



Automating Production of Cross Media Content for Multi-channel Distribution

www.AXMEDIS.org

DE4.3.1 Content Composition and Formatting

Version: 1.14

Date: 15/11/2005

Responsible: DSI (Ivan Bruno) (revised and approved by DSI)

Project Number: IST-2-511299

Project Title: AXMEDIS

Deliverable Type: Report and Prototype

Visible to User Groups: Yes

Visible to Affiliated: Yes

Visible to Public: Yes

Deliverable Number: DE4.3.1

Contractual Date of Delivery: M13 (end of September 2005)

Actual Date of Delivery: 15/11/2005

Work-Package contributing to the Deliverable: WP4.3

Task contributing to the Deliverable: WP4.2

Nature of the Deliverable: Report and Prototype

Author(s): DSI, IRC, XIM, HP

Abstract:

The report deals with problems related to the Automatic Content Production and more in general with content processing, therefore the current status of the Content Processing prototype and tools are reported. The report is structured in 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... Finally the state of the art in the field of automatic formatting and adaptation was investigated to introduce the AXMEDIS formatting tools: a templates-based system with functionalities for automatic layout selection and optimization .

Keyword List:

Content production, Javascript, Adaptation tools, formatting, composition, transcoding

Table of Contents

1	EXECUTIVE SUMMARY AND REPORT SCOPE	7
2	INTRODUCTION	8
2.1.1	Specification of T4.3.1/2 Composition and Formatting algorithms and tools (DSI)	8
2.1.2	Specification of T4.3.3 Workflow Support (IRC)	12
3	AXMEDIS ARCHITECTURE FOR CONTENT PROCESSING (DSI)	22
4	AXMEDIS RULE DEFINITION AND MODEL	24
	IN THIS SECTION THE STATUS OF WORK PERFORMED IN DEFINING THE MODEL OF AXCP RULE IS REPORTED. THIS REPORT SHOWS THE MAIN CONCEPTS, FORMALIZATION AND IMPLEMENTATION OF THE MODEL.	24
4.1	TECHNICAL DETAILS	24
4.2	AXCP RULE XML FORMALISATION	24
4.2.1	AXCP Rule Model	38
4.3	AXRULE LOADER AND SAVER MODULES (DSI)	40
4.3.1	AxRuleVisitor Class	41
4.3.2	AxRuleLoader Class	41
4.3.3	AxRuleSaver Class	41
5	AXMEDIS RULE EDITOR (DSI)	43
5.1	TECHNICAL DETAILS	43
5.2	DESCRIPTION AXMEDIS RULE EDITOR (WORK DONE)	43
6	AXMEDIS RULE SCHEDULER (DSI)	47
6.1	TECHNICAL DETAILS	47
6.2	DESCRIPTION OF RULE SCHEDULER (WORK DONE)	47
6.2.1	Rule Scheduler Gui	48
6.2.2	Core Scheduler	50
6.2.3	Grid Interface and architecture	51
7	AXMEDIS RULE EXECUTOR (DSI)	52
7.1	TECHNICAL DETAILS	52
7.2	DESCRIPTION OF RULE EXECUTOR (WORK DONE)	52
7.2.1	Rule ExecutorManager	52
7.2.2	Script Executor/JSEngine	53
7.2.3	Profile of the Rule Executor (Peer)	54
7.3	AXMEDIS GRID ARCHITECTURE	58
7.3.1	Structure of messages exchanged between Scheduler and Remote Executor	59
8	JSAXCLASSES STATUS (DSI PLUS ALL)	61
8.1	JS_AXOM: AXMEDIS DATA MODEL JS WRAPPING (DSI)	61
8.2	JS_CRAWLER: CRAWLER JS WRAPPING (FOCUSEEK SUBCONTRACT, DSI)	62
8.3	JS_SELECTION AND QUERY AND DATA BASE ACCESS (DSI, EXITECH, CRS4)	63
8.4	JS_FUNCTION: INTEGRATION WITH EXTERNAL RESOURCES AND LIBRARIES (DSI)	63
8.5	JS_DUBLIN CORE (UNIVLEEDS)	63
8.6	JS_FORMATTING (DSI)	63
8.6.1	JS_Template (DSI)	63
8.6.2	JS_Style (DSI)	64
8.6.3	JS_Format (DSI)	64
9	CONTENT COMPOSITION, EXAMPLES (DSI)	65
9.1	COMPOSITION PROCESS, EXAMPLE	65

9.2	CONTENT PROCESSING, EXAMPLE	66
10	CONTENT COMPOSITION AND FORMATTING (DSI)	67
10.1	STATE OF THE ART	67
10.2	SYSTEM OUTLINE	68
10.3	FORMATTING PROCESS	69
10.4	SYSTEM ARCHITECTURE OVERVIEW	70
10.5	FORMATTING ENGINE ARCHITECTURE	73
10.6	TEMPLATE LANGUAGE	75
10.7	STYLE-SHEET LANGUAGE	78
10.8	DOCUMENT OPTIMIZATION	78
10.9	OPTIMIZATION ALGORITHM	79
10.10	FORMAT EXAMPLE	80
11	CONTENT FORMATTING TOOLS (DSI)	88
11.1	TECHNICAL DETAILS	88
11.2	TEMPLATE EDITOR	88
11.3	AUTOMATIC TEMPLATE SELECTOR	90
11.4	STYLE-SHEET EDITOR	90
11.5	AUTOMATIC STYLE-SHEET SELECTOR	91
11.6	STYLE-SHEET OPTIMIZER	92
11.7	STYLE-SHEET PROCESSOR	92
11.8	TEMPLATE AND DOCUMENT VIEWER	92
12	TRANSCODING AND ADAPTATION (FHGIGD)	93
13	TRANSCODING AUDIO (EPFL)	95
13.1	TECHNICAL DETAILS	95
13.2	AUDIO: STATE OF THE ART	95
13.3	AUDIO: THE PROBLEMS	95
13.4	AUDIO: WORK PERFORMED	96
14	TRANSCODING VIDEO (FHGIGD)	99
14.1	TECHNICAL DETAILS	99
14.2	VIDEO: STATE OF THE ART	99
14.3	VIDEO: THE PROBLEMS	99
14.4	VIDEO: WORK PERFORMED	100
14.5	REFERENCES	102
15	TRANSCODING DOCUMENTS AND TEXT (DIPITA)	103
15.1	TECHNICAL DETAILS	103
15.2	DOCUMENTS AND TEXT: STATE OF THE ART	104
15.3	DOCUMENTS AND TEXT: THE PROBLEMS	105
15.4	DOCUMENTS AND TEXT: WORK PERFORMED	105
16	TRANSCODING IMAGES (DSI, IRC)	107
16.1	TECHNICAL DETAILS	107
16.2	IMAGES : STATE OF THE ART	107
17	TRANSCODING MULTIMEDIA (EPFL)	112
17.1	TECHNICAL DETAILS	112
17.2	MULTIMEDIA: STATE OF THE ART	112
18	TRANSCODING/ADAPTATION PAR AND LICENSES (FUPF)	116
18.1	PAR AND LICENSES: STATE OF THE ART	116
18.1.1	MPEG-21 Rights Expression Language (REL)	116
18.1.2	ODRL	120
18.1.3	OMA DRM Rights Expression Language	121
18.1.4	PAR and Licenses: The problems	122

