Automating Production of Cross Media Content for Multi-channel Distribution

www.AXMEDIS.org

DE3.1.2C Framework and Tools Specifications (Content Production)

Version: 2.2
Date: 15/03/2005
Responsible: DSI

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 C
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): DSI, EPFL, UNIVLEEDS, DIPITA< FUPF, FHGIGD, CRS4, EXITECH.

Abstract: This part of the specification deals with problems related to the Automatic Content Production and more in general with content processing, therefore the specification of the Content Processing and tools are reported. Specification is structured in 3 sections dealing with different aspects of content production: Automatic Content Processing Area based on rules and a distributed system, using JavaScript language for rules and definition of AXMEDIS Data Types for JavaScript Engine, Adaptation Tools and Algorithms for content processing, formatting, etc…

Keyword List:
Content production, Javascript, Adaptation tools

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)

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 document contains Part C only.

This document is focused on producing a first version of the specification without having deep details to be adopted as a baseline in the specification tasks of the different work packages that will be developed in the first 18 months of the project.
The part C of the specification deals with problems related to the Content Production and more in general with content processing, therefore the specification of the Content Processing and tools are reported.
Specification Part C is structured in 3 sections dealing with different aspects of content production:

- AXMEDIS Content Processing Area (under responsibility of DSI): this section describes the goals, the activity and tools of AXMEDIS Content Processing Area. This area will cope with the problem of automatic content production, adaptation and protection of AXMEDIS object and their publication on a P2P environment (AXEPTool). The proposed solution will be based on rules that will include a procedural description using the Javascript language (script) and a Javascript engine for the their execution derived from SpiderMonkey by Mozilla. To cope with the amount of needed resources (computational, time, etc…) during the content processing activity, a distributed environment will be defined and based on GRID computing. The AXMEDIS Content Processing Area specification is then further illustrated by means of UML diagrams, scenarios, tables, GUI definitions and design, definition of a Grid communication protocol and the XML schema for rules.
- AXMEDIS Data Types for JavaScript Engine: In this section how to use C++ classes and functions in the Spidermonkey Javascript engine are described. This mechanism will be used with the set of classes (methods and attributes) and functions that will have to be wrapped to be used with the Javascript code are reported. A first set of classes and functions are also reported.
- Adaptation Tools and Algorithms: in this section external tools and libraries for content processing, content format transcoding, content formatting and adaptation (digital audio, video, animation, multimedia, metadata, etc…) are discussed. They will be used to implement and customise the set of functions and algorithms to be used inside the automatic content processing area.

2 AXMEDIS Content Processing Area (DSI)

The AXMEDIS Content Processing Area will provide a set of digital content processing tools and will aid the content designer to:

- efficiently collect the components needed, using advanced query options
- find/produce alternatives for the components that present potential distribution problems (files too big).
- structure the components, highlighting the semantic relations between them.
- bind the structure content to some presentation styles.
- the same broadcast/broadband-quality content must also be formatted for delivery to a variety of channels, necessitating repurposing and even re-authoring.

- cope with different delivering channels according to different formatting styles and constrains about the profile of the final user device.
- cope with digital content protection and security
- cope with technical metadata and licensing,
- cope with composition, formatting and adaptation of digital resources
- cope with automatic definition of DRM rules for the new composite item, etc…
- cope with publication of contents on a P2P network (AXEPTool)

The structuring information provided by the user is of high value and should be stored and used when building other similar activities. This structuring information could also be shared with the other content designers of the P2P network. To this end the AXMEDIS Content Processing Area will be governed by rules. Rules will be the way to described the work to do for producing and distributing digital contents in automatic and secure manner. More in general, the AXMEDIS content processing area will group:

1. **Content Production** – New AXMEDIS objects will be produced by means of composition, formatting, adaptation of digital resources according to the publication, distribution channel and user profile requirements.
2. **Content Protection** – AXMEDIS objects will be protected in terms of DRM definitions, PAR, encryption, etc…
3. **Content Publication on AXEPTool** – It will perform content sharing and distribution among content producers and distributors. It will work directly with the P2P Network. It will enable the sharing of both content under DRM rules and unprotected content , monitoring of content movement and exploitation, etc.

In the user requirements analysis, the AXMEDIS objects Production (in terms of Composition and Formatting), Protection and Publication on AXEPTool were considered as separate areas with own tools and engines. A deep analysis shows that all these areas have many common aspects:

- the architecture
- the way to work
- the script language for implementing rules

These aspects suggest to design a unified solution that will provide many advantages:

- The definition of a single shared rule engine
- The definition of a unique rule format
- The design of a unique rule editor
- A single script language support
- A common set of data types derived from AXMEDIS Framework
- The possibility to assume such engine as a tool to be used in different context in the AXMEDIS framework: AXEPTool, Metadata manipulation, Content Production (Composition and formatting) and Protection. In this way, all these activity could be executed all together or separately or in group according to the content designer and publisher needs.

In this context, all types of rules involved in the three areas of content processing will be named as AXCP Rules. This unified solution will provide a versatile and customizable way to produce and manage digital content respecting legal aspects, DRM, ownership, user and publishers requirements.

2.1 Content Production (DSI)

The automatic production of contents, called AXMEDIS Object, will be based on the composition and formatting process.

The composition is the action of putting together content component to create a new digital item in an almost automatic manner. The final result is a new composite AXMEDIS object. The compositional activity should allow composing different kinds of raw assets such as Text, Images, Audio, Video (actual shot), Animation (synthetic), etc… coming from the AXMEDIS database. The composition could generate:

Basic Combined assets, for example:

- Texts + Image
- Texts + Audio

- Texts + Image + Audio
- Audio + Texts
- Animation + Texts
- Video + Texts
- Video compilations
- Audio compilations
- Image compilations
- All other different combinations of raw assets

Advanced Combined assets, for example:

- Multimedia presentation embedding sets of raw assets such as MPEG4, etc.
- Multimedia presentations composed of basic combined objects, such as HTML, etc..

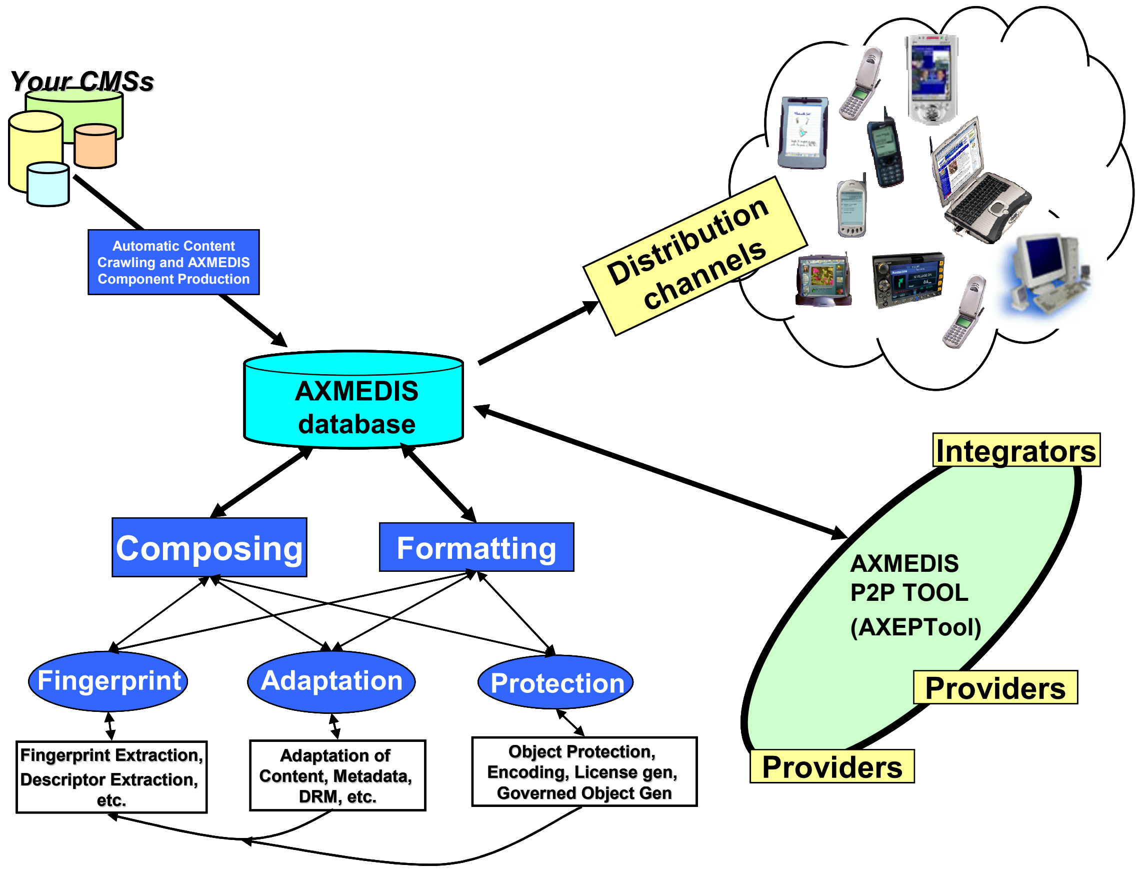
The composition could produce homogenous (where all digital resources are the same type) or heterogeneous (different kinds of digital resources) composite AXMEDIS objects.

Example:
Let suppose to get a Selection of digital documents from a large database of content. The Selection can be a the identification of a AXMEDIS components (digital objects) Italian singer Eros Ramazzotti, that is:

- 100 Audio, 50 Documents, 30 Images

Some solutions for different compositions could be:

- Any composition of 2 audio, 2 images and 4 documents
- Collection year by year with audio, documents and images of the same year
- Any two audio ordered by time, take these images and the doc number 45 which is the biography
- Any combination of 2 over the 100, plus put in any new object a specific document (for instance a given version of the bio plus its image, that could be the cover).


Your CMSs
Automatic Content Crawling and AXMEDIS Component Production
AXMEDIS database
Distribution channels
Composing
Formatting
Fingerprint
Adaptation
Protection
Fingerprint Extraction, Descriptor Extraction, etc.
Adaptation of Content, Metadata, DRM, etc.
Object Protection, Encoding, License gen, Governed Object Gen
Integrators
AXMEDIS P2P TOOL (AXEPTool)
Providers
Providers

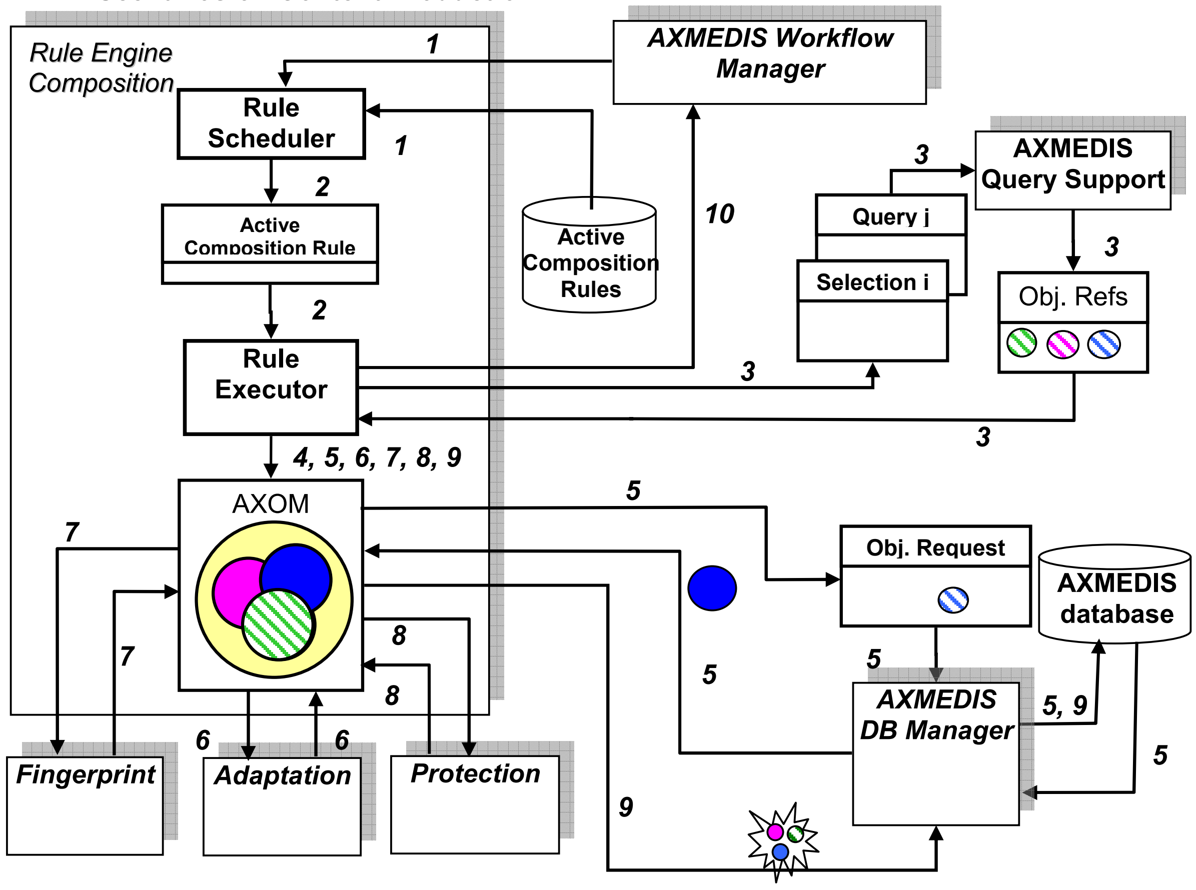
Formatting is the process to exploit the contained components in some integrated visualisation (editorial) format for their distribution and usage from the end user. A simple compounded object comprised of several parts (e.g., an audio, a video and a document), can be formatted in several different ways according to several manner according to different formatting styles (graphic layout, temporal scheduling of the content, speech generation from text, etc.) producing final content for i-TV, mobile, PC usage, etc…
These activities will be based on content features, generic user profile and needs, specific user profile (in the case of composition on demand), formatting style, optimisation parameters, end-user device profile, interactivity level and paradigms, content type and features, metadata, categorization, business information (price, localization, etc.), temporal evolution, DRM rules, delivery time, etc…
The formatting has to take into account the specific problems of the distribution channel:

- Location of the content and time to delivering
- Business and transaction models, thus DRM of content
- Model of delivering: streaming, download, off-line distribution, etc.
- Format for delivering: MPEG-4, simple audio files, documents, video, etc.

A wide set of formatting algorithms should be developed during project life-time. Each of such algorithms should be based on:

- AXMEDIS components with their description content type,
- Different content usage paradigms, which are the editorial formats.
    - page or formatting style/layout,
    - evolution of page style along the time,
    - Final format to be produced: MPEG-4, WAP, HTML, AXMEDIS integrated format, etc….
- Selected on the basis of formatting rules based on
    - User profiling, Program or channel profile, User's request, interactivity level,
    - availability of the content components and their costs
    - final device on which the content has to be received and used
    - Duration and complexity of each content component combined for creating the page.
- Shaping details for each content type…..
    - Video: time, size, frame rate, compression
    - Audio: time, sample rate, compression
    - Image: size, complexity, size
    - Text: #words, size of text (proportionally scalable)
    - Music: #measures, time, voices
    - Animation: size and time

2.1.1 Scenarios on Content Production

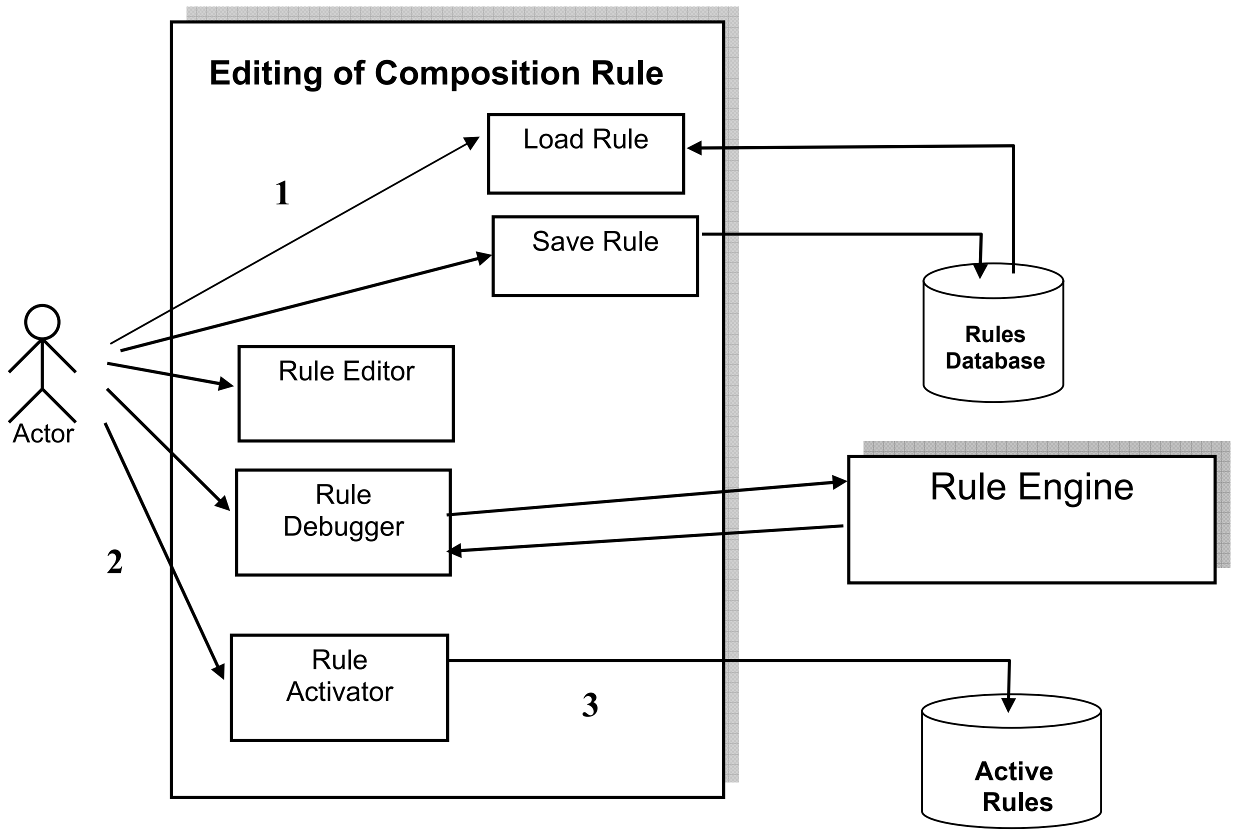


Scenario on Automatic Content Composition – Rule Engine for composition

Scenario description:

1. **Start composition process**. The Rule Engine receives a running rule request coming from the AXMEDIS Workflow Manager or the internal scheduler activates a rule from the Active Composition Rules.
2. **Rule execution request**. The scheduler sends a rule execution request to the rule executor with the corresponding rule.
3. **AXMEDIS Objects selection request**. For each selection and/or query specified in the rule, the rule executor sends queries to the AXMEDIS Query Support to obtain references to AXMEDIS objects that match the request.
4. **Embedding request**. An embedding object request with the relative object reference are sent to the AXOM to perform the inclusion.
5. **Physical Objects request**. If the physical embedding option is specified in the rule the physical object is requested to the AXMEDIS Database by means its reference, else the object is embedded as a reference.
6. **Adaptation request**. This request is performed via AXOM in order to adapt objects according to the adaptation parameters specified in the rule.
7. **Fingerprint request**. This request is performed via AXOM in order to apply the fingerprint to the embedded object according to the fingerprint parameters specified in the rule.
8. **Protection request**. This request is performed via AXOM. A protection request is sent to the Protection tool in order to apply protection to the new AXEMDIS composite object.
9. **Storing AXMEDIS object**. The new AXMEDIS Object is stored into the AXMEDIS Database.

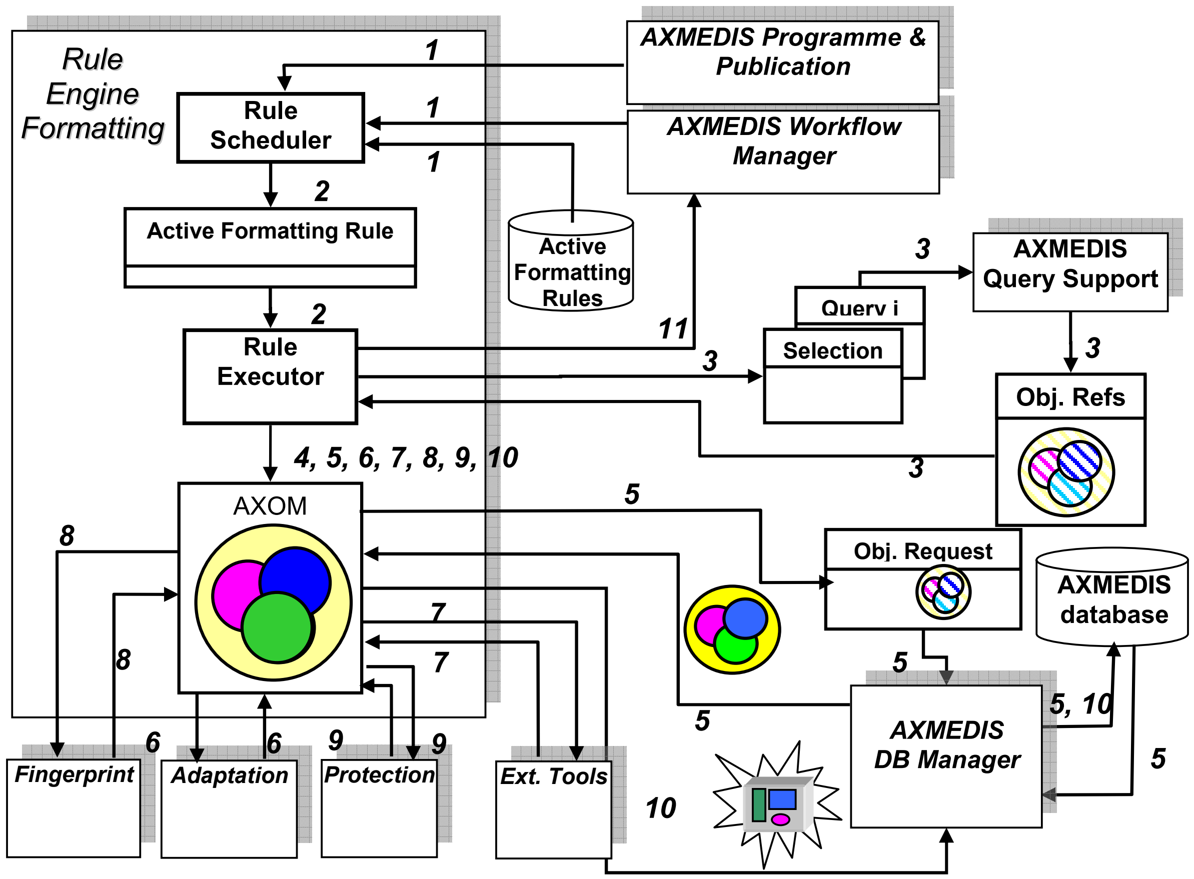
10. **End process notification**. The End of the composition is notified to the AXMEDIS Workflow Manager.



Scenario on Editing of Composition Rule

Scenario description for the activation of an existing rule:

1. The actor loads the Composition rule from the rules database
2. The actor activate the rule
3. The rule is sent to the Active Composition rules repository



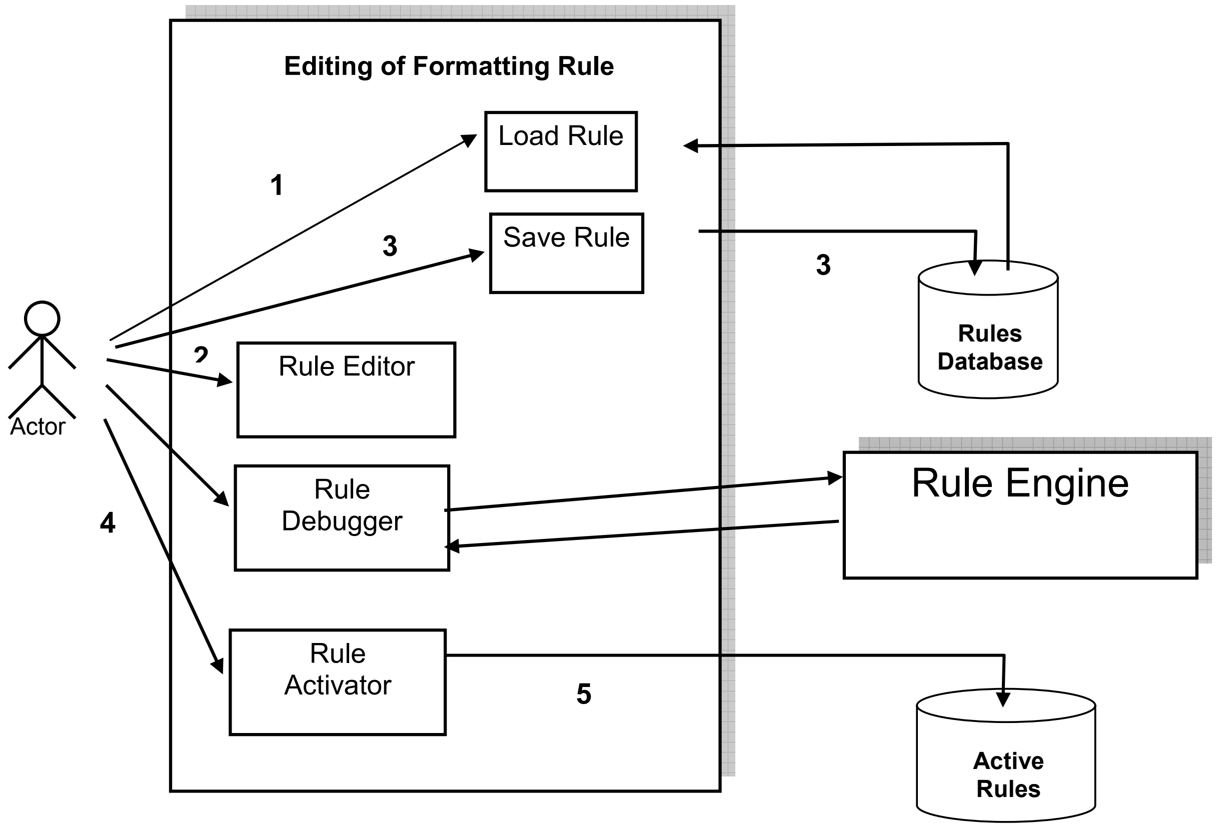
Scenario on Automatic Content Formatting – Rule Engine for formatting

Scenario description:

1. **Start formatting process**. The Rule Engine receives a running rule request coming from the AXMEDIS Workflow Manager, or from the AXMEDIS Publication & Programme, or from the internal scheduler that activates a rule from the Active Composition Rules.
2. **Rule execution request**. The scheduler sends the rule execution request to the rule executor with the corresponding rule (*Active formatting rule*).
3. **AXMEDIS Objects selection request**. For each selection and/or query specified in the rule, the rule executor sends queries to the AXMEDIS Query Support to obtain references to AXMEDIS objects that match the request.
4. **AXOM using**. An embedding object request with the relative object reference are sent to the AXOM to perform the inclusion.
5. **Physical Objects request**. The physical object is requested to the AXMEDIS Database by means its reference.
6. **Adaptation request**. This request is performed via AXOM in order to perform a formatting paradigm or a set of customised formatting parameters. This phase could:
    a. Perform adaptation algorithm (change resolution, change dimension, time or spatial best fitting, etc…)
    b. Apply spatial and temporal constrains specified in the rule (i.e. graphic layout, temporisation, transitions effects, etc…)
    c. Perform synchronisation algorithm (audio and text audio and images, video and text, etc…)
    d. Convert the whole formatted object into a specific output format (i.e. MPEG4).
7. **External tools calling**. This request allow calling external functionalities available on external formatting tools. In this way some formatting operation can be delegated and performed in other

formatting environment. These call are properly coded in the formatting rule and refer to portion of code written by using for example the script language available on the external tool.

8. **Fingerprint request**. This request is performed via AXOM in order to apply the fingerprint to the formatted object according to the fingerprint parameters specified in the rule.
9. **Protection request**. This request is performed via AXOM. A protection request is sent to the Protection tool in order to apply protection to the new AXEMDIS formatted object.
10. **Storing AXMEDIS object**. The new formatted AXMEDIS Object is stored into the AXMEDIS Database.
11. **End process notification**. The End of the formatting process is notified to the AXMEDIS Workflow Manager.



Scenario on Editing of Formatting Rule

Scenario description for the activation and modification of an existing formatting rule:

1. The actor loads an existing formatting rule from the rules database
2. The actor edits the rule by the RULE Editor
3. The actor saves the new rule into the rules database
4. The actor activate the rule
5. The rule is sent to the Active rules repository.

2.2 Content Protection (FHGIGD)

The Content Processing Area provides support for the protection of AXMEDIS objects and the license generation and verification. In this context the Content Protection will include:

- Applying Protection to AXMEDIS object : encryption, scrambling, compression, FP, … and creation of new Protection Information
- Sending the Protection Information to the database of the AXCS via the PMS.
- Creating a new object with license (called *AXMEDIS Governed Object*).
- Generating a license from license model and additional information.
- Check/Verification of an issued License or an existing PAR against some RIGHTS written in clear such as: "the play on the AXOID 34 in July 2005 for 5 times, the print of AXOID 56 in Spain in May 2006 at least one, etc."
- Check/Verification if it is possible to issue/generate a License with some RIGHTS written in clear such as: "the play on the AXOID 34 in July 2005 for 5 times, the print of AXOID 56 in Spain in May 2006 at least one, etc."
- Addition of rights or removal from a PAR/license (PAR/license adaptation): Generation of a new license (with new or less rights) AND Revocation of the old licenses in ONE TRANSACION.

A further detailed descriptions of the different scenarios about Protection can be found in section 5.1 of part H of the AXMEDIS framework specification "Scenarios on Content Protection Area".

2.3 Content Publication on AXEPTool (CRS4)

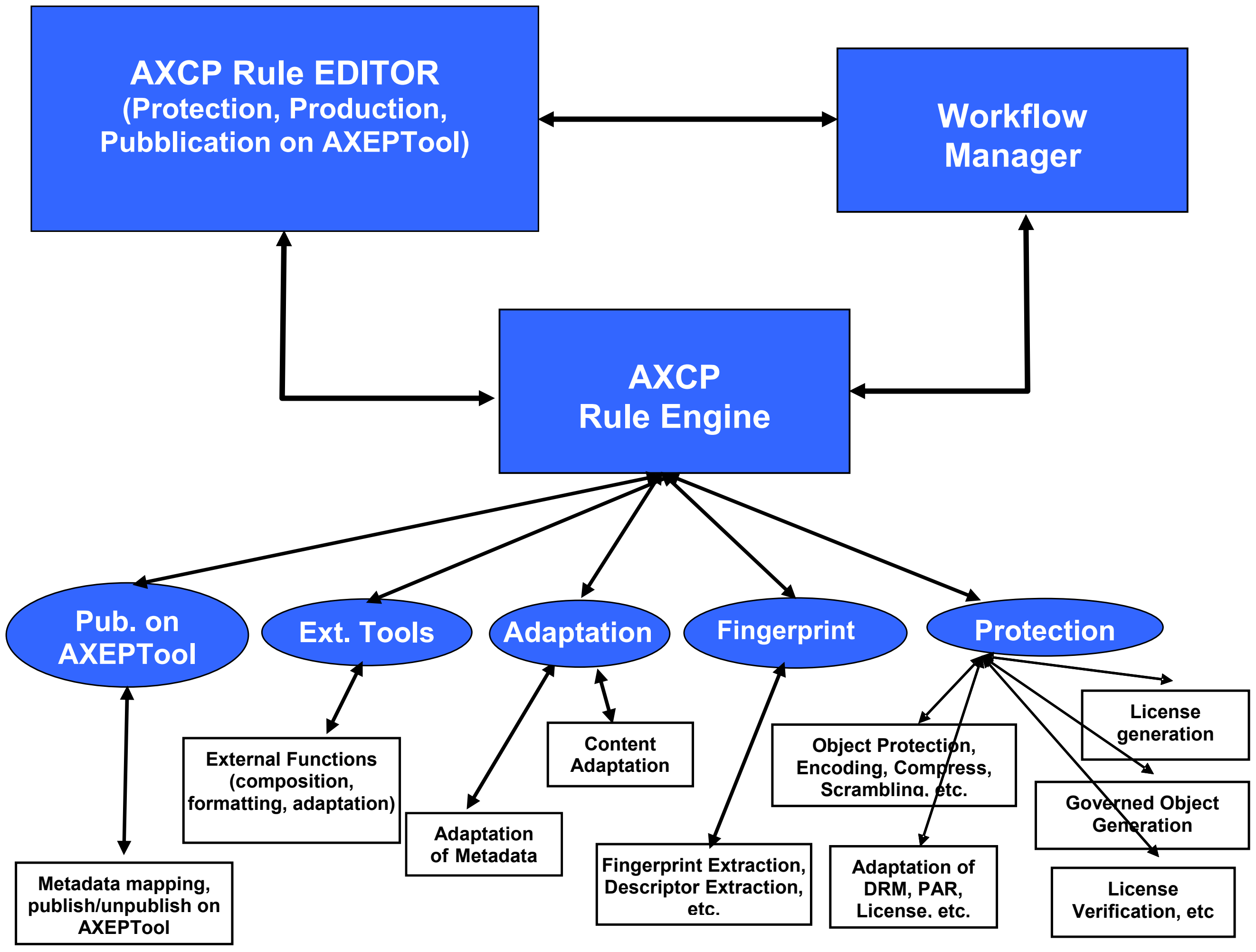
Axmedis contents are handled in the AXEPTool domain to publish, download and load objects. The AXEPTool performs content sharing and distribution among content producers and distributors. It works directly with the P2P Network. It will enable the sharing of both content under DRM rules and unprotected content, monitoring of content movement and exploitation, etc. It is involved in data flows between itself and the AXMEDIS Database Area and performs collaborative tasks. The main modules involved in content flow are the Loading Module, the Downloading Module and the Publication Module. It can be viewed as a stack due to the fact that it involves layer with different level of abstraction with respect to the network, data, and users.
Content publishing and loading in the AXEPTool is driven by AXCP Rules. Three different types of rules are used for publishing and loading:

- Publication Rule: when executed publishes a selection of objects in the P2P network. The component performing the publication is the **Publication Component of AXEPTool**
- Loading Rule: when executed loads a selection of objects from AXDBIN to AXDB. The component performing the publication is the **Loading Component of AXEPTool**
- Downloading Rule: when executed downloads a selection of objects from the P2P network to the AXDBIN. The component performing the publication is the **P2P Active Selections Component of AXEPTool**

For more details see document DE3.1.2F Framework and Tools Specifications (AXEPTool and Progr. and Pub.).

2.4 AXCP Content Processing Flow Diagram

In this section, main actors involved in the AXMEDIS Content Processing Area are described. The picture, reported below, shows the unified solution and relationships among the AXMEDIS Workflow Manager, the AXCP Rule Editor and Engine.


