editor.wsdl" targetNamespace="urn:ax"
elementFormDefault="unqualified" attributeFormDefault="unqualified">
    <xsi:import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
    <xsi:complexType name="Database-result">
        <xsi:sequence>
            <xsi:element name="result" type="xsd:boolean"/>
            <xsi:element name="errmsg" type="xsd:string"
nillable="true" minOccurs="0"/>
            <xsi:element name="errorCode" type="xsd:int"/>
        </xsi:sequence>
    </xsi:complexType>
    <!-- operation response element -->
    <xsi:element name="return" type="ax:Database-result"/>
    <!-- operation request element -->
    <xsi:element name="AXRQID" type="xsd:string"/>
    <!-- operation response element -->
    <xsi:element name="result" type="xsd:boolean"/>
    <xsi:element name="AXOID" type="xsd:string"/>
    <xsi:element name="Version" type="xsd:string"/>
</xsi:schema>
</types>
<message name="DB_Notification">
    <part name="AXRQID" element="ns:AXRQID"/>
    <part name="Results" element="ns:result"/>
    <part name="AXOID" element="ns:AXOID"/>
    <part name="Version" element="ns:Version"/>
</message>
<message name="getNotificaitonResult">
    <part name="parameter" element="ns:result"/>
</message>
<portType name="DBNotificationPortType">
    <operation name="DB_Notification">
        <input message="y:DB_Notification"/>
        <output message="y:getNotificaitonResult"/>
    </operation>
</portType>
<binding name="DB_Notification" type="y:DBNotificationPortType">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="DB_Notification">
        <soap:operation soapAction="urn:#DB_Notification"/>
        <input>
            <soap:body use="literal"/>
        </input>
        <output>
            <soap:body use="literal"/>
        </output>
    </operation>

```

```

        </operation>
    </binding>
    <service name="DB_Notification">
        <port name="DB_Notification" binding="y:DB_Notification">
            <soap:address location="http://www.AXMEDIS.org/DBNotification.cgi"/>
        </port>
    </service>
</definitions>

```

6 AXMEDIS Integration with Available Workflow Environments

6.1 AXMEDIS integration and usage with the Open-Source AXWF: Openflow

The objective of this area of work was to identify open-source or proprietary Workflow Management Systems (WFMS) that are both dominant in the industrial sectors of interest (digital media production and distribution) and technologically suitable as candidates for having Plug-ins developed for them to support the AXMEDIS Integration Demonstrators. Accordingly an extensive state-of-the-art review of WFMSs was performed as documented in the AXMEDIS Requirements Document. It was concluded that Openflow as an open-source WFMS and BizTalk as a proprietary WFMS were the most suitable as will be justified later in this Section (4.1.1, 4.2.1).

All the open-source products feature:

- A Web Application based User Interface
- An API which can be easily used for developing the interfaces to the various AXMEDIS components

6.1.1 The Rationale for Choice of Openflow as an AXWF

1. Openflow has been chosen over other workflow engines due to its maturity and the availability of a simple and effective API interface. Furthermore it is not based on Java.
2. Regarding the other workflow engines that had been analysed, Jbpm the nearest contender is limited in terms of its development tools and this would impact negatively on the development cost (it is also Java based); Enhydra Shark, ObjectWeb Bonita and Open WFE are Java based engines and there is a negative bias towards Java based platforms from some of the leading consortium members.

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.

In the next two sections the interface between Openflow and the AXMEDIS tools and Engines is described. The analysis has been divided into two sections: a first section relating to the communication coming from Openflow to the AXMEDIS tools and a second section relating to the communication coming from the AXMEDIS tools to the AXWF.

6.1.2 AXMEDIS Tools Interacting with Openflow

Applications external to Openflow such as AXMEDIS components can use **the XML-RPC protocol** to call the methods of an OpenFlow object.