18.1.5	PAR and Licenses: Work performed.....	122
18.1.5.1	OMA-based MPEG-21 REL DTD	122
18.1.5.2	Interoperability between MPEG-21 REL and OMA DRM REL v2.0.....	123
19	TRANSCODING METADATA (UNIVLEEDS)	127
19.1	TECHNICAL DETAILS	127
19.2	METADATA: STATE OF THE ART	127
19.3	METADATA: THE PROBLEMS	128
19.4	METADATA: WORK PERFORMED	128
19.4.1	AxMetadata Class	128
19.4.2	AxmetadataElement Class	128
19.4.3	AxMetadataAttribute Class	128
19.4.4	AxMetadataSAXImplementation Class	128
20	WORKFLOW MANAGEMENT AND DATABASE (IRC)	130
20.1	TECHNICAL DETAILS	130
20.2	WRITING AND DESCRIBING WORKFLOW, HARMONISING AXMEDIS TOOLS	131
21	WORKFLOW INTEGRATION OF TOOLS (IRC).....	141
21.1	TECHNICAL DETAILS	141
21.2	INTEGRATION SUPPORT WITH CONTENT PROCESSING TOOLS (AXCP PROCESSING TOOLS: ENGINE AND SCHEDULER)	143
21.3	INTEGRATION SUPPORT WITH EDITORS (AXCP RULE EDITOR AND AXMEDIS EDITOR)	145
2.2	Interface between the WF AXOM Request Adapter and the WF Editor Request Gateway	146
21.4	INTEGRATION SUPPORT WITH AXEPTOOLS	147
21.5	INTEGRATION SUPPORT WITH QUERY SUPPORT.....	150
21.6	INTEGRATION SUPPORT WITH AXMEDIS P&P EDITOR.....	153
22	BIBLIOGRAPHY	154
23	OTHER REFERENCE	155

AXMEDIS Copyright Notice

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

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

1. DEFINITIONS

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

2. LICENCE

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

3. TERM AND TERMINATION

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

4. USE

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

5. COPYRIGHT NOTICES

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

6. WARRANTY

1. The Licensor warrants the licensee that the present licence is issued on the basis of full Copyright ownership or re-licensing agreements granting the Licensor full licensing and enforcement power.

2. For the avoidance of doubt the licensee should be aware that although the Copyright in the documents is given under warranty this warranty does not extend to the content of any document which may contain references or specifications or technologies that are covered by patents (also of third parties) or that refer to other standards. AXMEDIS is not responsible and does not guarantee that the information contained in the document is fully proprietary of AXMEDIS consortium and/or partners.
 3. Licensee hereby undertakes to the Licensor that he will, without prejudice to any other right of action which the Licensor may have, at all times keep the Licensor fully and effectively indemnified against all and any liability (which liability shall include, without limitation, all losses, costs, claims, expenses, demands, actions, damages, legal and other professional fees and expenses on a full indemnity basis) which the Licensor may suffer or incur as a result of, or by reason of, any breach or non-fulfilment of any of his obligations in respect of this Licence.
- 7. INFRINGEMENT**
1. Licensee undertakes to notify promptly the Licensor of any threatened or actual infringement of the Copyright which comes to licensee notice and shall, at the Licensor's request and expense, do all such things as are reasonably necessary to defend and enforce the Licensor's rights in the Copyright.
- 8. GOVERNING LAW AND JURISDICTION**
1. This Licence shall be subject to, and construed and interpreted in accordance with Italian law.
 2. The parties irrevocably submit to the exclusive jurisdiction of the Italian Courts.

Please note that:

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

1 Executive Summary and Report Scope

This report is focussed on describing the AXMEDIS Content Production Area (AXCP) prototype, tools, work done and analysis of the state of the art regarding formatting and media transcoding tools. This document is structured in sections dealing with:

- AXMEDIS Content Processing Area (under responsibility of DSI): this section describes the current status and the work done in the development of tools in AXMEDIS Content Processing Area. This area copes with the problem of automatic content production, adaptation and protection of AXMEDIS object and their publication on a P2P environment (AXEPTool). The actual prototype is based on rules that include a procedural description using the Javascript language (script) and schedule information that describe the rule firing inside the AXMEDIS Rule Engine. Such engine is based on a Javascript engine derived from SpiderMonkey by Mozilla. To cope with the amount of needed resources (computational, time, etc...) during the content processing activity, an early version of a distributed environment was defined and based on GRID computing. Therefore the Rule Engine was decomposed in two main component: the AXMEDIS Rule Scheduler and AXMEDIS Rule Executor. The AXMEDIS Content Processing Area results are further illustrated by means of UML diagrams, tables, GUI definitions and snapshots, description of a Grid communication protocol and the XML schema for rules.
- JS Classes for the JavaScript Engine: the composition and formatting process by means javascript required the extension of the object data type inside the javascript language. The set of classes and functions that currently extend the capabilities of the Spidermonkey Javascript engine are described.
- Content Composition and Formatting: this section performs a state of the art analysis to identify limits and problems of pre-existent tools for automatic adaptation and formatting; then, architecture and status of AXMEDIS tools are reported. The actual prototype is based on SMIL for document structure description and XSLT for layout definition; techniques for layout optimization are sketched. UML diagrams and screenshots illustrate architecture and functionalities of these tools.
- Adaptation and Transcoding Tools: in this part the work done and analysis on external tools and libraries for content processing, content format transcoding, content formatting and adaptation (digital audio, video, animation, multimedia, metadata, etc...) are discussed. Part of them was used to implement and customise the set of functions and algorithms to call inside the automatic content processing area.

2 Introduction

The problems related to automatic content production should be decomposed into composition and formatting. In addition, these aspects have to be integrated with those related to content production workflow. Content workflow monitoring and management is considered in WP4.4, which deals with the P2P model behind AXEPTool.