AXCP Rule EDITOR (Protection, Production, Pubblication on AXEPTool)
Workflow Manager
AXCP Rule Engine
Pub. on AXEPTool
Ext. Tools
Adaptation
Fingerprint
Protection
Metadata mapping, publish/unpublish on AXEPTool
External Functions (composition, formatting, adaptation)
Adaptation of Metadata
Content Adaptation
Fingerprint Extraction, Descriptor Extraction, etc.
Object Protection, Encoding, Compress, Scrambling, etc.
Adaptation of DRM, PAR, License, etc.
License generation
Governed Object Generation
License Verification, etc

1. **Workflow Manager** – It will perform the role of supervisor by monitoring and controlling the AXCP Rule engine and editor activity.
2. **AXCP Rules Editor –** It will be an editor for writing protection, production and publication on AXEPTool rules. It will be supported by a repository of rule rules and will interact with the AXCP Rule Engine.
3. **AXCP Rule Engine –** It will perform the automatic production, protection, adaptation process by means specific AXCP rules. It will generate new composite, formatted, adapted, protected AXMEDIS objects. Such objects will be successively stored in the AXMEDIS database or delivered via distribution channels or published by means the AXEPTool. The Engine will be supported by following modules and tools:
    a. **Protection** – It will provide functionalities and algorithms performing the protection of the AXMEDIS object via Protection Support
    b. **Fingerprint** – It will provide functionalities and algorithms performing the Fingerprint estimation of a new AXMEDIS object. The fingerprint could be based on component's fingerprint or could be a new one.

c. **Adaptation** – It will provide functionalities and algorithms performing the Content adaptation for different distribution channels and format paradigm.
d. **External tools** - Tools for using external formatting functionalities. It will provide a set of plug-ins that will allow using external tools (Macromedia suite, Adobe suite, etc…) and extending functionalities of the composition and formatting engine and AXMEDIS Editor.
e. **Publication on AXEPTool** – It will provide functionalities and algorithms performing metadata manipulation and mapping, publication of AXMEDIS object on the AXEPTool.

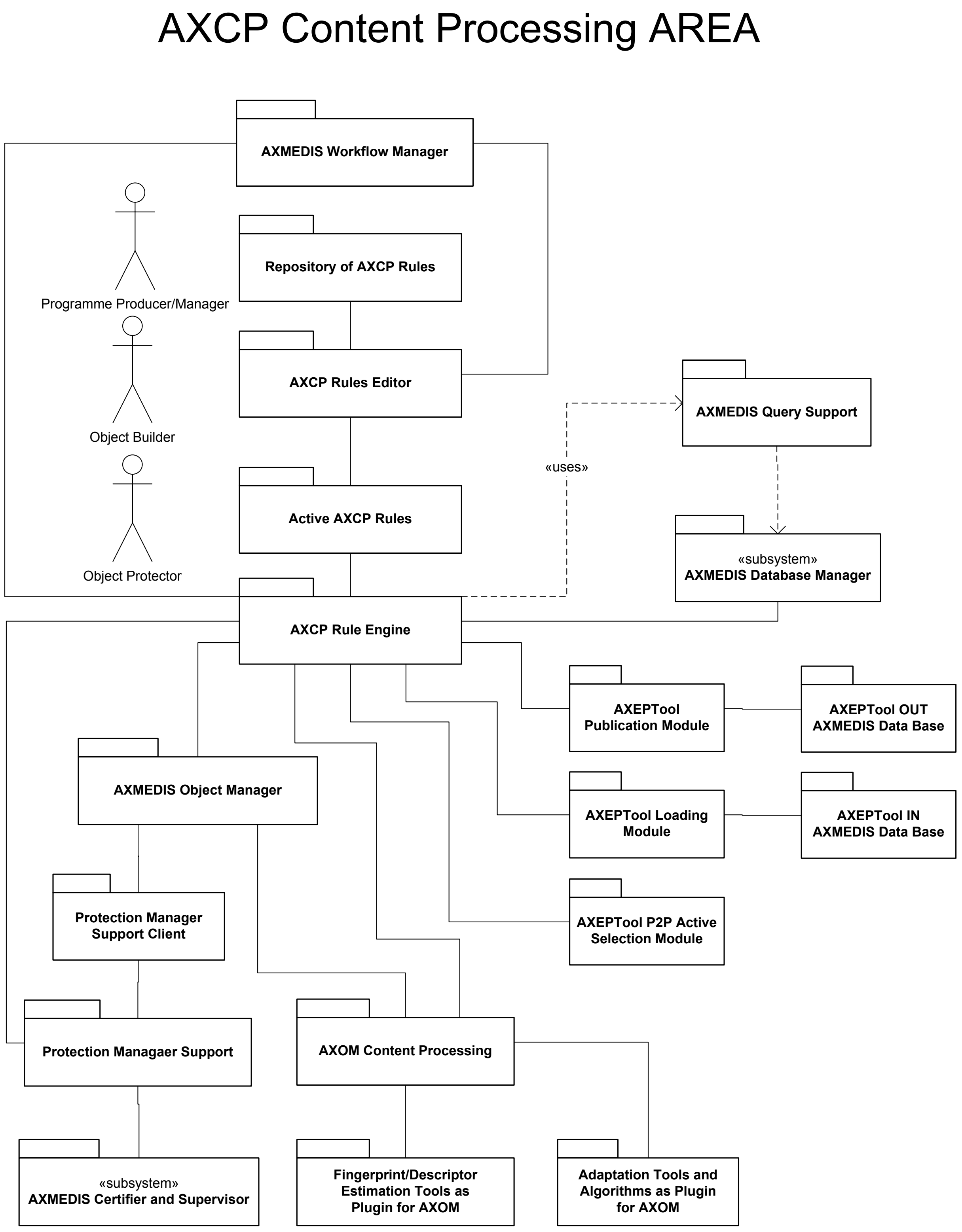
2.5 AXCP Content Processing area UML Decomposition

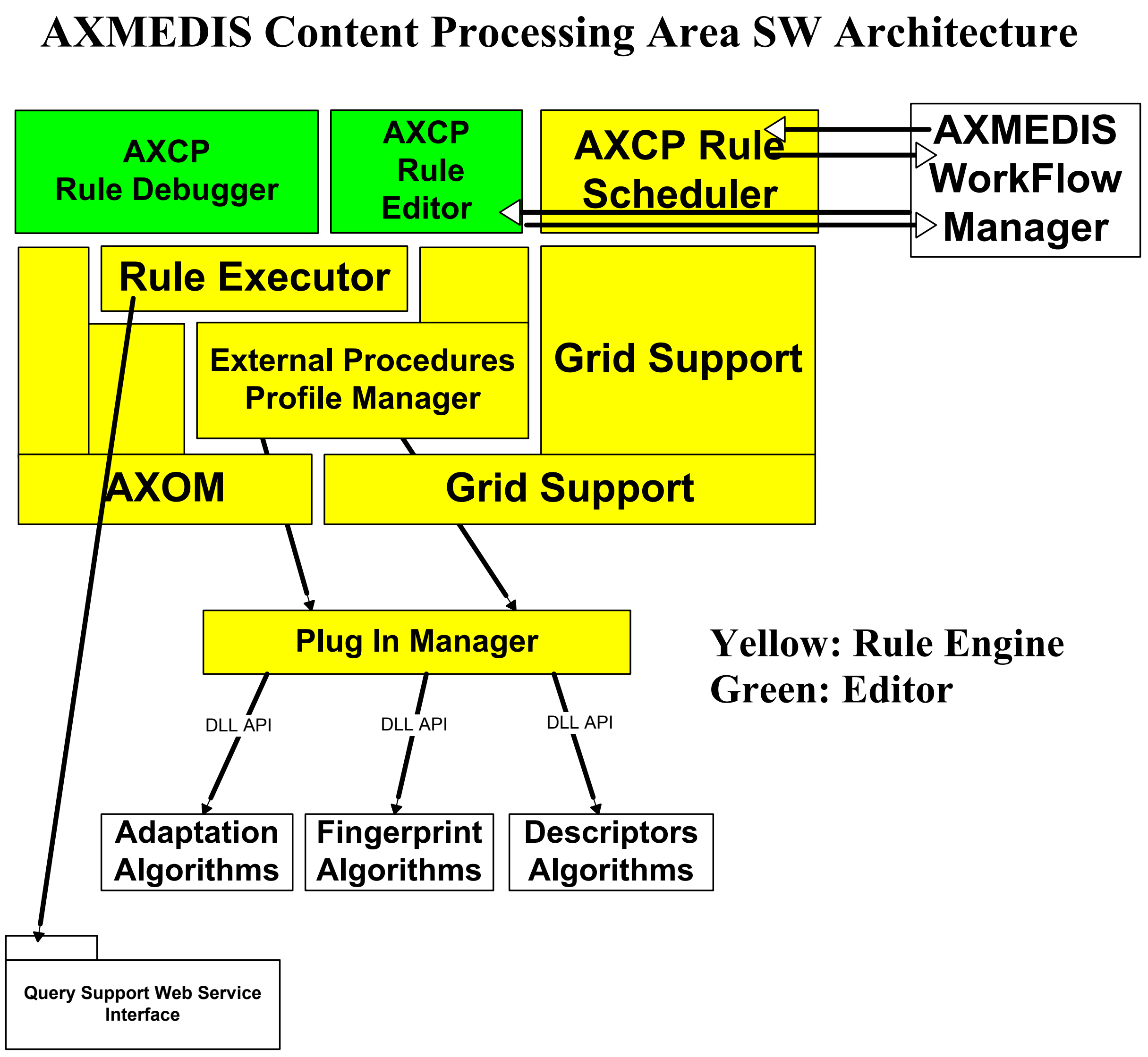
In this section the UML decomposition of the AXMEDIS Content Processing Area is described. It represents the unified solution that integrates the following engines:

- AXMEDIS Compositional/Formatting Engine
- AXEPTool Loading Tool Engine
- AXEPTool Publication Tool Engine
- AXEPTool P2P Active Selection Engine
- The Protection Tool Engine

According to the UML diagram, the AXMEDIS Content Processing Area will include:

- **AXCP Rule Editor**: A graphic editor that will allow writing and editing composition, formatting, protection and publication on AXEPTool rules.
- **Repository of AXCP Rules**: It will be a simple repository of rules; it will be the file systems or a database. The rules will be described by means an XML schema.
- **Active Rules**: They are rules that will be scheduled to be run by the rule engine.
- **AXCP Rule Engine**: It will be the rule executor and will be comprised of a scheduler and a rule engine.
- **AXMEDIS Database Manager** – It will allow the engine to retrieve AXMEDIS objects involved in the execution of a rule.
- **AXMEDIS Query Support** – It will allow the engine to make queries to the AXMEDIS Database Manager
- **AXMEDIS Object Manager (AXOM)** – It will allow managing AXMEDIS objects during the execution of a rule. It will provide functions and methods for manipulating and managing resources and metadata,
- **AXOM Content Processing –** It will based on a Plugin Manager and the External Procedures Profile Manager. It will allow extending the engine capabilities by providing the interface to Plug-ins such as:
  - **Adaptation Tools and Algorithms as Plugin for AXOM**: They will be a collection of algorithms and tools that will provide functions for content adaptation . The role of such component will be to provided different methods to manipulate digital contents in order to satisfy several and different user profile.
  - **Fingerprint/Descriptor Estimation Tools as Plugin for AXOM -** They will be a collection of algorithms and tools that will provide functions for fingerprint/descriptors estimation from digital contents.
- **Protection Manager support –** It will provide the support to manage the protection of AXMEDIS objects.
- **AXEPTool Publication Module** – it will provide the support to manage the publication aspects in the AXEPTool
- **AXEPTool Loading Module -** it will provide the support to manage the publication aspects in the AXEPTool
- **AXEPTool P2P Active Selection Module -** it will provide the support to manage the publication aspects in the AXEPTool


AXCP Content Processing AREA
AXMEDIS Workflow Manager
Programme Producer/Manager
Object Builder
Object Protector
Repository of AXCP Rules
AXCP Rules Editor
Active AXCP Rules
AXCP Rule Engine
«uses»
AXMEDIS Query Support
«subsystem»
AXMEDIS Database Manager
AXEPTool Publication Module
AXEPTool OUT AXMEDIS Data Base
AXEPTool Loading Module
AXEPTool IN AXMEDIS Data Base
AXEPTool P2P Active Selection Module
AXMEDIS Object Manager
Protection Manager Support Client
Protection Managaer Support
AXOM Content Processing
«subsystem»
AXMEDIS Certifier and Supervisor
Fingerprint/Descriptor Estimation Tools as Plugin for AXOM
Adaptation Tools and Algorithms as Plugin for AXOM



2.6 AXCP Rule General Format

The entire production process will be driven by rules called AXMEDIS Content Processing rules (AXCP Rule). They will allow an automatic and customizable process that responds to the distribution and end user needs.

A rule will have to:

- describe what resources are involved in the processing (i.e. extracting digital resources from the AXMEDIS database by means of queries built on metadata and licensing information or from a composite AXMEDIS object);
- describe distribution channel properties, user device features, user profile, etc…
- describe the final output using a specific integration format (MPEG-4, SMIL,…) or using DIP capabilities provided by MPEG-21 objects
- describe how to combine different digital resources and create relationships in terms of:
    - spatial relationships (for graphic layout, resource adaptation, …)
    - time relationships (for synchronisation, transitions effect, fitting (shrinking or stretching, cutting,)…);

- describe how to manage and combine DRM rules for the new formatted resource;
- describe operations or actions that have to be performed during the formatting process, for example:
  - which formatting algorithms have to be used (synchronisation, image scaling, resolution scaling, format conversion, etc…)
  - which external functionalities (by dynamic call to services provided by external tools) have to be used
  - Fingerprint estimation and application for the new composite item
  - Object ID assignment for the new composite item.
- describe how to protect resources

In general, a rule could be formalised as a function in the following way:

$$R = f(S1,S2,..,Sn,P1,…,Pm)$$

Where:

- S_i – It defines a selection. It is a sequence of query to be sent to the Query Support for AXMEDIS objects retrieval or references to digital resource embedded into an composite AXEMDIS object;
- P_i – It is a parameter (basic type as integer, string, Boolean);
- f is the identifier of rule (name of rule or other);
- R is the resultant of rule application. It will be a new AXMEDIS object, or a metadata manipulation, the protection of an AXMEDIS object, etc…

In this section the structure of a rule is described. A rule is constituted of three main sections:

- **Header –** General metadata about the AXCP rule
- **Schedule –** Temporal metadata that describes conditions for firing the AXCP rule
- **Definition –** The definition of the AXCP rule

Header

This section contains metadata related to general information associated with a rule. It is constituted of::

Header				
Data	**Type**	**Values/Format**	**Description**	**Issues**
Rule Name	String	e.g. “Audio Collection”	It defines the name of the rule	
AXRID	String		It defines the AXMEDIS Rule ID	
Rule Version	String	e.g. “1.0”	It defines the version of the rule	
Rule Type	String	AXCP Rule, AXPnP Rule	It defines the type of rule	AXCP rules identifies all rules related to the Content Processing Area, whereas the AXPnP rules are the rule of the P&P area
Software Name	String	“Axmedis Rule Editor”	It specifies the name of software used	
Version of software	String	e.g. “2.0”	It defines the version of software used.	
Date of production	Date	dd.mm.yy	It defines when the rule has been created	
Time of production	Time	hh.mm.ss am/pm	It defines at what time the rule has been created	
Author	String	e.g. “John Brown”	It defines the name of author who has created the rule	

Affiliation	String	e.g. "DSI"	It defines the name of Affiliation	
URL	String	e.g. "http://www...."	It defines the Internet address of the Affiliation	
Comment	String		It allows describing what the rule does	
Last_Modification	Date		Who is last modified	
Terminal_ID	String		The Id of the terminal used to write the rule.	
Cost	Enum		Estimation of Cost	
Work_Item_ID	String		External reference, for instance the commitment	

Schedule
This section contains the sequence of metadata for programming the activation of a rule:

Schedule				
Data	**Type**	**Values/Format**	**Description**	**Issues**
Run	Section		It defines a subsection of metadata that describe information needed for scheduling the execution of the rule.	
Status	String	"Active", "Inactive"	It defines if a rule: • is active and can be executed • is inactive	The list of status identifiers could be extended if it is necessary.

Run				
Data	**Type**	**Values/Format**	**Description**	**Issues**
Start Date	Date	dd.mm.yy	It defines when the rule has to be executed by the engine in terms of day, month and year.	
Start Time	Time	hh.mm.ss am/pm	It defines when the rule has to be executed by the engine in term of time clock.	
Periodicity	String	"Monthly", "Daily", "Weekly", etc…	It defines if a rule has to be executed periodically	Optional
Expiration date	Date	dd.mm.yy	To stop the periodicity	Optional
Expiration time	Time	hh.mm.ss am/pm	To stop the periodicity	Optional

Definition
This section include the *AXCP Rule* section containing the procedural description of the rule.

AXCP Rule				
Data	**Type**	**Values/Format**	**Description**	**Issues**
Arguments	Section		It includes the set of *selections* and *parameters* that rule has in input.	
Preconditions	Section		It includes the list of DLL/functions, Plug In used by the script preconditions	This is information could be used to check the feasibility of rule before running it
Rule Body	Section		It includes the JavaScript code that defines the Rule	

Files Complexity	TBD	TBD	It will specify the complexity of the rule in terms of computational, file transfer parameters, estimated amount of disk space required by the digital resources involved in the rule and other parameters	This will be better defined during the project life

The Arguments subsection contains the list of Selections and Parameters that will be used by the rule.

For each *Selection* see the "XML Selection Schema" in Part E.

Each *Parameter* is defined as following:

Parameter				
Data	**Type**	**Values/Format**	**Description**	**Issues**
Name	String		It specifies the name of the parameter	
Type	String	Integer, String, Float, Double, etc…	It specifies the type of parameter	
Value	string		It specifies the value that the parameter assumes	

The Preconditions subsection contains information about the AXMEDIS Editor Plug In that could be required by the Rule Body. This mechanism is similar to the import directive in JAVA language.

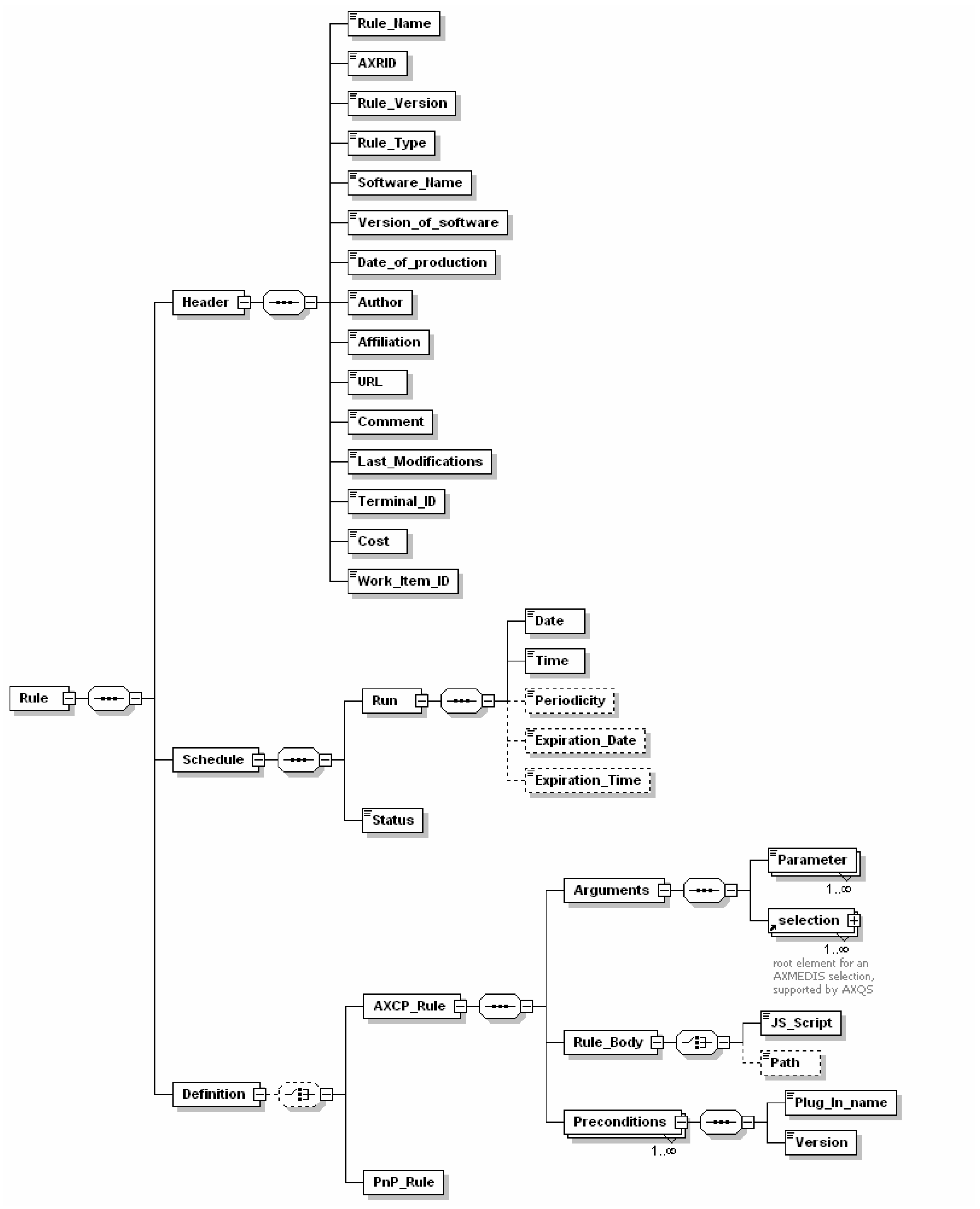
Preconditions				
Data	**Type**	**Values/Format**	**Description**	**Issues**
Plug_In_Name	String	e.g.: "Adaptation"	It provides the name of the AXMEDIS Editor Plug In used by the script. This information has to be matched with that provided by the DLL from its profile.	
Version	String	e.g.: "2.0"	Version of the Plug In. This information has to be matched with that provided by the DLL from its profile.	

The Rule Body section provides two possible ways to refer the adopted script:

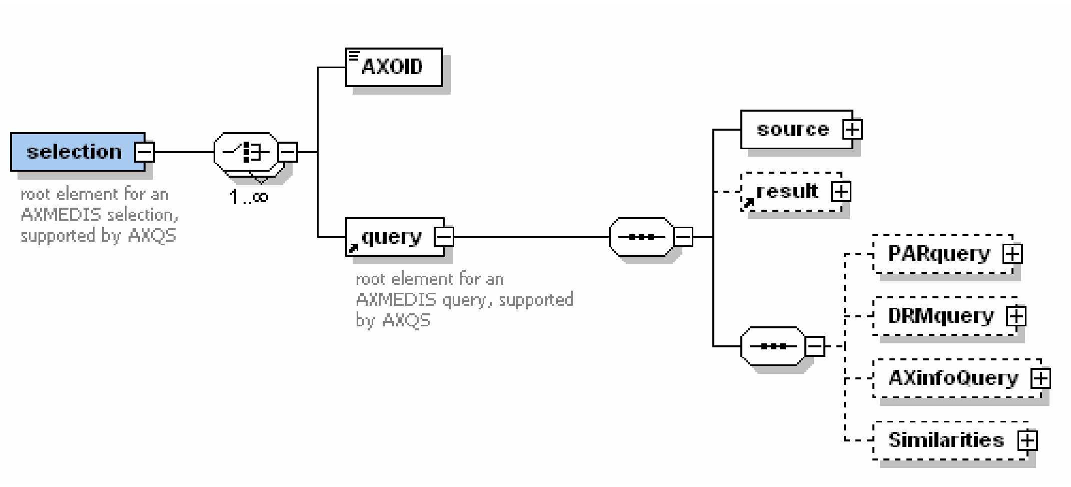
Rule Body				
Data	**Type**	**Values/Format**	**Description**	**Issues**
JS_Script	String		It could be used to embed the whole script (JavaScript code) inside the XML rule format.	Choice
Path	URL		It could be used to specifies reference to a ".js" file that contains the source script of the current rule (JavaScript code).	Optional

2.6.1 AXCP Rule XML formalisation

The set of metadata defined previously could be formalised by means of the following XML Schema:



Where the Selection XML schema will be:


selection
root element for an AXMEDIS selection, supported by AXQS
1..∞
AXOID
query
root element for an AXMEDIS query, supported by AXQS
source
result
PARquery
DRMquery
AXinfoQuery
Similarities

The file associated with the rule is an XML file whose name will be generated as following:

< Rule_Filename > = <Rule_Name> + <Rule_Version> + AXRID + “.xml “

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v2004 rel. 3 U (http://www.xmlspy.com) by Paolo Nesi (University of Florence) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
    <xs:include schemaLocation="Selection-v1-4.xsd"/>
    <xs:element name="Rule">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="Header">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="Rule_Name" type="xs:string"/>
                            <xs:element name="AXRID" type="xs:string"/>
                            <xs:element name="Rule_Version" type="xs:string"/>
                            <xs:element name="Rule_Type">
                                <xs:simpleType>
                                    <xs:restriction base="xs:string">
                                        <xs:enumeration value="Compositional"/>
                                        <xs:enumeration value="Formatting"/>
                                        <xs:enumeration value="Comp_Form"/>
                                        <xs:enumeration value="Prog_Pub"/>
                                    </xs:restriction>
                                </xs:simpleType>
                            </xs:element>
                            <xs:element name="Software_Name" type="xs:string"/>
                            <xs:element name="Version_of_software" type="xs:string"/>
                            <xs:element name="Date_of_production" type="xs:date"/>
                            <xs:element name="Author" type="xs:string"/>
                            <xs:element name="Affiliation" type="xs:string"/>
                            <xs:element name="URL" type="xs:anyURI"/>
                            <xs:element name="Comment" type="xs:string"/>
                            <xs:element name="Last_Modifications" type="xs:date"/>
                            <xs:element name="Terminal_ID" type="xs:ID"/>
                            <xs:element name="Cost" type="xs:string"/>
                            <xs:element name="Work_Item_ID" type="xs:string"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
                <xs:element name="Schedule">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="Run">
                                <xs:complexType>
                                    <xs:sequence>
                                        <xs:element name="Date" type="xs:date"/>
                                        <xs:element name="Time" type="xs:time"/>
                                        <xs:element name="Periodicity" minOccurs="0">
                                            <xs:simpleType>
                                                <xs:restriction base="xs:string">
                                                    <xs:enumeration value="Daily"/>
                                                    <xs:enumeration value="Weekly"/>
                                                    <xs:enumeration value="Monthly"/>
                                                    <xs:enumeration value="Yearly"/>
                                                </xs:restriction>
                                            </xs:simpleType>
                                        </xs:element>
                                        <xs:element name="Expiration_Date" type="xs:date" minOccurs="0"/>
                                        <xs:element name="Expiration_Time" type="xs:time" minOccurs="0"/>
                                    </xs:sequence>
                                </xs:complexType>
                            </xs:element>
                            <xs:element name="Status">
                                <xs:simpleType>
                                    <xs:restriction base="xs:string">
                                        <xs:enumeration value="Active"/>
                                        <xs:enumeration value="Inactive"/>
                                    </xs:restriction>
                                </xs:simpleType>
                            </xs:element>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
                <xs:element name="Definition">
                    <xs:complexType>
                        <xs:choice minOccurs="0">
                            <xs:element name="AXCP_Rule">
                                <xs:complexType>
                                    <xs:sequence>
                                        <xs:element name="Arguments">
                                                <xs:complexType>
                                                    <xs:sequence>
                                                        <xs:element name="Parameter" type="xs:string"
maxOccurs="unbounded"/>
                                                        <xs:element ref="selection" maxOccurs="unbounded"/>
                                                    </xs:sequence>
                                                </xs:complexType>
                                        </xs:element>
                                        <xs:element name="Rule_Body">
                                                <xs:complexType>
                                                    <xs:choice>
                                                        <xs:element name="JS_Script" type="xs:string"/>
                                                        <xs:element name="Path" type="xs:anyURI" minOccurs="0"/>
                                                    </xs:choice>
                                                </xs:complexType>
                                        </xs:element>
                                        <xs:element name="Preconditions" maxOccurs="unbounded">
                                                <xs:complexType>
                                                    <xs:sequence>
                                                        <xs:element name="Plug_In_name" type="xs:string"/>
                                                        <xs:element name="Version" type="xs:time"/>
                                                    </xs:sequence>
                                                </xs:complexType>
                                        </xs:element>
                                    </xs:sequence>
                                </xs:complexType>
                            </xs:element>
                            <xs:element name="PnP_Rule"/>
                        </xs:choice>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

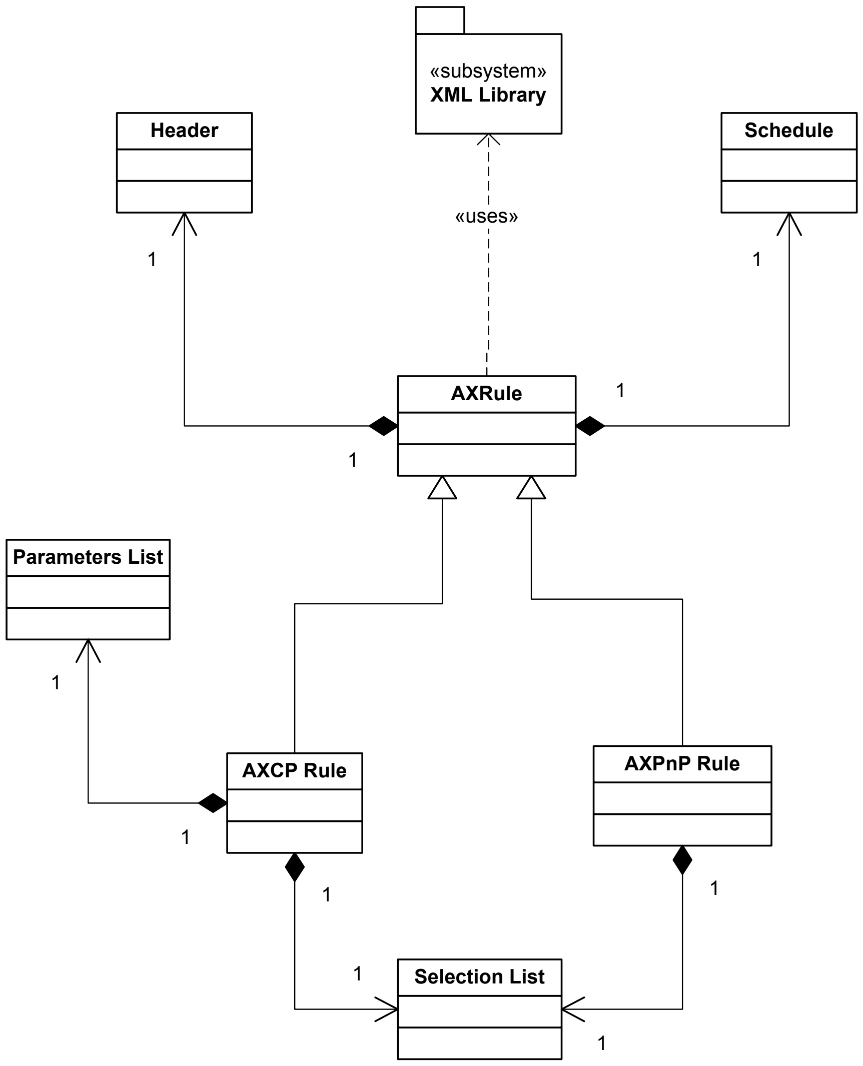
The following example shows the XML structure for a generic rule:

```
<Rule>
        <Header>
                <Rule_Name> Audio Collection </Rule_Name>
                <AXRID> 0010001 </AXRID>
                <Rule_Version> 1.0 </Rule_Version>
                <Rule_Type> Compositional </Rule_Type>
                <Software_Name> Axmedis Rule Editor </Software_Name>
                <Version_of_software> 2.0 </Version_of_software>
                <Date_of_production> 24.12.2004 </Date_of_production>
                <Time_of_production> 12:00 am </Time_of_production>
                <Author> John Brown </Author>
                <Affiliation> DSI </Affiliation>
                <URL> http://www.dsi.unifi.it </URL>
                <Comment> This rule embeds only audio file </Comment>
                <Last_Modifications> 25.12.2004 </Last_Modifications>
                <Terminal_ID> </Terminal_ID>
                <Cost> What is cost? </Cost>
                <Work_Item_ID> What? </Work_Item_ID>
        </Header>
        <Schedule>
                <Run>
                        <Date> 24.12.2004 <Date>
                        <Time> 12:00 pm </Time>
                        <Periodicity> Weekly </Periodicity>
                        <Expiration_Date> 01.01.2005 </Expiration_Date>
                        <Expiration_Time> 12:00 pm </Expiration_Time>
                 </Run>
                <Status> Active </Status>
        </Schedule>
        <Definition>
                <AXCP_Rule>
                        <Arguments>
                                <selection name="TEST" timestamp="2005-01-20T18:20:46.275+01:00">
                                        <AXOID>3y7932469236</AXOID>
                                        <AXOID>824375832741723</AXOID>
                                        <query>
                                                <source>
                                                        <location>CRAWLER</location>
                                                </source>
                                                <AXinfoQuery>
                                                        <querycondition>
                                                                <nesting>
                                                                        <test>
                                                                                <field>AUTHOR</field>

                                        <operator>STARTWITH</operator>
                                                                                <value>MOZ</value>
                                                                        </test>
                                                                </nesting>
                                                        </querycondition>
                                                </AXinfoQuery>
                                        </query>
                                </selection>
                                <Parameter name="count" type = "integer" > 20 </ Parameter >
                                ……
                                <Parameter>  ……</ Parameter >
                        </ Arguments>
                        <Preconditions>
                                <Plug_In_Name> Adaptation </Plug_In_Name >
                                <Version> 2.0 </Version>
                        </Preconditions>
                        <Preconditions>
                                ………….
                        </Preconditions>
                        <Rule_Body>
                                   …
                        </Rule_Body>
                </AXCP_Rule>
        </Definition>
</Rule>
```

2.6.2 AXCP Rule Class Diagram

According to the XML schema, the data model of the rule will be realised by means of the following class diagram:



The *AXRule* is the main class for a rule. It will encapsulate a *Header* and a *Schedule* class. They will model respectively the Header and Schedule section according to Rule XML Schema.
The AXRule class will be the common class for the AXCP and AXPnP rules. Each of them will define the *Definition* section according to Rule XML Schema and then specialise the AXRule.