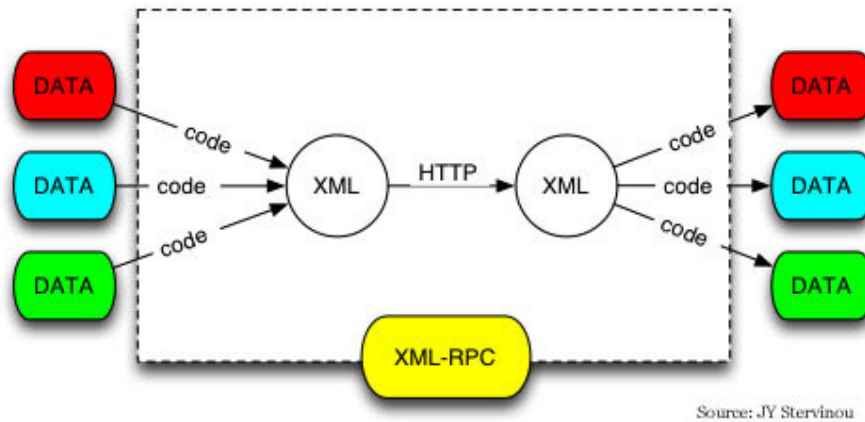
6.1.2.1 XML-RPC Protocol Description

The XML-RPC protocol is a set of implementations that allow software running on disparate operating systems and in different environments to make procedure calls over the Internet.

XML-RPC consists of Remote Procedure Calls that will use HTTP for the transport and XML for the encoding. It is designed to be as simple as possible, while allowing complex data structures to be transmitted, processed and returned.

An XML-RPC message is made of an HTTP-POST request. The body of the request is in XML. A procedure executes on the server and the value it returns is also formatted in XML. Procedure parameters can be scalars, numbers, strings, dates, etc.; and can also be complex record and list structures.

For further information, on the implementations page (<http://www.XMLRPC.com/directory/1568/implementations>) there is a list of the accomplishments of the XML-RPC community, a set of compatible XML-RPC implementations that span all operating systems, programming languages, dynamic and static environments, open-source and commercial, for Perl, Python, Java, Frontier, C/C++, Lisp, PHP, Microsoft .NET, Rebol, Real Basic, Tcl, Delphi, WebObjects and Zope. The figure below illustrates the XML-RPC usage.



The main methods provided by the Openflow via XML-RPC are listed below:

- 1 **addInstance**(process_ID, customer, comments, title, activation=0, RESPONSE=None):
 - CREATES an instance.
 - Returns the instance_ID.
- 2 **startInstance**(instance_ID, subflow_depth=0, REQUEST=None):
 - Creates a workitem for the activity "Begin", so Begin is assigned to someone.
 - This method is called automatically if the user gives "activation=1" to addInstance method.
- 3 **activateWorkitem**(instance_ID, workitem_ID, actor=None, REQUEST=None):
 - Activates the workitem (workitem_ID) of the instance (instance_ID).
 - When an application starts to run a job, it must activate the workitem with this method.
 - The value of workitem_ID is generated and given by Openflow to the application.
- 4 **completeWorkitem**(instance_ID, workitem_ID, REQUEST=None):
 - Signals to the Openflow that the task has been done.
 - If the activity is set as "Auto-Finish",
 - then Openflow will move to the next set of activities calling forwardWorkitem(),
 - assigning them (or running them if they are automatic).
- 5 **forwardWorkitem**(instance_ID, workitem_ID, path=None, REQUEST=None)
 - After having completed the current task with completeWorkitem(),
 - If the activity is not "Auto-Finish", you have to explicitly "move" to the following activities.
 - Calling forwardWorkitem() will assign (or run if they are automatic) the following activities. There is no need to call forwardWorkitem (it is called by completeWorkitem) when the activity is a subflow, or when the "Auto-Finish" flag is set.

- It is possible to explicitly indicate the path to follow (i.e. to indicate what are the following activities to move to?), by inserting in the path parameter a list of the IDs of the transitions to follow.
- It is possible to accommodate externally-signalled arbitrary termination of any processes i.e. situations when an explicit signal is needed to ascertain that an action is terminated. This is useful in an embedded environment or in the context of a lot of non-automated manual work including supervisor over-rides.

6 getInstance(instance_ID, REQUEST=None):

- Returns the Instance object.
- This method is useful to perform specific actions over the Instance object, for example to store data in it with: getInstance(instance_ID), manage_addProperty("issue_subject", issue_subject, "string")

7 deleteInstance(inst_IDs=None, RESPONSE=None):

- REMOVES an instance.

6.1.3 Openflow as the AXWF Interfacing with the AXMEDIS Components

Calls from external applications (AXMEDIS tools and engines) to Openflow are made via http protocol, using the classical GET method (parameters given in the URL).

Five parameters are always passed to the application: openflow_ID, process_ID, activity_ID, instance_ID and workitem_ID.

For additional parameters, the format is a python-like dictionary: comma-separated key-values pairs expressed as key:value, all wrapped in braces.

Example 1: if it is intended to pass "int_one" parameter with value 1 (int) and "str_one" parameter with value "one" (string), one should write: {"int_one": 1, "str_one": "one"}.