2.1.1 Specification of T4.3.1/2 Composition and Formatting algorithms and tools (DSI)

Major partners involved

DSI with EPFL, ILABS, UNIVLEEDS, XIM, DIPITA

State of the art

Compositional and formatting tools belong to the set of digital content production tools. They have to help content designer to:

- efficiently collect needed components, using advanced query options
- find/produce alternatives for those components that may present distribution problems (e.g. files too big, IPR or usage clearance issues, etc.)
- structure components, highlighting the semantic relations among them
- bind content structure to some presentation styles
- format broadcast/broadband-quality content for delivery to a variety of channels, eventually requiring repurposing or even re-authoring
- support different delivery channels according to various formatting styles and constraints reported in the final user's device profile

Automatic content production problems are decomposed in two separate phases: composition and formatting. The **Compositional** and **Formatting Engine** will respectively perform such phases. Such engine can use services provided by some tools involved in AXMEDIS object production.

Composition is the action of putting together content components to create a new digital item in an almost automatic manner, based on a set of user-defined rules. 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), Animations (synthetic), etc... coming from AXMEDIS database. The composition may produce homogenous (where all digital resources are the same type) or heterogeneous (different kinds of digital resources) composite AXMEDIS Objects.

A challenging part in the automation of composition will be the management of metadata as in such activity the user will be highly relying on objects metadata to identify, select and operate on digital assets. This is already the habit in most editorial environments and tools. In more detail it is important to note that metadata are used during the search phase, and are crucial in the classification and archival phase. Research concerning automatic extraction of high level metadata, and related state of the art, is covered in T4.2.2 and specified in the related section.

Formatting is the process of exploiting digital resources in a combined AXMEDIS Object to some integrated visualization (editorial) format for distribution to the end user. A simple composite object comprising several parts (e.g., an audio, a video and a document) may be formatted in different ways according to various formatting styles (graphic layout, spatial constraints, file size, quality limitations, content temporal scheduling, speech generation from text, synchronization between audio and images, etc.)

and adapted to produce a “final content” (Digital Item Adaptation) to be distributed via different channels to users’ platforms such as i-TV, mobile, PC, etc...

Generally speaking, a formatted multimedia object can be seen as a unique object characterized mainly by:

- *Spatial relationships* – They are defined in the spatial domain and describe how visible resources are organized in the visualization space (layout, margins, etc...)
- *Time relationships* – They are defined in the time domain and describe not only the basic orchestration (i.e. when media items appear and disappear from the screen), but also which synchronization relations need to be maintained (e.g. audio streams lip sync with a given video stream) and how it is possible to control speed and other basic behaviors of animated content (in time domain).

Other relevant aspects to be taken into account are the navigation structure and hierarchies inside the object. For example in many multimedia objects there are at least two different sets of possible navigation paths that allow the user to exploit the content. Usually there is a linear or sequential path and a hyper-textual one. This basically allows defining some additional relationships to be added to the previously mentioned ones and that may be present or not:

- *Navigation and connection relationships* – They are defined in the content structure domain and provide information on the content components structuring and navigation path and relations. Such relationship may be limited to object structure, but can also connect several different object or content sources. In the latter case there is a direct connection with the *Context and dependency relationships*.
- *Context and dependency relationships* – They are defined in the metadata structure domain and provide information on the content context and dependencies relations.

This latter point may account for objects level of complexity and (in case of occurrence) imply some limitation to the results achievable through automatic formatting.

All above-mentioned relationships and aspects (spatial, time, and hierarchical) will be analyzed in this task and linked to the corresponding views developed in T4.1.3.

All the formatting activity shall be based on content features, generic user profile and needs, specific user profile (in case of formatting on demand), formatting style, optimization 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 will have to take into account also the specific problems of the distribution channel:

- Content location and time to delivery (*time for processing and eventually storing of content to be manipulated*).
- Business and transaction models, thus DRM of content
- Delivery model: streaming, download, off-line distribution, etc.
- Delivery format: MPEG-4, simple audio files, documents, video, etc.

For example a user working on a PDA or a mobile wants to play a video clip, which requires a broadband network to be played; the formatting process has to adapt this video clip so to satisfy transmission constraint and to provide to the user a service with an adequate quality.

As with the composition activity, AXMEDIS aims to automate the formatting activity as far as possible, by allowing the user to define formatting rules (based on the above characteristics) which will then be applied to multimedia objects to prepare them for the required distribution channels, business models, delivery formats, etc.

For the analysis of the state of the art the following activities will be performed:

- Review of commercial and non commercial tools
- Analysis of the state of the art in adaptation algorithms for different media
- Monitoring and analysis of the standardization regarding DIA in MPEG-21
- Collecting the internal state of the art, what the partners are using for content composition and formatting
- Compilation of a list of references
- Identification of the main tools on the market, verification of their features and capabilities
- Preparation of a review document of the state of the art
- Identification of constraints imposed by the
 - distribution channel
 - final format for delivery
 - client power and profile

Research and development plan in composition and formatting

The MPEG-21 and WEDELMUSIC models are compositional models in which any kind of content can be integrated. In addition, in both models the DRM aspects and several relationships between the content components included can be defined.

In AXMEDIS we intend to study the following aspects:

- definition of a language for content composition/formatting process description
- implementation of an engine for processing scripting for content composition/formatting
- language dedicated to the layout of the content format. This can be the style of the content format
- definition and implementation of the rule structures for AXMEDIS Composition and Formatting Rules Editor
- definition of a set of rules for assisted speech to text synchronization
- definition and implementation of AXMEDIS Composition and Formatting Process Engine
- elicitation and analysis of the expertise and current manual processes used in the production workflow in order to create tools based on artificial intelligence technologies
- connection of composition/formatting algorithms and tools to the engine and the editor, which will be:
 - Based on technical metadata and licensing aspects (see above)
 - For automatic definition of DRM rules (Possible Available Rights) based on profiles: distribution, device, channel, user, player tool, etc.
 - Produced by example, the user could imagine to provide an example of composition and the tools could process the other files accordingly
 - Based on artificial intelligence and actions on sets such as the merge and selections
 - Coming from artificial intelligence: logic engine interpretation and execution of scripting for automatic processing of composition and formatting
- connection of composition/formatting algorithms and tools with several available integration and optimization technologies, such as:
 - For graphic formatting tools to allocate in the visual and temporal domains different visual sources, integrated with optimization issues
 - For optimizations such as those used for solving knapsack problems in the spatial domain, and scheduling optimization algorithms in temporal domain: DSI has a considerable skill on optimizing processes of formatting on both these domains such as performed in SAMOPROS, OPTAMS, WEDELMUSIC, projects and solutions, etc., by using taboo search, genetic algorithms, knapsack and multi knapsack solutions, several scheduling solutions, MILLA formatting language used in WEDELMUSIC tools, etc.
 - For synchronization, DSI, DIPITA, EPFL and UNIVLEEDS have a considerable skill on these aspects used in solutions for WEDELMUSIC, CARROUSO, etc. In this area, the research partners will develop algorithms for the automatic synchronization of content as performed by DSI for WEDELMUSIC and by EPFL for real-time MPEG-4 scenarios. Synchronization of audio/video with other content: documents, text, voice, images, etc., considering temporal