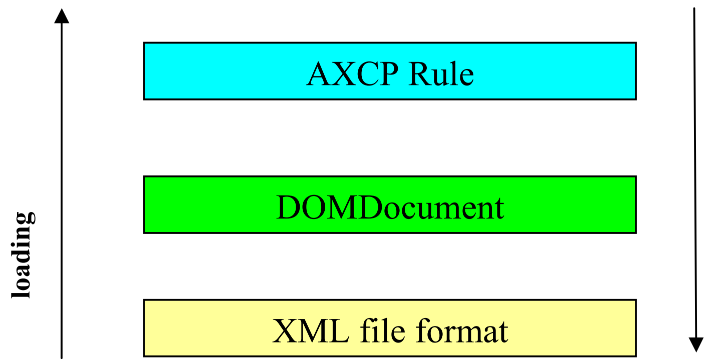
2.6.3 AXRule Loader and Saver Modules (DSI)

To manage the repository of rules it is necessary to have the possibility to load and save a rule in/from the system. For this purpose, the following modules respond to such requirement.
AXRule Loader - It is the module for loading an XML representation of the rule in the AXCP Rule Editor and AXCP Rule Engine. It will work according to the XML rule specification and it will be based on an XML library for managing metadata. It provides the following functionalities:

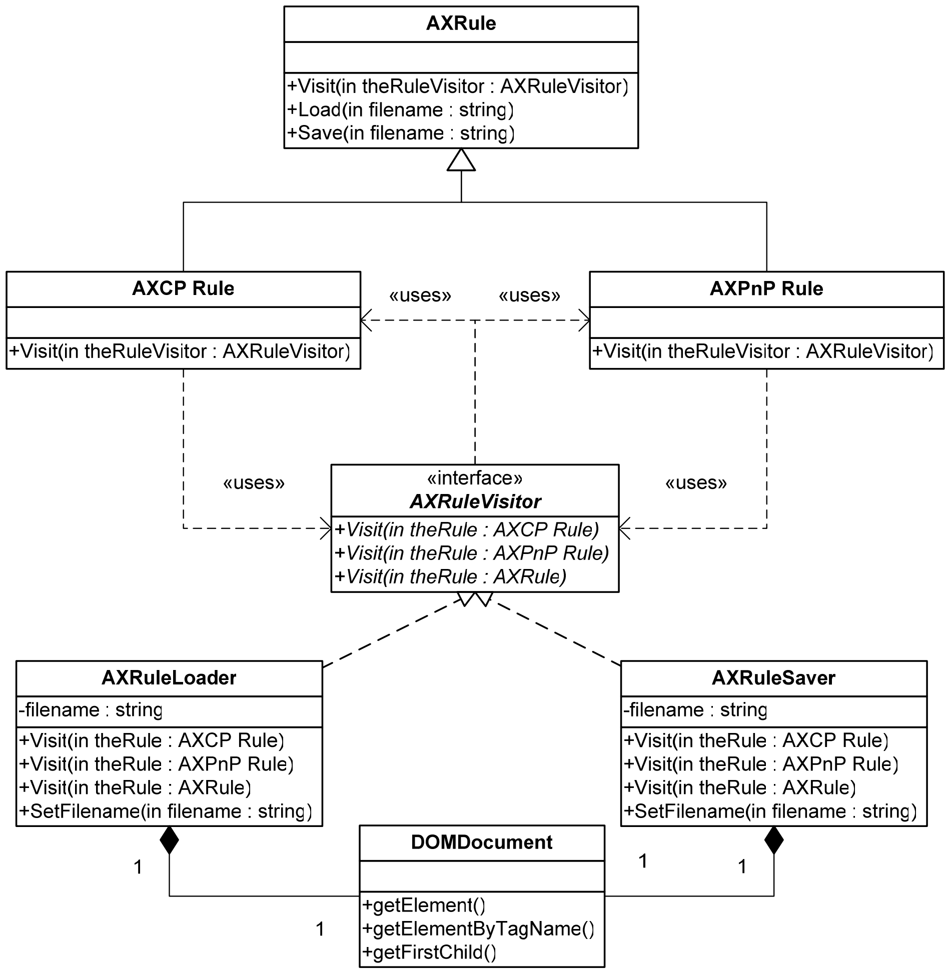
- Load the XML file of the AXCP rule from disk and generates an AXCP memory representation of rule (AXCP Rule object)

AXRule Saver – It is the module for saving an XML representation of the rule on disk. It will work according to the XML rule specification. It provides the following functionalities:

- Save the XML representation of the rule by replacing the existing one
- Save as function for saving the XML representation of the rule with a name



Both modules will be implemented by using an abstract class called *AXRuleVisitor*. This solution will allow building an *AXRuleLoader* and an *AXRuleSaver* class that could manage different types of rules by implementing different *Visit* methods (see the class diagram reported below). Both classes are related to a DOM in order to perform the necessary read/write operations on an XML file. The XML representation of a rule is stored in the *DOMDocument* class from which it is possible to build the memory representation of the AXCP rule. The *AXRule* class will have a *Load* and *Save* method and a virtual method *Visit* that will have to be redefined in the *AXCP Rule* class. In this way, the *Visit* method of AXCP Rule will call the *Visit* method of *AXRuleLoader* on the AXCP Rule object by using the *this* pointer. In the following picture, the class diagram for the Rule Loader and Saver modules is reported:


AXRule
+Visit(in theRuleVisitor : AXRuleVisitor)
+Load(in filename : string)
+Save(in filename : string)
AXCP Rule
+Visit(in theRuleVisitor : AXRuleVisitor)
«uses»
«uses»
AXPnP Rule
+Visit(in theRuleVisitor : AXRuleVisitor)
«uses»
«interface»
AXRuleVisitor
+Visit(in theRule : AXCP Rule)
+Visit(in theRule : AXPnP Rule)
+Visit(in theRule : AXRule)
«uses»
AXRuleLoader
-filename : string
+Visit(in theRule : AXCP Rule)
+Visit(in theRule : AXPnP Rule)
+Visit(in theRule : AXRule)
+SetFilename(in filename : string)
AXRuleSaver
-filename : string
+Visit(in theRule : AXCP Rule)
+Visit(in theRule : AXPnP Rule)
+Visit(in theRule : AXRule)
+SetFilename(in filename : string)
DOMDocument
+getElement()
+getElementByTagName()
+getFirstChild()
1
1
1
1

The adopted solution can be used to implement a loader and saver also for the AXPnP rule.

2.7 AXCP Rule Editor (DSI)

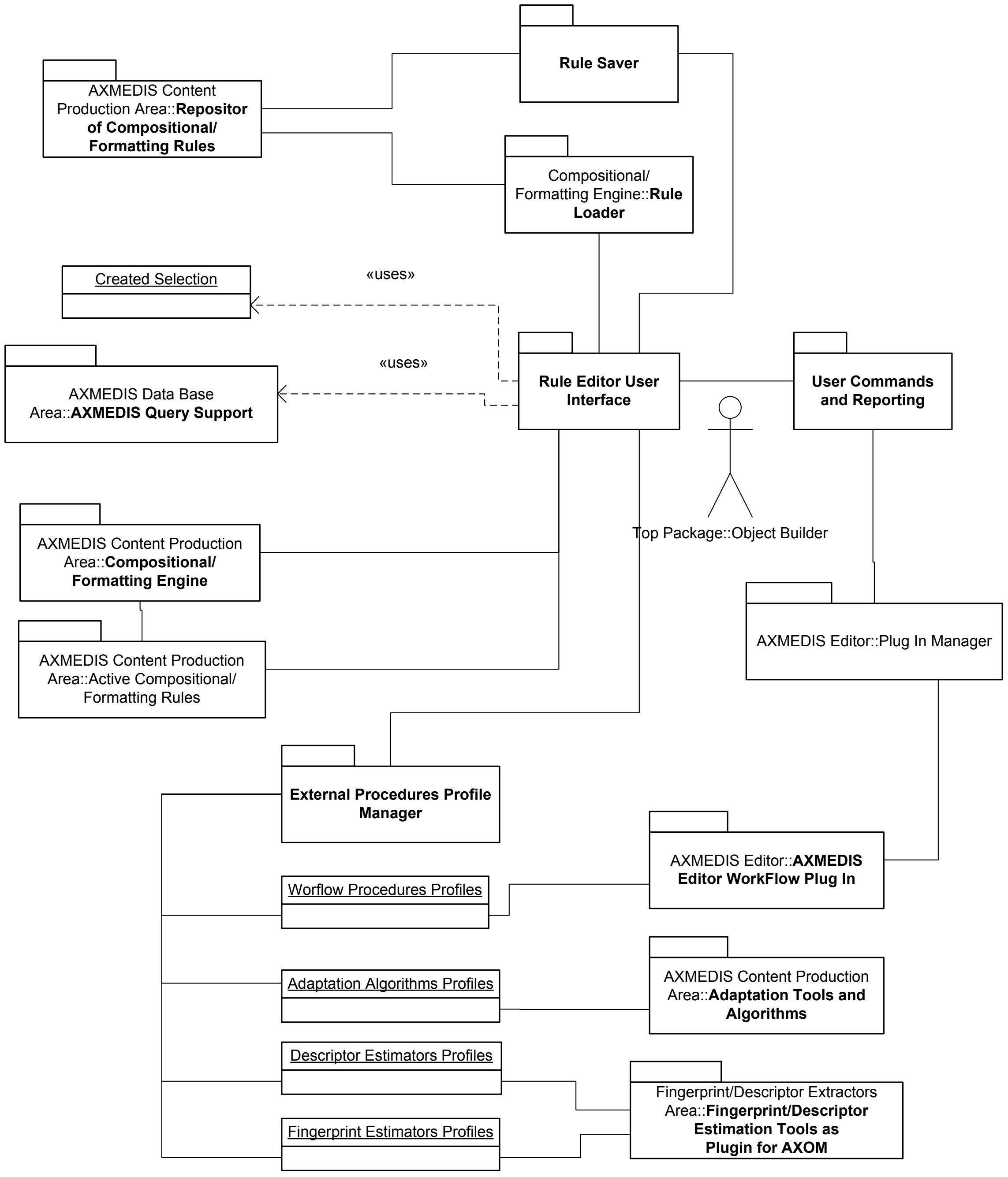
Module Profile		
AXCP Rule Editor		
Executable or Library(Support)	Executable	
Single Thread or Multithread	Multi-Thread	
Language of Development	C++	
Responsible Name	Ivan Bruno	
Responsible Partner	DSI	
Status (proposed/approved)	Proposed	
Platforms supported	Microsoft Windows, Linux, MACOS X	
Interfaces with other tools:	Name of the communicating tools	Communication model and format (protected or not, etc.)
AXMEDIS Workflow manager	Command and reporting	Web Service
AXCP Rule Engine		
File Formats Used	Shared with	File format name or reference to a section
Config Files		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
Views Manager	C++	wxWidgets 2.4.2 C++ library
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not
wxWidgets 2.4.2	wxWidgets 2.4.2 C++ library	LGPL
XERCES-C++	XERCES 2.6.0	LGPL
FL C++ Lib contribution to wxWidgets Lib		LGPL
STC C++ based on Scintilla editor, contribution to the wxWidgets Lib		LGPL

The AXCP Rule Editor is the tool that will permit to obtain well formed rules by the means of both graphic and scripting tools. The AXCP Rule Editor will be launched by the Workflow Web User Interface when the user/Rule producer will click on a specified button in the Web page. During the launching an incomplete Rule XML will be sent to the AXCP Rule Editor. Some information could be already available in the header of rule (rule type, AXRID, comments etc…) and in the schedule. The Rule Editor will be able to interact with the Rule Executor derived from the AXCP Rule Engine in order to perform debugging and feasibility check on rules. The main capabilities provided by the editor will be:

- AXCP rule editing
- AXCP Rule Activation
- Internal textual editor for writing the script code
- Debugging support to test and verify the script
- Communication support with the AXCP Rule Engine
- Communication support with the AXMEDIS Workflow Manager

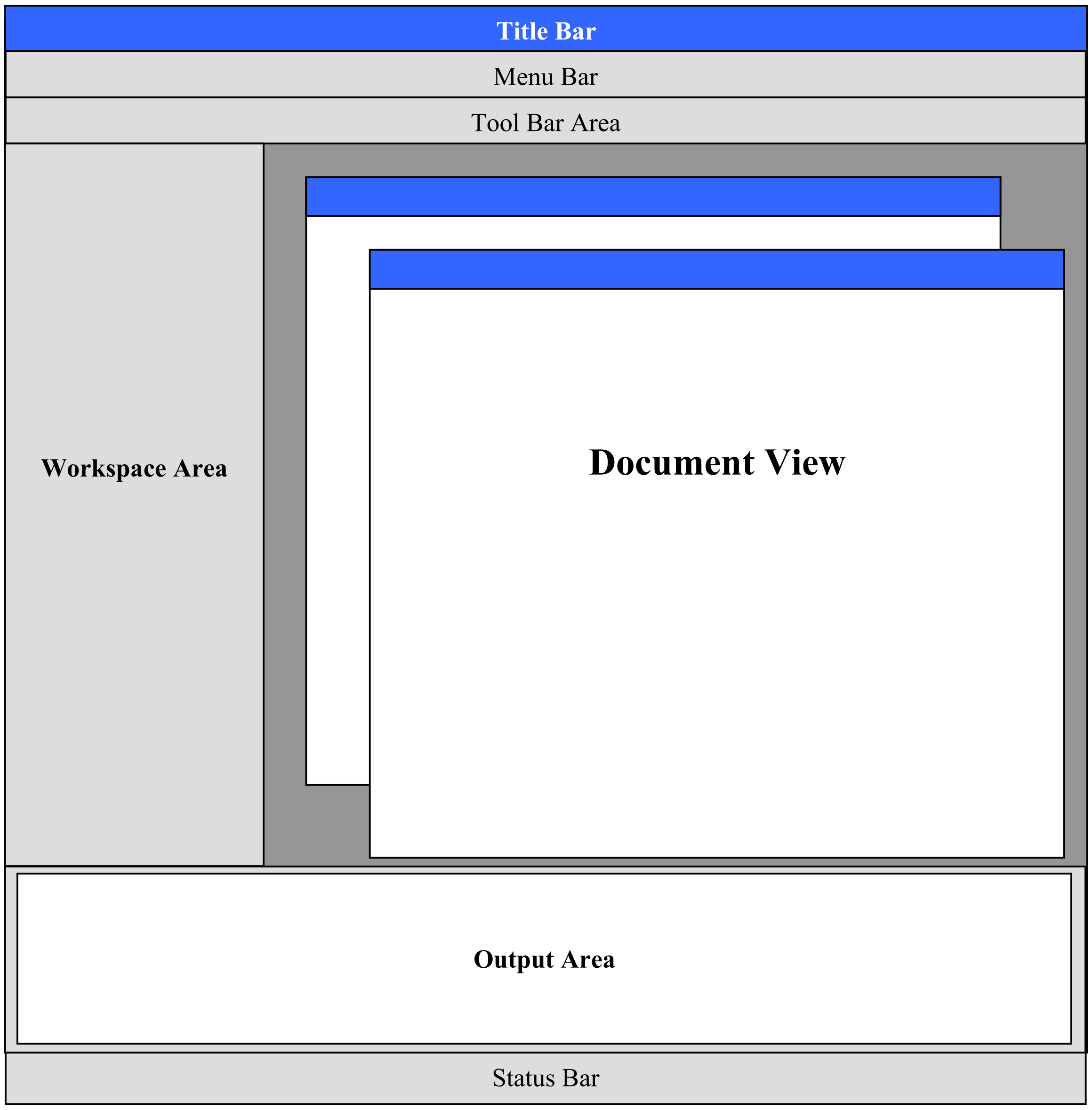
The specific components and aspects regarding protection and DRM can be found in section 5.2 of part H of the AXMEDIS framework specification "Protection Rule Editor".

Compositional/Formatting Rule Editor
To be taken as an example of others Rule/Selection EDITOR

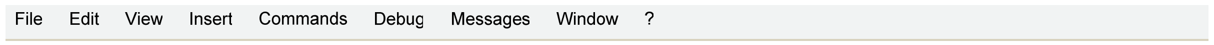

Rule Saver
AXMEDIS Content Production Area::Repositor of Compositional/ Formatting Rules
Compositional/ Formatting Engine::Rule Loader
Created Selection
«uses»
«uses»
AXMEDIS Data Base Area::AXMEDIS Query Support
Rule Editor User Interface
User Commands and Reporting
Top Package::Object Builder
AXMEDIS Content Production Area::Compositional/ Formatting Engine
AXMEDIS Content Production Area::Active Compositional/ Formatting Rules
AXMEDIS Editor::Plug In Manager
External Procedures Profile Manager
AXMEDIS Editor::AXMEDIS Editor WorkFlow Plug In
Worflow Procedures Profiles
Adaptation Algorithms Profiles
AXMEDIS Content Production Area::Adaptation Tools and Algorithms
Descriptor Estimators Profiles
Fingerprint Estimators Profiles
Fingerprint/Descriptor Extractors Area::Fingerprint/Descriptor Estimation Tools as Plugin for AXOM

2.7.1 AXCP Rule Editor User Interface (DSI)
The AXCP Rule Editor will be a multithread application and will be developed by using the wxWidgets ver. 2.4.2 library. This should allow having a multiplatform rule scheduler.

The AXCP Rule Editor GUI will be a MDI window that will manage a rule document. It will be developed to provide a set of tools and views to help the user during the editing and building of rule. It will host an instance of the rule executor in order to provide functionalities for debugging, testing and validating the script code associated with a rule. To help the user in writing rule, the editor will be equipped with an Help on line and area where the user will be able to access to a library of script functions. The main architecture of GUI will be based on the following structure:



The Menu Bar
The menu bar will be constituted of the following main entries:



File

- **New** – create a new rule document
- **Open** – Open a document in the Rule Editor (rules and java script)
- **Close** – Close the current rule document
- **Save** – save the current rule using the current file name
- **Save as** – save the current rule by name
- **Recent Files** – History of files
- **Exit** – Quit the editor

Edit

- **Copy** – copy a selection in the clipboard
- **Paste** – paste a selection available in the clipboard
- **Cut** – delete and copy a selection in the clipboard
- **Delete** – delete a selection
- **Find…** – Search a word in the text
- **Replace…** – replace a word with another
- **Select All** – select all content
- **Go to…** – go to a specific line text

View

- **Workspace** – It opens the Workspace area
- **Output –** It opens the Output area

Insert

- **Selection –** Adds a selection item in the rule
- **Parameter –** Adds a parameter item in the rule
- **Schedule –** Adds a schedule item in the rule
- **Script -** Adds a script item in the rule
- **Precondition -** Adds a precondition item in the rule

Commands

- **Activate Rule** – It is the activate rule command and will allow sending the current rule to the scheduler and the notification to the AXMEDIS Workflow Manager. A connection with the Rule Engine Scheduler will be open in order to perform the installation of rule in the Scheduler.
- **Check rule** – Tests the feasibility of the rule (like a compiler plus some tests on AXMEDIS objects and estimation of some parameters such as the files complexity and required workload)
- **Rules List…** – Shows the list of rules inside the repository of the Rule Editor
- **Find Rule…** – Allows making queries to the rules repository of the Rule Editor

Debug

- **Run** – Enter in the debug mode or if the script is stopped, continue execution until the script is finished, or a breakpoint is reached.
- **Stop** – Stop the script execution and close the debug mode
- **Step -** Executes the current line of the script, then pauses. This differs from the "Trace" command in that it will not step into functions and scripts that are called by the current line.
- **Trace** – Executes the current line of the script, then pauses. This differs from the "Step" command in that if the current line calls a function, or another script, the debugger will trace into the called function or script.

- **Skip -** Skips over current line without executing it. The script will resume execution on the subsequent line.
- **Watches –** It will open the table of current variables showing name and values
- **Stack –** It will open the stack of functions
- **Set Breakpoint** – Set a breakpoint on the currently selected line of the script code. Every time the selected line is reached, the debugger will stop
- **Remove Breakpoint -** Clear a breakpoint from the currently selected line of the script code.
- **Remove All Breakpoint -** Clear a all breakpoint in the script code.

Messages

- **Last message –** Displays the last message sent by the AXMEDIS Workflow Manager
- **Messages List -** Displays the list of messages sent by the AXMEDIS Workflow Manager

Window (provided automatically by the MDI GUI)

- **Cascade**
- **Tile Horizontal**
- **Tile Vertical**
- **Next –** Activate the next document view
- **Previous -** Activate the previous document view
- **Arrange Icons –** Arrange the all minimised document views
- **Close All –** Close all document views
- **Windows list**

?

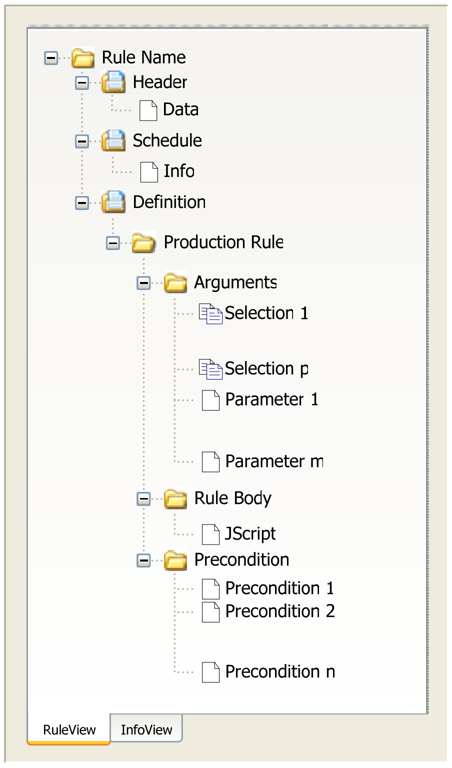
- **Help** – Call the on line help
- **About** – Information about the authors, version, etc

ToolBar Area
The toolbar area will host a set of icon buttons that will allow calling functions without accessing to the menu bar. The toolbar area will be based on dockable toolbars and will allow the dynamic customisation by adding or removing sub-toolbars. For this end the editor will provide sub-toolbars for:

- *Standard* – it will provide main functionalities for managing rule files and editing (new, open, save, etc…)
- *Debug* – it will provide main controls
- *Messages* – it will provide the access to the message list and the last message
- *Commands* – it will provide main commands

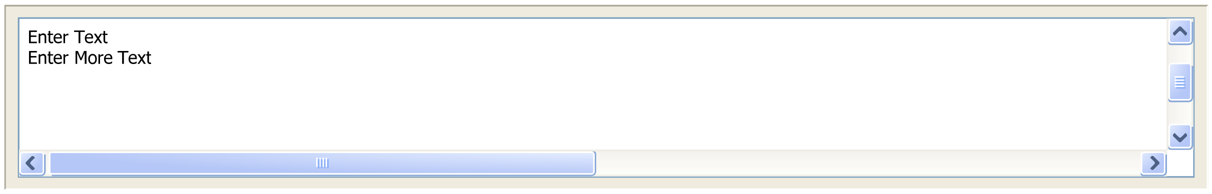
Workspace Area
This will be a resizable panel and will include a notebook control constituted of the following view items:

1. **Rule View –** In this area the structure of rule is displayed. It will be realised by using a Tree control that will permit to show and browse components according the rule XML schema.
   A dynamic popup menu will be available for a quick access to functions that will allow the quick management of items (edit and view metadata, delete,…). Appropriate icons will be also drawn in order to identify intuitively components of rule in the tree control view. In the following picture the structure of the Rule View area is depicted:



2. **Info View –** It will be as an on-line book that could be used as help by the user**.** It will display the set of functionalities provided by the Plugins installed and automatically detected by the editor. It will be realised by using a Tree control that will permit to show and browse plugins module and the functionalities that they provide according to their profile. The profile will be used to build on the fly an html or txt documentation page. The user will be able to see the documentation associated with each selected function by interacting with each item of the tree. To this end a contextual popup menu will be designed. The selected documentation will be displayed in the **Text/Html document view.**

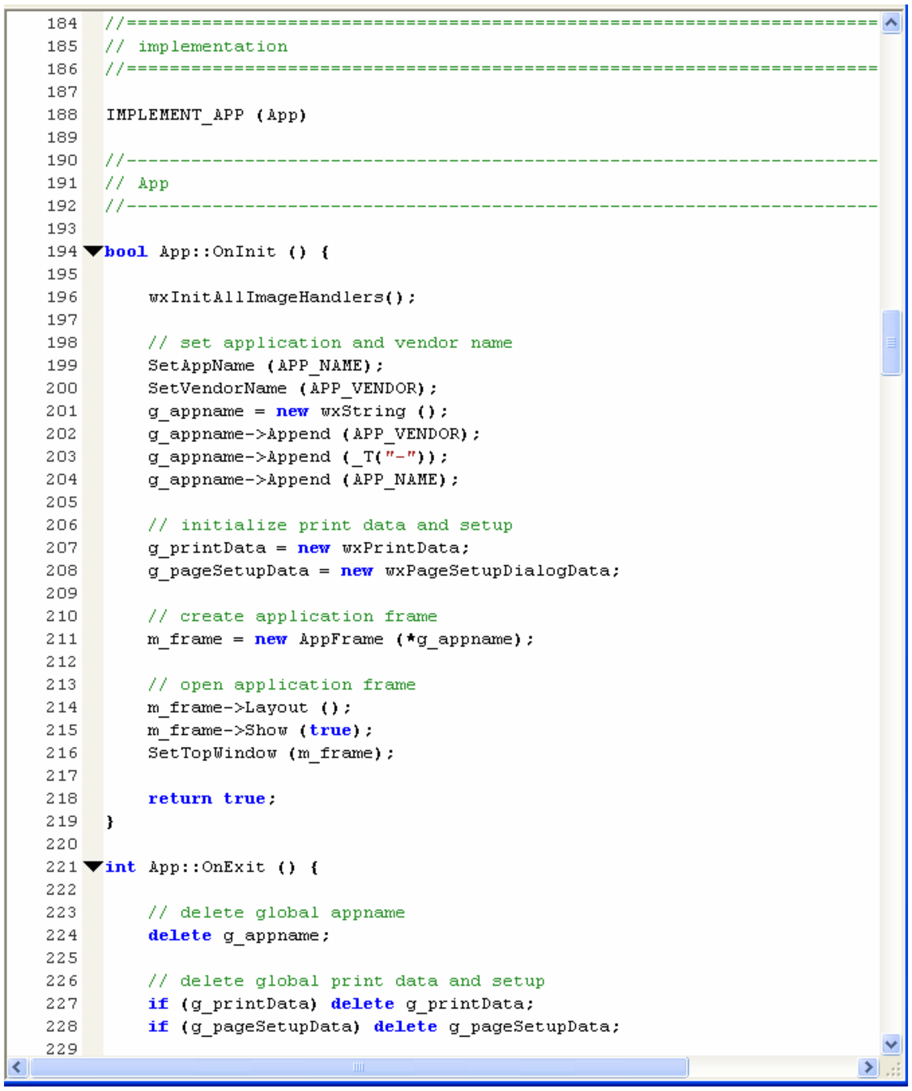
Output Area
This is a text control where messages, textual description, errors, debugging info, alert, etc… will be displayed.


Enter Text
Enter More Text

Tools, Viewers and Editors
Some different types of tools and editor will be designed for visualizing and/or editing different type of documents.

1. **Textual script editing window** – This is the window client where the user will be able to write the script code. It will be based on a multiline text control where it will be possible to edit the script. The textual editor will support some facilities such as:
   - Auto completion of words - a window listing possible completions for strings the user has typed
   - Syntax highlighting – keywords will be colourised
   - Brace highlighting

- Folding/Hiding - making lines invisible or visible. It shows or hides a range of lines.
- Multiple views - to have multiple views of the same Document. (Split view)
- Breakpoint insertion/removal – to control the code in the debugging session
- Visualisation of line numbers



```
//=====================================================================
// implementation
//=====================================================================

IMPLEMENT_APP (App)

//---------------------------------------------------------------------
// App
//---------------------------------------------------------------------

bool App::OnInit () {

    wxInitAllImageHandlers();

    // set application and vendor name
    SetAppName (APP_NAME);
    SetVendorName (APP_VENDOR);
    g_appname = new wxString ();
    g_appname->Append (APP_VENDOR);
    g_appname->Append (_T("-"));
    g_appname->Append (APP_NAME);

    // initialize print data and setup
    g_printData = new wxPrintData;
    g_pageSetupData = new wxPageSetupDialogData;

    // create application frame
    m_frame = new AppFrame (*g_appname);

    // open application frame
    m_frame->Layout ();
    m_frame->Show (true);
    SetTopWindow (m_frame);

    return true;
}

int App::OnExit () {

    // delete global appname
    delete g_appname;

    // delete global print data and setup
    if (g_printData) delete g_printData;
    if (g_pageSetupData) delete g_pageSetupData;

```

2. **Text/Html document view –** This is the window for the visualisation of the documentation provided by the help on line. It will be opened when the user will make double click on a voice of the index in the *Info view* or when the internal help is called. It will provide functionalities for browsing TXT or HTML pages. For example, all the information related to the description of a function selected from the *Info view* will be shown in such window.

3. **Selection Editor -** It will be an interactive html page that will be displayed by means the HTML document viewer. It will provide functionality for:
   a. Edit a selection
   b. Save/Load a selection
   c. Actualise the selection

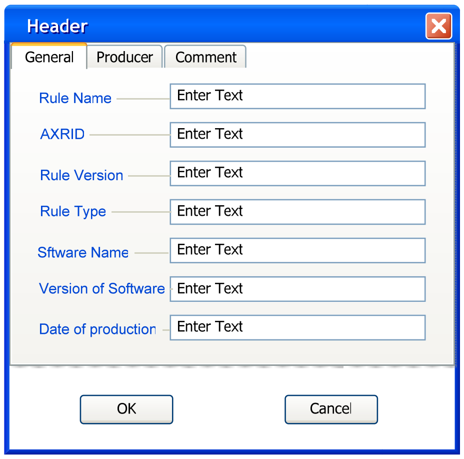
For more details about the Selection editing see the DE3.1.2E Framework and Tools Specifications (Database and Gathering)

4. Mapper Editor [CRS4]

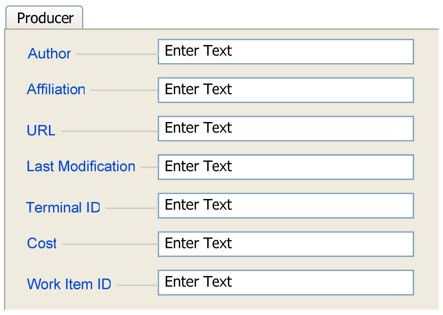
Mapper editor GUI will be an editor for Metadata Mapping AXEPTool. It will be the interface used to creates the map for incoming metadata translation. The GUI allows the user to decide which origin fields have to be converted in destination fields. The GUI will be invoked by the AXCP Editor.
For more details see document DE3.1.2F Framework and Tools Specifications (AXEPTool and Progr. and Pub.).

Interactive Dialogs
The editor will provide a set of dialogs for facilitate the editing of a rule. To this end, the following dialogs will be designed:

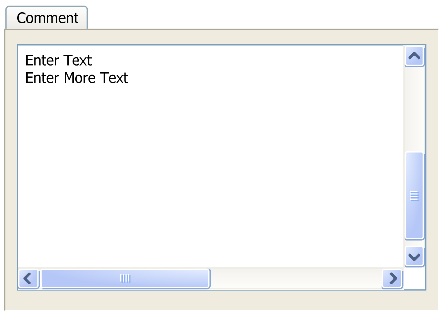
1. **Header Rule Edit Dialog –** This is the dialog that will allow to fill fields of the header section. The dialog will be designed as an OK/Cancel modal dialog in a notebook style with General, Producer and Comment tab where the list of items to edit.will be displayed.



Rule Name	Enter Text
AXRID	Enter Text
Rule Version	Enter Text
Rule Type	Enter Text
Sftware Name	Enter Text
Version of Software	Enter Text
Date of production	Enter Text

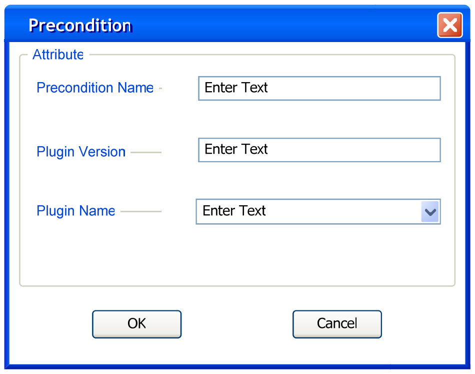


Author	Enter Text
Affiliation	Enter Text
URL	Enter Text
Last Modification	Enter Text
Terminal ID	Enter Text
Cost	Enter Text
Work Item ID	Enter Text



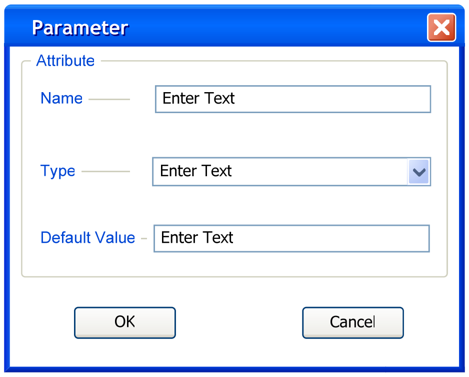
Enter Text
Enter More Text

2. **Precondition Edit Dialog -** This will be a dialog that will allow to fill fields for a precondition item. The dialog will be designed as an OK/Cancel modal dialog and will display the list of items to edit. The dialog will show the list of plugin installed in order to facilitate the choice.



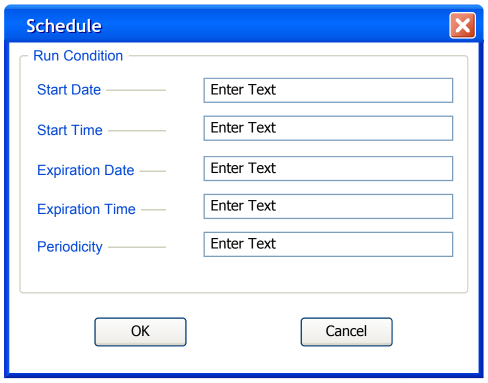
Precondition Name	Enter Text
Plugin Version	Enter Text
Plugin Name	Enter Text

3. **Parameter Edit Dialog -** This will be a dialog that will allow editing/filling fields for a parameter item. The dialog will be designed as an OK/Cancel modal dialog and will display the list of items to edit.



Name	Enter Text
Type	Enter Text
Default Value	Enter Text

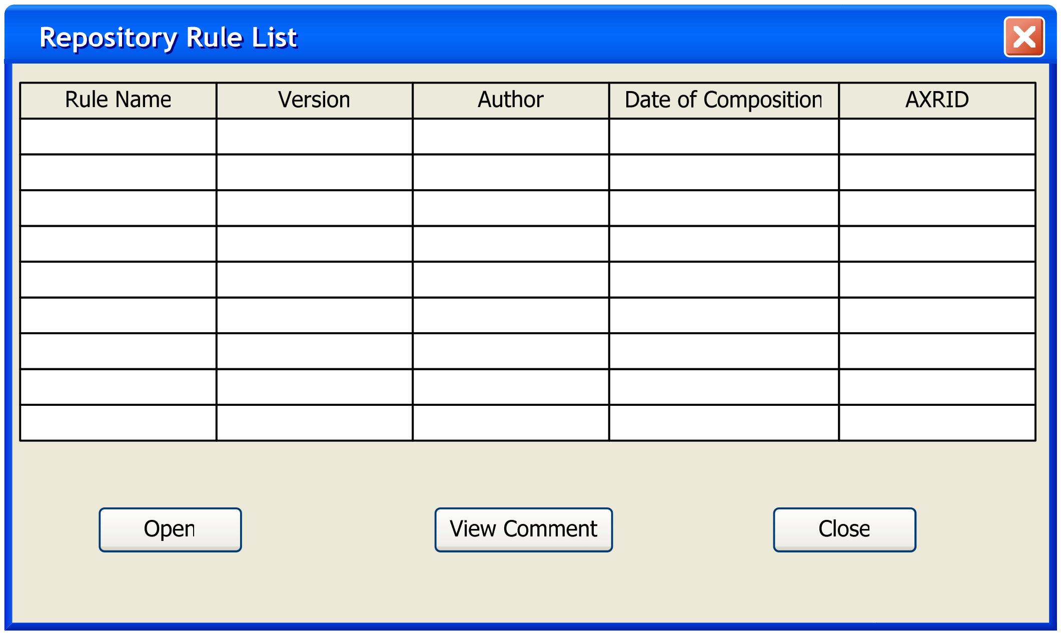
4. **Schedule Edit Dialog –** This will be a dialog that will allow filling/editing fields for a schedule item. The dialog will be designed as an OK/Cancel modal dialog and will display the list of items to edit.