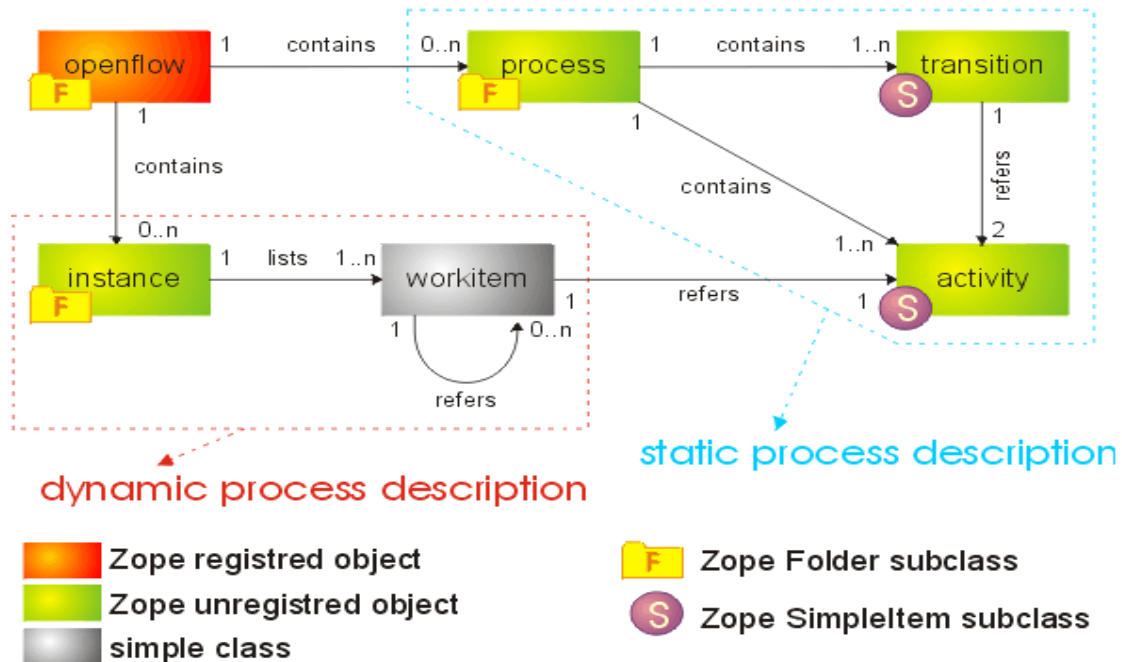
It is possible to pass items belonging to instance, activity, process and Openflow, like properties, python scripts results or other.

Example 2: {"int_one": instance.int_one, "str_one": process.id, "b": activity.id, c }

6.1.4 Openflow Structures and Terminology

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.

6.2 AXMEDIS Integration and Usage with the Proprietary AXWF: BizTalk

The Workflow and Interfaces development team within AXMEDIS have examined the various constraints as well as the pros and cons of operation and integration potential of several available proprietary WFMSs as candidates for integration within the AXMEDIS Platform at this stage.

Accordingly we have concluded that Microsoft's BizTalk, as a proprietary WFMS, represents the best choice for a variety of reasons as outlined in the following section.

6.2.1 The Rationale for the Choice of BizTalk as an AXWF

- 1) Today there are tens of WFMSs available on the market. Some are "standalone" products which had been intended to serve just as workflow engines, others originate from Document Management Systems (e.g. FileNet), yet others come from "application integration suites", or are rooted in the CRM/Trouble Ticketing type of Enterprise Applications (e.g. PeopleSoft, Siebel, ARS/Remedy).
- 2) As commercial WFMS type products have come from so many different stables, they tend to exhibit different features and advantages/disadvantages relative to a particular targeted context of application such as AXMEDIS.
- 3) As AXMEDIS is basically an integration framework and not a Document Management or CRM or other kind of specialised application, it is envisaged that the most suitable candidates for integration with AXMEDIS at this stage are to be drawn from the Enterprise Application Integration arena.
- 4) The following is a list of major commercial integration products:
 - Microsoft .NET (and then BizTalk)
 - IBM WebSphere
 - BEA WebLogic Integration
 - Oracle Application Server
 - Tibco InConcert
- 5) It is expected that the following features are to be given priority in the selection of candidate WFMSs to be considered for adoption as the AXMEDIS Workflow Systems integrated with the AXMEDIS Framework:
 - widely adopted
 - not too focused on any single sector or context of enterprise applications
 - capable of being easily integrated with the AXMEDIS framework
- 6) Following the extensive process of Multi-Sector Requirements Knowledge Elicitation and Analysis as undertaken in the initial stages of the AXMEDIS project together with the state-of-the-art review of all the major WFMSs it was concluded that:
 - there is no dominant WFMSs currently deployed in the multi-media Production/Distribution sector
 - MS NET BizTalk offered the greatest scope for this AXMEDIS integration.