- evolution of content.
- For integration of accessible algorithms and solutions for the comparison with the innovative algorithms that will be developed
- For integration of content formatting engine with commercial or open source tools via plug-ins
- For integration of adaptation and content processing algorithms for several media
- For integration of adaptation and content processing, including synchronization and editing
- For integration of the MPEG-4 real-time semi-automatic composition tool developed in CARROUSO for MPEG-4 in a production tool, possibly adding (MPEG-7) metadata as from T4.2.2
- For managing relations between textual information and acoustic signals in multimedia objects at two levels (DIPITA): 1) assisted methods for reducing time-consuming alignment procedures; 2) strategies for error reduction in automatic alignment. The automatic detection of significant cues in the speech signal is a key issue in synchronization tasks. Pause detection does not ensure a sufficient threshold of information. The relation between various detectable cues that have been demonstrated relevant during linguistic research will be exploited. More specifically the interaction among turn alternation, music/voice transitions, and frequency analysis and intensity changes will be tested in connection with the implementation of synchronization technologies.
- research activity on Content Composition and Formatting and integration into the several workflow processes of content producers, aggregators and distributors
- adaptation of the content producer partners' workflows to support AXMEDIS framework and integration of the new tools into these workflows; this will include:
 - creating initial composition and formatting rules for each content producer partner
 - integrating AXMEDIS tools via plug-ins and interfaces with the producers' existing production tools and WFMS (where currently used)
- definition of the modality of testing in the form of a Test Plan: identification of the model for the production/collection of test cases
- verification and production of several test cases derived from the defined Test Plan
- validation of the new environment for content composition/formatting by using content produced by WP8. This will involve each content producer partner in:
 - working through the Test Plan, running test cases
 - identifying and documenting any missing rules, formatting language elements or critical functionalities in the prototype AXMEDIS tools
 - measuring comparative production costs with and without AXMEDIS tools. It may be possible that content producer partners may not want to make public the production cost in EURO. It is more likely that they will share measurements of effort and workload, learning cost and human resources, productivity (efficiency, e.g. number of steps needed to complete a test scenario, and effectiveness, that is the extent to which AXMEDIS tools provide the functionality needed to perform a test scenario) and of performance, e.g. time to perform a test scenario
 - logging benefits and other results not documented elsewhere

Research and development plan in adaptation

The following aspects and algorithms will be taken into account:

- Formalization of distribution channels, devices, tools, user profiles and the class hierarchy for their loading and saving, and algorithms for their processing
- Formalization of adaptation algorithm profiles and the class hierarchy for their loading and saving, and algorithms for their processing
- Definition and implementation of a common framework and library for the adaptation algorithms and its usage from AXMEDIS Editor, Composition and Formatting Engine
- Definition and implementation of AXMEDIS Object Manager (AXOM) Content Processing interface for external plug-ins in conjunction with AXOM
- Digital Item adaptation for several media:

- Documents and speech (DIPITA)
- Video (FHGIGD, XIM)
- Audio (EPFL, DSI, UNIVLEEDS)
- Images: (DSI, EPFL)
- Multimedia MPEG-4 (EPFL)
- Multimedia IE (DSI, UNIVLEEDS, XIM)
- PAR: Possible Available Rights (FUPF)
- Metadata and AXINFO adaptation (UNIVLEEDS)
- Etc.
- Integration of available algorithms and solutions for performance comparisons
- Definition of transcoding algorithms according to an extensible plug and play model standardized into AXMEDIS tool and framework
- Definition of transcoding algorithms according to portability issues that may concern PDAs, mobile devices, etc. (resolution, definition, illumination and number of supported colors, available memory storage, available computational capabilities, easiness of usage of the device and its GUI)
- Multilingual support for production and verification of multilingual metadata
- Usage and integration of commercial and open source text translation algorithms and tools: leading technology on translating text into several languages, and technology in vocal synthesis from text
- Collection and management of AXMEDIS Digital Item Adaptation Support
- Verification and production of several test cases

Planned schedule

- M9: Definition of rules structure and syntax for content composition and formatting
- M12: First results about content composition and formatting algorithms, rule based engine
- M12: First results about the formatting algorithms
- M13: A first collection and version of digital item adaptation algorithms, transcoding, reduction, etc.
- M16: First version of synchronization algorithms between audio and text, audio and notes, etc.
- M24: Intermediate version of engines for content formatting and composition
- M24: An intermediate version of the set of algorithms for composition, transcoding and formatting of content. This will be used as trial version of their integration into the AXMEDIS framework
- M24: Final version of engines for content formatting and composition
- M36: The final version of the set of algorithms for composition, transcoding and formatting of content. This will replace the set used for the AXMEDIS framework

2.1.2 Specification of T4.3.3 Workflow Support (IRC)

Major partners involved

IRC, with HP, XIM

Knowledge Engineering Pre-requisites

The design and specification of the Workflow Integration to be undertaken as the planned activity for the forthcoming phase was based on three earlier preparatory stages of activity undertaken to accomplish a systematic requirements analysis, state-of-the-art research review and knowledge engineering-led transactions and states analysis. Thus in setting out the rationale for the proposed research work as part of the planned activity we shall suffice to give only brief references to some of the above results as these were fully reported in documents submitted earlier. Thus this section will focus on the salient aspects of the requirements and specification of the plug-in interfaces between the two selected Workflow Management Supports (WFMSs) to be integrated with the relevant AXMEDIS components. Accordingly the present specification is based on the conclusions of three distinct planks of preparatory research carried out by the workflow group (IRC, HP, XIM); outlined as follows:

A) Established the state-of-the-art from the standpoint of the relevant business semantics, workflow knowledge and scenarios that are applicable to the three key sectors targeted for exploitation of AXMEDIS Framework, namely e-book, e-media, e-music production and distribution workflows.

The required domain knowledge elicitation was supported by the design and distribution of a questionnaire and a series of follow-on semi-structured interviews with selected practitioners from each of the above sectors. The results were documented in the multi-sectorial Workflow Requirements Elicitation and Domain Knowledge Analysis part of DE2.1.1a, *User Requirements* and use cases.