Start Date	Enter Text
Start Time	Enter Text
Expiration Date	Enter Text
Expiration Time	Enter Text
Periodicity	Enter Text

5. **Repository Rule List Dialog -** The *Rule List* command will open a rules list modal dialog displaying all rules stored in the repository of the AXCP Rule editor. In this window, the list of rules will be organised in a table built on the following subset of metadata:
   - *Rule Name*
   - *Rule Version*
   - *Author*
   - *Date of composition*
   - *Rule ID (AXRID)*

The user will be able to select a specific rule in order to open it in the rule editor. Such operation will be possible by pushing the *Open* button or double clicking on the line of the chosen rule. The user will be able to visualize the comment associated with rule by pushing the *View Comment* button, the comment will be displayed the *Output Area*. Otherwise the user will be able to cancel the operation by closing the dialog or pushing the *Close* button.



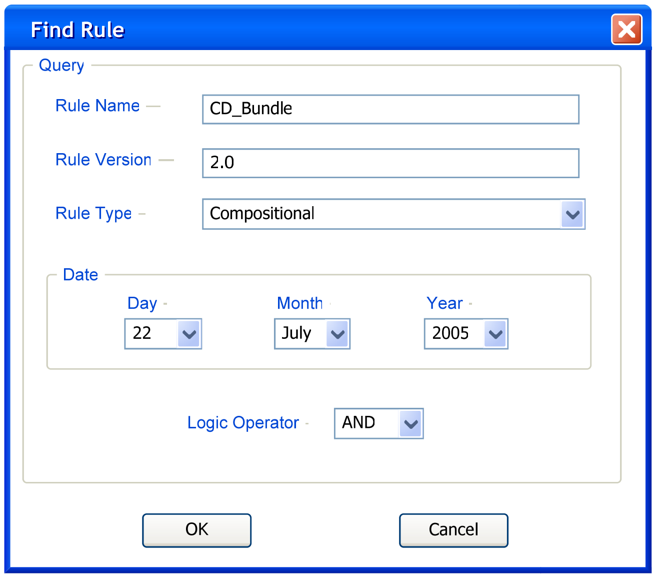
Rule Name	Version	Author	Date of Composition	AXRID

6. **Find –** This will be a common/standard find text no modal dialog where the user will put strings he wants to search in the text.

7. **Go To –** This will be an OK/Cancel no modal dialog where the user will put the number of the line where he wants to set the cursor

8. **Find Rule dialog –** The find rule command will open a dialog that will allow setting a query in order to search a specific rule inside the repository. The query will be built on the following set of metadata:
   - *Rule Name*
   - *Rule Version*
   - *Rule Type*
   - *Date (day, month and year)*

Two logical operators will be available to make query: OR and AND.



Rule Name: CD_Bundle
Rule Version: 2.0
Rule Type: Compositional
Day: 22
Month: July
Year: 2005
Logic Operator: AND

In event of some metadata missing, the query on the repository will be done with the available metadata. An empty query will be not executed, this will be controlled by activating the *OK* button when at least a metadata has been input.
Clicking on the *OK* button will start the search in the rule repository. The *Cancel* button aborts the operation.

2.7.2 AXCP Rule Editor Configuration (DSI)

In this section the set of parameters regarding the configuration of the editor are listed:

- **Rules Repository** – it is the directory where the rule will be saved
- **Rule Engine URL** – it is the network address of the rule engine.
- **Workspace Area –** it indicates if the workspace area will have to be shown or not when the application starts
- **Output Area –** it indicates if the area will be shown or not when the application starts
- **Toolbar –** it indicates which toolbars will have to be put in the main frame
- **Client Size –** it is the information about the last width and height of the main frame
- **Client Position -** It is the information about the last position (x,y) of the main frame
- **Plugins Path –** It is the directory where the DLL of plug-ins with their profiles (workflow, adaptation, descriptor and fingerprint estimators) are stored.

This set will be extended if necessary during the life of the project.

2.7.3 Debugging Rules (DSI)

The editor will provide the debug mode for rule debugging. The debugging mode will be possible by using an instance of the Script Executor in the AXCP Rule Editor. The Rule Executor will be used in the Debugging mode. The debug mode will be based on the following set of functions:

- o **Run** – Enter in the debug mode or if the script is stopped, continue execution until the script is finished, or a breakpoint is reached.
- o **Stop** – Stop the script execution and close the debug mode
- o **Step** - Executes the current line of the script, then pauses. This differs from the "Trace" command in that it will not step into functions and scripts that are called by the current line.
- o **Trace** – Executes the current line of the script, then pauses. This differs from the "Step" command in that if the current line calls a function, or another script, the debugger will trace into the called function or script.

  - **Skip -** Skips over current line without executing it. The script will resume execution on the subsequent line.
  - **Set Breakpoint** – Set a breakpoint on the currently selected line of the script code. Every time the selected line is reached, the debugger will stop.
  - **Remove Breakpoint -** Clear a breakpoint from the currently selected line of the script code.
  - **Remove All Breakpoint -** Clear a all breakpoint in the script code.

To this end the editor will be equipped with a module that will manage the debug mode and specific functions and data structure to:

- maintain the list of breakpoints to put in the code
- put traps in the code corresponding to breakpoints
- open the watch and stack windows
- call the rule engine and initialise it to the debugging mode
- control the debug mode by means the previous set of function.

2.7.4 User Commands and Reporting - AXMEDIS Workflow Manager interaction (DSI)

It provides communication support from/to the AXMEDIS Workflow Manager. Such services are divided in commands for the user and reporting.

- Messages coming from the AXMEDIS Workflow Manager:
  - Description of the work to perform
  - AXRID to associate with the rule to be prepared
  - Rule type
  - Schedule information
  - Some other information related to header of the rule (according to the XML schema)
- Reporting to the AXMEDIS Workflow Manager
  - Notification (end work with success)
  - Exception

Communication with the AXCP Rule Engine

- Commands to the AXMEDIS Rule Engine:
  - Get list of rules
  - Activate rule
  - Deactivate rule
  - Remove rule
  - Install rule in the scheduler (xml file transfer)
  - Get Rule
- Notification, messages, files and exception returned by the AXMEDIS Rule Engine.

The communication between the AXCP Rule Editor and the Engine will be based on the same protocol used by the Workflow Manager for communicating with the AXCP Rule Engine. It will based on Web Service. This will allow avoiding duplications.

2.7.5 External Procedures Profile Manager (FHGIGD, DSI)

To enable AXMEDIS to be a flexible structure, which can be extended according to the specific user needs, plug-ins can be easily be integrated into the AXMEDIS framework. The plug-ins are handled by the Plug-In Managers (AXMEDIS Editor::Plug-In Manager and the Collector Engine::Collector Plug-In Manager).

The Plug-in Managers takes care about the installation, registration, and loading of plug-ins. Different kinds of plug-ins are supported, including:

- Data-manipulation plug-ins shall be able to modify AXMEDIS object structure, i.e. plug-ins which shall be able to delete or move existing components, insert new components, etc…

- Metadata show/manage plug-ins shall be used by Metadata View to adequately display and modify user-defined sets of metadata;
- Metadata production shall be able, through AXMEDIS object (and parts thereof) analysis, to produce metadata to be included into the object;
- Configuration plug-ins shall be used by AXMEDIS Editor Configuration Manager to manage and display specific configuration information;
- Workflow plug-ins which shall permit interaction of AXMEDIS Editor with AXMEDIS Workflow subsystem;
- Protection plug-ins, which contain protection algorithm enriching the set of those available for the Protection Processor;

To enable an effective management, a profile manager is responsible for the handling of the profile descriptions. This profile manager is very close to the Plug-In manager. It is responsible for loading the profiles from installed libraries. The relevant information includes:

- the category of the plug-in, e.g. content processing;
- the unique identifier of the plug-ins
- the signature of the plug-in
- data specific for the kind of plug-in
- the signature

As the external procedures profile is closely related to Plug-in Manager and the content processing algorithms details can be found in DE3-1-2B (Framework and Tools Specification – Viewers and Players) and in DE3-1-2D (Framework and Tools Specification – Fingerprint and Descriptors).

2.8 AXCP Rule Engine (DSI)

Module Profile		
AXCP Rule Engine		
Executable or Library(Support)	Executable	
Single Thread or Multithread	Multi-Thread	
Language of Development	C++	
Responsible Name	Ivan Bruno	
Responsible Partner	DSI	
Status (proposed/approved)	Proposed	
Platforms supported	Microsoft Windows, Linux, MACOS X	
Interfaces with other tools:	Name of the communicating tools	Communication model and format (protected or not, etc.)
AXMEDIS Workflow manager	Command and reporting	Web Service
File Formats Used	Shared with	File format name or reference to a section
Config Files		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
Views Manager	C++	wxWidgets 2.4.2 C++ library

		SpiderMonkey JavaScript Engine ver. 1.5 by Mozilla
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not
wxWidgets 2.4.2	wxWidgets 2.4.2 C++ library	LGPL
SpiderMonkey JavaScript Engine	SpiderMonkey JavaScript Engine ver. 1.5 by Mozilla	LGPL
XERCES-C++	XERCES 2.6.0	LGPL

In this section, the specification of the AXCP Rule Engine will be discussed. As already discussed, the content processing activity (production, protection and publication on AXEPTool) will be based on a unified and shared solution. In these terms, the AXCP Rule Engine will play the role of:

- AXMEDIS Compositional/Formatting Engine
- AXEPTool Loading Tool Engine
- AXEPTool Publication Tool Engine
- AXEPTool P2P Active Selection Engine
- The Protection Tool Engine

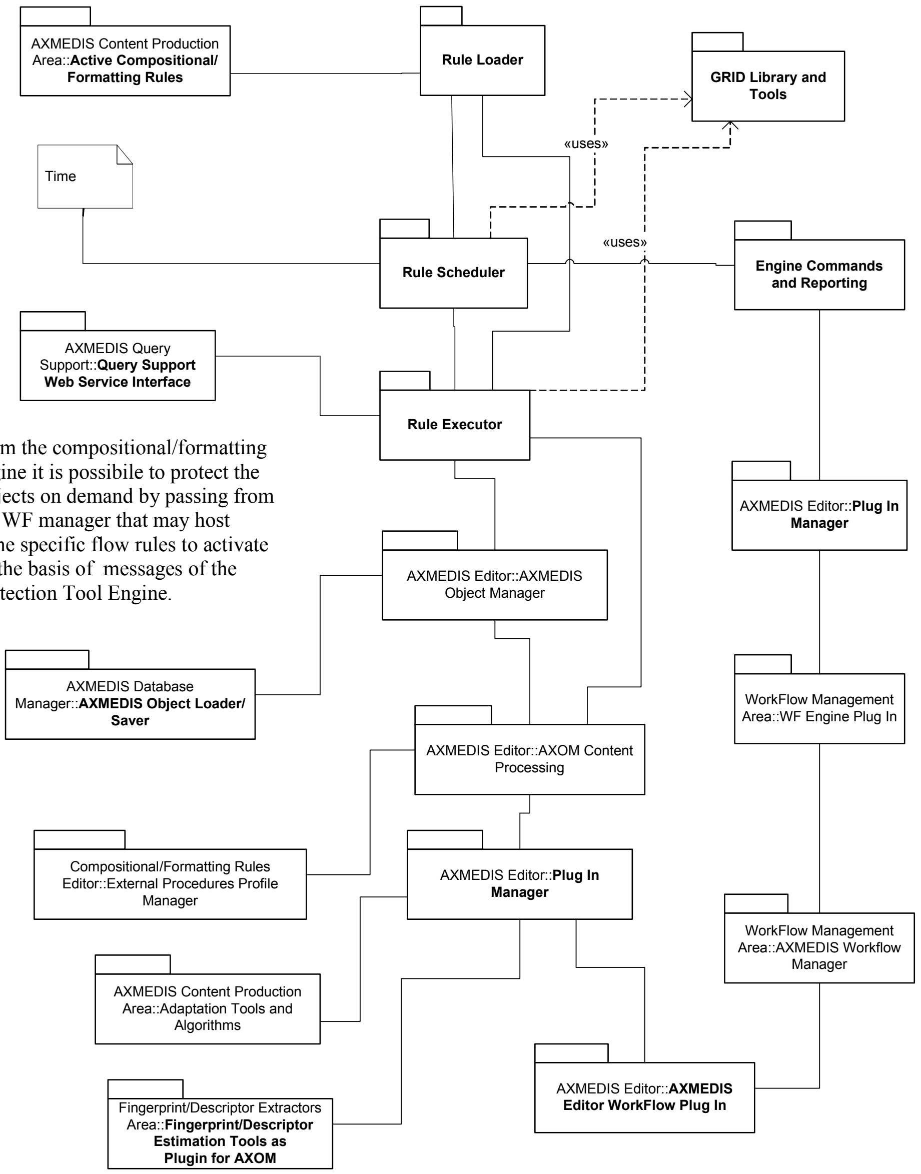
By delegating the processing activity to a single rule engine seems to be not the best solution since the amount of work and the dimension of data that the engine will have to manage is high. The main idea is to design a distributed environment of engines for the AXMEDIS object processing based on GRID. This solution will maintain advantages of a unified solution and allow enhancing the capabilities of the AXMEDIS content processing area by running rules in parallel.
According to the UML diagram, the AXCP Rule engine will be divided in two main components:

- **Rule Scheduler (Server Side)** – It consists of the Scheduler and Dispatcher. It performs the operations of rule firing, rule executor discovering and management, rules dispatching, communication with the AXMEDIS environment, etc….

- **Rule Remote Executor** (**Client Side**) – It is the executor of rules and consists of a script engine based on JavaScript (JS) SpiderMonkey released by Mozilla. It receives the JavaScript code associated with rule and performs the necessary operations for: Script preparation, JS Engine initialisation, JS Engine running script

The **Grid infrastructure** will be realised by means P2P technology. For these reason both the Rule Scheduler and the Rule Executors will be equipped with a P2P communication support.

Compositional/Formatting Engine
To be taken as an example of others AXMEDIS Engine for Rules


AXMEDIS Content Production Area::Active Compositional/ Formatting Rules
Rule Loader
GRID Library and Tools
«uses»
Time
Rule Scheduler
«uses»
Engine Commands and Reporting
AXMEDIS Query Support::Query Support Web Service Interface
Rule Executor
AXMEDIS Editor::Plug In Manager
AXMEDIS Editor::AXMEDIS Object Manager
AXMEDIS Database Manager::AXMEDIS Object Loader/ Saver
WorkFlow Management Area::WF Engine Plug In
AXMEDIS Editor::AXOM Content Processing
Compositional/Formatting Rules Editor::External Procedures Profile Manager
AXMEDIS Editor::Plug In Manager
WorkFlow Management Area::AXMEDIS Workflow Manager
AXMEDIS Content Production Area::Adaptation Tools and Algorithms
Fingerprint/Descriptor Extractors Area::Fingerprint/Descriptor Estimation Tools as Plugin for AXOM
AXMEDIS Editor::AXMEDIS Editor WorkFlow Plug In

From the compositional/formatting engine it is possibile to protect the Objects on demand by passing from the WF manager that may host some specific flow rules to activate on the basis of messages of the Protection Tool Engine.

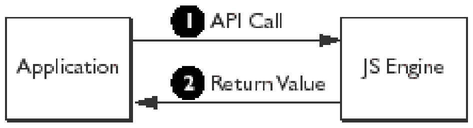
2.8.1 JSENGINE (SpiderMonkey by Mozilla)

The AXMEDIS Rule Executor will be equipped with a JavaScript Engine based on SpiderMonkey, the Mozilla's C implementation of JavaScript ([1]). The JS engine supports JS 1.0 through JS 1.5, JS 1.4., JS 1.3 and greater conform to the ECMAScript-262 specification. At its simplest, the JS engine parses, compiles, and executes scripts containing JS statements and functions. The engine handles memory allocation for the JS data types and objects needed to execute scripts, and it cleans up--garbage collects--the data types and objects in memory that it no longer needs. Generally, the JS engine is built as a shared resource. For example, the engine is a DLL on Windows and Windows NT, and a shared library on Unix. The JS engine's API provides functions that fall into the following broad categories:

- Data Type Manipulation
- Run Time Control

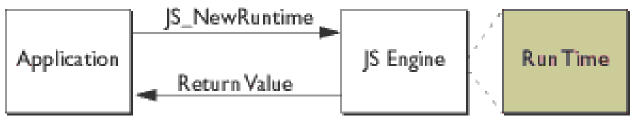
- Class and Object Creation and Maintenance
- Function and Script Execution
- String Handling
- Error Handling
- Security Control
- Debugging Support

Conceptually, the JS engine is a shared resource on the system. By embedding engine API calls in the application, requests can be passed to the JS engine for processing. The engine, in turn, processes requests, and returns values or status information back to the application. The following picture illustrates this general relationship:



In truth, the actual relationship between the application and the JS engine is somewhat more complex than shown in Figure 1.1. For example, it assumes that you have already built the JS engine for your platform. It assumes that your application code includes `jsapi.h`, and it assumes that the first call your application makes to the engine initialises the JS run time.

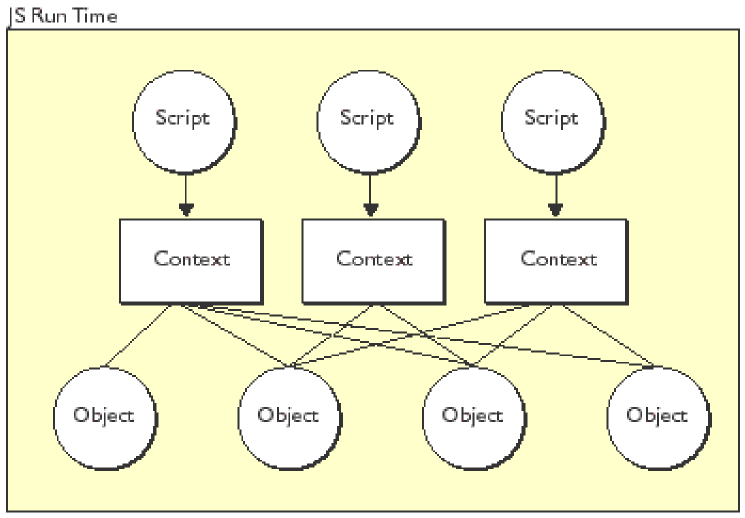
When the JS engine receives an initialisation request, it allocates memory for the JS run time. The picture reported below illustrates this process:



The run time is the space in which the variables, objects, and contexts used by the application are maintained. A context is the script execution state for a thread used by the JS engine. Each simultaneously existent script or thread must have its own context. A single JS run time may contain many contexts, objects, and variables.

Almost all JS engine calls require a context argument, so one of the first things the application must do after creating the run time is call `JS_NewContext` at least once to create a context. The actual number of contexts the application needs depends on the number of scripts expected to use at the same time in the application. One context is needed for each simultaneously existing script in the application. On the other hand, if only one script at a time is compiled and executed by application, then you need only create a single context that you can then reuse for each script. After contexts creation, the built-in JS objects in the engine must be initialized by calling `JS_InitStandardClasses`. The built-in objects include the `Array`, `Boolean`, `Date`, `Math`, `Number`, and `String` objects used in most scripts.

Most applications will also use custom JS objects. These objects are specific to the needs of applications. They usually represent data structures and methods used to automate parts of your application. To create a custom object, you populate a JS class for the object, call `JS_InitClass` to set up the class in the run time, and then call `JS_NewObject` to create an instance of your custom object in the engine. Finally, if your object has properties, you may need to set the default values for them by calling `JS_SetProperty` for each property. Even though you pass a specific context to the JS engine when you create an object, an object then exists in the run time independent of the context. Any script can be associated with any context to access any object. The following figure illustrates the relationship of scripts to the run time, contexts, and objects.

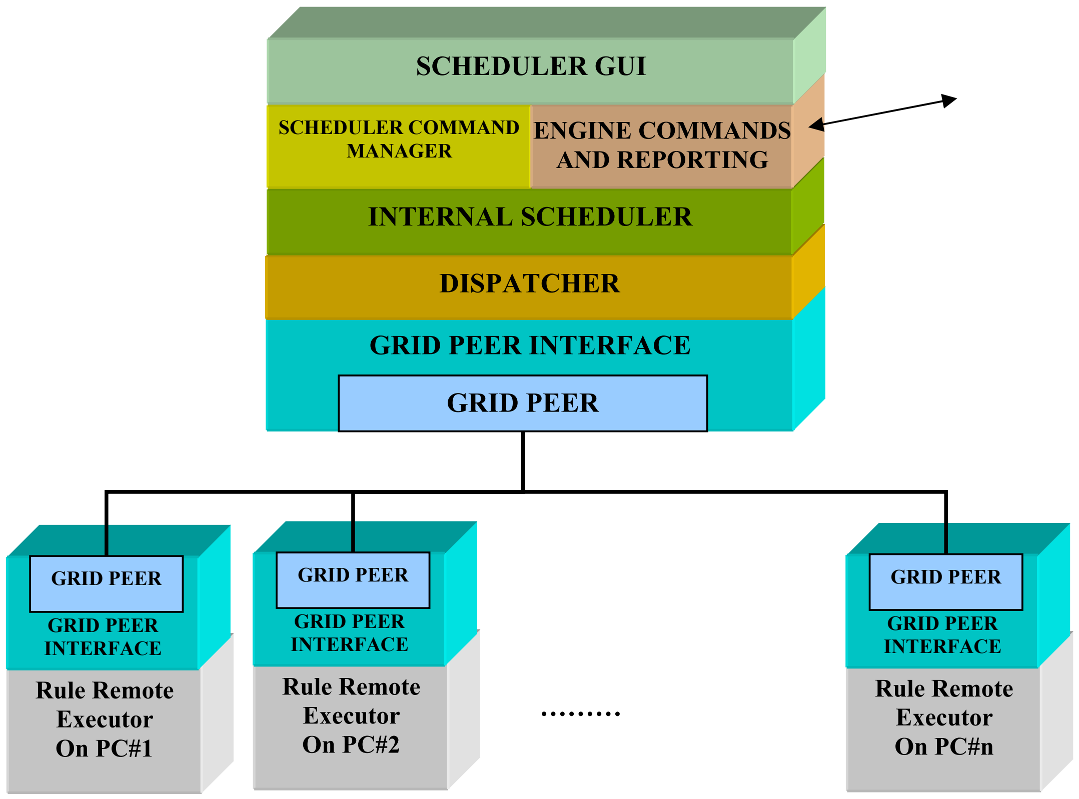


As the previous figure also illustrates, scripts and contexts exist completely independent from one another even though they can access the same objects. Within a given run time, an application can always use any use any unassigned context to access any object. There may be times when you want to ensure that certain contexts and objects are reserved for exclusive use. In these cases, create separate run times for your application: one for shared contexts and objects, and one (or more, depending on your application's needs) for private contexts and objects.

NOTE: Only one thread at a time should be given access to a specific context.

2.9 Rule Scheduler

The Rule Scheduler is the application involved in the rules engines management. It plays the role of server in the distributed environment. It will be a multithread application and will be developed by using the wxWidgets ver. 2.4.2 library. This should allow having a multiplatform rule scheduler.
Referring to the picture, it is constituted of following parts:



AXMEDIS Distributed Engines

2.9.1 Scheduler Command Manager

The Scheduler Manager is the main application and provides the interface to the Scheduler GUI. The main provided functionalities are:

- Load/Save the configuration file.
- Initialisation of the scheduler
- Starting the scheduler
- Stopping the scheduler
- Exporting the list of rules and remote executors
- Browsing the list of jobs/rules and remote executors
- Performing backup of current status (list of rules)
- Performing the restore of the last list of rules
- Load/Install rules
- Providing commands for managing, controlling and monitoring the execution of rules and the activity of remote executors.
- Logs and reports management

The Scheduler Manager will manage the configuration parameters stored in a "Schedeuler.ini" configuration file. The set of parameters will contain information about:

- **Backup Time** - Backup interval for logging the set of submitted rule and tracing operations. It will be expressed in minutes.
- **Time Out** - Time out on client activity. It will be expressed in seconds.
- **Time Resolution** - Time Resolution of the scheduler. It will be expressed in seconds.
- **Refresh Time** - Time Resolution for discovering new rule executors
- **Rules Path** - Rule Repository Path
- **Log Path** - Log Repository Path
- **Profile Path** - Executor Profile Repository Path
- **Backup Path** – The path where the scheduler periodically will save the current rules list.

The management of configuration parameters will be done by using the AXMEDIS Configuration Manager.

2.9.2 Engine command and reporting (AXMEDIS Workflow Manager interaction)

It will provide the communication and the interaction layer with the AXMEDIS Workflow Manager and to external applications such as the AXMEDIS Rules Editor. It will based on Web Service (GSOAP) and Workflow PlugIn. This module provides communication functionalities and allows managing:

- **Commands coming from the AXMEDIS Workflow Manager:**
  - *Run rule(AXRID, arguments, when)* – Ask for a running of a rule specified by means the AXRID, this command could override the arguments value of the rule by sending the xml descriptions about the *arguments* section(according to the XML schema) and could specify when the rule has to be run by overriding the *schedule*. If the command does not override the arguments the rule will be executed with the current arguments. If the command does not override the schedule, the rule will be executed immediately. Also all other combinations of overriding will be considered.
  - *Activate rule(AXRID)* – Switch the status of the rule specified by means the AXRID in the "ACTIVE" status
  - *Deactivate rule(AXRID)* - Switch the status of the rule specified by means the AXRID in the "INACTIVE" status
  - *Remove rule(AXRID)* – Remove the rule specified by means the AXRID from the Scheduler
  - *Pause rule*(AXRID) – Put the rule specified by means the AXRID in the "PAUSE" status
  - *Suspend rule*(AXRID) - Put the rule specified by means the AXRID in the "SUSPENDED" status for a specific time.
  - *Resume rule(AXRID)* – Resume the rule specified by means the AXRID
  - *Kill rule(AXRID)* – Stop the running of the rule specified by means the AXRID
  - *Install and activate rule (rule xml file)* – Install a new rule in the Scheduler by sending the XML file.
  - *Reschedule Rule(AXRID, schedule*) – Override the Schedule information of the rule specified by the AXRID. The schedule parameter will be the xml description according to the XML schema.
  - *Get list of rules* – Request the list of AXRIDs related to rules currently in the scheduler.
  - *Get logs(AXRID)* – Request the current action log generated by the Rule Scheduler. The action log will be pre-filtered to make available to the workflow an action log that is structured per each rule.
  - *Get rule status(AXRID)* – Request the current status of the rule specified by means the AXRID and the start time if rule is running or the next running time if it is not
  - *Get xml rule(AXRID)* - Request the XML file of the rule specified by means the AXRID

- **Reports, messages and files returned to the AXMEDIS Workflow Manager:**
  - Error notification (failure messages)
  - Complete rule notification
  - Log of the current activity

- List of AXRIDs related to the currently scheduled rules
- Xml file of Rule
- Rule status (current status plus schedule information)

2.9.3 Communication with the AXMEDIS Rule Editor

- Commands coming from the AXMEDIS Rule Editor:
    - Get list of rules
    - Activate rule
    - Deactivate rule
    - Remove rule
    - Install rule in the scheduler (xml file transfer)
    - Get Rule

The communication between the AXCP Rule Editor and the Engine will be based on the same protocol used by the Workflow Manager. It will based on Web Service. This will allow avoiding duplications.

2.9.4 Internal Scheduler

The internal scheduler is the manager of active rules. It has to detect, fire, launch and manage the execution of a rule. During its activity, the internal scheduler has to:

1. preserve the scheduled work from interruption of service (crash of the application) giving the possibility to restore the last status of activity
2. manage and update the list of rules to be scheduled and their status
3. manage and update the list of available rule executors
4. notify to the AXMEDIS Workflow Manager messages due to:
    - errors during the phase of rule association with an executor
    - errors due to the launching phase
    - errors during the rule execution on remote executor.
    - errors due to the time out deadline missing (the executor did not respond to request)

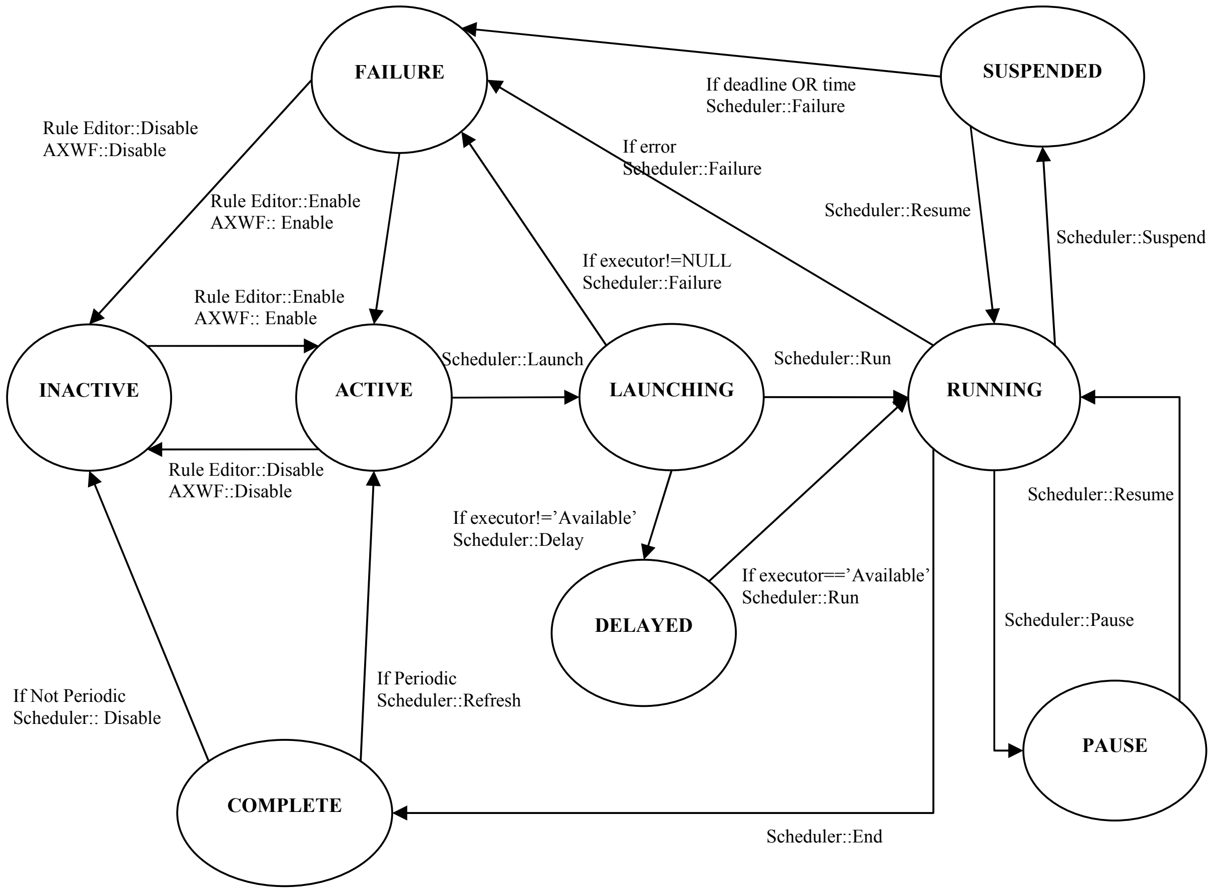
To this end, the functionalities provided by the rule scheduler are:

- Select from the internal scheduled rules the rule that matches conditions for the execution. This is performed by:
    - checking the execution time and date
    - receiving an immediate run command from the AXMEDIS Workflow Manager
- Modify and set the time resolution for the control of rules execution
- Add a new submitted rule in the list of jobs
    - Loading the corresponding rule xml file from the repository directory
    - Extracting the metadata for scheduling
    - Generating and assigning a Job Id to the rule
- Remove a rule from the list of jobs
- Run a rule on demand
- Reschedule a rule (by overriding the schedule information)
- Override rule arguments (by replacing the current arguments)
- Check expiration conditions of a rule
- Provide the list of jobs/rules
- Update firing conditions of a periodic rule
- Browse the list of jobs/rules
- Modify the status of rules
- Remove an executor from the list of executors
- Provide the list of executors
- Browsing the list of executors
- Save periodically on disk a backup copy of the list of jobs
- Restore the last status by loading the backup copy of the list of jobs

- Route messages coming from the dispatcher and the remote executors to the AXMEDIS Workflow Manager

Rule Life Cycle
The life cycle of rule represents the evolution of a rule from the activation to its completion. The evolution is strictly linked to the activities performed by the scheduler, the dispatcher and the executor and it can be described by means of a status attribute. The status of a rule will assume the following values:

1. **Active** – The rule is waiting for the execution
2. **Inactive** – The rule will be not executed
3. **Failure** – An error occurred during the execution or control of rule. The execution is blocked and the executor is released.
4. **Launching** – The rule will be associated with a remote executor.
5. **Delayed** – The launch of rule has been delayed. Available executors are temporary busy with another rule.
6. **Running** – The rule is running on a remote executor
7. **Pause** – The run of rule has been stopped temporary
8. **Suspended** – The run of rule is suspended by defining a temporal interval. The time unit is second.
9. **Complete** – The run is finished



State Diagram for a rule

The general evolution of the rule status is depicted in the State Diagram representation, whereas the description of each transition is reported in the following.

Complete to Active Transition – The transition from COMPLETE to ACTIVE status is performed if the rule has to be executed periodically. In this case, the rule is re-submitted to the scheduler and its run-conditions are updated on the basis of the specified period.

Complete to Inactive Transition – The transition from COMPLETE to INACTIVE status is performed when the rule has to be executed once. In this case, the rule is ready to be removed from the scheduler or to be modified by means of the Rule Editor or to be run on demand.

Active to Launching – The transition from ACTIVE to LAUNCHING status is performed when the scheduler fires a rule. The rule has to be associated with a remote executor.

Launching to Running Transition – The transition from LAUNCHING to RUNNING status is performed when a rule is running on the remote executor.

Launching to Delayed Transition – The transition from LAUNCHING to DELAYED status is performed when the executor that could run the rule is busy. The rule is placed temporally in a delay queue to be run when the executor will be ready.

Delayed to Running Transition – The transition from DELAYED to RUNNING status is performed when the remote executor is ready to run the rule.

Launching To Failure Transition – The transition is performed when during the rule check operation, the rule profile does not match any available executor profiles.

Running to Failure Transition – The transition from RUNNING to FAILURE status is performed if the following conditions occur:

- when during the execution of rule the corresponding rule executor sends a run-time error message.
- when during the execution of the rule, the executor does not respond to sever call (Time out).

Running to Complete Transition – The transition from RUNNING to COMPLETE status is performed when the execution of rule is successfully completed.

Running to Pause Transition – The transition from RUNNING to PAUSE status is performed when a pause rule request comes from the Scheduler. The status of PAUSE can be conditioned by deadline condition.

Pause to Running Transition – The transition from PAUSE to RUNNING status is performed when a resume rule request comes from the Scheduler.

Failure to Inactive Transition – The transition from FAILURE to INACTIVE status is performed when a disable rule request comes from the AXMEDIS Workflow Manager or from the Rule Editor.

Failure to Active Transition – The transition from FAILURE to ACTIVE status is performed when a disable rule request comes from the AXMEDIS Workflow Manager or from the Rule Editor.