6.2.2 Outline Description of the BizTalk

Microsoft states:

In today's global economy, companies need to integrate applications, systems, and technologies from a variety of sources. To facilitate integration Microsoft delivers best-of-breed integration technology through BizTalk Server 2004 and expands the offering with industry accelerators and adapters directly and through partners.

BizTalk Server 2004, a Windows Server System product, helps customers efficiently and effectively integrate systems, employees, and trading partners faster than ever before. BizTalk Server can interact with already deployed back-end and legacy systems to automate transactions between business partners, and to automate day-to-day business workflow.

The BizTalk Server 2004 engine provides expanded capabilities and services, such as:

- A new way to specify business rules
- Better ways to manage and monitor applications
- Support for single sign-on
- New services for information workers, including:
 - A group of Business Activity Services (BAS) that enable business users to manage their business partners and processes
 - Support for business process provisioning and configuration
 - Services that information workers can use to set up and manage interactions with trading partners
 - Business Activity Monitoring (BAM) features for analysing running business processes
 - Support for ad-hoc and semi-structured workflow through Human Workflow Services (HWS), which enables you to create business processes that people can interact with through client applications such as Microsoft Windows® SharePointA, A™ Services, Microsoft Office InfoPathA, A™, and Microsoft Office

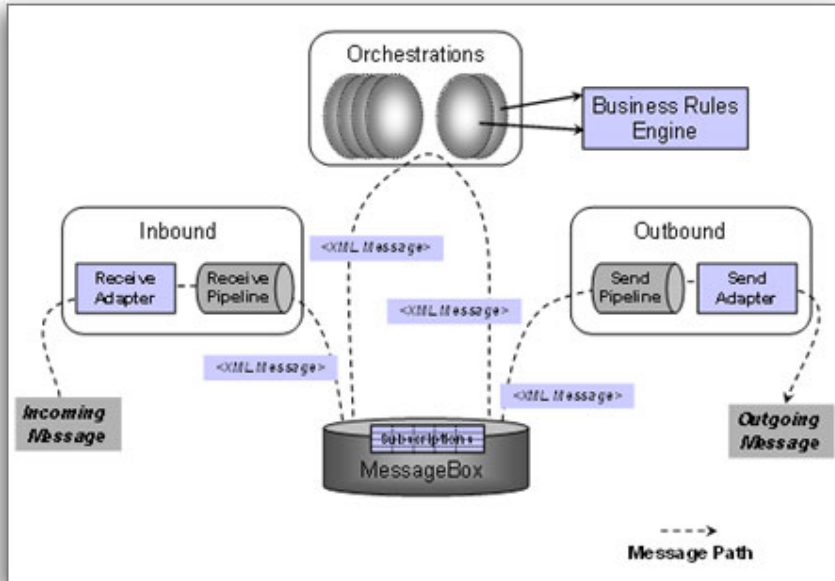
Unlike its COM-based predecessors, BizTalk Server 2004 is built completely around the Microsoft .NET Framework and Microsoft Visual Studio® .NET. It also has native support for communicating by using Web services, along with the ability to import and export business processes described in Business Process Execution Language (BPEL).