This led to

- a novel analysis of AXMEDIS Object lives and their LifeCycles
- the analysis of the metadata and descriptor states required for the workflow-triggered traversal of reasoning over such LifeCycles hierarchy that takes place in the normal course of AXMEDIS Object tracking and control
- and how this involved three distinct interaction spaces i.e. three workflow-centric inferencing environments re Object states reasoning, tracking and control

It was then suggested that the above three possible sub-spaces of AXWFM-AXMEDIS interaction could help clarify and streamline the design of AXWFM-AXMEDIS interfaces as well as direct the focus of any metadata and database partitioning (i.e. AXWFM DB- AXdb complementarity) that could be efficiently deployed using the canonical 10P-stamp situation assessment classes of Object metadata states; with its two sub-fields namely *purpose* and *period* used for filtering the focus on metadata in a context-sensitive manner. This was set out in the respective Workflow Sections of DE2.1.1a, *User Requirements* and use cases.

This work in turn led to suggestions re the design of the AXMEDIS Object Schema in terms of the specific states and descriptors required to be included with the Object metadata (AXinfo) and/or in AXWFDB and/or AXdb or in the PMSdb to allow reasoning over all aspects of Object history and full Object traceability and tracking.

This foundational requirements engineering work led to the conceptualisation of the prototypical scenarios of workflow usage and thus the derivation of the 23 Use Cases and associated Test Cases for workflow interaction. This in turn supported the rationale for a four-Channel set of interfaces for a streamlined integration of workflow and AXMEDIS.

B) Established the state-of-the-art of the available workflow technologies both as open-source and proprietary products and performed comparative analysis of their compatibility for integration with AXMEDIS framework. This has led to a rationalization of the choices re the adoption of the candidate workflow systems for integration with AXMEDIS Framework. This was reported in the respective Workflow Section of DE2.1.1a, *User Requirements* and Use cases.

C) Established the knowledge engineering and software development analysis base from the standpoint of the need for specification of the sub-systemic workspace boundaries for states representation, and, *the technology and the protocols* for the above integration leading to the specification and design of the transaction Semantics, Syntax and Protocols for the interfaces required to achieve the various integration scenarios as specified.

In what follows we will first outline the state-of-the-art functional requirements from all workflow participants (Users, AXFWM, AXMEDIS components) as it has helped set the guidelines for the research and development work to be performed in the next phase to achieve the Workflow Integration within AXMEDIS Framework. Thereafter we will briefly refer to the rationale for the planned research in terms of the choice of existing workflow systems to be adapted and interfaced to AXMEDIS Framework and the knowledge engineering-led choices re the selected sub-systemic boundaries for the required 17 distinct

interface instances to be delivered within the following four-channel architecture for the workflow integration:

- i. AXWFM and AXMEDIS Workflow Editor,
- ii. AXWFM and AXMEDIS Rule Editor/Viewers,
- iii. AXWFM and AXMEDIS Engines
- iv. AXWFM and AXMEDIS Query Support Interface

We shall conclude with an outline of the design specification and the technical environment of its implementation as part of the future planned activities together with the associated schedule of milestones for the deliverables

Established Requirements for the Integrated AXWFM-AXMEDIS

This is the outline of the requirements now established for AXWFM Integration as follows:

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

AXMEDIS is expected to deliver a number of innovations to the media production workflow:

- Support single and distributor customisable workflow for all channels - content should need to be authored only once
- Support for peer-to-peer production workflow – collaborative production and integration of content across remote ad hoc virtual organisations that may form and re-form on a project-by-project basis
- Support artificial intelligence-assisted workflow - repetitive tasks, in particular re-purposing for multiple channels, is to be automated

- Enable integration into existing workflows - AXMEDIS is to provide a framework to support workflow, rather than dictate a single, inflexible model. This will allow production houses to retain their unique styles and original creative aspects of their current workflows

The interface will be developed to ensure maximum compatibility with the requirements of the various parts of AXMEDIS Framework to deploy the workflow environment for example for automated content composition and formatting, or content distribution through various channels such as B2C over P2P networks.

The Workflow is to be tightly integrated with AXMEDIS Viewer/Editor tools, in order to be able to automatically streamline editing/viewing activities inside publishing and distribution processes. With this kind of integration, the Workflow automatically launches AXMEDIS Editors/Viewers on AXMEDIS Objects.

The AXMEDIS Viewer/Editor tools can be executed outside the Workflow environment. This is to accommodate those enterprises that wish to work with AXMEDIS Objects but without necessarily adopting structured processes – this is to be supported through the provision of a simple AXMEDIS Object check-in/check-out interface.

On the Client side, the AXWF User Interface will be the home interface for all users and actors involved in the product development and distribution processes. Through the AXWF User Interface the actors logs-in and see all the workitems in which they are individually committed and via accessing any workitems they can perform the actions required through launching the appropriate tools. All actions performed by the actors/users are logged by the AXWF in the AXMEDIS Object repository, as well as the new revisions and status changes.

Through the AXWF User Interface it must be possible to create new or delete existing Objects or Components which in turn will create or delete (sub) process instances. Once the actor/user has selected a workitem to work with and an editing activity to be performed (e.g. editing, composing, formatting, etc an Object/component), the AXWFM will lock the Object/component in the Object repository and will copy it to a work area for exclusive access by the user where it will be processed by the proper tools (authoring/formatting/rendering/composing/ packaging/bundling).

Accordingly the key patterns of deployment in scenarios for the integration of any AXMEDIS-adopted workflow management system (AXWFM) with AXMEDIS Framework involve the AXWFM launching the execution of various plug-ins as follows:

- AXMEDIS Object Editors/Viewers via the AXMEDIS Editor WorkFlow Plug-in
- AXMEDIS Compositional/Formatting Engine via the AXMEDIS WorkFlow Engine Plug-in
- AXMEDIS Collector Internal Engine via the AXMEDIS WorkFlow Engine Plug-in
- AXMEDIS Program and Publication Engine via the AXMEDIS WorkFlow Engine Plug-in
- AXEPTool Loading Tool Engine via the AXMEDIS WorkFlow Engine Plug-in
- AXEPTool Publication Tool Engine via the AXMEDIS WorkFlow Engine Plug-in
- AXMEDIS Loader/Mover via the AXMEDIS WorkFlow Query and Database Interface
- AXMEDIS Query Support via the AXMEDIS WorkFlow Query and Database Interface
- AXMEDIS Compositional/Formatting Rule Editor via the AXMEDIS WorkFlow Rule Editor Plug-in
- AXMEDIS Program and Publication Rule Editor via the AXMEDIS WorkFlow Rule Editor Plug-in
- AXEPTool Publication/Loading Rule Editor via the AXMEDIS WorkFlow Rule Editor Plug-in
- AXMEDIS Protection Rule Editor via the AXMEDIS WorkFlow Rule Editor Plug-in

Thus the AXWFM Workflow Manager will be capable of interacting with the AXMEDIS Object Manager via the AXMEDIS Editor WorkFlow Plug-in

It is expected that every AXMEDIS tool will update the AXMEDIS Object tracking information, while performing its actions. However, when the check-in / check-out interface is used, it is up to the Workflow Object Manager to update such information. Unfortunately the Workflow cannot know exactly what action the user performed between the Check-in and Check-out stage. Accordingly a field is to be provided for users to describe the work that was done on the Object concerned; thus allowing status updates.