Suspended to Failure Transition – The transition from SUSPENDED to FAILURE status is performed when the suspension misses the deadline condition.

Running to Suspended Transition – The transition from RUNNING to SUSPENDED status is performed when a rule suspension request comes from the Scheduler. The status of SUSPENDED is conditioned by a deadline condition (e.g. a temporal interval).

Suspended to Running Transition – The transition from SUSPENDED to RUNNING status is performed when a resume rule request comes from the Scheduler. Since the status of SUSPENDED can be conditioned by deadline condition, a rule is automatically resumed by the scheduler when the deadline condition is matched.

Jobs as Rules
From the scheduler point of view, a rule is a job to be executed. Each job is described by using the metadata contained in the *Header*, *Schedule* and *Definition* section of the rule.
Jobs are organised in a table, called **Table of Jobs.** Such table will be monitored by the internal scheduler periodically to detect the rule to be fired. A periodic backup of the table will have to be performed to guarantee a certain degree of reliability in case of critical problems that could require to restart the scheduler. The backup copy will permit to restore the last status of the scheduler.

Job	
Attributes	**Description**
Rule Name	The name of rule. It comes from the corresponding data in the Header section of the rule
Rule Version	The version of rule. It comes from the corresponding data in the Header section of the rule
Rule Type	The type of rule. It comes from the corresponding data in the Header section of the rule
Start Time	The start time of execution. It comes from the corresponding data in the Schedule section of the rule
Start Date	The start date of execution. It comes from the corresponding data in the Schedule section of the rule
Periodicity	The name of rule. It comes from the corresponding data in the Schedule section of the rule
Expiration Time	The expiration time of job life. It comes from the corresponding data in the Schedule section of the rule
Expiration Date	The expiration date of job life. It comes from the corresponding data in the Schedule section of the rule
Executor ID	It is the identifier of the current executor associated with the rule by the Dispatcher
N° of Run	Number of times that the rule has been run
Job ID	Identifier associated with the rule by the scheduler
Profile	List of dependencies (AXMEDIS Plug In, external tools, etc…). It comes from the Uses data in the Definition section of the rule
URL	The complete path of the rule xml file.

2.9.5 Rule Scheduler User Interface

The Scheduler GUI will be the main window that will allow the interaction with the Scheduler. It will be constituted of:

1. A menu bar
2. Two main areas where the list of rules and the list of remote executors are displayed.
3. A status bar where the current clock and the current date are displayed.

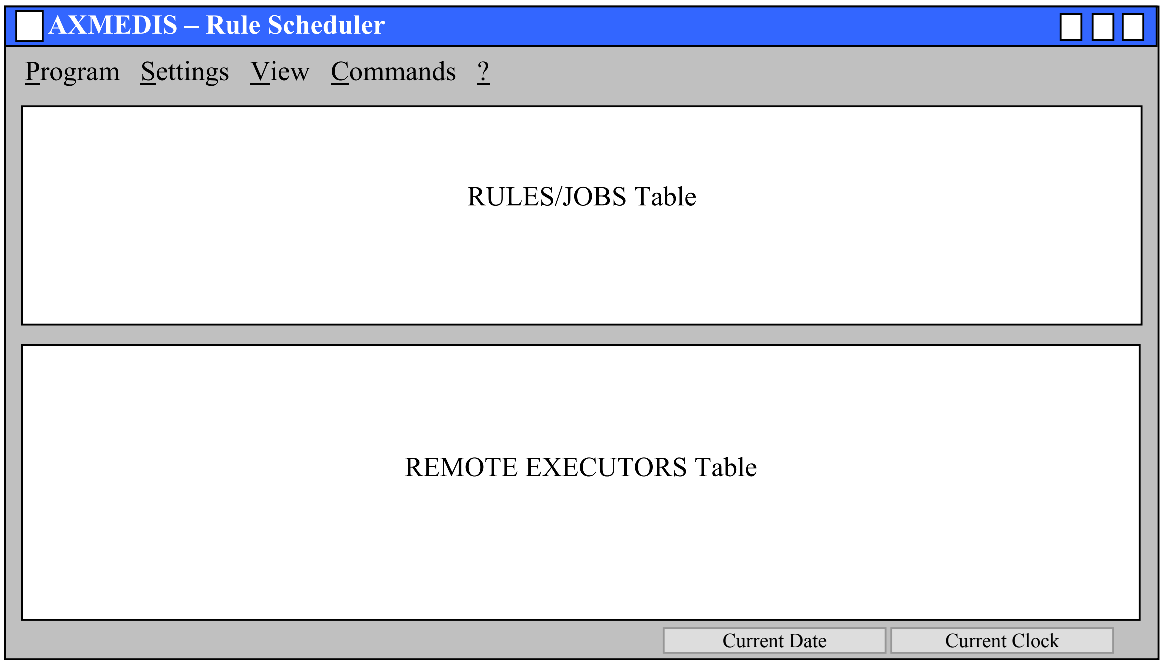
Menu bar – It will provide the access to the following set of functions:

1. **Program**
    a. *Launch scheduler* - Start the scheduler activity.
    b. *Stop scheduler* - Stop the scheduler activity.
    c. *Restore* - Backup Copy of the last jobs list.
    d. *Minimize* - It reduces at icon on the taskbar.
    e. *Exit* - Close the application.
2. **Settings**
    a. *Preferences* - Open an editable dialog with the set of configuration parameters.
3. **View**
    a. *Refresh* – Update the list of jobs and list of remote executors.

   b. *Arrange* – Repainting modes of tables in the main frame
      i. *Top* – It shows only the top table (Table of rules)
      ii. *Bottom* – It shows only the bottom table (Table of executors)
      iii. *Vertical* – It shows tables vertically
      iv. *Horizontal* – It shows tables horizontally
   c. *Rule Properties…* - Open a Rule Properties dialog.
   d. *Executor Profile…*- Open an Executor Profile dialog.
   e. *Logs…*- Open a dialog to show the list of log messages

4. **Commands**
   a. *Activate Rule* - Put in the "ACTIVE" status the current selected inactive rule.
   b. *Deactivate Rule* - Put in the "INACTIVE" status the current selected active rule.
   c. *Kill Rule* - Kill the current execution of the current selected rule.
   d. *Pause Rule* - Put in pause the execution the current selected rule.
   e. *Resume Rule* - Resume the execution of the current selected rule.
   f. *Remove Rule* – Remove the rule from the list of rules
   g. *Suspend Rule…* - Open a dialog to edit the temporal interval for rule resuming and then suspend the current selected rule.
5. **?**
   a. *Help* - Open the On Line help.
   b. *About* - Open a dialog with credits.

All this functionalities will be also accessible by means shortcuts.



Rules/Jobs Table - It is the area where scheduled rules are displayed. It will be a list control constituted of a set of columns where the following list of metadata will be displayed:

- *Rule name* – it will display the name of the rule
- *Rule version* – it will display the version of rule
- *Rule status* – it will display the current status of rule
- *Rule ID* – it will display the identifier of rule
- *Executor ID* – it will display the identifier of the executor associated with rule
- *Start Time* – it will display the time to fire the rule
- *Start Date* – it will display the date to fire the rule

- *Periodicity* – it will display the periodic attribute
- *N° Runs* – it will display the number of time the rule was fired.

Name	Version	Status	ID	Executor ID	Start Date	Start Time	Periodicity	N° Runs

The following functionalities will be provided by means a contextual popup menu:

- Ordering rules alphabetically by name
- Ordering rules by start running time
- Ordering rules by ID

Remote Executors Table - It is the area where remote executors are displayed. It will be a list control constituted of a set of columns where the following list of metadata will be displayed:

- *Name* - Computer Name
- *IP* - IP address
- *CPU* - CPU & Clock
- *OS* - OS & Version
- *Ping* – The network capabilities in term of transmission time.
- *HD Space* – The space available on the disk of the executor
- *Status* – The status of the executor
- *Rule ID* – The ID of the running rule
- *Executor ID* – The Id of the executor assigned by the scheduler
- *Start Time* – At what time the run is started.

Name	IP	CPU	OS	Ping	HD Space	Status	ID	RuleID	Start Time

The following functionalities will be provided by means a contextual popup menu:

- Ordering executors alphabetically by computer name
- Ordering executors by ID

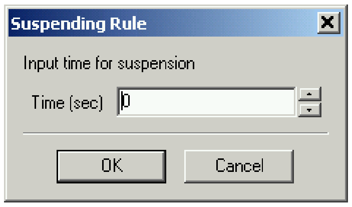
Auxiliary dialogs
The Scheduler GUI will be supported by the following set of dialogs:

Rule Properties Dialog - It will be an editable no modal dialog where the properties of the selected rule are displayed. Part of such properties will be extracted from the XML file associated with rule.

Executor Profile Dialog - It will be a not editable no modal dialog where the properties of the selected executor are displayed. Such properties will be extracted from the executor profile.

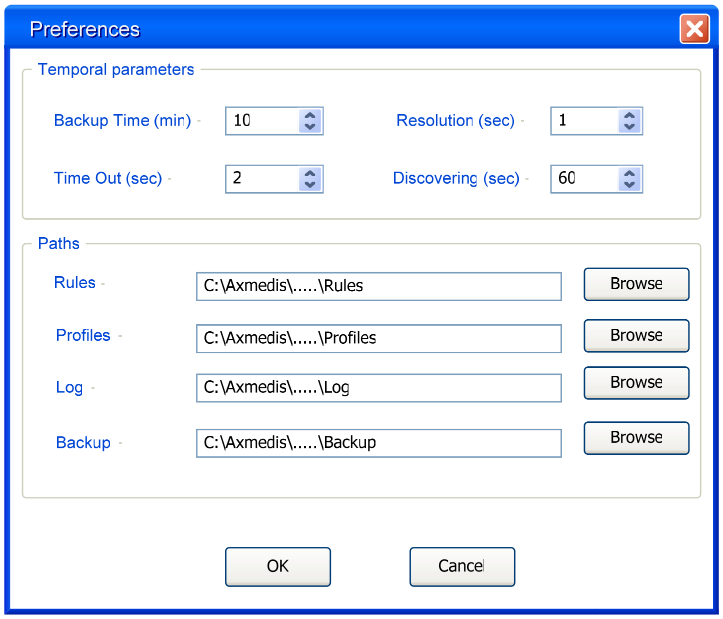
Logs Dialog – It will be a not editable no modal dialog where the log messages will be played.

Suspend Rule Dialog – It will be an editable no modal dialog where the user will put the time for the suspension.



Time (sec) 0

Preferences Dialog – It will be a no modal dialog that will display configuration parameters. Such parameters will be editable and will allow customizing the configuration of the scheduler. The main structure of the dialog is shown in the following picture:



Backup Time (min) 10
Resolution (sec) 1
Time Out (sec) 2
Discovering (sec) 60
Rules C:\Axmedis\.....\Rules
Profiles C:\Axmedis\.....\Profiles
Log C:\Axmedis\.....\Log
Backup C:\Axmedis\.....\Backup

The dialog is divided in two main areas: (i) *Temporal parameters* and (ii) *Paths*. The former area will display parameters related to temporal constrains and deadlines. The latter area will display the paths for rules, profiles of executors, log and backup files storage.
The OK button confirms the value of each parameter and updates settings.
The Cancel button rejects possible update maintaining the old settings.

2.9.6 Dispatcher of the Rule Scheduler

The main role of the dispatcher is to:

- associate a remote executor with the rule to be run
- engage the selected remote executor
- launch the execution of rule on the remote executor
- monitoring the status of remote executors
- managing the possible errors messages and notifications coming from executors
- creating logs and tracing the activity of each remote executor involved in the execution of a rule
- discovering new remote executor
- requesting and receiving the profile from remote executors

Remote Executors

A remote executor is the virtual image on the scheduler side of a real machine equipped with the rule executor. Knowing the availability and capabilities of a remote executor is mandatory to identify the machine that will execute the rule. To this end, the association of executors with rules will be based on the list of available remote executors (computer) and their profiles. Such list will be persistent and it will be managed at run-time. Each remote executor belonging to the list will be described by means of internal attributes (managed by the scheduler) and a profile that will be provided by each real executor during the discovery and refreshing phase. The profile will contain a set of metadata that will describe the capabilities of the remote executor (see Section 2.10.4).

Executor	
Attributes	**Description**
Current Status	It provides the status of the executor
Executor ID	Identifier of the executor
Profile	It is the set of information related to : 1. Identity of the executor (computer name, IP address, location, etc…) 2. Computational capabilities: (CPU, RAM, Clock, etc…) 3. Functionalities that the executor provides: • AXMEDIS Plug-In installed (For each plug in the name and version are provided). • External tools Plug-In installed For each plug in the name and version are provided).

To realise that, the dispatcher will be divided in four main components.

Resource Controller – It periodically will control and refresh the availability of remote executors in the network of AXMEDIS factory. To this end, it will perform the following activities:

- discovering new remote executors
- requesting profiles of remote executors
- refreshing and managing the list of available remote executors (adding a new discovered executor, remove an executor from the list, loading the profile)
- generating and assigning an unique Executor ID (process Id) with the remote executor
- initialising the remote executor (by sending the Executor ID, the port number to use for sending messages to the Rule Monitor)
- managing and updating the status of each remote executor.
- generating an error in case of time out or deadline missing (the remote executor did not respond to a request within a time interval specified by a time out)

Optimizer – It will receive rules to be launched. Such rules will be put in an internal queue that will include rules to be associated with a remote executor. The choice of an executor will be performed by checking the rule profile with the best profile among available remote executors. If all available remote executors do not

match the profile of rule, the association fails, the status of rule is set to "Failure" and an error message is generated and sent to the internal scheduler. If all remote executors matching the profile of the current rule are running different rules, the launch of the current rule is delayed and the rule is maintained in the internal queue of rules waiting for an available executor.
Future version of the optimizer will include optimisation algorithm based on artificial intelligence such as: Taboo Search, Genetic Algorithm, etc…that will improve the scheduler capability.

Rule Launcher – The role of the rule launcher will be to:

1. send commands generated by the scheduler (kill, pause, run, resume) to remote executors
2. engage the remote executor associated with the rule and launch the execution of rule. If the engaging fails an launch error is generated.

To this end, the rule launcher will provide functionalities for:

- communicating with the remote executor (by sending and receiving messages and commands)
- transferring the rule to the remote executor
- communicating errors messages to the internal scheduler if the launch phase fails
- tracing the commands and controls sending to remote executors by updating an activity log file

Rule Monitor – It will monitor persistently the execution of rules by:

- listening to messages and notifications coming from remote executors
- interpreting messages
- managing the log file and trace the activity of each rule executor (by reporting all messages and notifications in input)
- communicating the status of a rule to the internal scheduler for updating
- routing possible errors messages and notifications to the internal scheduler

To this end, the rule monitor will provide functionalities for:

- communicating with the remote executor (by receiving messages and notifications)
- parsing messages of executors
- providing and managing a queue of input messages
- updating log files associated with each remote executor for each message or notification received
- routing errors and messages to the internal scheduler coming from executors

2.9.7 Grid Peer Interface

It is the interface to the Grid Peer. It will provide functionalities for:

- Sending message to a peer
- File transfer to a peer
- Discovering peers
- Engaging/Launching a peer
- Managing messages coming from other peers

2.9.8 Grid Peer

It will provide the support for the distributed system management.
It will be based on TCP/UDP socket and will be constituted of the following components:

- **Peer Explorer** – It will provide functionalities and support for querying the presence of other peers. It will be based on UPD broadcast messages.
- **Peer Communicator** – It will provide functionalities and support for communicating with available peers (already discovered). It will be based on TCP connection.
- **Peer File Transfer** – It will provide functionalities and support for transferring file to a selected peer. It will be based on TCP connection.
- **Peer Event Consumer** – It will provide functionalities and support for handling events of communication, file transfer and discovering.

These components will be used singularly and independently. It will be developed in C++ with STL and WindowsSocket library.

2.9.9 Structure of messages exchanged between Scheduler and Remote Executor

Messages exchanged between the Scheduler and the Remote Executor can be different types and grouped in two set of messages: (i) from Scheduler to Rule Executor and (ii) from Rule Executor to Scheduler.

Messages from Scheduler to Rule Executor:

1. **Command** – the message is a specific command

Messages from Rule Executor to Scheduler:

2. **Notification** – the message is a notification
3. **Error** – the message reports an error
4. **Response** – the message is a response to a request or a command

The main idea is to have a common message structure that allows covering all these types. In addition, to guarantee a fast delivery on the network, messages shall be light. To this end, they will be based on a formatted text and structured according to the following EBNF formalisation:

<message> := <Sender ID>'#'<Type_Msg>

```
<Sender ID> := <string>
<Type_Msg> := <CMD_MSG> | <REQ_MSG> | <NOTIFY_MSG> | <ERR_MSG> | <RESP_MSG>

<CMD_MSG> := 'COMMAND#'<ID_MSG>'#'<command>
<command> := RUN | KILL | PAUSE | RESUME | GET <request> | SET <attribute>  <value>
<request> := PROFILE | STATUS | ID
<attribute> := ID | …
 <value> := <string>

<NOTIFY_MSG> := 'NOTIFICATION#'<what notified>
<what notified> := 'END PROCESS' | <msg>
<msg> := 'MSG' <string>

<ERR_MSG> := 'ERROR#'<error from>'#'<error description>
<error from> := 'RULE' | 'EXECUTOR'
<error description> := <error code> | <string>

<RESP_MSG> := 'RESPONSE#'<to msg>'#'<response argument>
<response argument> := <status> | <executor ID> | 'CMD OK'
<to msg> = <ID MSG>

<ID MSG> = <timestamp>
```

Where:
<timestamp>: it indicates the generation time of a message and allows indexing a message. It could be used as reference to link a response message to command messages and to monitor the activity of the rule executor.

<Sender ID>: it indicates the identifier of the sender. By default, the ID of the Scheduler is '0', whereas for all rule executors will be the Executor ID

<error code>: it reports the code of the error

Example 1:
The scheduler requests the profile to a rule executor by means the message:

```
0#COMMAND#12:00:00 pm#GET PROFILE
```

where:
- '0' is the sender ID associated with the scheduler (server).

Example 2:
The scheduler sends to the rule executor its Executor ID:

```
0#COMMAND #12:20:00 pm#SET ID 34
```

The scheduler requests the value of status to the executor identified by "34" by means of the message:

```
0#COMMAND #12:20:00 pm#GET STATUS
```

The rule executor "34" responds to the request by means of:

```
34#RESPONSE#12:20:00 pm#'value of status'
```

where:
- '34' is the sender ID associated with the Executor ID of the rule executor.

Example 4:
The rule executor '34' sends to the scheduler:
- a message generated by the rule:

```
34#NOTIFICATION#MSG "AXMEDIS Database connection error"
```

- a run time error

```
34#ERROR#EXECUTOR#"Disk Full" or 34#ERROR#EXECUTOR#001
```

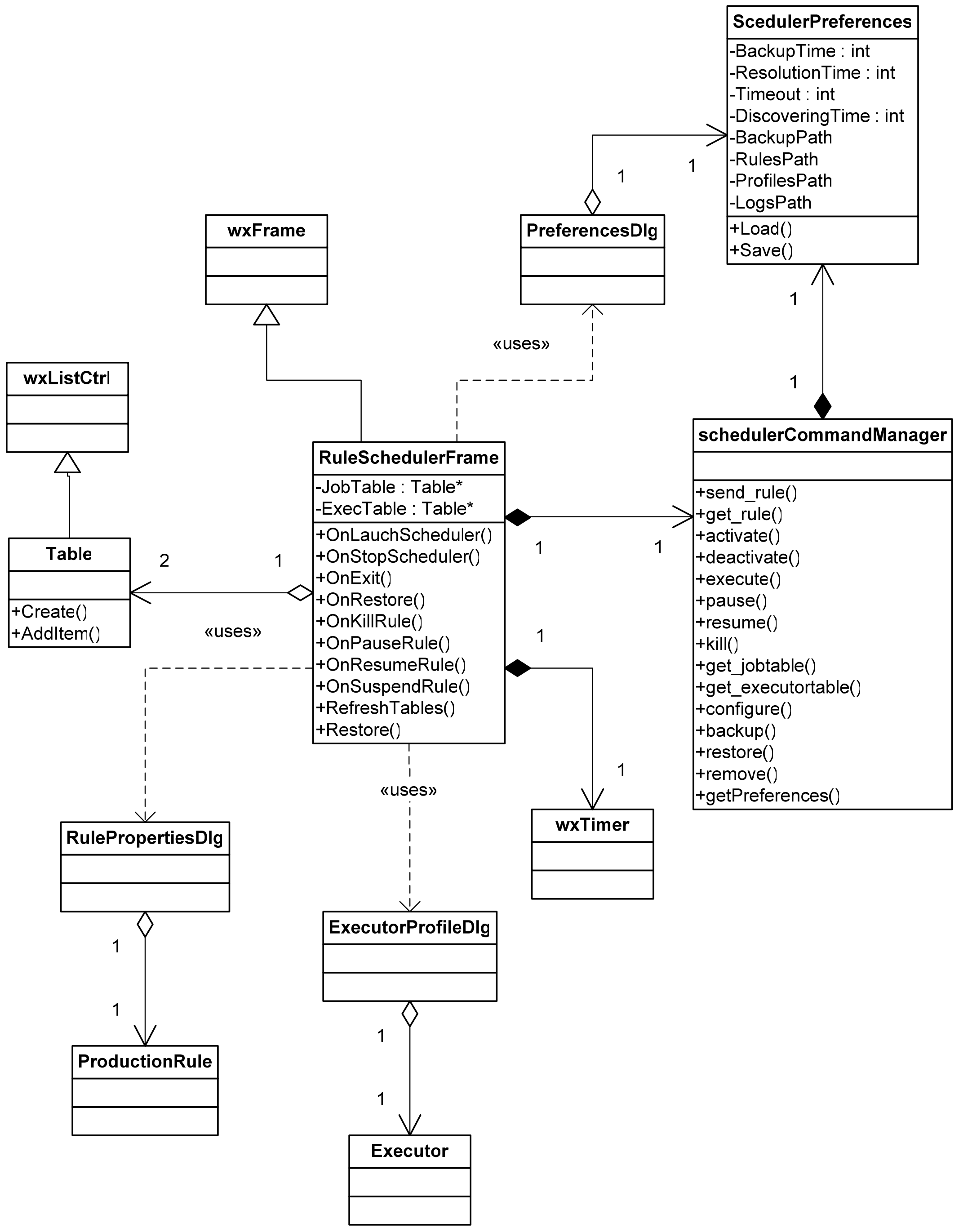
where '001' could be for instance the error code associated with "Disk Full"

- an end process notification

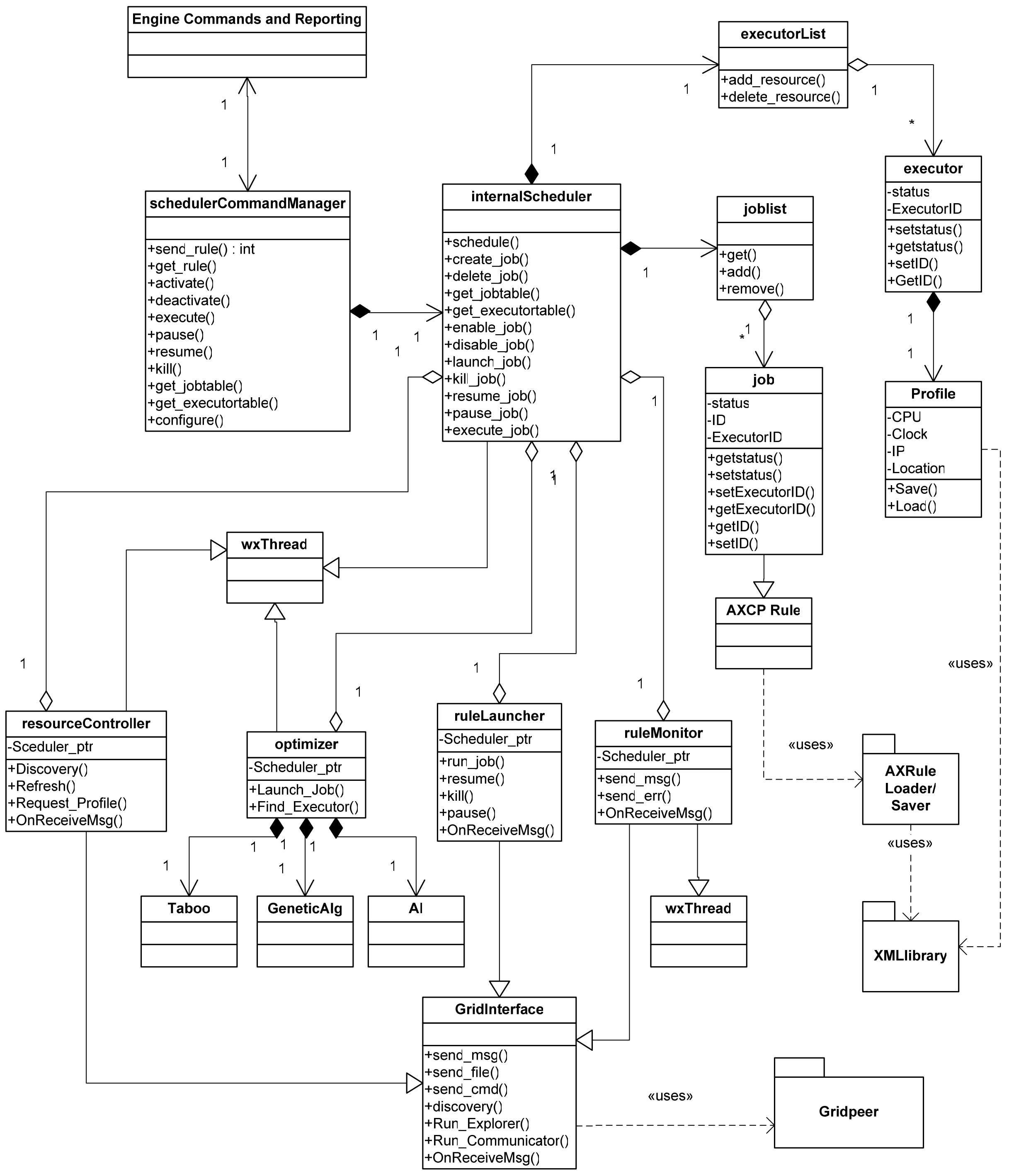
```
34#NOTIFICATION#END PROCESS
```

2.9.10 Rule Scheduler Class Diagram

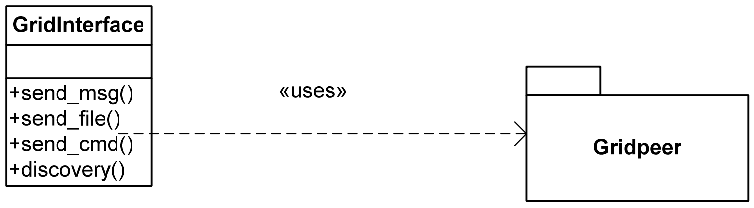
Class diagram of Scheduler GUI

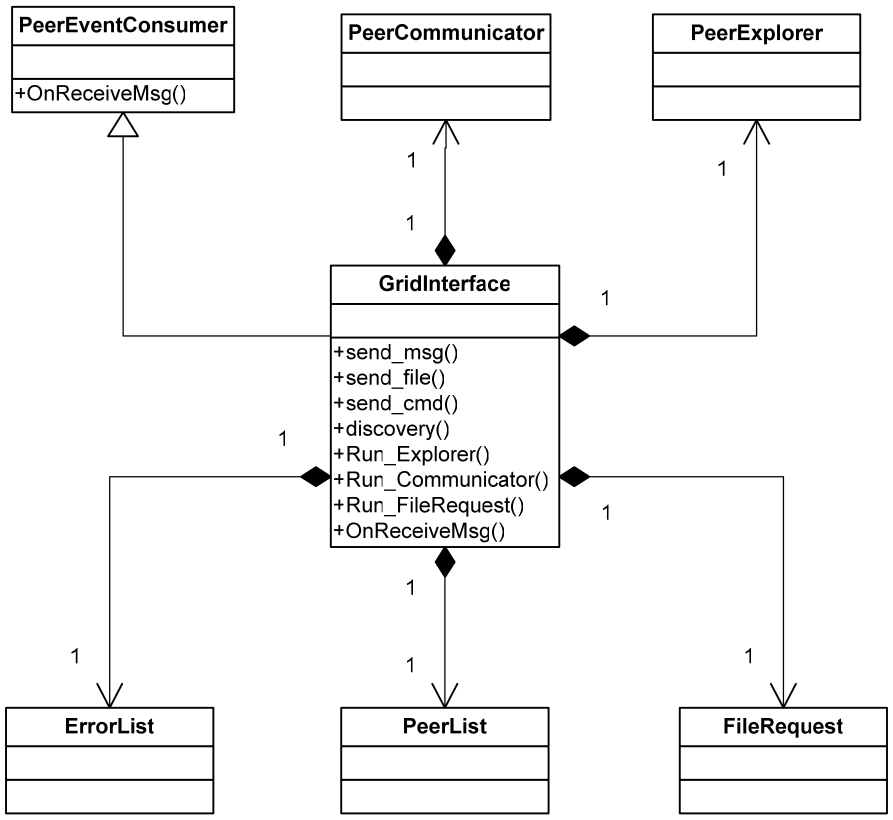


Class diagram of Internal Scheduler



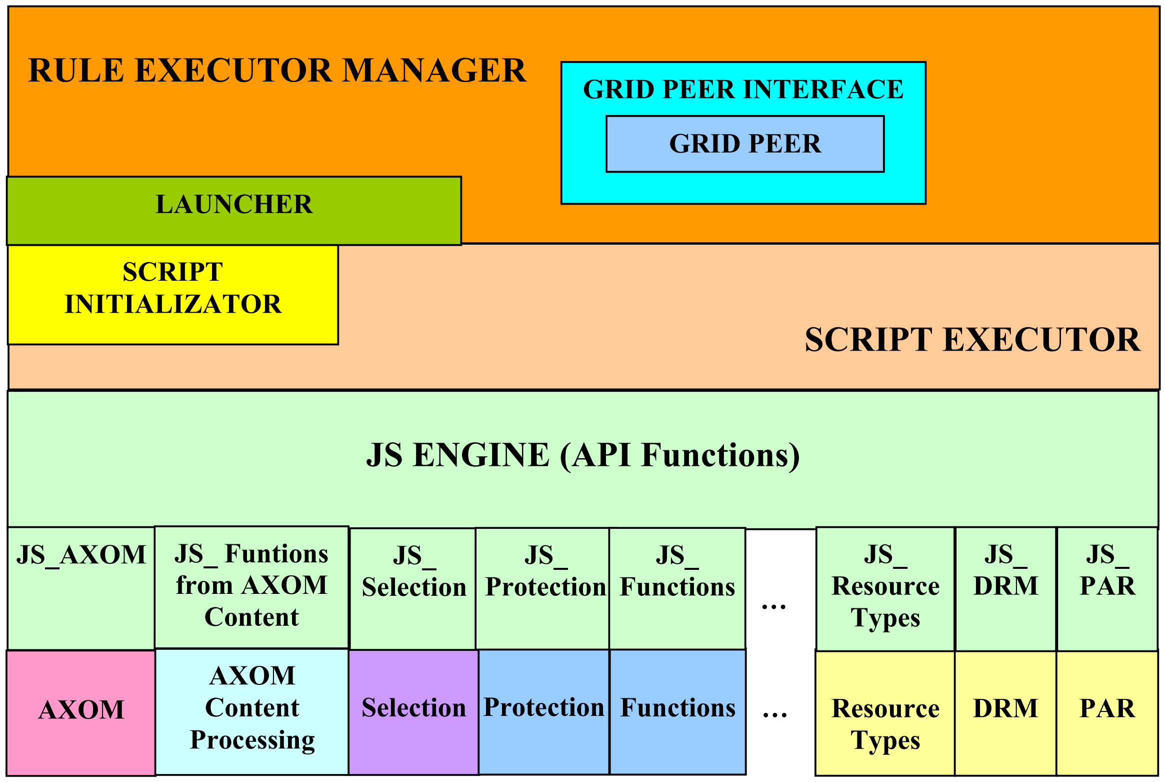
Class diagram of the GridInterface class





2.10 Rule Executor

The rule executor is an application running on a remote computer. It is a computational unit in the distributed rule engines environment. It will be the based on Javascript engine and will execute JavaScript code. To this end it will host the SpiderMonkey Javascript Ending realised by Mozilla. The main architecture of the Rule Executor is depicted in the following picture:



AXMEDIS Rule Executor

The main components will be:

- **Grid Peer Interface** – the communication support with the AXCP rule scheduler
- **Rule Executor Manager** – the command interface of the engine
- **Script Executor** – It will host the SpiderMonkey Javascript Engine (called JS Engine)

2.10.1 Rule Executor Manager

The Rule Executor Manager is the main program and the interface between the script executor and the scheduler. It will host a Grid Peer Interface for the communication in the distribute environment based on the technology of the Grid Peer on the scheduler side. This interface will be the same hosted by the Scheduler or a specialised version if necessary and will allow:

1. receiving commands, messages, requests and files from the Scheduler
2. sending messages, notifications and files to the Scheduler
3. being discovered by the Scheduler during the discovering phase.

The main activity is to:

- generate the profile of the executor to send to the Scheduler

- receive the rule file from the Scheduler
- load the rule in the executor
- manage the launch of the rule execution by means the *Launcher*
- provide to the *Script Executor* the support of communication with the Scheduler.
- notify errors, status and the end of execution to the Scheduler.

To realize that, the Rule Executor Manager will provide functionalities for:

- Routing messages produced by internal components to the scheduler
- Receiving control messages and commands from the Scheduler
- Parsing and executing commands coming from the scheduler such as:
  - Launch the execution of rule
  - Kill the execution of rule
  - Pause the execution of rule
  - Resume the execution of rule
  - Request profile
  - Request status
- File Transferring to:
  - send the profile of the Rule Executor
  - receive the rule to be executed
- Sending messages and notification to the Scheduler
- Creating the profile of the executor according to the XML schema
- Managing the status of the Executor

Status of the Rule Executor
The Rule Executor status describes the activity of the rule engine. If it is available for running a rule the status value will be “READY”, otherwise if it is working with a rule the value will be “RUNNING”. In event of errors that could break the execution, if they could be managed by software, the executor notifies them to the Scheduler stopping definitively the current execution and resetting the status to the “Ready” value.

2.10.2 Launcher

The role of the Launcher is to start the execution of the script. The main steps that the Launcher has to perform, are:

- Loading the rule XML file received by the Scheduler
- Extracting the script included in the Rule (all the information included in the *Definition* section of the XML file)
- Calling the *Script Initializator* for preparing the script
- Calling the *Script Executor* for executing the script

Script Initializator – Before running the script, the Launcher calls the *Script Initializator* to check and prepare the script code for the execution on JS Engine. In this phase, The Engine and the JavaScript script code are prepared according to the SpiderMonkey JS Engine guideline. All the functions related to AXMEDIS plug-ins and arguments of rule will be initialised. The arguments initialisation will allow defining global variables associated with the arguments used in the script by actualising them with values specified in XML rule description. In event of possible errors during the script initialization, a failure message will be generated and sent to Rule Executor Manager that will route it to the Scheduler.

2.10.3 Script Executor

The Script Executor receives the script code and arguments (Selections and parameters), then, it performs the necessary operations for:

- Invoking and initialising the JS Engine and variables.
- Sending the script to the JS Engine.