BizTalk Server 2004 can be implemented in the organisation in a variety of ways. When BizTalk Server is used for application integration, the following scenarios are most important:

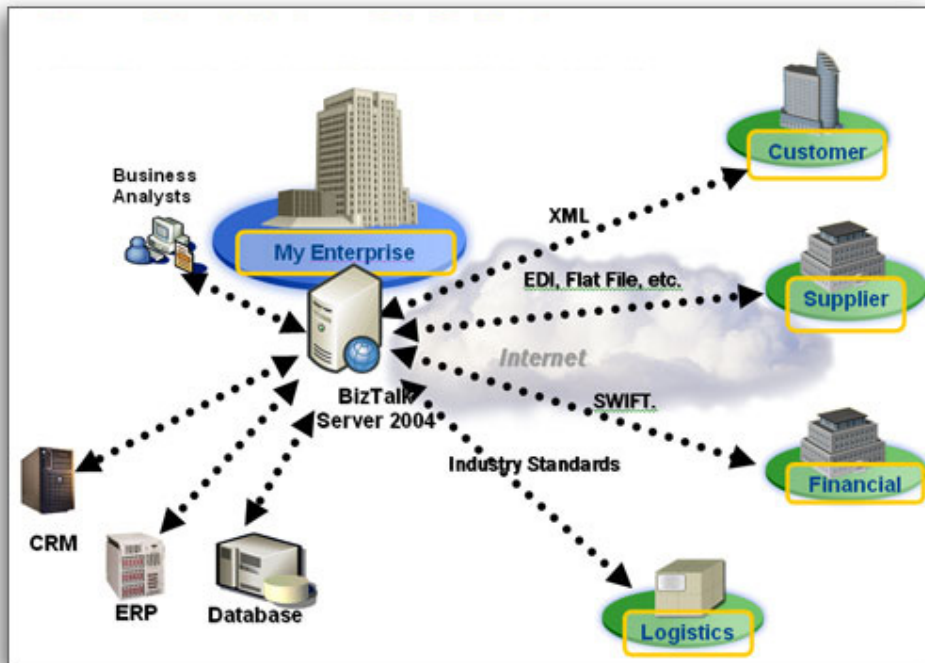
- Connecting applications within a single organisation, commonly referred to as Enterprise Application Integration (EAI)
- Connecting applications in different organisations, commonly referred to as business-to-business (B2B) Integration

6.2.3 Overview of BizTalk Architecture

The following figure shows the main components of BizTalk Server 2004. BizTalk Server consists of receive and send adapters, receive and send pipelines, orchestrations, the BizTalk Server message box, and the business rules engine.



Connecting trading partners and integrating systems is no longer the end goal of the enterprise integration. Companies require highly automated business process management functionality, with the flexibility to incorporate a human touch at appropriate stages throughout the workflow, as provided by BizTalk Server 2004 orchestration services. Additionally, with the BizTalk Server 2004 rules engine, companies can implement flexible business rules and make them visible to the information worker.



Accelerators are used to support a broad spectrum of business scenarios and industries, from high tech to healthcare. Accelerators for BizTalk Server add new functionality and complement Microsoft Windows Server System™ integrated server software, vastly reducing the time, effort, and costs associated with solution development, deployment, and management. BizTalk Server accelerators include a powerful combination of product enhancements, simple-to-use tools, documentation, and samples that are developed in concert to ensure they work well together. This translates into rapid deployments, a lower overall cost of ownership, and improved efficiency.

6.3 Open-Source Licensing of AXMEDIS-Developed Workflow Integration Software

In principle if one modifies open-source code for any purpose, then one has to release the thus *modified* open-source code back into the open-source community. For what concerns the Workflow Plug-ins software to be developed for AXMEDIS Integration, we foresee the need to use the open-source XMLRPClib, but in doing so the XMLRPClib will be left *un-modified*. This will obviate the need for the Plug-in code developed by AXMEDIS in this way to be made available to the open-source community.

For what concerns OpenFlow, it will probably be necessary to make some *modifications* to the OpenFlow data model, including accommodating specific AXMEDIS data (e.g. AXMEDIS AXOID). As the Openflow data model *thus modified* is part of the open-source code, in principle it *has to be released* back into the open-source community.

This means that the adaptation of the OpenFlow Data model to allow AXMEDIS Integration *will* become publicly available. However, the processes (activities, transitions and so on) implemented in the OpenFlow are not included in the open-source code, so they do not need to be made available to as open-source.

7 Glossary of Terms and Abbreviations (Part G)

7.1 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 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.

7.1 Glossary of Terms used with Workflow Management Systems Integration (continued)	
Term	Meaning
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 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.

7.1 Glossary of Terms used with Workflow Management Systems Integration (continued)

Term	Meaning
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

7.2 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.

Table of Acronyms Relevant to Workflow Management Systems Integration (continued)	
Term/Acronym	Meaning
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.

8 References to WFMS, Openflow, Zope, Python etc and Downloads

8.1 Openflow Workflow Management System

- <http://www.openflow.it/EN/Documentation/EN/Documentation>
- http://www.openflow.it/EN/index_html

8.2 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

8.2.1 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. xmirplib-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

8.3 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>

8.4 Workflow and Re-engineering International Association (waria)

- <http://www.waria.com/>
- <http://www.waria.com/books/study-volume3.htm>

8.5 XPDL

XML Process Definition Language (XPDL) of the Workflow Management Coalition (WfMC)

<http://www.wfmc.org/standards/docs.htm#Interface%20-%20Process%20Definition%20Interchange>