Specification of AXWF-AXMEDIS Interface Types

In summary four types of interfaces are expected to form the basis of AXWFM interactions within the AXMEDIS framework. These types will depend on the characteristics of the different applications or tools that the AXWFM must exchange information with.

These interfaces are outlined below:

- a) AXWFM-AXMEDIS native tools interface: the AXMEDIS client tools form a crucial part of the AXMEDIS framework and will therefore present a closely knit form of interaction. These native tools which deal with aspects related to authoring, formatting, rendering, composing, packaging, and bundling applications all perform specific actions on an AXMEDIS Object.
- b) AXWFM-CMS interface: External CMSs must be allowed for as users must not be impeded in their work. Interfaces to such systems are necessarily less well integrated relative to native AXMEDIS tools. An essential part of this type of interface will be the check-in and check-out operations of an AXMEDIS Object by an authorised actor. The former operation performs tasks related to maintaining Object consistency by locking it in the repository and authorising its download exclusively to such actor. The latter operation releases this lock once the upload of the manipulated AXMEDIS Object has been successfully completed and status / revision entries have been inserted in the AXWFM.
- c) AXWFM-AXMEDIS Server Engines interface: The web services technology can be used to integrate the AXWFM and server-based AXMEDIS applications such as the AXMEDIS Object Manager, Program and Publications Engine, Protection Tool Engine, Publication Tool Engine of AXEPTool among others. Using web services, the AXWFM can invoke the proper methods over HTTP(s), enabling one to interface any server-side component, regardless of language or platform or location and vice-versa.
- d) AXWFM-Query support interface: this type of interface is a particular case of the AXWFM-AXMEDIS server engine interface which is to be based on the web services technology. In this case an authorised user may perform AXMEDIS-related queries through the AXWF user interface. The user must be afforded the possibility of performing advanced searches on characteristics connected to the AXMEDIS Objects by invoking external search engines and thus retrieving the information being sought.

State of the art

This part gives an outline of the rationale for the choice of the two selected workflow systems (i.e. the open-source workflow, Openflow, and the proprietary workflow system BizTalk). Later we shall describe the framework for the design of the plug-in input/output structures to be realised such that they are compatible with the protocols and method invocations of the adopted workflow systems.

The main WFMSs that have been examined are as follows:

- i. Openflow

- ii. JBoss jBpm
- iii. ObjectWeb Bonita
- iv. Enhydra Shark.
- v. OpenWFE

Rationale for the Choice of Openflow

The Open-Source Workflow tool was to be selected on the basis of the following criteria:

- Best coverage of the functions needed by Axmedis
- Openness of the interfaces
- Usability considerations
- Robustness, based on project references

The Rationale for choice of BizTalk

- a) Following the extensive process of Multi-Sector Requirements Knowledge Elicitation and Analysis as undertaken in the initial stages of AXMEDIS project together with the state-of-the-art review of all the major WFMSs it was concluded that:
- b) there is no dominant WFMSs currently deployed in the multi-media Production/Distribution sector
- c) MS NET BizTalk offered the greatest scope for this AXMEDIS integration.

Research and development plan

AXMEDIS technical annex sets out the following Objectives in the present context:

- Adaptation of the content producer partners' workflows to support the AXMEDIS framework and integration of the new tools into these workflows
- Definition of the expertise and current manual processes used in the production workflow in order to create tools based on artificial intelligence technologies
- Research activity on Content Composition and Formatting and integration into the several workflow processes of content producers, aggregators and distributors
- Consolidation of tools and workflow developments to be compliant with the overall AXMEDIS framework

The research and development activity for the next period will mainly include the examination of the specification of the plug-ins from the point of view of the integration technology required for the delivery of a fully interoperable set of the plug-ins with the APIs available from Openflow.

This is to serve the integration with AXMEDIS native engines, tools and the Query Support Interface. This implies the design of the plug-ins for full Connectivity, messaging Communication, Command and Control (C4) as well as AXMEDIS Object transfers across the relevant interfaces in particular such that the workflow will be able to provide seamless interaction with all the relevant AXMEDIS components involved in any of the *eight generic Workflow Scenarios* that are already proposed and established by the workflow team.

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

- Programmes and Publications Engine
- AXEPTool Publishing Engine
- AXEPTool Loading Engine
- AXEPTool P2P Active Selection Engine
- AXEPTool Objects Monitoring Tool

- AXEPTool Publication & Loading Editor/Viewer
- Programme & Publications Rule Editor/Viewer
- Composition Engine
- Compositional Commands & Reporting
- Formatting Engine
- Collector Engine
- Protection Engine
- AXOM Editor Plug-in
- Compositional Rule Editor/Viewer
- Protection Tool
- Protection Tool Editor/Viewer
- User Interface

Research will be carried out for the design and development of the above interfaces between the AXMEDIS Workflow (AXWFM) and the relevant core AXMEDIS components (tools/engine/Query Support) to facilitate indexing, tracking, storing as well as optionally to support Authentication, Authorisation & Auditing (AAA) with single sign-on and all-in single-billing.

Accordingly Research will be carried out to finalise the Exchange Format and Metadata states required. An Exchange Format must satisfy the following specification criteria:

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

Each Exchange is an Exchange-instance and has to be indexable as an Exchange_Instance_ID. Any Exchange between the adopted AXMEDIS workflow system (AXWFM) and relevant AXMEDIS service providing components (Editors, Engines, Tools, Query Support) can thus be uniquely distinguished by means of an AXWF-Exchange-instance –ID or WF-Exchange_ID for short.

All these exchanges will be developed in order to be compatible to the selected workflow environments and are a neutral exchange format for accessing any transaction between the AXWFM environment and any relevant AXMEDIS component whilst imposing minimal metadata requirement on the AXMEDIS Object Model. An efficient way of handling the transactions between the transactor (typically AXWFM) and other tools will be developed along with history information of such transactions over the period of any sessions so as to be able to provide the data structures to support a viewer-friendly re-call of all types of session exchanges as well as the session costs, rights granted and used, DRM and billing.