- Running and managing the communication with the JS Engine **according to the capabilities and functionalities provided by the JS Engine.**
- Routing errors coming from the JS Engine to the Rule Executor Manager.
- Sending Messages coming from the script in execution to the Rule Executor Manager.

The Script Executor will be developed to be used also in the AXCP Rule Editor. As depicted in the software architecture, the editor will be equipped with a Rule Executor. When used inside the AXCP Rule Editor it will be able to work also in two different modes: the rule debugging mode and rule check mode. This will be useful during the definition of a rule since the user will be able to test the rule and to solve possible errors

Script Executor: Debugging Mode
The Executor will be realised by using the debug function provided by JSDebug API of SpiderMonkey and to be controlled by the AXCP Rule Editor. The Spidermokey APIs permit to :
- put traps in the code corresponding to breakpoints (interrupting the execution)
- watch variables
- manage the stack of functions
- realise the interface for debug functions and controls for the AXCP Rule Editor.

Script Executor: Check Mode
This modality will be mainly used by AXCP Rule Editor when it will be necessary to check the feasibility of a rule. In the testing mode, the rule will be executed in order to:
- verify the correctness of the rule before to send it to the AXCP Rule Engine
- estimate some parameters related to the complexity of the rule. Such parameters will be identified and defined during the project life. They will be used to define a complete profile of the rule in terms of required computational resources.

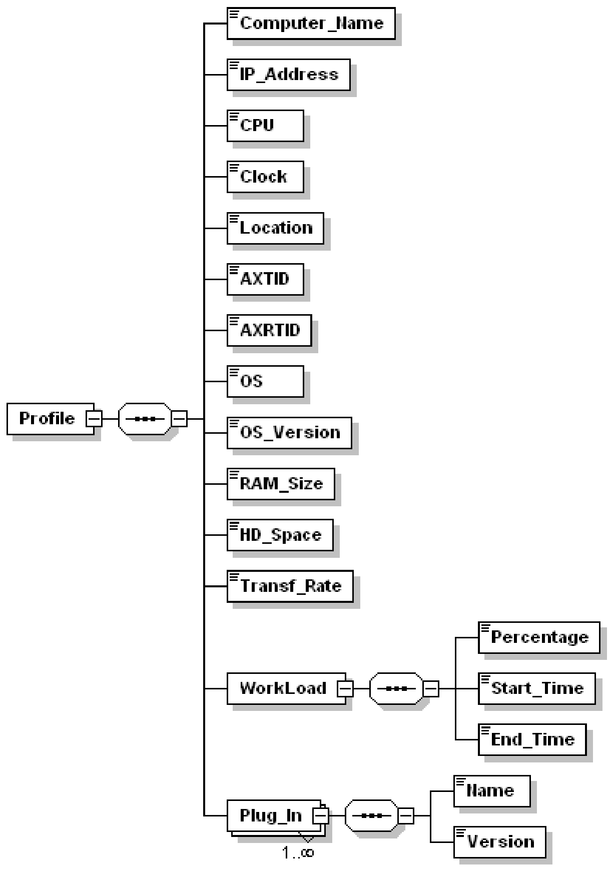
2.10.4 Executor Profile and XML formalisation

The executor profile is the set of metadata that allows to describe the executor in terms of:
1. Computational capabilities
2. Functionalities that the executor provides such as:
   - AXMEDIS Plug-In installed.
   - External tools Plug-In installed

The following table describes each metadata of profile:

Attributes	Description
Computer Name	Name of the computer hosting the rule executor
URL	IP address
AXTID	The ID of a specific instance related to the AXTID
AXRTID	The AXMEDIS Registered Tool Id associated with the Rule Executor application
SO	The Operating System
Version	The SO version
CPU	The type of CPU
Clock	The clock of CPU
RAM size	The amount of memory
HD-space	The amount of disk space available
Location	Where the computer is located
Workload	The percentage of availability during the time period
Transfer rate from AXDB	The network capability for transferring a file from the AXMEDIS Database to the rule executor machine.
Plug-In	It provides the name and the version. Since the executor can host many plug-in, this is field represents a list of Plug-In

The set of metadata is organised in the XML format as depicted in the following schema:



In the following, the textual representation of the XML schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v2004 rel. 3 U (http://www.xmlspy.com) by Paolo Nesi (University of Florence) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
    <xs:element name="Profile">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="Computer_Name" type="xs:string"/>
                <xs:element name="IP_Address" type="xs:anyURI"/>
                <xs:element name="CPU" type="xs:string"/>
                <xs:element name="Clock" type="xs:float"/>
                <xs:element name="Location" type="xs:string"/>
                <xs:element name="AXTID" type="xs:string"/>
                <xs:element name="AXRTID" type="xs:string"/>
                <xs:element name="OS" type="xs:string"/>
                <xs:element name="OS_Version" type="xs:string"/>
                <xs:element name="RAM_Size" type="xs:string"/>
                <xs:element name="HD_Space" type="xs:string"/>
                <xs:element name="Transf_Rate" type="xs:unsignedInt"/>
                <xs:element name="WorkLoad">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="Percentage" type="xs:float"/>
                            <xs:element name="Start_Time" type="xs:time"/>
                            <xs:element name="End_Time" type="xs:time"/>
                        </xs:sequence>
                    </xs:complexType>
                    </xs:element>
                    <xs:element name="Plug_In" maxOccurs="unbounded">
                        <xs:complexType>
                            <xs:sequence>
                                <xs:element name="Name" type="xs:time"/>
                                <xs:element name="Version" type="xs:string"/>
                            </xs:sequence>
                        </xs:complexType>
                    </xs:element>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
</xs:schema>
```

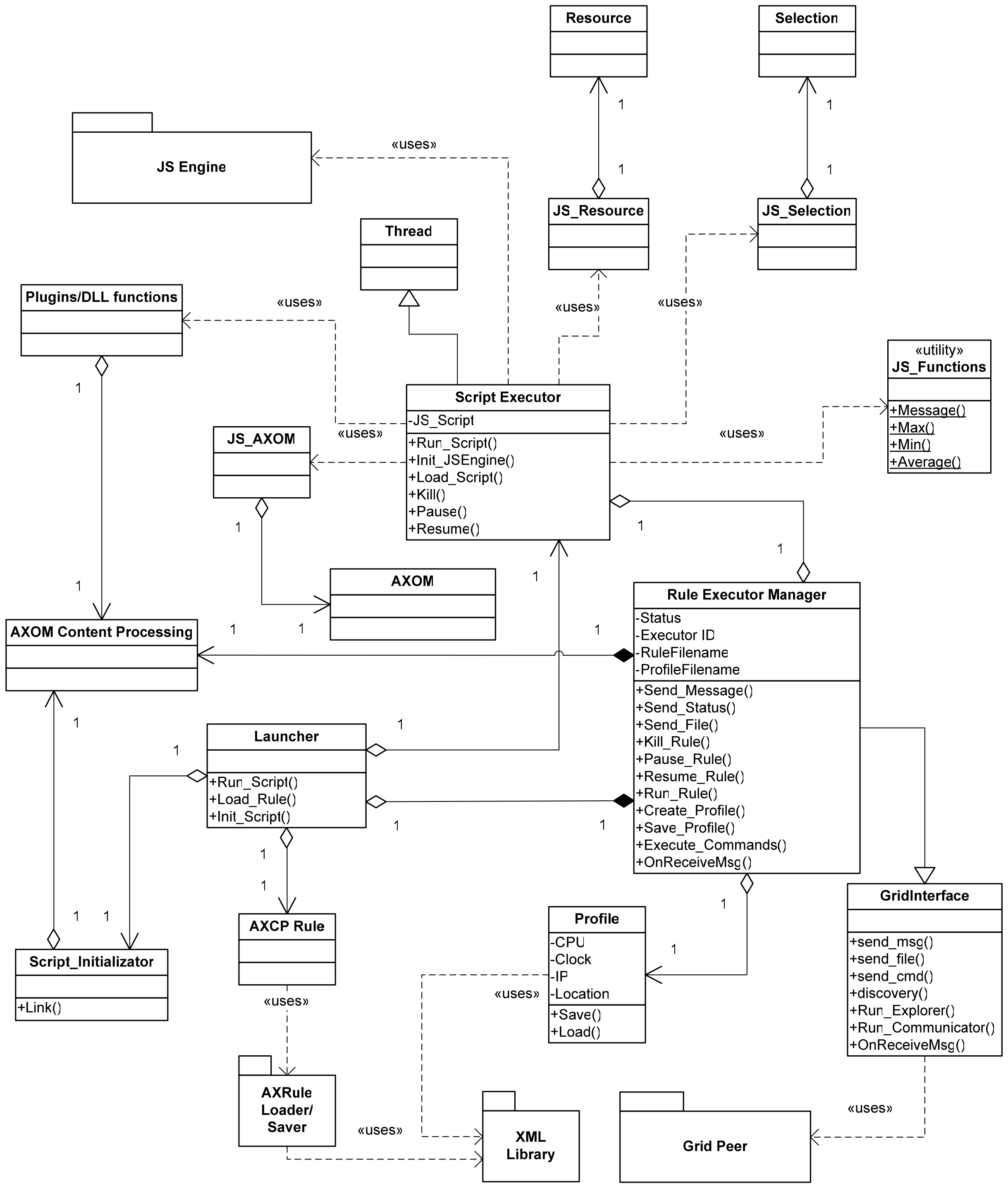
2.10.5 Rule Executor Configuration file

The rule executor will need some configuration parameters:

- **Disk Unit -** The logical disk unit to use for storage
- **Rule Path -** The path of the download folder where the executor will receive rules
- **Workload -** The specification of workload availability in terms of:
    - Percentage
    - Start Time
    - End Time
- **Location** – Where the computer is placed.

Plus other parameters that will be defined during the project life.

2.10.6 Rule Executor Class Diagram



3 The AXMEDIS DATA Type and Functions for JS

In addition to using the engine's built-in objects, it is possible to create, initialize, and use own JS objects and JS Functions. This is especially true if the JS engine is used with scripts to automate the application. Custom JS objects can provide direct program services, or they can serve as interfaces to program's services. For example, a custom JS object that provides direct service might be one that handles all of an application's network access, or might serve as an intermediary broker of database services. Or a JS object that mirrors data and functions that already exist in the application may provide an object-oriented interface to C code that is not otherwise, strictly-speaking, object-oriented itself. Such a custom object acts as an interface to the application itself, passing values from the application to the user, and receiving and processing user input before returning it to the application. Such an object might also be used to provide access control to the underlying functions of the application.

There are two ways to create custom objects that the JS engine can use:

- Write a JS script that creates an object, its properties, methods, and constructor, and then pass the script to the JS engine at run time.
- Embed code (wrapping) in the application that defines the object's properties and methods, call the engine to initialize a new object, and then set the object's properties through additional engine calls. An advantage of this method is that the application can contain native methods that directly manipulate the object embedding.

3.1 A JavaScript class in C++

In this section is reported an example of how to wrap a C++ class by embedding code without using inheritance ([2], [3]). For more details about JS API please see the official site of SpiderMonkey on Mozilla web page. The class used in the example is the following:

```
class Customer
{
public:
 int GetAge() { return m_age; }
 void SetAge(int newAge) { m_age = newAge; }
 std::string GetName() { return m_name; }
 void SetName(std::string newName) { m_name = newName; }

private:
 int m_age;
 std::string m_name;
};
```

3.1.1 Step 1 - The JavaScript class.

Create a new C++ class that derives from the C++ class you want to use in JavaScript or create a new C++ class which has a member of the type of that C++ class.

A class is defined in JavaScript with a JSClass structure. Create a static member of this type. Declare it as a public member, because this structure can be useful for other classes. It can be used by other classes to determine the type of an object. (see JS_InstanceOf API)

```
// JSCustomer.h
class JSCustomer
{
 public:
  JSCustomer() : m_pCustomer(NULL)
  {
  }

  ~JSCustomer()
  {
    delete m_pCustomer;
    m_pCustomer = NULL;
  }

  static JSClass customerClass;
```

```
    protected:
     void setCustomer(Customer *customer)
     {
       m_pCustomer = customer;
     }

     Customer* getCustomer()
     {
       return m_pCustomer;
     }

    private:
     Customer *m_pCustomer;

   };
```

The JSClass structure contains the name of the JavaScript class, some flags and the name of callbacks used by the engine. For example a callback is used when the engine needs to retrieve a property from your class. Define the JSClass structure in the implementation file of the C++ class as below.

```
// JSCustomer.cpp
JSClass JSCustomer::customerClass =
{
 "Customer", JSCLASS_HAS_PRIVATE,
 JS_PropertyStub, JS_PropertyStub,
 JSCustomer::JSGetProperty, JSCustomer::JSSetProperty,
 JS_EnumerateStub, JS_ResolveStub,
 JS_ConvertStub, JSCustomer::JSDestructor
};
```

The used callbacks are JSCustomer::JSGetProperty, JSCustomer::JSSetProperty and JSCustomer::JSDestructor. JSGetProperty is called when the engine needs a property, JSSetProperty is called when the engine sets a property and JSDestructor is called when the JavaScript object is destroyed.
The flag JSCLASS_HAS_PRIVATE is used so that the engine provides memory you can use to attach some data to a JavaScript object. You can use this to store a pointer to your class.
The callbacks are static member functions of the C++ class.

```
static JSBool JSGetProperty(JSContext *cx, JSObject *obj, jsval id, jsval *vp);
static JSBool JSSetProperty(JSContext *cx, JSObject *obj, jsval id, jsval *vp);
static JSBool JSConstructor(JSContext *cx, JSObject *obj, uintN argc,
                            jsval *argv, jsval *rval);
static void JSDestructor(JSContext *cx, JSObject *obj);
```

3.1.2 Step 2 - Initialize your JavaScript object

Create another static method called JSInit. See below for an example. This method will be called by the Application that creates the JavaScript runtime.

```
static JSObject *JSInit(JSContext *cx, JSObject *obj, JSObject *proto);
```

The implementation looks like this

```
JSObject *JSCustomer::JSInit(JSContext *cx, JSObject *obj, JSObject *proto)
{
  JSObject *newObj = JS_InitClass(cx, obj, proto, &customerClass,
                                  JSCustomer::JSConstructor, 0,
                                  NULL, JSCustomer::customer_methods,
                                  NULL, NULL);
  JS_DefineProperties(cx, newObj, JSCustomer::customer_properties);
  return newObj;
}
```

The static method JSConstructor will be called when your object is instantiated in a script. This method is very handy to attach your data to the object using the JS_SetPrivate API.

```
 JSBool JSCustomer::JSConstructor(JSContext *cx, JSObject *obj, uintN argc,
                                  jsval *argv, jsval *rval)
 {
   JSCustomer *p = new JSCustomer();
```

```
    p->setCustomer(new Customer());
    JS_SetPrivate(cx, obj, p);
    return JS_TRUE;
  }
```

This constructor method can have multiple arguments, which you can use to initialize your class. Now that you've created a pointer on the heap, you also need a way to destroy the pointer. This is done in the static method JS_Destructor.

```
void JSCustomer::JSDestructor(JSContext *cx, JSObject *obj)
{
  JSCustomer *p = JS_GetPrivate(cx, obj);
  delete p;
  p = NULL;
}
```

3.1.3 Step 3 - Adding properties

Add a static array member of the type JSPropertySpec. This array will contain the information of a property. Create also an enum for the property ids.

```
static JSPropertySpec customer_properties[];
enum
{
 name_prop,
 age_prop
};
```

Initialize this array in the implementation file as follows

```
JSPropertySpec JSCustomer::customer_properties[] =
{
  { "name", name_prop, JSPROP_ENUMERATE },
  { "age", age_prop, JSPROP_ENUMERATE },
  { 0 }
};
```

The last element of the array must be a null element. Each element contains another array with 3 elements. The first element is the name that will be used in JavaScript. The second is a unique id for the property. This will be passed to the callback functions. And the third one is a flag. JSPROP_ENUMERATE means that a script will see this property when it's enumerating the properties of the Customer object. You can also specify JSPROP_READONLY to indicate that the property can't be changed in the script.

Now you can implement the callbacks for getting and setting properties.

```
JSBool JSCustomer::JSGetProperty(JSContext *cx, JSObject *obj, jsval id, jsval *vp)
{
  if (JSVAL_IS_INT(id))
  {
    Customer *priv = (Customer *) JS_GetPrivate(cx, obj);
    switch(JSVAL_TO_INT(id))
    {
      case name_prop:

        break;
      case age_prop:
        *vp = INT_TO_JSVAL(priv->getCustomer()->GetAge());
        break;
    }
  }
  return JS_TRUE;
}

JSBool JSCustomer::JSSetProperty(JSContext *cx, JSObject *obj, jsval id, jsval *vp)
{
  if (JSVAL_IS_INT(id))
  {
    Customer *priv = (Customer *) JS_GetPrivate(cx, obj);
    switch(JSVAL_TO_INT(id))
    {
      case name_prop:
        break;
      case age_prop:
        priv->getCustomer()->SetAge(JSVAL_TO_INT(*vp));
        break;
    }
  }
  return JS_TRUE;
}
```

It's recommanded to return JS_TRUE in the property callbacks. When you return JS_FALSE a prototype will not be searched when the property is not found in your object.

3.1.4 Step 4 - Adding methods

Create a static member array of JSFunctionSpec type.

```
static JSFunctionSpec customer_methods[];
```

Initialize this array in the implementation file as follows

```
JSFunctionSpec wxJSFrame::wxFrame_methods[] =
{
  { "computeReduction", computeReduction, 1, 0, 0 },
  { 0 }
};
```

The last element of the array must always be a null element. Each element is another array with 5 elements. The first element is the name of the method that's used in the script. The second one is the name of a global or static member function. The third element is the number of arguments of this method. The last two elements are ignored.

Create a static method in the class

```
 static JSBool computeReduction(JSContext *cx, JSObject *obj, uintN argc,
                                jsval *argv, jsval *rval);
```

You return JS_TRUE when the function is successful. Otherwise you return JS_FALSE. The actual return of your JavaScript method is placed in the rval argument.

A sample implementation of this method

```
JSBool JSCustomer::computeReduction(JSContext *cx, JSObject *obj, uintN argc,
                                    jsval *argv, jsval *rval)
{
  JSCustomer *p = JS_GetPrivate(cx, obj);
  if ( p->getCustomer()->GetAge() < 25 )
    *rval = INT_TO_JSVAL(10);
  else
    *rval = INT_TO_JSVAL(5);
  return JS_TRUE;
}
```

3.1.5 An example

The following script uses the previously created object

```
var c = new Customer();
c.name = "Franky";
c.age = 32;
var reduction = c.computeReduction();
```

Don't forget to initialize the JavaScript object when you create the context:

```
JSObject *obj = JSCustomer::JSInit(cx, global);
```

3.2 Wrapping functions

To wrap a native function you use JS_DefineFunction or to register multiple functions with one API call JS_DefineFunctions.

```
/* Define a bunch of native functions first: */
static JSBool
my_abs(JSContext *cx, JSObject *obj, uintN argc, jsval *argv, jsval *rval)
{
  jsdouble x, z;

  if (!JS_ValueToNumber(cx, argv[0], &x))
    return JS_FALSE;
  z = (x < 0) ? -x : x;
  return JS_NewDoubleValue(cx, z, rval);
}

. . .

/*
 * Use a JSFunctionSpec array terminated with a null name to define a
 * bunch of native functions.
 */
static JSFunctionSpec my_functions[] = {
  /*    name          native          nargs    */
  {"abs",         my_abs,         1},
  {"acos",        my_acos,        1},
  {"asin",        my_asin,        1},
  . . .
  {0}
};

/*
 * Pass a particular object to define methods for it alone.  If you pass
 * a prototype object, the methods will apply to all instances past and
 * future of the prototype's class (see below for classes).
 */
JS_DefineFunctions(cx, globalObj, my_functions);
```

3.3 JS_AXOM (DSI)

JS_AXOM Class is the mapping of AXOM for JavaScript. According to the specification of the AXOM, JS_AXOM will provide and wrap AXOM methods that will permit to:

- Create an empty AXMEDIS object. The root of the object will be identified by the ElementId fixed to 0;
- Load an AXMEDIS object from the AXMEDIS database by using the AXOID
- Create/Remove an Element to the AXMEDIS object. The creation of an element will return an ElementId
- Get all ElementIds by giving the parent ElementId. It will return an array of ElementId
- Get Element type by giving the ElementId. It will return a string descriptor type.
- Add Resource, it will add a digital resource (audio, video, text, etc…) to a specific Element using the ElementId. It will return the ResourceId
- Remove a Resource (audio, video, text, etc…) by using the ResourceId
- Get a Resource by using the ResourceId. It will return a Resource object
- Add an AXInfo metadata object. It will return a MetadataId.
- Add a Dublin Core metadata object. It will return a MetadataId
- Add a generic XML metadata object by using a string containing an XML description. It will return a MetadataId
- Remove any metadata object by using the MetadataId.
- Store the AXMEDIS object on: file system or AXMEDIS Database

The ResourceId, ElementId and MetadataId will be integer numbers.

3.4 JS_Functions from AXOM_CONTENT_PROCESSING (DSI)

JS_AXOM_CONTENT_PROCESSING class will refer and wrap the AXOM Content Processing. It will provide and manage the access to the set of functions exposed by DLLs discovered by the Plugin Manager and related to AXMEDIS content production, protection, adaptation algorithms and tools.

Functions provided by the Axmedis Plugins for content processing

- *Fingerprint functions*
- *Digital Resource Adaptation functions*
- *Protection functions*
- *Metadata Adaptation functions*
- *Functions for using External tools*

3.5 JS_AXINFO (DSI)

JS_AXINFO will map and allow managing the metadata of the AXINFO in the JavaScript. This class manages the access to individual elements and fields in AXINFO metadata, this class will map all the functionalities provided by AxInfo class (see section 8.2.2.2 of Part A for a complete description).
It will allow to manage:

- ObjectCreator information
- Owner information
- Distributor information
- Object Status information
- PromoOf information
- Workflow information
- Fingerprints information
- PAR information
- Additional Metadata information
- Object History information

3.6 JS_DUBLIN_CORE (UNIVLEEDS)

JS_DUBLIN_CORE maps the metadata in the JavaScript. This class manages the access to individual elements and fields in the Dublin Core metadata (Get and Set methods).

Creating an DC object

- JSCreateDC(AXOID)
  - create an object of the Dublin Core

Composing and Editing a DC object for adaptation

- JSAddDCElement(“dc_element”, value)
  - Add elements to the DC object
- JSDeleteDCElement(“dc_element”, ref_num=1)
  - To delete a particular DC element. To delete all instances of
- JSSetDCElement
  - Set the text field related to the specified element
- JSGetDCElement
  - retrieve the text filed related to the specified element

Saving the new DC object

- JSUpdateDC()
  - Update the object to the originator

3.7 JS_Selection (DSI)

JS_Selection class maps the Selection in the JavaScript. This class allows using Selection objects to manage the access and to make queries to the AXMEDIS database, and to retrieve AXMEDIS objects ID (AXOID). It will manage the array of AXOID.

3.8 JS_Resource (DSI)

It could be whatever digital resource type: Image, video, animation, etc…They derive from classes that model the single resource in AXMEDIS object model.
This class will wrap the AxResource class (see section 8.2.2.7 of Part A).
It will provide functionalities to:

- access to the mime type
- access to the byte stream of the resource
- create a new resource and to embed a file inside a resource object

3.9 JS_CrawlerDB access (DSI with subcontract)

The Crawler class will provide all the functionalities exposed by its SOAP interface to:

- get a Focuseek File Format (FFF) version of a document giving the document id
- get the original document by using the document id
- perform queries on the crawler DB
- update a FFF version of a document
- and other functionality for crawler management

Moreover a class to access in a structured way to specific portions of FFF documents will be provided.

3.10 JS_Protection (FHGIGD)

The JS_Protection class provides the protection methods, which are needed by the rules editor, to the JavaScript.

The following functionalities are provided by JS_Protection as a JavaScript stub using AXOM and PMS:

- Requesting keys from PMS (encryption keys)
- Applying Protection to AXMEDIS object : encryption, scrambling, compression, FP.
- Creation of new Protection Information
- Sending the Protection Information (keys and parameter, see IPMP standard of MPEG21) to the database of the AXCS via the PMS

JS_ProtectionInfo
The protection information holds the IPMP information as stated in MPEG-21 Part 4 IPMP standard and more. The information may include:

- How each element of an AXMEDIS object has been protected, i.e. encrypted, encoded, compressed and scrambled.
- How each chunk of a resource has been protected, e.g. specifying that a given set of protection tools has to be applied from byte X to byte Y of a given resource (and not to the whole resource). In that way, different protection can be applied to a resource along its consumption.
- It is based on an XML schema which allows to describe sort of protection procedures as en-/decryption, (de-)compression, (de-)scrambling

The syntax and semantics is still under discussion, the actual state of the standard is contained in the output document w6772 of the 70th MPEG meeting (see http://mpeg.nist.gov/).
MPEG-21 Part 4 divides protection information into two XML schemas:

- one is used to declare the list of needed protection tools (or commands as defined in this section) to unprotect the whole digital item;

- the other is used to describe, for each protected element, how to use those tools (e.g. the execution order, keys, initialization parameters, etc…) to unprotect a specific element.

JS_ProtectionStamp
To identify the correct ProtectionInfo for an AXMEDIS object also the so-called ProtectionStamp is needed which binds an object to the different protection models that can be applied to the object.

3.11 JS_DRM (FHGIGD)

JS_DRM defines the DRM data type and methods for the Protection Tool Engine.

The following functionalities are provided by JS_DRM as a JavaScript stub using AXOM and PMS:

- Creating a new governed object (with license)
- Sending a License to the PMS and saving the license into the database
- Loading a License from the database via the License ID
- Loading a License Model from the database via the License Model ID
- Generating a license from license model and additional information (principal, AXOID)
- Check/Verification of an issued License against some RIGHTS written in clear such as: "the play on the AXOID 34 in July 2005 for 5 times, the print of AXOID 56 in Spain in May 2006 at least one, etc."
- Check/Verification if it is possible to issue/generate a License with some RIGHTS written in clear such as: "the play on the AXOID 34 in July 2005 for 5 times, the print of AXOID 56 in Spain in May 2006 at least one, etc."
- Check/Verification of existing PAR against some RIGHTS written in clear such as: "the play on the AXOID 34 in July 2005 for 5 times, the print of AXOID 56 in Spain in May 2006 at least one, etc."
- Addition of rights or removal from a license (license adaptation): Generation of a new license (with new or less rights) AND Revocation of the old licenses in ONE TRANSACION
- Addition of rights or removal from a PAR (PAR adaptation): Generation of a new PAR (with new or less rights) AND Revocation of the old licenses in ONE TRANSACION
- Check/Verification of license against PAR

In order to express DRM rules associated to AXMEDIS objects it has been decided to use MPEG-21 REL as primary rights expression language. A common structure is imposed for licenses and PAR.

JS_License – class that models a License
The following objects are needed to fully represent a *license* and all of its components in JavaScirpt:

- JS_License
- JS_Issuer
- JS_GrantGroup
- JS_Grant
- JS_Right
- JS_Principal
- JS_Resource
- JS_Condition
    - Fee
    - Territory
    - Number
    - Interval

Each *license* has an *issuer* and a *GrantGroup*. Each *GrantGroup* contains a set of *Grants*. Each *Grant* contains the information of the *right* granted, the *resource*, the *principal* and an optional set of *conditions* related to that right. In addition, we have to realise that a *resource* can be a *GrantGroup* (in case of Distributor Licenses).

For expressing the different types of conditions JS_CONDITION, the following information was defined:
- ConditionType: It indicates which kind of condition we are expressing.
- Five Tvalue fields and two NValue fields (more can be added if desired), whose values depend on the conditionType.

JS_PAR – class that models a PAR
The following objects are needed to fully represent a *PAR* and all of its components in JavaScript:
- JS_PAR
- JS_Grant
- JS_Right
- JS_Resource
- JS_Condition
    - Fee
    - Territory
    - Number
    - Interval

The relationship between the different objects and the meaning is the same as for JS_LICENSE.

3.12 JS_Publisher (CRS4)

It is the class loaded by the script engine and wrapped by a JS_Publisher Javascript class, so that its methods can be invoked in scripts.
- publish() publishes the actualized selection
- unpublish() un-publishes the actualized selection
- publishedObjects() returns the ID of objects published by the current instance of the Publisher

As Meatadata mapping is wrapped within loading process, mapping support is not required
For more details see document DE3.1.2F Framework and Tools Specifications (AXEPTool and Progr. and Pub.).

3.13 JS_Downloader(CRS4)

It is the class loaded by the script engine and wrapped by a JS_Downloader Javascript class, so that its methods can be invoked in scripts. It contains one fundamental method:
- download(), download the actualized selection of objects

As Meatadata mapping is wrapped within loading process, mapping support is not required
For more details see document DE3.1.2F Framework and Tools Specifications (AXEPTool and Progr. and Pub.).

3.14 JS_Loader (CRS4)

It is the class loaded by the script engine and wrapped by a JS_Loader Javascript class, so that its methods can be invoked in scripts.
- load() publishes the actualized selection
- loadedObjects() returns the ID of objects loaded by the current instance of the Loader

As Meatadata mapping is wrapped within loading process, mapping support is not required
For more details see document DE3.1.2F Framework and Tools Specifications (AXEPTool and Progr. and Pub.).

3.15 JS_Functions

JS Functions will be a set of auxiliary functions wrappred into Javascript. They will be divided into the following categories:

Statistical

- MAX – return the maximum value of an numerical array
- MIN - return the minimum value of an numerical array
- VAR – return the variance of an numerical array
- AVERAGE – return the average of an numerical array
- MODE – return the mode of an numerical array
- ….

Combinatorial

- Data Permutation
- Sort data

Set Management

- *Intersection* – $A \cap B$, it will return the list of common items
- *Union* – $A \cup B$, it will return the list of items as union of sets
- *Inclusion* – return true if $A \subseteq B \neq \emptyset$

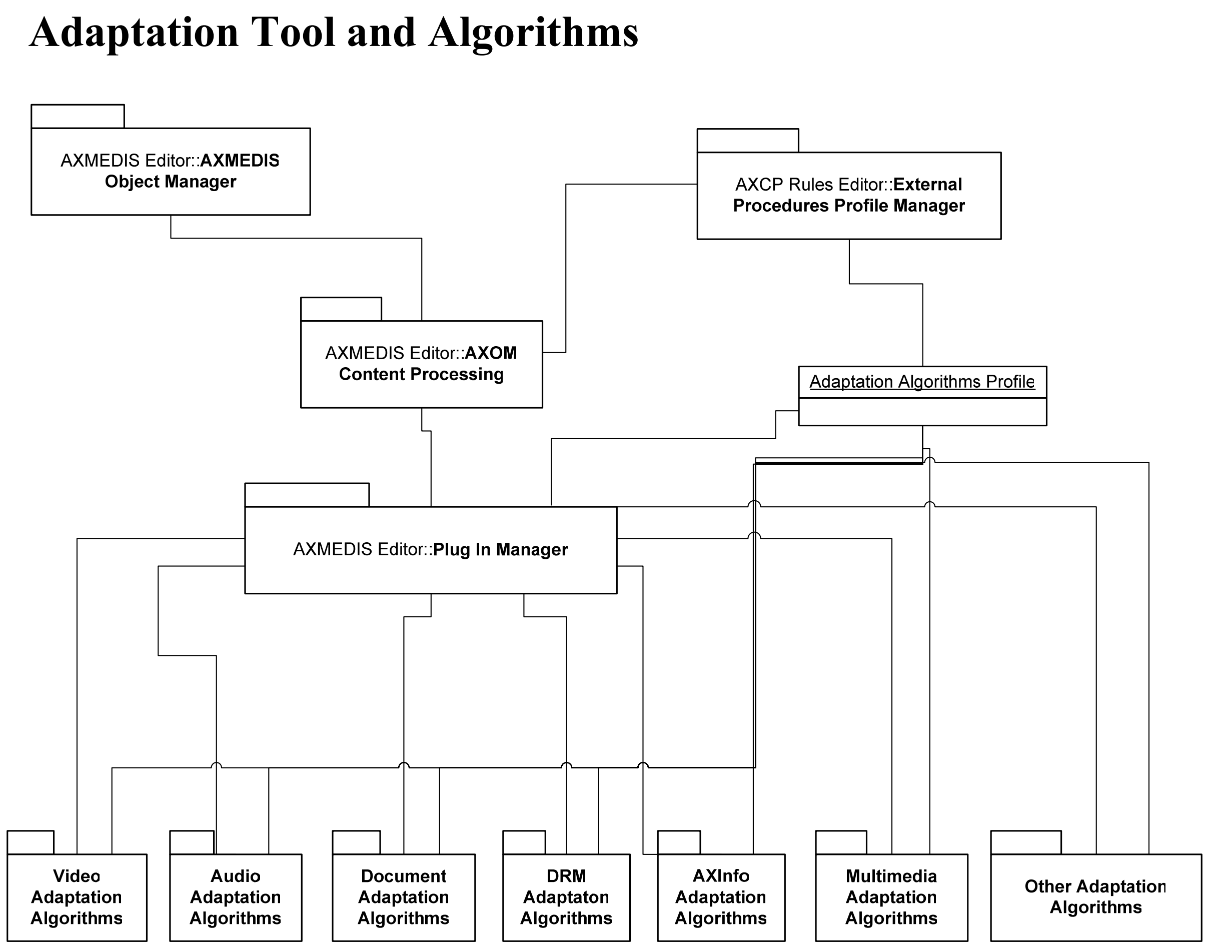
Generic

- File system functions – File exists, dir exists, create dir, etc…
- Communication functions – A *message()* function will provide the support for the communication via JavaScript. Messages will be routed via the Engine to the Axmedis Workflow Manager.
- Typeof function – it will return the type of data by returning the string that describes the type.

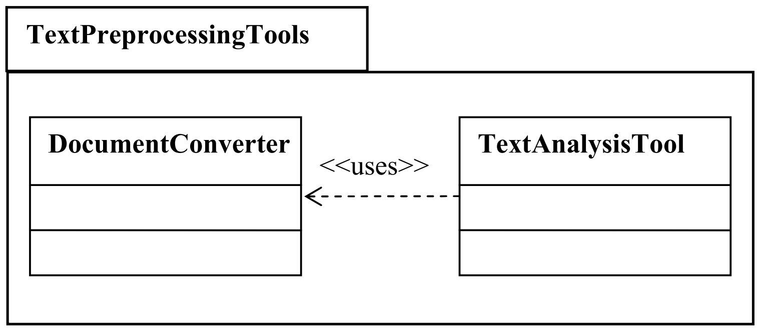
Other functions could be added during the life of the Project.

4 Adaptation Tools and Algorithms (DSI, EPFL, UNIVLEEDS)

In this section, tools and algorithms that will be used in the content adaptation task will be described. According to the UML diagram reported below, several content adaptation modules will be developed in order to cope with different types of contents, in particular: video, audio, document, multimedia, DRM, metadata. Each of them will be based on existing library, executable or others. Other and specific algorithms could be added during the life of project.


Adaptation Tool and Algorithms
AXMEDIS Editor::AXMEDIS Object Manager
AXCP Rules Editor::External Procedures Profile Manager
AXMEDIS Editor::AXOM Content Processing
Adaptation Algorithms Profile
AXMEDIS Editor::Plug In Manager
Video Adaptation Algorithms
Audio Adaptation Algorithms
Document Adaptation Algorithms
DRM Adaptaton Algorithms
AXInfo Adaptation Algorithms
Multimedia Adaptation Algorithms
Other Adaptation Algorithms

4.1 Tools and algorithms for Documents Adaptation (DIPITA, DSI, EXITECH)



IDL of the module that contains classes used to convert text documents an to extract low-level information from plain text files.

```
module TextPreprocessingTools {

       /** Text formats accepted by the tool
        *  Acronyms are taken from DE3-1-2J §4.6
        */
       enum DOCUMENT_FORMATS { PDF, HTML, TXT, RTF, PS };

       /** ****** TO BE REVIEWED ***********
        * DocumentFile is the input format of the document.
        * It's defined as a octet sequence just for being the
        * most general (BLOB format). It will be changed as soon as it will
        * be defined input types specifications for plugins.
        */
       typedef sequence <octet> DocumentFile;

       /** This interface provides the description of classes that are able to
        *  convert a text document.
        */
       interface DocumentConverter {

              /** The document to be analyzed */
              attribute DocumentFile document;

              /** Contains the original format of the document.
               * This attribute can be set in case the user
               * knows the document format.
               */
              attribute DOCUMENT_FORMATS type;
              /** Sometimes the document cannot be converted.
               *  E.g.: copy protected PDF, ill-formed RTF,...
               *  This attribute shows if the conversion returned a success value.
               */
              readonly attribute boolean conversion_success;
              /** Converts the document from its original format to
               *  the one specified by the 'out_type' parameter.
               *  'name_out' specifies the name of the new file.
               */
              DocumentFile convert(in DOCUMENT_FORMATS type_in, in DOCUMENT_FORMATS
type_out, in string name_out);
              /** Tries to guess the input document format and then
               *  converts it to the format specified by the 'out_type' parameter.
               *  'name_out' specifies the name of the new file.
               */
              DocumentFile guessInputFormatAndConvert(in DOCUMENT_FORMATS type_out, in
string name_out);
       };

       /** This interface provides some utility methods to
        *  manage plain text file and to get some technical metadata
        */
       interface TextAnalysisTool {

              /** Output type of text tokenization */
              typedef sequence <wstring> TextTokens;

              /** Sets the document to be analyzed */
              void setDocument(in DocumentFile doc2analyze);
              /** Delete all whitespaces and formatting chars
               *  from the plain text document, then returns it.
               */
              wstring getTextWithoutSpaces();
              /** Delete all punctuation chars from the plain
```

```
                 *  text document, then returns it.
                 */
                wstring getTextWithoutPunctuation();
                /** Delete all consonants chars from the plain text
                 *  document, then returns it.
                 */
                wstring getOnlyVowels();
                /** Delete all vowels chars from the plain text
                 *  document, then returns it.
                 */
                wstring getOnlyConsonants();
                /** Get the byte size of the plain text file */
                unsigned long getSizeInByte();
                /** Get the chars number of the plain text file */
                unsigned long getCharsNumber();
                /** Get the words of the plain text file */
                TextTokens getWords();
                /** Get the lines of the plain text file */
                TextTokens getLines();
                /** Get the periods of the plain text file */
                TextTokens getPeriods();
                /** Get the paragraphs of the plain text file */
                TextTokens getParagraphs();
                /** Get the graphic words number of the plain text file
                 */
                unsigned long getWordsNumber();
                /** Get lines number of the plain text file */
                unsigned long getLinesNumber();
                /** Get periods number of the plain text file */
                unsigned long getPeriodsNumber();
                /** Get the paragraphs number of the plain text file
                 */
                unsigned long getParagraphsNumber();
                /** Get the character encoding format of the plain text file.
                 * ******** TO BE ASSESSED ************
                 * Define which encodings are supported
                 * ************************************
                 */
                string getCharacterEncoding();
        };
};
```

WSDL of the module that contains classes used to convert text documents an to extract low-level information from plain text files.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2005 sp2 U (http://www.altova.com) by Marco Fabbri (LABLITA) -->
<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:tns="http://axmedis.org/wsdl/TextPreprocessingTool/"
xmlns:ns="http://axmedis.org/wsdl/TextPreprocessingTool/types/"
targetNamespace="http://axmedis.org/wsdl/TextPreprocessingTool/">
        <types>
                <xs:schema
targetNamespace="http://axmedis.org/wsdl/TextPreprocessingTool/types/"
xmlns="http://axmedis.org/wsdl/TextPreprocessingTool/types/"
elementFormDefault="unqualified" attributeFormDefault="unqualified">
                        <xs:element name="DocumentFile" type="xs:hexBinary"/>
                        <xs:element name="DOCUMENT_FORMATS">
                                <xs:simpleType>
                                        <xs:restriction base="xs:string">
                                                <xs:enumeration value="PDF"/>
                                        <xs:enumeration value="HTML"/>
                                        <xs:enumeration value="TXT"/>
                                        <xs:enumeration value="RTF"/>
                                        <xs:enumeration value="PS"/>
                                    </xs:restriction>
                                </xs:simpleType>
                            </xs:element>
                        </xs:schema>
                        <xs:schema
targetNamespace="http://axmedis.org/wsdl/TextPreprocessingTool/types/"
xmlns="http://axmedis.org/wsdl/TextPreprocessingTool/types/"
elementFormDefault="unqualified" attributeFormDefault="unqualified">
                            <xs:element name="HashSignature" type="xs:string"/>
                            <xs:complexType name="TextTokens">
                                <xs:sequence>
                                    <xs:element name="token" type="xs:string"/>
                                </xs:sequence>
                            </xs:complexType>
                        </xs:schema>
                    </types>
                    <message name="doc2analyze">
                        <part name="parameters" element="ns:DocumentFile"/>
                    </message>
                    <message name="signature_out">
                        <part name="parameters" element="ns:HashSignature"/>
                    </message>
                    <message name="docFormat_out">
                        <part name="parameter" element="ns:DOCUMENT_FORMATS"/>
                    </message>
                    <message name="docFormat_in">
                        <part name="parameter" element="ns:DOCUMENT_FORMATS"/>
                    </message>
                    <message name="conversion_success">
                        <part name="parameter" type="xs:string"/>
                    </message>
                    <message name="doc_out">
                        <part name="parameter" element="ns:DocumentFile"/>
                    </message>
                    <message name="convert_in">
                        <part name="type_out" element="ns:DOCUMENT_FORMATS"/>
                        <part name="type_in" element="ns:DOCUMENT_FORMATS"/>
                        <part name="name_out" type="xs:string"/>
                    </message>
                    <message name="guess_in">
                        <part name="type_out" element="ns:DOCUMENT_FORMATS"/>
                        <part name="name_out" element="" type="xs:string"/>
                    </message>
                    <message name="text_out">
                        <part name="parameters" type="xs:string"/>
                    </message>
                    <message name="number_out">
                        <part name="parameters" type="xs:unsignedLong"/>
                    </message>
                    <message name="toks_out">
                        <documentation>Output type of text tokenization</documentation>
                        <part name="parameters" type="ns:TextTokens"/>
                    </message>
                    <portType name="DocumentConverter">
                        <documentation>This interface provides the description of classes with which
                     the hash signature of a document can be calculated.
                     Different classes must be implemented in order to menage
                     binary and plain text file.
                    </documentation>
                        <operation name="getConversion_success">
                            <output message="tns:conversion_success"/>
                        </operation>
                        <operation name="setDocument">
                    <documentation>Sets the document to be analyzed</documentation>
                    <input message="tns:doc2analyze"/>
             </operation>
             <operation name="getDocument">
                    <output message="tns:doc2analyze"/>
             </operation>
             <operation name="guessInputFormatAndConvert">
                    <output message="tns:doc_out"/>
                    <input message="tns:guess_in"/>
             </operation>
             <operation name="convert">
                    <output message="tns:doc_out"/>
                    <input message="tns:convert_in"/>
             </operation>
      </portType>
      <portType name="TextAnalysisTool">
      <operation name="getTextWithoutSpaces">
                    <documentation>Delete all whitespaces and formatting chars
                    from the plain text document, then returns it.</documentation>
                    <output message="tns:text_out"/>
             </operation>
             <operation name="getTextWithoutPunctuation">
                    <documentation>Delete all punctuation chars from the plain
                    text document, then returns it.</documentation>
                    <output message="tns:text_out"/>
             </operation>
             <operation name="getOnlyVowels">
                    <documentation>Delete all consonants chars from the plain text
                    document, then returns it.</documentation>
                    <output message="tns:text_out"/>
             </operation>
             <operation name="getOnlyConsonants">
                    <documentation>Delete all vowels chars from the plain text
                    document, then returns it.</documentation>
                    <output message="tns:text_out"/>
             </operation>
             <operation name="getSizeInByte">
                    <documentation>Get the byte size of the plain text
file.</documentation>
                    <output message="tns:number_out"/>
             </operation>
             <operation name="getCharsNumber">
                    <documentation>Get the chars number of the plain text
file.</documentation>
                    <output message="tns:number_out"/>
             </operation>
             <operation name="getWords">
                    <documentation>Get the words of the plain text file.</documentation>
                    <output message="tns:toks_out"/>
             </operation>
             <operation name="getLines">
                    <documentation>Get the lines of the plain text file.</documentation>
                    <output message="tns:toks_out"/>
             </operation>
             <operation name="getPeriods">
                    <documentation>Get the periods of the plain text
file.</documentation>
                    <output message="tns:text_out"/>
             </operation>
             <operation name="getParagraphs">
                    <documentation>Get the paragraphs of the plain text
file.</documentation>
                    <output message="tns:text_out"/>
             </operation>
             <operation name="getWordsNumber">
                    <documentation>Get the graphic words number of the plain text
file.</documentation>
```

```
                        <output message="tns:number_out"/>
                </operation>
                <operation name="getLinesNumber">
                        <documentation>Get lines number of the plain text
file.</documentation>
                        <output message="tns:text_out"/>
                </operation>
                <operation name="getPeriodsNumber">
                        <documentation>Get periods number of the plain text
file.</documentation>
                        <output message="tns:text_out"/>
                </operation>
                <operation name="getParagraphsNumber">
                        <documentation>Get the paragraphs number of the plain text
file.</documentation>
                        <output message="tns:text_out"/>
                </operation>
                <operation name="getCharacterEncoding">
                        <documentation>Get the character encoding format of the plain text
file.
                        * ******** TO BE ASSESSED ************
                        * Define which encodings are supported
                        * ************************************
                        </documentation>
                        <output message="tns:text_out"/>
                </operation>
        </portType>
        <binding name="bind_DocumentConverter" type="tns:DocumentConverter">
                <operation name="getConversion_success">
                        <output/>
                </operation>
                <operation name="setDocument">
                        <input/>
                </operation>
                <operation name="getDocument">
                        <output/>
                </operation>
                <operation name="guessInputFormatAndConvert">
                        <input/>
                        <output/>
                </operation>
                <operation name="convert">
                        <output/>
                        <input/>
                </operation>
        </binding>
        <binding name="bind_TextAnalysisTool" type="tns:TextAnalysisTool">
        <operation name="getTextWithoutSpaces">
                        <output/>
                </operation>
                <operation name="getTextWithoutPunctuation">
                        <output/>
                </operation>
                <operation name="getOnlyVowels">
                        <output/>
                </operation>
                <operation name="getOnlyConsonants">
                        <output/>
                </operation>
                <operation name="getSizeInByte">
                        <output/>
                </operation>
                <operation name="getCharsNumber">
                        <output/>
                </operation>
                <operation name="getWords">
                        <output/>
```

```
            </operation>
            <operation name="getLines">
                  <output/>
            </operation>
            <operation name="getPeriods">
                  <output/>
            </operation>
            <operation name="getParagraphs">
                  <output/>
            </operation>
            <operation name="getWordsNumber">
                  <output/>
            </operation>
            <operation name="getLinesNumber">
                  <output/>
            </operation>
            <operation name="getPeriodsNumber">
                  <output/>
            </operation>
            <operation name="getParagraphsNumber">
                  <output/>
            </operation>
            <operation name="getCharacterEncoding">
                  <output/>
            </operation>
            </binding>
      <service name="serviceName"/>
</definitions>
```

Module Profile		
Tools and Algorithms for Document Adaptation		
Executable or Library(Support)	Library	
Single Thread or Multithread		
Language of Development	C++	
Responsible Name	Moneglia	
Responsible Partner	DIPITA	
Status (proposed/approved)	Purposed	
Platforms supported	MS WINDOWS, Mac OS X, Linux	
Interfaces with other tools:	Name of the communicating tools	Communication model and format (protected or not, etc.)
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	License status: GPL. LGPL. PEK,

		proprietary, authorized or not
DOCFRAC		LGPL
GNU Ghostscript		GPL
XPDF		GPL
HTMLDOC		GPL

Document adaptation tools provide functions which can convert a text document file modifying its format and can extract some low-level information. TextPreprocessingTools is the module containing classes which will be used to deal with document adaptation: it's the same module described in DE3-1-2D §2.6.1.
As described in that section some libraries distributed under GPL will be used, so the conversion part of the tool will be developed as a separated executable program: this part (that will be distributed under GPL) will contain code that will use those libraries while the plug-in will make calls to the converter and won't make direct calls to the libraries so it has not to be licensed under GPL.
The communication protocol between the two parts of the tool has to be defined.

The following conversion libraries will be used for text document conversion:
DOCFRAC (http://docfrac.sourceforge.net/)
Conversion Formats

- RTF to HTML
- RTF to TEXT
- HTML to RTF
- HTML to TEXT
- TEXT to RTF
- TEXT to HTML

Uses

- converting many documents at a time;
- active web pages; and
- converting output from Microsoft's Internet Explorer RTF control to HTML.

Platforms

- Windows;
- Linux command line; and
- programming kit (ActiveX and DLL).

DocFrac is free. It is released under the LGPL.

GNU Ghostscript (http://www.cs.wisc.edu/~ghost/)
Ghostscript is the name of a set of software that provides:

- An interpreter for the PostScript (TM) language and the Adobe Portable Document Format, and
- A set of C procedures (the Ghostscript library) that implement the graphics and filtering (data compression / decompression / conversion) capabilities that appear as primitive operations in the PostScript language and in PDF.

Versions entitled "GNU Ghostscript" are distributed with the GNU General Public License.

XPDF (http://www.foolabs.com/xpdf/)
Xpdf is an open source viewer for Portable Document Format (PDF) files. The Xpdf project also includes a PDF text extractor, PDF-to-PostScript converter, and various other utilities.
Xpdf runs under the X Window System on UNIX, VMS, and OS/2. The non-X components (pdftops, pdftotext, etc.) also run on Win32 systems and should run on pretty much any system with a decent C++ compiler.
Xpdf is designed to be small and efficient. It can use Type 1, TrueType, or standard X fonts.
Xpdf is licensed under the GNU General Public License (GPL), version 2.

HTMLDOC (http://www.easysw.com/htmldoc/)
HTMLDOC converts HTML source files into indexed HTML, PostScript, or Portable Document Format (PDF) files that can be viewed online or printed.
The program is free software and is distributed under GPL.

4.2 Tools and algorithms for Video Adaptation (EPFL)

Module Profile Tools and Algorithms for Video Adaptation		
Executable or Library(Support)	Library	
Single Thread or Multithread	Single	
Language of Development	C++	
Responsible Name	Schmucker	
Responsible Partner	FHGIGD	
Status (proposed/approved)	Purposed	
Platforms supported	Linux, Max OS, Win32 (MinGW)	
Interfaces with other tools:	Name of the communicating tools	Communication model and format (protected or not, etc.)
File Formats Used	Shared with	File format name or reference to a section
See the tables below		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
Simple direct call		
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not
FFMPEG		LGPL
FOBS		LGPL

Format transcoding is one of the main adaptation functions needed by the AXMEDIS Framework as it implies bitrate reduction when transcoding among compressed formats. In the case of video objects (but this also applies to audio and multimedia objects), the FFMPEG and the FOBS library may be used.

FFMPEG:

FFMPEG is a complete solution to record, convert and stream audio and video. It is developed under Linux but it can be operated under most operating systems, including Windows.

FFMPEG provides a C API and two libraries:

- Libavcodec: a library containing all the FFMPEG audio/video encoders and decoders; most codecs were developed from scratch to ensure best performances and high code reusability;
- Libavformat: a library containing parsers and generators for all common audio/video formats.

FFMPEG is licensed under LGPL. However, it incorporates several modules that are covered under the GPL, notably liba52 (a library for decoding ATSC A/52 streams) and libpostproc (a library for post-processing). If these components are used in a project, then the all project should be distributed under the GPL. Yet, it is possible to avoid linking to these GPL libraries ensuring a full LGPL use of FFMPEG.

FOBS:

FOBS is an object-oriented wrapper for FFMPEG library (FOBS are Ffmeg OBjectS). It is a set of object oriented APIs to deal with media. It relies on the FFMPEG library but provides a much simpler programming interface. FOBS is currently available in C++ and has been successfully tested in a range of platforms (Linux, Max OS, Win32 (MinGW)). FOBS is released under the LGPL licence.

Here is a short C++ example of how to transcode a video file using FOBS' API. The example transcodes an "avi" file into an "mp4". The examples illustrates how to set the parameters of the transcoded output file (bit rate, frame rate, number of audio channels…).

```
#include <iostream>
#include "transcoder.h"

int main()
{
        omnivedia::fobs::returnCode         error;
        std::string                         inputFile( "test.avi" );
        std::string                         outputFile( "test.mp4" );

        // create transcoder object
        omnivedia::fobs::transcoder         t ( inputFile.c_str(), outputFile.c_str() );

        // choose output video codec:
        //      - width:        352 pixels
        //      - height:       288 pixels
        //      - bit rate:     400 kb/s
        //      - frame rate:   25 f/s
        //      - codec:        msmpeg4
        error = t.chooseVideoCodex( 352, 288, 400, 25, "msmpeg4" );
        if( isError( error ) ) {
                std::cout << "Error choosing video codec" << std::endl;
                exit(-1);
```

```
            }

            // choose output audio codec:
            //          - samples per second:           44100
            //          - number of channels:           2
            //          - bit rate:                     64 kb/s
            //          - codec:                        mp2
            error = t.chooseAudioCodec( 44100, 2, 64, “mp2” );
            if( isError( error ) ) {
                        std::cout << “Error choosing audio codec” << std::endl;
                        exit(-1);
            }

            // choose output file format:
            //          - output file format:           mp4
            error = t.chooseFormat( “mp4” );
            if( isError( error ) ) {
                        std::cout << “Error choosing file format” << std::endl;
                        exit(-1);
            }

            // perform the actual transcoding:
            error = t.transcode();
            if( isError( error ) ) {
                        std::cout << “Error in transcoding” << std::endl;
                        exit(-1);
            }

            return 0;

}
```

Here is a list of the file formats supported by FFMPEG through the libavformat library; “X” means that encoding (resp. decoding) is supported:

Supported File Format	**Encoding**	**Decoding**	**Comments**
MPEG audio	X	X	
MPEG 1 systems	X	X	Muxed audio and video
MPEG 2 PS	X	X	Also known as VOB file
MPEG 2 TS		X	Also known as DVB transport stream
ASF	X	X	
AVI	X	X	
WAV	X	X	
Macromedia flash	X	X	Only embedded audio is decoded
FLV	X	X	Macromedia flash video files
Real audio and video	X	X	
Raw AC3	X	X	

Raw MJPEG	X	X	
Raw MPEG video	X	X	
Raw PCM 8/16 bits, mulaw/Alaw	X	X	
Raw CRI ADX audio	X	X	
SUN AU format	X	X	
NUT	X	X	NUT open container format
Quicktime	X	X	
MPEG4	X	X	MPEG4 is a variant of Quicktime
Raw MPEG4 video	X	X	
DV	X	X	
4xm		X	4X Technologies format, used in some games
Playstation STR		X	
Id RoQ		X	Used in Quake III, Jedi Knight II, other computer games
Interplay MVE		X	Format used in various Interplay computer games
WC3 Movie		X	Multimedia format used in Origin’s Wing Commnader III computer game
Sega FILM/CPK		X	Used in many Sega Saturn console games
Westwood Studios VQA/AUD		X	Multimedia formats used in Westwood Studios games
Id Cinematic (.cin)		X	Used in Quake II
FLIC format		X	.fli / ,flc files
Sierra VMD		X	Used in Sierra CD-ROM games
Sierra Online		X	.sol files used in Sierra Online games
Matroska		X	
Electronic Arts Multimedia		X	Used in various EA games; files have extensions like WVE and UV2

Furthermore, FFMPEG can read and write images for each frame of a video sequence. The following image formats are supported:

Supported Image Format	**Encoding**	**Decoding**	**Comments**
PGM, PPM	X	X	
PAM	X	X	PAM is a PNM extension with alpha support
PGMYUV	X	X	PGM with U and V components in YUV 4:2:0
JPEG	X	X	Progressive JPEG is not supported
.Y.U.V.	X	X	One raw file per component
Animated GIF	X	X	Only uncompressed GIF are generated
PNG	X	X	2 bit and 4 bit/pixel not supported yet
SGI	X	X	SGI RGB image format

Here is a list of video codecs supported by FFMPEG through the libavcodec library:

Supported Codec	**Encoding**	**Decoding**	**Comments**
MPEG1 video	X	X	
MPEG2 video	X	X	
MPEG4	X	X	Also known as DIVX 4/5
MSMPEG4 V1	X	X	
MSMPEG4 V2	X	X	
MSMPEG4 V3	X	X	Also known as DIVX 3
WMV7	X	X	
WMV8	X	X	Not completely working
H. 261	X	X	
H. 263 (+)	X	X	Also known as Real Video 1.0
H. 264		X	
MJPEG	X	X	
Lossless MJPEG	X	X	
Apple MJPEG-B		X	
Sunplus MJPEG		X	Fourcc: SP5X
DV	X	X	
Huff YUV	X	X	
FFmpeg video 1	X	X	Experimental lossless codec (fourcc: FFV1)
FFmpeg snow	X	X	Experimental wavelet codec (fourcc: SNOW)
Asus v1	X	X	Fourcc: ASV1
Asus v2	X	X	Fourcc: ASV2
Creative YUV		X	Fourcc: CYUV
Sorenson Video 1	X	X	Fourcc: SVQ1
Sorenson Video 3		X	Fourcc: SVQ3
On2 VP3		X	Still experimental
Theora		X	Still experimental
Intel Indeo 3		X	Only works on i386 right now
FLV	X	X	Sorenson H. 263used in Flash
ATI VCR1		X	Fourcc: VCR1
ATI VCR2		X	Fourcc: VCR2
Cirrus Logic AccuPak		X	Fourcc: CLJR
4X video		X	Used in certain computer games
Sony Playstation MDEC		X	
Id RoQ		X	Used in Quake III, Jedi Knight II, other computer games
Xan/WC3		X	Used in Wing Commander III .MVE files
Interplay video		X	Used in Interplay .MVE files
Apple Animation		X	Fourcc: 'rle'
Apple Graphics		X	Fourcc: 'smc'
Apple Video		X	Fourcc: rpza
Apple Quickdraw		X	Fourcc: qdrw
Cinepak		X	
Microsoft RLE		X	
Microsoft Video-1		X	
Westwood VQA		X	
Id Cinematic Video		X	Used in Quake II
Planar RGB		X	Fourcc: 8BPS
FLIC Video		X	
Duck TrueMotion v1		X	Fourcc: DUCK

VMD Video		X	Used in Sierra VMD files
MSZH		X	Part of LCL
ZLIB	X	X	Part of LCL, encoder experimental
TechSmith Camtasia		X	Fourcc: TSCC
IBM Ultimotion		X	Fourcc: ULTI
Miro VideoXL		X	Fourcc: VIXL

4.3 Tools and algorithms for Images Adaptation (DSI)

Module Profile **Tools and Algorithms for Images Adaptation**		
Executable or Library(Support)	Library	
Single Thread or Multithread	Single	
Language of Development	C++	
Responsible Name	Ivan Bruno	
Responsible Partner	DSI	
Status (proposed/approved)		
Platforms supported	Windows and probably also LINUX and MAC	
Interfaces with other tools:	Name of the communicating tools	Communication model and format (protected or not, etc.)
File Formats Used	Shared with	File format name or reference to a section
Many formats see the following table		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
Simple direct call		C+ compiler libs
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not
Image Magik	Version 6.1.9.4	Similar to LGPL, ImageMagick is available for free, may be used to support both open and proprietary applications, and may be redistributed without fee.

This module will give the possibility to use algorithms and tools for adaptation of images. The main adaptation functions needed by the AXMEDIS Framework could be summarised in:

- Scaling
- Resolution improvement/reduction
- Colour to Greyscale conversions
- Format transcoding
- Composition with other images
- Adding widgets and graphic motifs

- Text drawing
- Image decomposition

These functions will be implemented by defining specific algorithms or using graphic libraries. An example of library is given by the *ImageMagick Library*.

The ImageMagick Library
ImageMagick^{TM}, version 6.1.9-4 (http://www.imagemagick.org), is a free software suite for the creation, modification and display of bitmap images. It can read, convert and write images in a large variety of formats. Images can be cropped, colors can be changed, various effects can be applied, images can be rotated and combined, and text, lines, polygons, ellipses and Bézier curves can be added to images and stretched and rotated. ImageMagick is free software: it is delivered with full source code and can be freely used, copied, modified and distributed. Its license is compatible with the GPL. ImageMagick is available for free, may be used to support both open and proprietary applications, and may be redistributed without fee. It runs on all major operating systems. Most of the functionality of ImageMagick can be used interactively from the command line; more often, however, the features are used from programs written in the programming languages Perl, C, C++, Python, PHP, Ruby or Java, for which ready-made ImageMagick interfaces (PerlMagick, Magick++, PythonMagick, MagickWand for PHP, RubyMagick, and JMagick) are available. This makes it possible to modify or create images automatically and dynamically.

Features and Capabilities - Here are just a few examples of what ImageMagick can do:

- Convert an image from one format to another (e.g. TIFF to JPEG)
- Resize, rotate, sharpen, color reduce, or ali special effects to an image
- Create a montage of image thumbnails
- Create a transparent image suitable for use on the Web
- Turn a group of image into a GIF animation sequence
- Create a composite image by combining several separate image
- Draw shapes or text on an image
- Decorate an image with a border or frame
- Describe the format and characteristics of an image

ImageMagick includes a number of ready-made ImageMagick interfaces. This makes it possible to modify or create images automatically and dynamically. The following table shows supports to different programming languages.

Programming language	Tool/library
C	Use MagickWand to convert, compose, and edit images from the C language. There is also the low-level MagickCore library but is only recommended for wizard-level developers.
C++	Magick++ provides an object-oriented C++ interface to ImageMagick.
Java	JMagick provides an object-oriented Java interface to ImageMagick.
Perl	Use PerlMagick to convert, compose, and edit images from the Perl language.
PHP	MagickWand for PHP a native PHP-extension to the ImageMagick MagickWand API.
Python	PythonMagick an object-oriented Python interface to ImageMagick.
Ruby	RubyMagick is an interface between the Ruby programming language and the ImageMagick image processing libraries.

Supported Image Formats - ImageMagick supports reading over 90 major file formats (not including sub-formats). The following table provides a summary of the supported image formats. The *Mode* column reports the availability to read and/or write the format.

Tag	Mode	Description	Notes
ART	R	PFS: 1st Publisher	Format originally used on the Macintosh (MacPaint?) and later used for PFS: 1st Publisher clip art.
AVI	R	Microsoft Audio/Visual Interleaved	
AVS	RW	AVS X image	
BMP	RW	Microsoft Windows bitmap	
CGM	R	Computer Graphics Metafile	Requires ralcgm to render CGM files.
CIN	RW	Kodak Cineon Image Format	Cineon Image Format is a subset of SMTPE DPX.
CMYK	RW	Raw cyan, magenta, yellow, and black samples	Set -size and -depth to specify the image width, height, and depth.
CMYKA	RW	Raw cyan, magenta, yellow, black, and alpha samples	Set -size and -depth to specify the image width, height, and depth.
CUR	R	Microsoft Cursor Icon	
CUT	R	DR Halo	
DCM	R	Digital Imaging and Communications in Medicine (DICOM) image	Used by the medical community for images like X-rays.
DCX	RW	ZSoft IBM PC multi-page Paintbrush image	
DIB	RW	Microsoft Windows Device Independent Bitmap	DIB is a BMP file without the BMP header. Used to support embedded images in compound formats like WMF.
DPX	RW	Digital Moving Picture Exchange	
EMF	R	Microsoft Enhanced Metafile (32-bit)	Only available under Microsoft Windows.
EPDF	RW	Encapsulated Portable Document Format	
EPI	RW	Adobe Encapsulated PostScript Interchange format	Requires Ghostscript to read.
EPS	RW	Adobe Encapsulated PostScript	Requires Ghostscript to read.
EPS2	W	Adobe Level II Encapsulated PostScript	Requires Ghostscript to read.
EPS3	W	Adobe Level III Encapsulated PostScript	Requires Ghostscript to read.
EPSF	RW	Adobe Encapsulated PostScript	Requires Ghostscript to read.
EPSI	RW	Adobe Encapsulated PostScript Interchange format	Requires Ghostscript to read.
EPT	RW	Adobe Encapsulated PostScript Interchange format with TIFF preview	Requires Ghostscript to read.
FAX	RW	Group 3 TIFF	See TIFF format. Note that FAX machines use non-square pixels which are 1.5 times wider than they are tall but computer displays use square pixels so FAX images may appear to be narrow unless they are explicitly resized using a resize specification of "150x100%".
FIG	R	FIG graphics format	Requires TransFig.
FITS	RW	Flexible Image Transport	

		System	
FPX	RW	FlashPix Format	Requires FlashPix SDK.
GIF	RW	CompuServe Graphics Interchange Format	8-bit RGB PseudoColor with up to 256 palette entires. Specify the format "GIF87" to write the older version 87a of the format.
GPLT	R	Gnuplot plot files	Requires gnuplot3.5.tar.Z or later.
GRAY	RW	Raw gray samples	Use -size and -depth to specify the image width, height, and depth.
HPGL	R	HP-GL plotter language	Requires hp2xx-3.2.0.tar.gz
HTML	RW	Hypertext Markup Language with a client-side image map	Also known as "HTM". Requires html2ps to read.
ICO	R	Microsoft icon	Also known as "ICON".
JBIG	RW	Joint Bi-level Image experts Group file interchange format	Also known as "BIE" and "JBG". Requires jbigkit-1.0.tar.gz.
JNG	RW	Multiple-image Network Graphics	JPEG in a PNG-style wrapper with transparency. Requires libjpeg and libpng-1.0.2 or later, libpng-1.2.5 or later recommended.
JP2	RW	JPEG-2000 JP2 File Format Syntax	Requires jasper-1.600.0.zip
JPC	RW	JPEG-2000 Code Stream Syntax	Requires jasper-1.600.0.zip
JPEG	RW	Joint Photographic Experts Group JFIF format	Requires jpegsrc.v6b.tar.gz
MAN	R	Unix reference manual pages	Requires that GNU groff and Ghostcript are installed.
MAT	R	MATLAB image format	
MIFF	RW	Magick image file format	Open ImageMagick's own image format (with ASCII header) which ensures that no image attributes understood by ImageMagick are lost.
MONO	RW	Bi-level bitmap in least-significant-byte first order	
MNG	RW	Multiple-image Network Graphics	A PNG-like Image Format Supporting Multiple Images, Animation and Transparent JPEG. Requires libpng-1.0.2 or later, libpng-1.2.5 or later recommended.
MPEG	RW	Motion Picture Experts Group file interchange format (version 1)	Requires mpeg2vidcodec_v12.tar.gz.
M2V	RW	Motion Picture Experts Group file interchange format (version 2)	Requires mpeg2vidcodec_v12.tar.gz.
MPC	RW	Magick Persistent Cache image file format	The native "in-memory" ImageMagick uncompressed file format. This file format is identical to that used by Open ImageMagick to represent images in memory and is read in "zero time" via memory mapping. The MPC format is not portable and is not suitable as an archive format. It is suitable as an intermediate format for high-performance image processing. The MPC format requires two files to support one image. When writing the MPC format, a file with extension ".mpc" is used to store information about the image, while a file with extension ".cache" stores the image pixels. The storage space required by a MPC image (or an image in memory) may be calculated by the equation (5*QuantumDepth*Rows*Columns)/8.