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

WF-Exchange_ID Protocols and Methods

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

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

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

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

In this way a generic model for AXWFM-AXMEDIS transactions is to be specified such that it can abstract from the idiosyncrasies of design/processing modes within individual adopted workflow engines whilst providing a vehicle for sufficient and necessary status data/parameters to be exchanged between any AXWFM and the relevant AXMEDIS components through a single composite string structure (i.e. WF-Exchange-ID) that is transparently and uniquely decodable and decomposable within the individual *participant sub-systems* to provide all the necessary lifecycles information of interest locally as well as to be able to index various sub-fields that need to be combined to provide multiple or particularised views.

The Neutral Exchange Messaging Format has to be specified for the plug-ins as the logical, necessary and sufficient transaction protocol to remain both AXMEDIS-compliant as well as technologically realisable and consistent with the capabilities of existing workflow systems given their available method invocations and transaction protocols. Thus the reason for adoption of a *neutral exchange messaging format* is to provide a uniform and generic transaction standard so as to allow portability of the interfaces to various workflow systems and facilitate the development of new plug-ins.

Research will also be done for efficient tracking of digital assets within AXMEDIS framework. For this the Object's metadata information is considered to be that which allows relevant access to the status and tracking information regarding the processing and progress of the work done on any Objects and the results of various actors/tools/engines acting upon the Objects through the lifecycle of various projects. In general, the chief requirements for tracking and control is *focused situation assessment* and the semantics types recommended for the metadata and status descriptors are intended to provide the situation assessment on current state and progress of any digital assets developed during any project.

This can be done by metadata partitioning and assigning a *10-P situation assessment criterion* (as defined in DE3.1.2G Framework and Tools Specifications. This 10-P status data type is meant to provide a sufficient basis for tracking the status of AXMEDIS Objects through their development life cycles whilst allowing purpose-specifically filtered metadata-view and reasoning to serve the current focus of the actors/user/tools concerned.

The workflow rules will be defined based on the development process involved in the production and distribution of multimedia artefacts, which have been studied with the cooperation of the AXMEDIS partners involved in the respective fields (e.g. OD2, ILABS, XIM, AFI, ANSC, SERGER, etc)

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

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

It must be noted therefore that the WF-Exchange-ID to be developed must contain the *method invocation*, as well as INPUT/OUTPUT/NOTIFICATION parameters i.e. it is not just another parameter in the INPUT/OUTPUT/NOTIFICATION class invocation. So the WF-Exchange-ID format (which is basically a text string) should be compatible with any communication protocol as desired.

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

Openflow as the AXWFM Interfacing with the AXMEDIS Components

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

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

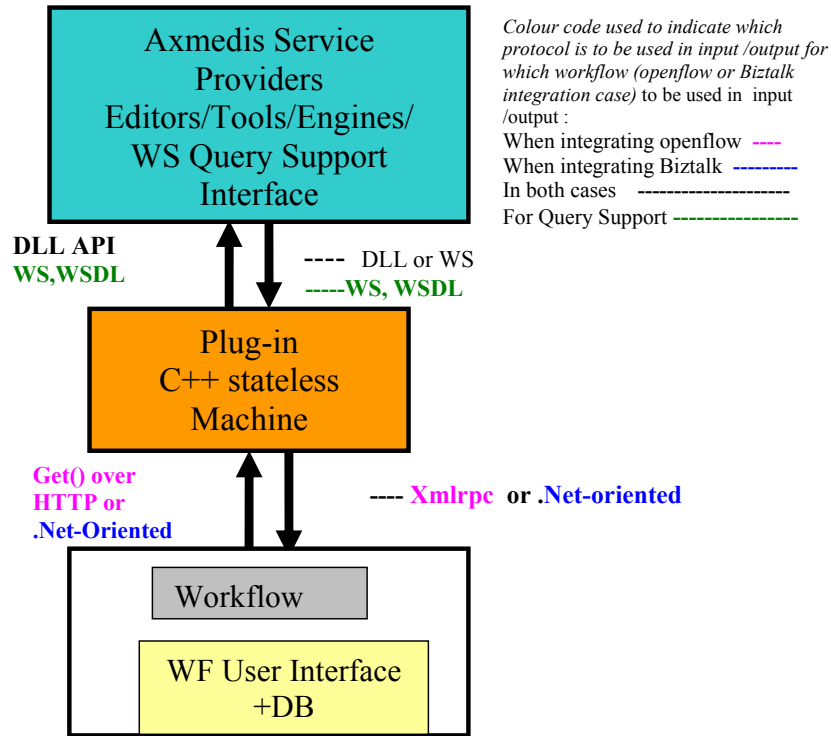
The WorkFlow Manager will be a customisation of one of the adopted Workflow Management System with some functions added, if not already present (e.g. search engine).

The Workflow User Interface will also require some customisation to serve the integration of the adopted WFMS with the AXMEDIS Framework. The User Interface will use the WorkFlow Manager class methods via the internal interface of the adopted WFMS.

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

In the case of most potential AXWFs, including Openflow, five parameters tend to be always required to be passed to an application e.g. to an AXMEDIS service provider: openflow_ID, process_ID, activity_ID, instance_ID and workitem_ID.

The Interface Protocols Diagram below illustrates the protocols to be deployed for the range of AXWFM-AXMEDIS plug-ins as listed above.



Planned schedule

- M9: Analysis and requirements specification for the workflow support for content composition
- M11: Identification of suitable workflow engines and selection of the most suitable open-source workflow environment to integrate into the AXMEDIS framework
- M13: Analysis and requirements specification of the interface to the selected workflow support for the AXMEDIS content composition and the AXEPTool
- M16: First version of the workflow support interface for the AXMEDIS content composition and the AXEPTool
- M24: Final revised specification of the workflow support interface for the AXMEDIS content composition and the AXEPTool
- M28: Final version of the workflow support interface for the AXMEDIS content composition and the AXEPTool
- M33: Updated final specification of the workflow support interface for the AXMEDIS content composition and the AXEPTool
- M36: Updated final version of the workflow support interface for the AXMEDIS content composition and the AXEPTool

--- this section is mandatory in all deliverables ---

3 AXMEDIS Architecture for Content Processing (DSI)

The AXMEDIS Content Processing Area is mainly realized as a distributed tool for processing content on the basis of rules which are written as scripts. This solution allows considering such AXCP Rule Engine as a tool to be used for content processing according to the content designer and publisher needs.

The Rules and the Engine allow the definition of algorithms for composition, formatting, adaptation, transcoding, extraction of fingerprint and descriptors, protection, license manipulation, potential available rights manipulation, resource manipulation, load and save from databases and file system, mathematics, combinatorial, etc.

The AXCP Rules can be written by referring to a large set of common data types derived from AXMEDIS Framework, MPEG21, and general resource definition such as: images, documents, video, licenses, etc. It is a sort of engine that allows to script middleware algorithms for content processing.