MSL	RW	Magick Scripting Language	MSL is the XML-based scripting language supported by the conjure utility.
MTV	RW	MTV Raytracing image format	
MVG	RW	Magick Vector Graphics.	The native ImageMagick vector metafile format. A text file containing vector drawing commands accepted by convert's -draw option.
OTB	RW	On-the-air Bitmap	
P7	RW	Xv's Visual Schnauzer thumbnail format	
PALM	RW	Palm pixmap	
PBM	RW	Portable bitmap format (black and white)	
PCD	RW	Photo CD	The maximum resolution written is 768x512 pixels since larger images require huffman compression (which is not supported).
PCDS	RW	Photo CD	Decode with the sRGB color tables.
PCL	W	HP Page Control Language	For output to HP laser printers.
PCX	RW	ZSoft IBM PC Paintbrush file	
PDB	RW	Palm Database ImageViewer Format	
PDF	RW	Portable Document Format	Requires Ghostscript to read.
PFA	R	Postscript Type 1 font (ASCII)	Opening as file returns a preview image.
PFB	R	Postscript Type 1 font (binary)	Opening as file returns a preview image.
PGM	RW	Portable graymap format (gray scale)	
PICON	RW	Personal Icon	
PICT	RW	Apple Macintosh QuickDraw/PICT file	
PIX	R	Alias/Wavefront RLE image format	
PNG	RW	Portable Network Graphics	Requires libpng-1.0.2 or later, libpng-1.2.5 or later recommended.
PNM	RW	Portable anymap	PNM is a family of formats supporting portable bitmaps (PBM) , graymaps (PGM), and pixmaps (PPM). There is no file format associated with pnm itself. If PNM is used as the output format specifier, then ImageMagick automatically selects the most appropriate format to represent the image. The default is to write the binary version of the formats. Use +compress to write the ASCII version of the formats.
PPM	RW	Portable pixmap format (color)	
PS	RW	Adobe PostScript file	Requires Ghostscript to read.
PS2	RW	Adobe Level II PostScript file	Requires Ghostscript to read.
PS3	RW	Adobe Level III PostScript file	Requires Ghostscript to read.
PSD	RW	Adobe Photoshop bitmap file	

PTIF	RW	Pyramid encoded TIFF	Multi-resolution TIFF containing successively smaller versions of the image down to the size of an icon. The desired sub-image size may be specified when reading via the -size option.
PWP	R	Seattle File Works multi-image file	
RAD	R	Radiance image file	Requires that *ra_ppm* from the Radiance software package be installed.
RGB	RW	Raw red, green, and blue samples	Use -size and -depth to specify the image width, height, and depth.
RGBA	RW	Raw red, green, blue, and alpha samples	Use -size and -depth to specify the image width, height, and depth.
RLA	R	Alias/Wavefront image file	
RLE	R	Utah Run length encoded image file	
SCT	R	Scitex Continuous Tone Picture	
SFW	R	Seattle File Works image	
SGI	RW	Irix RGB image	
SHTML	W	Hypertext Markup Language client-side image map	Used to write HTML clickable image maps based on a the output of montage or a format which supports tiled images such as MIFF.
SUN	RW	SUN Rasterfile	
SVG	RW	Scalable Vector Graphics	Requires libxml2 and freetype-2. Note that SVG is a very complex specification so support is still not complete.
TGA	RW	Truevision Targa image	Also known as formats "ICB", "VDA", and "VST".
TIFF	RW	Tagged Image File Format	Also known as "TIF". Requires tiff-v3.6.1.tar.gz or later. Note that since Unisys claims a patent on the LZW algorithm (expiring in the US as of June 2003) used by LZW-compressed TIFF files, ImageMagick binary distributions do not include support for the LZW algorithm so LZW TIFF files can not be written. Although a patch is available for libtiff to enable building with LZW support. Users should consult the Unisys LZW web page before applying it.
TIM	R	PSX TIM file	
TTF	R	TrueType font file	Requires freetype 2. Opening as file returns a preview image.
TXT	RW	Raw text file	
UIL	W	X-Motif UIL table	
UYVY	RW	Interleaved YUV raw image	Use -size command line option to specify width and height.
VICAR	RW	VICAR rasterfile format	
VIFF	RW	Khoros Visualization Image File Format	
WBMP	RW	Wireless bitmap	Support for uncompressed monochrome only.
WMF	R	Windows Metafile	Requires libwmf. By default, renders WMF files using the dimensions specified by the metafile header. Use the -density option to adjust the output resolution, and thereby adjust the ouput size. The default output resolution is 72DPI so "-density 144" results in an image twice as large as the default. Use **-background color** to specify the WMF background color (default white) or **-texture**

			filename to specify a background texture image.
WPG	R	Word Perfect Graphics File	
XBM	RW	X Windows system bitmap, black and white only	Used by the X Windows System to store monochrome icons.
XCF	R	GIMP image	
XPM	RW	X Windows system pixmap	Also known as "PM". Used by the X Windows System to store color icons.
XWD	RW	X Windows system window dump	Used by the X Windows System to save/display screen dumps.
YCbCr	RW	Raw Y, Cb, and Cr samples	Use -size and -depth to specify the image width, height, and depth.
YCbCrA	RW	Raw Y, Cb, Cr, and alpha samples	Use -size and -depth to specify the image width, height, and depth.
YUV	RW	CCIR 601 4:1:1	

4.4 Tools and algorithms for Audio Files Adaptation (FHGIGD, UNIVLEEDS, EPFL)

Module Profile Tools and Algorithms for Audio Adaptation		
Executable or Library(Support)	Library	
Single Thread or Multithread	Single	
Language of Development	C++	
Responsible Name	Mattavelli	
Responsible Partner	EPFL	
Status (proposed/approved)		
Platforms supported	Linux, Win32 (MinGW)	
Interfaces with other tools:	Name of the communicating tools	Communication model and format (protected or not, etc.)
File Formats Used	Shared with	File format name or reference to a section
See the tables below		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
Simple direct call		
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not
FFMPEG		LGPL
FOBS		LGPL
LIBSNDFILE		LGPL
SoundTouch		LGPL

4.4.1 FFMPEG/FOBS

Format transcoding is one of the main adaptation tool needed by the AXMEDIS framework as it implies bitrate reduction when transcoding into a compressed format. For audio transcoding, the FFMPEG and FOBS library should be used. For more details on those two libraries, see section 4.2 (Tools and Algorithms for video adaptation)

Here is a list of audio codecs supported by FFMPEG through the libavcodec library ("I" means that an integer only version is available too, ensuring highest performances on systems without hardware floating point support; "X" means that encoding (resp. decoding) is supported):

Supported Codec	**Encoding**	**Decoding**	**Comments**
MPEG Audio Layer 2	IX	IX	
MPEG Audio Layer 1/3	IX	IX	MP3 encoding is supported through the external library LAME
AC3	IX	IX	Liba52 (GPL) is used internally for decoding
Vorbis	X	X	Supported through the external library libvorbis
WMA V1/V2		X	
AAC	X	X	Supported through the external library libfaac/libfaad
Microsoft ADPCM	X	X	
MS IMA ADPCM	X	X	
QT IMA ADPCM		X	
4X IMA ADPCM		X	
G. 726 ADPCM	X	X	
Duck DK3 IMA ADPCM		X	Used in some Sega Saturn console games
Duck DK4 IMA ADPCM		X	Used in some Sega Saturn console games
Westwood Studios IMA ADPCM		X	Used in Westwood Studios games like Command and Conquer
SMJPEG IMA ADPCM		X	Used in certain Loki game ports
CD-ROM XA ADPCM		X	
CRI ADX ADPCM	X	X	Used in Sega Dreamcast games
Electronic Arts ADPCM		X	Used in various EA titles
Creative ADPCM		X	
RA 144		X	Real 14400 bit/s codec
RA 288		X	Real 28800 bit/s codec
RADNET	X	IX	Real lowbitrate AC3 codec, liba52 (GPL) is used for decoding
AMR-NB	X	X	Supported through an external library
AMR-WB	X	X	Supported through an external library
DV audio		X	
Id RoQ DPCM		X	Used in Quake III, Jedi Knight II, other computer games
Interplay MVE DPCM		X	Used in various Interplay computer games
Xan DPCM		X	Used in Origin's Wing Commander IV AVI files
Sierra Online DPCM		X	Used in Sierra Online game audio files
Apple MACE 3		X	
Apple MACE 6		X	
FLAC		X	
FFmpeg Sonic	X	X	Experimental lossly/lossless codec

4.4.2 LIBSNDFILE (http://www.mega-nerd.com/libsndfile/)

Libsndfile is a C library for reading and writing files containing sampled sound (such as MS Windows WAV and the Apple/SGI AIFF format) through one standard library interface. It is released in source code format under the Gnu Lesser General Public License.

The library was written to compile and run on a Linux system but should compile and run on just about any Unix (including MacOSX). It can also be compiled and run on Win32 systems using the Microsoft compiler and MacOS (OS9 and earlier) using the Metrowerks compiler. There are directions for compiling libsndfile on these platforms in the Win32 and MacOS directories of the source code distribution.

It was designed to handle both little-endian (such as WAV) and big-endian (such as AIFF) data, and to compile and run correctly on little-endian (such as Intel and DEC/Compaq Alpha) processor systems as well as big-endian processor systems such as Motorola 68k, Power PC, MIPS and Sparc. Hopefully the design of the library will also make it easy to extend for reading and writing new sound file formats.

It has been compiled and tested (at one time or another) on the following systems:

- i586-pc-linux-gnu (Linux on PC hardware)
- powerpc-unknown-linux-gnu (Linux on Apple Mac hardware)
- powerpc-apple-darwin7.0 (Mac OS X 10.3)
- sparc-sun-solaris2.8 (using gcc)
- mips-sgi-irix5.3 (using gcc)
- QNX 6.0
- i386-unknown-openbsd2.9
- Win32 (Microsoft Visual C++)

At the moment, each new release is being tested on i386 Linux, PowerPC Linux, MacOSX on PowerPC and Win32.
Features

libsndfile has the following main features :

- Ability to read and write a large number of file formats.
- A simple, elegant and easy to use Applications Programming Interface.
- Usable on Unix, Win32, MacOS and others.
- On the fly format conversion, including endian-ness swapping, type conversion and bitwidth scaling.
- Optional normalisation when reading floating point data from files containing integer data.
- Ability to open files in read/write mode.
- The ability to write the file header without closing the file (only on files open for write or read/write).
- Ability to query the library about all supported formats and retrieve text strings describing each format.

libsndfile has a comprehensive test suite so that each release is as bug free as possible. When new bugs are found, new tests are added to the test suite to ensure that these bugs don't creep back into the code. When new features are added, tests are added to the test suite to make sure that these features continue to work correctly even when they are old features.

The following table lists the file formats and encodings that libsndfile can read and write. The file formats are arranged across the top and encodings along the left edge.

	Micro-soft WAV	SGI / Apple AIFF / AIFC	Sun / DEC / NeXT AU / SND	Header-less RAW	Paris Audio File PAF	Commo-dore Amiga IFF / SVX	Sphere Nist WAV	IRCAM SF	Creative VOC	Sound forge W64	GNU Octave 2.0 MAT4	GNU Octave 2.1 MAT5	Portable Voice Format PVF	Fasttracker 2 XI	HMM Tool Kit HTK
Unsigned 8 bit PCM	R/W	R/W		R/W					R/W	R/W		R/W			
Signed 8 bit PCM		R/W	R/W	R/W	R/W	R/W	R/W						R/W		
Signed 16 bit PCM	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W		R/W
Signed 24 bit PCM	R/W	R/W	R/W	R/W	R/W		R/W	R/W		R/W					
Signed 32 bit PCM	R/W	R/W	R/W	R/W			R/W	R/W		R/W	R/W	R/W	R/W		
32 bit float	R/W	R/W	R/W	R/W				R/W		R/W	R/W	R/W			
64 bit double	R/W	R/W	R/W	R/W						R/W	R/W	R/W			
u-law encoding	R/W	R/W	R/W	R/W			R/W	R/W	R/W	R/W					
A-law encoding	R/W	R/W	R/W	R/W			R/W	R/W	R/W	R/W					
IMA ADPCM	R/W									R/W					
MS ADPCM	R/W									R/W					
GSM 6.10	R/W	R/W		R/W						R/W					
G721 ADPCM 32kbps			R/W												
G723 ADPCM 24kbps			R/W												

G723 ADPCM 40kbps			R/W												
12 bit DWVW		R/W		R/W											
16 bit DWVW		R/W		R/W											
24 bit DWVW		R/W		R/W											
Ok Dialogic ADPCM				R/W											
8 bit DPCM														R/W	
16 bit DPCM														R/W	

4.4.3 SoundTouch Audio Processing Library (http://sky.prohosting.com/oparviai/soundtouch/)

SoundTouch is an open-source audio processing library for changing the Tempo, Pitch and Playback Rates of audio streams or files:

- Tempo (time-stretch): Changes the sound to play at faster or slower speed than original, without affecting the sound pitch.
- Pitch (key) : Changes the sound pitch or key, without affecting the sound tempo or speed.
- Playback Rate : Changes both the sound tempo and pitch, as if an LP disc was played at wrong RPM rate.

The SoundTouch library is suited for application developers writing sound processing tools that require tempo/pitch control functionality, or just for playing around with the sound effects. The SoundTouch library source kit includes also an example utility SoundStretch that allows processing .wav audio files with command-line interface.

Features:

- Easy-to-use implementation of time-stretch, pitch-shift and sample rate transposing routines.
- High-performance object-oriented C++ implementation.
- Full source codes available for both the SoundTouch library and the example application.
- Clear and easy-to-use programming interface via a single C++ class.
- Supported audio data format : 16Bit integer or 32bit floating point PCM mono/stereo
- Capable of real-time audio stream processing:
  - input/output latency max. ~ 100 ms.
  - Processing 44.1kHz/16bit stereo sound in realtime requires a 133 Mhz Intel Pentium processor or better.
- Platform-independent implementation: The SoundTouch library can be compiled for any processor and OS platform supporting GNU C compiler (gcc) or Visual Studio, for example Win32, Linux, AIX.
- Additional assembler-level and Intel-MMX instruction set optimizations for Intel x86 compatible processors (Win32 & Linux platforms), offering several times increase in the processing performance.
- Compiled executable binaries available for Windows.
- Released under the GNU Lesser General Public License (LGPL).

4.5 Tools and algorithms for Multimedia Adaptation (EPFL)

Module Profile **Tools and Algorithms for Multimedia Adaptation**	
Executable or Library(Support)	Library
Single Thread or Multithread	
Language of Development	C++
Responsible Name	Mattavelli
Responsible Partner	EPFL
Status (proposed/approved)	
Platforms supported	MS Windows, Linux

Interfaces with other tools:	Name of the communicating tools	Communication model and format (protected or not, etc.)
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	License status: GPL. LGPL. PEK, proprietary, authorized or not

Multimedia documents are composed of a set of different media objects: text, picture, animations, synthesised and natural video, synthesized and natural audio. Hypermedia documents add the possibility for the user to interact with the document.

To allow for the composition of the various media streams, most multimedia formats define some XML-like language describing how the various media streams are associated. Here is a list of the major multimedia languages:

- VRML (Virtual Reality Modelling Language) and its new generation X3D (eXtensible 3D)
- XMT (eXtensible MPEG-4 Textual format)
- SMIL (Synchronised Multimedia Integration Language)

XML being at the basis of most of the major multimedia languages, it is possible to build compilers to transform files from one language to the other.

As far as known, there are only two tools available for multimedia documents manipulation. Actually these tools do not satisfy some basic requirements of AXMEDIS as one is written in Java and the other is released under a GPL licensing scheme. In the case of those formats that can be considered as multimedia (notably PDF, PS, and SVG), the **Lib ImageMagick** library can be used for adaptation (see section 4.3).

GPAC

GPAC is an implementation in ANSI C of the MPEG-4 Systems standard (ISO/IEC 14496-1), with additional audiovisual codecs. Its goal is to integrate multimedia standards such as SVG/SMIL, VRML, X3D or SWF into a single framework following the XMT specification.

GPAC is currently running under Windows and Linux platforms. Windows CE/Pocket PC platform is not actively maintained but GPAC 0.2.3 is running on an iPaq device.

GPAC is released under the GNU General Public License.

IBM Toolkit for MPEG-4

The IBM toolkit for MPEG-4 consists of a set of technologies compliant with the MPEG-4 standard. It is implemented as a set of Java classes and APIs which can be used to develop MPEG-4 applications for authoring and playback. It is the main toolset around which the XMT specification has been built, as a case study for compatibility of the different XML-based languages mentioned above.

Since the toolkit is Java-based, the applications built on top of it will run on any platform that supports Java.

The IBM toolkit for MPEG-4 is released under a commercial licensing scheme.

4.6 Tools and algorithms for Metadata/AXInfo Adaptation (UNIVLEEDS)

Module Profile **Tools and Algorithms for Metadata Adaptation**		
Executable or Library(Support)	Executable	
Single Thread or Multithread	Single Thread	
Language of Development	C++	
Responsible Name		
Responsible Partner	UNIVLEEDS	
Status (proposed/approved)	Proposed	
Platforms supported	Windows	
Interfaces with other tools:	Name of the communicating tools	Communication model and format (protected or not, etc.)
Query Support		
AXOM Commands and Reporting		
File Formats Used	Shared with	File format name or reference to a section
XML		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not
Xerces	Xerces C++ Parser v-2.6.0	Apache Software Licence, v-2.0
Xalan	Xalan-c++ v-1.9	The Apache Software License, v1.1

This module provides a collection of algorithms and tools for adaptation of XML metadata. The main adaptation functions needed by the AXMEDIS Framework could be summarised in:

- Scaling metadata by filtering elements

  - This can be done by specifying look-up tables (including XSLT) to define the valid/invalid elements.
- Adapting field (could be different name or size or others)
  - Xerces can also use a given schema to validate the elements. With this validation function, the elements can be detected for adaptation. Look-up tables (including XSLT) have to be setup in order to adapt metadata from one standard to the other.

For XML metadata transcoding, the *Xerces Libraries* can be used to parse a given piece of XML data.

4.6.1 Xerces: XML parsers in Java and C++ (plus Perl and COM)

Xerces (named after the Xerces Blue butterfly) provides XML parsing and generation. The library available for both C++ and Java, implementing the W3C XML and DOM (Level 1 and 2) standards, as well as the de facto SAX (version 2) standard. The parsers are highly modular and configurable. Initial support for XML Schema (draft W3C standard) is also provided.

A Perl wrapper is provided for the C++ version of Xerces. It also provides full access to Unicode strings. A COM wrapper (also for Xerces-C) provides compatibility with the Microsoft MSXML parser.

The libraries feature:

1. Source code, samples, and documentation is provided
2. Programmatic generation and validation of XML
3. Pluggable catalogs, validators and encodings
4. Customizable error handling

Xerces includes three Metadata interfaces making it possible to modify or create metadata automatically and dynamically. The following table shows support to the different programming languages (sources are taken from http://xml.apache.org/, http://xml.apache.org/xerces-c/, http://xml.apache.org/xerces2-j/, http://xml.apache.org/xerces-p/).

Programming language	Tool/library	Standards Supported
C++	Use Xerces-C++ Version 2.6.0 to read and write XML data. A shared library is provided for parsing, generating, manipulating and validating XML documents	• XML 1.0 (Third Edition) • XML 1.1 (First Edition) (Note: Normalization Checking has not been implemented) • DOM Level 1 Specification • DOM Level 2 Core Specification • DOM Level 2 Traversal and Range Specification • SAX 1.0 and SAX 2.0 • Namespaces in XML • XML Schema Part 1: Structure • XML Schema Part 2: Datatypes • Namespaces in XML 1.1 • DOM Level 3 Core Specification (Partial implementation) • DOM Level 3 Load and Save Specification
Java	Use Xerces2 Java Parser 2.6.2 and the Cerces Native Interface (XNI) to build parser components and configurations that is modular. Xerces2 is able to parse documents written according to the XML1.1 Recommendation.	• XML 1.0 (Third Edition) • Namespaces in XML • XML 1.1 (First Edition) • Document Object Model (DOM) Level 2 Core, Events, and Traversal and Range Recommendations • Simple API for XML (SAX) 2.0.1 Core, and Extensions

		• Java APIs for XML Processing (JAXP) 1.2 • XML Schema 1.0 Structures and Datatypes Recommendations
Perl	The XML::Xerces 2.3.0-4 Perl API is implemented using Xerces C++ API. Based on Xerces-C, provides a validating XML parser to read and write XML data and provides most of the C++ API. Classes are provided for parsing, generating, manipulating, and validating XML documents	• String I/O (Perl strings versus XMLch arrays) • List I/O (Perl lists versus DOM_NodeList's) • Hash I/O (Perl hashes versus DOM_NamedNodeMap's) • Combined List/Hash classes • DOM Serialization API • Implementing Perl handlers for C++ event callbacks • handling C++ exceptions • XML::Xerces::XMLUni unicode constants
COM	A COM wrapper (also for Xerces-C) provides compatibility with the Microsoft MSXML parser.	• See Perl standard supports

4.6.2 Xalan : XSLT stylesheet processors in Java & C++

Xalan is an XSLT processor for transforming XML documents into HTML, text, or other XML document types. Implementations for XSL Transformations (XSLT) Version 1.0 and the XML Path Language (XPath) Version 1.0, it works with the appropriate Xerces XML parser

Programming language	Tool/library	Description
C++	Use Xalan-C++ Version 1.9 to transform XML documents into HTML, text, or other document types.	• implements XSL Transformations (XSLT) Version 1.0 • XML Path Language (XPath) Version 1.0 • Works with a compatible release of the Xerces-C++ XML parser: Xerces-C++ version 2.6.0
Java	Use Xalan-Java Version 2.6.0 to transform XML documents into HTML, text, or other document types.	• implements XSL Transformations (XSLT) Version 1.0 • XML Path Language (XPath) Version 1.0 • implementation of the Transformation API for XML (TrAX) interfaces, part of the Java API for XML Processing 1.2 • builds on SAX 2 and DOM level 2

4.7 Tools and algorithms for DRM information adaptation (FUPF)

Module Profile Tools and Algorithms for DRM Adaptation		
Executable or Library(Support)	Executable	
Single Thread or Multithread	Multithread	
Language of Development	C++	
Responsible Name		
Responsible Partner	FUPF	
Status (proposed/approved)	Proposed	
Platforms supported	MS Windows	
Interfaces with other tools:	Name of the communicating tools	Communication model and format

		(protected or not, etc.)
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	License status: GPL. LGPL. PEK, proprietary, authorized or not
Xerces		
Xalan		

DRM adaptation involves the adaptation of the related licenses, as derived AXMEDIS objects or digital resources within the AXMEDIS objects can be seen as new creations with regard to original ones. Therefore, new licenses must be created during the adaptation process, always respecting the terms and conditions fixed in the original license or licenses for the adapted AXMEDIS objects or contents within these objects.

Nevertheless, DRM information (mainly licenses and PARs) inside the AXMEDIS project, that are related to AXMEDIS Objects will be expressed in XML language by using MPEG-21 REL.

In order to adapt this information to different rights expression languages, also based in XML or to adapt a license to be more compact in order to use it into portable devices (for instance, mobile phones or PDAs), we will make use of existing libraries for manipulating XML documents.

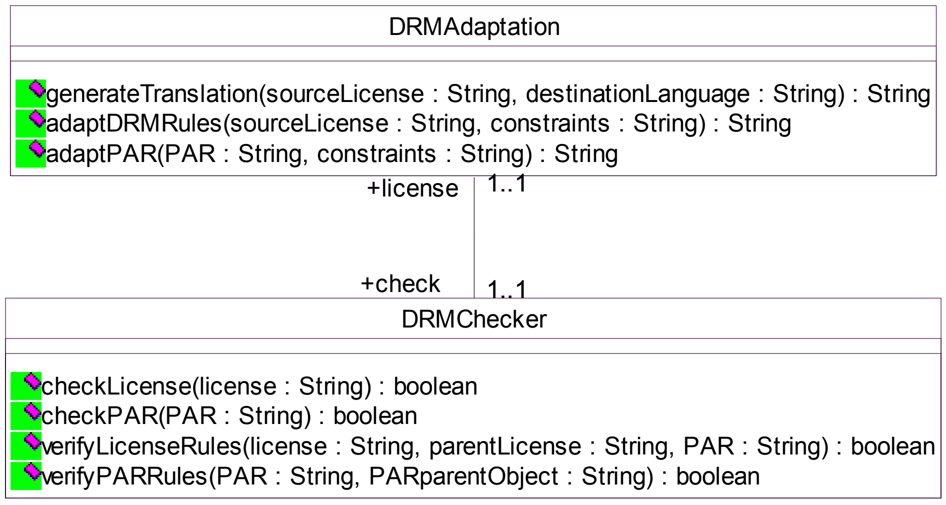
The main adaptation function produced by this module can be summarised in:

- Compacting licenses for their use in portable devices
- Translating licenses from one rights expression language to another
- Automatic generation of a license when an adaptation over the content it applies is done

For XML DRM rules transcoding, the *Xerces Libraries* can be used to parse a given piece of XML data.

4.7.1 Architecture of the module

The following figure shows the UML diagram of the Tools and Algorithms for DRM adaptation module.


DRMAdaptation
generateTranslation(sourceLicense : String, destinationLanguage : String) : String
adaptDRMRules(sourceLicense : String, constraints : String) : String
adaptPAR(PAR : String, constraints : String) : String
+license
1..1
+check
1..1
DRMChecker
checkLicense(license : String) : boolean
checkPAR(PAR : String) : boolean
verifyLicenseRules(license : String, parentLicense : String, PAR : String) : boolean
verifyPARRules(PAR : String, PARparentObject : String) : boolean

Class diagram for tools and algorithms for DRM Adaptation

DRMAdaptation: Provides the needed functionality to perform adaptation of DRM rules and PAR.

DRMChecker: Provides syntactical checking of licenses and PAR and verifying functionality of PAR and licenses against parent licenses and PAR.

DRMAdaptation	

WSDL	
	```
	<wsdl:definitions name='DRMAdaptation' targetNamespace='urn:DRMAdaptation'
	   xmlns:tns='urn:DRMAdaptation'
	   xmlns:ns0='http://systinet.com/xsd/SchemaTypes/'
	   xmlns:map='http://systinet.com/mapping/'
	   xmlns:soap='http://schemas.xmlsoap.org/wsdl/soap/'
	   xmlns:wsdl='http://schemas.xmlsoap.org/wsdl/'>
	   <wsdl:types>
	      <xsd:schema targetNamespace="http://systinet.com/xsd/SchemaTypes/"
	         elementFormDefault="qualified"
	         xmlns:xsd="http://www.w3.org/2001/XMLSchema"
	         xmlns:tns="http://systinet.com/xsd/SchemaTypes/">
	         <xsd:element name="sourceLicense" type="xsd:string"
	            nillable="true"/>
	         <xsd:element name="destinationLanguage" type="xsd:string"
	            nillable="true"/>
	         <xsd:element name="string_Response" type="xsd:string"
	            nillable="true"/>
	         <xsd:element name="constrains" type="xsd:string" nillable="true"/>
	         <xsd:element name="PAR" type="xsd:string" nillable="true"/>
	      </xsd:schema>

	   </wsdl:types>
	   <wsdl:message name='DRMAdaptation_adaptPAR_1_Request'>
	      <wsdl:part name='PAR' element='ns0:PAR'/>
	      <wsdl:part name='constrains' element='ns0:constrains'/>
	   </wsdl:message>
	   <wsdl:message name='DRMAdaptation_generateTranslation_Response'>
	      <wsdl:part name='response' element='ns0:string_Response'/>
	   </wsdl:message>
	   <wsdl:message name='DRMAdaptation_adaptDRMRules_1_Request'>
	      <wsdl:part name='sourceLicense' element='ns0:sourceLicense'/>
	      <wsdl:part name='constrains' element='ns0:constrains'/>
	   </wsdl:message>
	   <wsdl:message name='DRMAdaptation_generateTranslation_1_Request'>
	      <wsdl:part name='sourceLicense' element='ns0:sourceLicense'/>
	      <wsdl:part name='destinationLanguage' element='ns0:destinationLanguage'/>
	   </wsdl:message>
	```

```
  <wsdl:message name='DRMAdaptation_adaptDRMRules_Response'>
    <wsdl:part name='response' element='ns0:string_Response'/>
  </wsdl:message>
  <wsdl:message name='DRMAdaptation_adaptPAR_Response'>
    <wsdl:part name='response' element='ns0:string_Response'/>
  </wsdl:message>
  <wsdl:portType name='DRMAdaptation'>
    <wsdl:operation name='generateTranslation' parameterOrder='sourceLicense destinationLanguage'>
      <wsdl:input message='tns:DRMAdaptation_generateTranslation_1_Request'/>
      <wsdl:output message='tns:DRMAdaptation_generateTranslation_Response'/>
    </wsdl:operation>
    <wsdl:operation name='adaptDRMRules' parameterOrder='sourceLicense constrains'>
      <wsdl:input message='tns:DRMAdaptation_adaptDRMRules_1_Request'/>
      <wsdl:output message='tns:DRMAdaptation_adaptDRMRules_Response'/>
    </wsdl:operation>
    <wsdl:operation name='adaptPAR' parameterOrder='PAR constrains'>
      <wsdl:input message='tns:DRMAdaptation_adaptPAR_1_Request'/>
      <wsdl:output message='tns:DRMAdaptation_adaptPAR_Response'/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name='DRMAdaptation' type='tns:DRMAdaptation'>
    <soap:binding transport='http://schemas.xmlsoap.org/soap/http' style='document'/>
    <wsdl:operation name='generateTranslation'>
      <map:java-operation name='generateTranslation'
signature='KExqYXZhL2xhbmcvU3RyaW5nO0xqYXZhL2xhbmcvU3RyaW5nOylMamF2YS9sYW5nL1N0cmluZ
zs='/>
      <soap:operation
soapAction='urn:DRMAdaptationDRMAdaptation#generateTranslation#KExqYXZhL2xhbmcvU3RyaW5nO0xqY
XZhL2xhbmcvU3RyaW5nOylMamF2YS9sYW5nL1N0cmluZzs=' style='document'/>
      <wsdl:input>
        <soap:body use='literal'/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use='literal'/>
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name='adaptDRMRules'>
      <map:java-operation name='adaptDRMRules'
signature='KExqYXZhL2xhbmcvU3RyaW5nO0xqYXZhL2xhbmcvU3RyaW5nOylMamF2YS9sYW5nL1N0cmluZ
zs='/>
      <soap:operation
soapAction='urn:DRMAdaptationDRMAdaptation#adaptDRMRules#KExqYXZhL2xhbmcvU3RyaW5nO0xqYXZ
hL2xhbmcvU3RyaW5nOylMamF2YS9sYW5nL1N0cmluZzs=' style='document'/>
      <wsdl:input>
        <soap:body use='literal'/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use='literal'/>
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name='adaptPAR'>
      <map:java-operation name='adaptPAR'
signature='KExqYXZhL2xhbmcvU3RyaW5nO0xqYXZhL2xhbmcvU3RyaW5nOylMamF2YS9sYW5nL1N0cmluZ
zs='/>
      <soap:operation
soapAction='urn:DRMAdaptationDRMAdaptation#adaptPAR#KExqYXZhL2xhbmcvU3RyaW5nO0xqYXZhL2xh
bmcvU3RyaW5nOylMamF2YS9sYW5nL1N0cmluZzs=' style='document'/>
      <wsdl:input>
        <soap:body use='literal'/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use='literal'/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name='DRMAdaptation'>
    <wsdl:port name='DRMAdaptation' binding='tns:DRMAdaptation'>
      <soap:address location='http://cjlab1:6060/DRMAdaptation/'/>
    </wsdl:port>
  </wsdl:service>
```

	</wsdl:definitions>
Method	generateTranslation
Description	Generates the translation of a license to a given (different) rights expression language. If the translation is successful, the result is a license in the destination format.
Input parameters	sourceLicense: String, the license to be translated destinationLanguage: String, the language in which the license has to be translated
Output parameters	String, the translated license
Request Sample Message	<e:Envelope xmlns:e="http://schemas.xmlsoap.org/soap/envelope/" xmlns:d="http://www.w3.org/2001/XMLSchema" xmlns:i="http://www.w3.org/2001/XMLSchema-instance" xmlns:wn0="http://systinet.com/xsd/SchemaTypes/"> <e:Body> <wn0:sourceLicense i:type="d:string">&amp;lt;r:license/&amp;gt;</wn0:sourceLicense> <wn0:destinationLanguage i:type="d:string">ODRL</wn0:destinationLanguage> </e:Body> </e:Envelope>
Response Sample Message	<e:Envelope xmlns:e="http://schemas.xmlsoap.org/soap/envelope/" xmlns:d="http://www.w3.org/2001/XMLSchema" xmlns:i="http://www.w3.org/2001/XMLSchema-instance" xmlns:wn0="http://systinet.com/xsd/SchemaTypes/"> <e:Body> <wn0:string_Response i:type="d:string">&amp;lt;odrl:license/&amp;gt;</wn0:string_Response> </e:Body> </e:Envelope>
Method	adaptDRMRules
Description	Adapts the given license using the given constraints
Input parameters	SourceLicense:String, the license to be adapted Constraints:String, the adaptation constraints to be applied over the license
Output parameters	String, the adapted license
Request Sample Message	<e:Envelope xmlns:e="http://schemas.xmlsoap.org/soap/envelope/" xmlns:d="http://www.w3.org/2001/XMLSchema" xmlns:i="http://www.w3.org/2001/XMLSchema-instance" xmlns:wn0="http://systinet.com/xsd/SchemaTypes/"> <e:Body> <wn0:sourceLicense i:type="d:string">&amp;lt;r:license/&amp;gt;</wn0:sourceLicense> <wn0:constrains i:type="d:string">adapt</wn0:constrains> </e:Body> </e:Envelope>
Response Sample Message	<e:Envelope xmlns:e="http://schemas.xmlsoap.org/soap/envelope/" xmlns:d="http://www.w3.org/2001/XMLSchema" xmlns:i="http://www.w3.org/2001/XMLSchema-instance" xmlns:wn0="http://systinet.com/xsd/SchemaTypes/"> <e:Body> <wn0:string_Response i:type="d:string">&amp;lt;r:license/&amp;gt;</wn0:string_Response> </e:Body> </e:Envelope>
Method	adaptPAR
Description	Adapts the given PAR using the given constraints
Input parameters	PAR:String, the PAR to be adapted Constraints:String, the adaptation constraints to be applied over the PAR
Output parameters	String, the adapted PAR
Request Sample Message	<e:Envelope xmlns:e="http://schemas.xmlsoap.org/soap/envelope/" xmlns:d="http://www.w3.org/2001/XMLSchema" xmlns:i="http://www.w3.org/2001/XMLSchema-instance" xmlns:wn0="http://systinet.com/xsd/SchemaTypes/"> <e:Body> <wn0:PAR i:type="d:string">&amp;lt;PAR/&amp;gt;</wn0:PAR> <wn0:constrains i:type="d:string">adapt</wn0:constrains> </e:Body> </e:Envelope>

Response Sample Message	`<e:Envelope xmlns:e="http://schemas.xmlsoap.org/soap/envelope/"`
	`   xmlns:d="http://www.w3.org/2001/XMLSchema"`
	`   xmlns:i="http://www.w3.org/2001/XMLSchema-instance"`
	`xmlns:wn0="http://systinet.com/xsd/SchemaTypes/">`
	`   <e:Body>`
	`      <wn0:string_Response i:type="d:string">&amp;lt;PAR/&amp;gt;</wn0:string_Response>`
	`   </e:Body>`
	`</e:Envelope>`

DRMChecker	
Method	checkLicense
Description	Verifies a license syntactically against the schemas defined within the license.
Input parameters	xsd:string license: the license to be verified
Output parameters	xsd:boolean: true if the license is correct, false if not.
Method	checkPAR
Description	Verifies a PAR syntactically against the schemas defined within the PAR.
Input parameters	xsd:string PAR: the PAR to be verified
Output parameters	xsd:boolean: true if the PAR is correct, false if not.
Method	verifyLicenseRules
Description	Verifies if the license can be generated according to the PARs and the parent licenses (e.g. Distributor or Creator licenses).
Input parameters	xsd:string license: the license to be verified xsd:string parentLicense: distributor license xsd:string PARs: PARs of the corresponding AXMEDIS object
Output parameters	xsd:boolean: true if the license can be generated, false if not.
Method	verifyPARRules
Description	Verifies if the PAR can be generated according to the parent licenses and PARs.
Input parameters	xsd:string PAR: the PAR to be verified xsd:string PARs: PARs of the parent AXMEDIS object xsd:string parentLicense: distributor license
Output parameters	xsd:boolean: true if the PARs can be generated, false if not.

4.8 Adaptation Algorithms Profiles (FHGIGD)

Content adaptation algorithms as well as content description estimation and calculation algorithms are dynamically loaded into the AXMEDIS framework. The correct handling of the plug-ins is performed by the plug-in managers (AXMEDIS Editor::Plug-In Manager and the Collector Engine::Collector Plug-In Manager). They scan a specific directory upon their initialisation. The available plug-ins are loaded and their functionality is checked by loading the libraries description, the so-called algorithm profiles. In each profile human and machine readable information about the algorithms provided in the library are stored.

This description includes information like:

- General information about the library itself
- Information about the functions accessible in this library
- Information about the functions' parameters and the return value

Some of the information, like library name, vendor name, and version name, are relevant for the user. But this information can also be required for the management of the available plug-ins: For users a human understandable description is needed, e.g. a brief help function summarizing the main characteristics of the functions collected in this library.

The description of the functions depends on the individual function, which should be described. As content adaptation and content description algorithms can be distinguished but are closely related detailed information about content adaptation algorithms is available at DE3-1-2D (Framework and Tools Specification – Fingerprint and Descriptors). The relationship between the content adaptation and content description profiles and the plug-in interface can be found in DE3-1-2B (Framework and Tools Specification – Viewers and Players).

5 Bibliography

[1] “JavaScript C Engine Embedder's Guide”, http://www.mozilla.org/js/spidermonkey/apidoc/jsguide.html
[2] Jens Thiele, “Embedding SpiderMonkey - best practice” http://egachine.berlios.de/embedding-sm-best-practice/embedding-sm-best-practice-index.html
[3] “Scripting C++ with JavaScript using SpiderMonkey”, http://home.tiscali.be/franky.braem17/spidermonkey.htm
[4] The Extensible Stylesheet Language Family (XSL), http://www.w3.org/Style/XSL/
[5] Xerces, http://xml.apache.org/#xerces
[6] Xalan, http://xml.apache.org/#xalan