Thus the AXCS Engine and Rule are a versatile and customizable way to produce and manage digital content respecting legal aspects, DRM, ownership, user and publishers requirements.

In this terms, a rule:

- describes 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);
- works on content by using distribution channel properties, user device features, user profile, etc...
- generates the final output using a specific integration format (MPEG-4, SMIL,...) or using DIP capabilities provided by MPEG-21 objects
- describes 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,)...);
- describes how to manage and combine DRM rules for the new formatted resource;
- describes 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 addition, the AXCP area is capable to receive commands coming from the factory Workflow by means of a web service interface. Despite of several mentioned advantages, delegating the processing activity to a single AXCS Engine seems to be not the best solution since the amount of work in terms of elaboration time and the dimension of data that the engine has to manage can be very high in most of the content factories in which even millions of digital resources are managed per months. The main idea to solve this problem has been to design the AXCS Engine as a distributed environment of rule executors based on a **GRID infrastructure**. This solution maintains advantages of a unified solution and allows enhancing the capabilities of the AXMEDIS Content Processing area by running rules in parallel and rationally using the computational resources accessible in the content factory.

In the Figure, the relationships between the AXCP Rule Engine and the other parts of the AXMEDIS environment consisting of the *AXMEDIS Workflow Manager*, the *AXCP Rule Editor* and the *AXMEDIS Database*, are described. The figure shows also the main components and tools used by the AXCP Rule Engine.

AXMEDIS Workflow Manager to allow the reception of commands from connected workflow tools of the content factory and thus to control the AXCP Rule Engine and Editor activities.

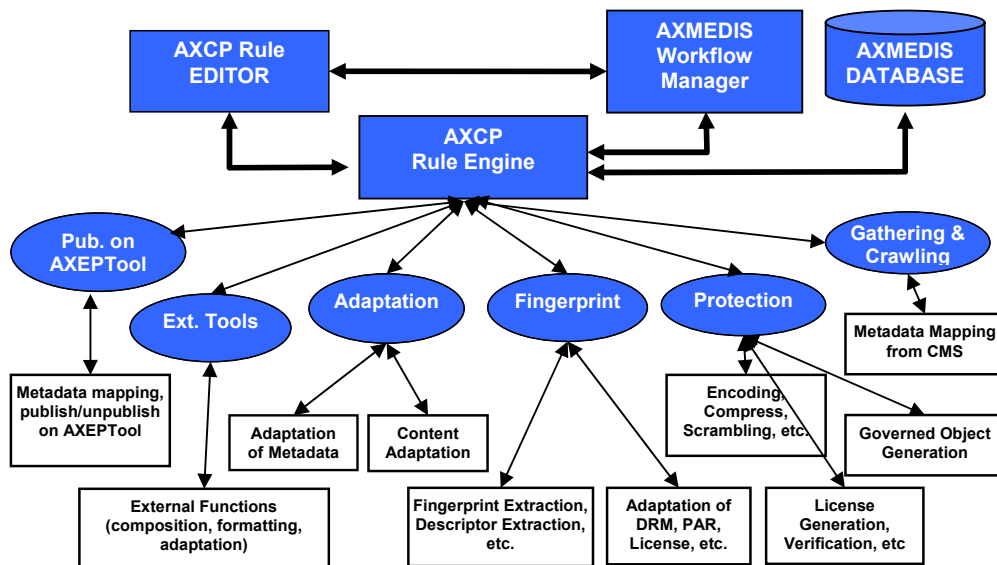
AXCP Rules Editor, which is front end tool to produce AXCP Rules for the user. It is supported by a repository of rules and interacts with the AXCP Rule Engine when a rule has to be executed.

AXCP Rule Engine to put in execution AXCP Rules. As a result of the rule execution, it may generate new AXMEDIS objects, licenses, etc. Such information may be stored in the AXMEDIS database to be further delivered via distribution channels. The Engine is supported by the set of plug-ins, called AXMEDIS plug-ins, that allows performing several functionalities and coping with:

1. **Protection** algorithms performing the protection of the AXMEDIS objects, license production and processing, etc.
2. **Fingerprint/Descriptor** algorithms for estimation of fingerprints and descriptors of AXMEDIS objects and contained digital resources.
3. **Adaptation** algorithms for content/resource adaptation, DRM and license adaptation, etc., that typically needed for different distribution channels and format paradigms.
4. **External tools** for the interaction with commercial tools and libraries for exploiting their formatting and processing functionalities.
5. **Publication** algorithms to perform metadata manipulation, gathering and mapping, publication of AXMEDIS objects on external channels and in particular on the B2B distribution.

This report describes the development activity performed in the AXMEDIS Content Processing Area and fix the current status of implemented features related to the following items:

- AXMEDIS Rule Model
- AXMEDIS AXCP Rule Editor
- AXMEDIS AXCP Rule Scheduler
- AXMEDIS AXCP Rule Executor
- Composition and Formatting tools, algorithms and formalizations
- Content Adaptation: Digital Resource Transcoding (audio, video, image, multimedia, etc....)



Content Processing Flow Diagram and Relationships

4 AXMEDIS Rule Definition and Model

In this section the status of work performed in defining the model of AXCP Rule is reported. This report shows the main concepts, formalization and implementation of the model.

4.1 Technical Details

reference to the AXFW location of the demonstrator	https://cvs.axmedis.org/repos/Framework/source/rulemodel/
List of libraries used	None
References to other components needed	Selection
Problems not solved	-- ---
Configuration and execution context	

The entire content production process is driven by rules called AXMEDIS Content Processing rules (AXCP Rule). A rule was formalised as a function in the following way:

$$R = f(S1, S2, \dots, Sn, P1, \dots, Pm)$$

Where:

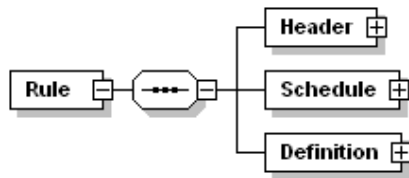
- S_i – It defines a selection. A selection is a sequence of query to be sent to the Query Support for AXMEDIS objects retrieval or references to digital resource embedded into an composite AXEMDIS object;
- P_i – It is a parameter (basic type as integer, string, boolean, float);
- f is the identifier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

4.2 AXCP Rule XML formalisation

The following XML schema refers to the “Rule_Axmedis.xsd” file.

element **Rule**

diagram



children

[Header](#) [Schedule](#) [Definition](#)

source

```

<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:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="Schedule"/>
      <xs:element name="Definition"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
  
```



```

<xs:simpleType>
  <xs:restriction base="xs:string">
    <xs:enumeration value="AXCP"/>
    <xs:enumeration value="AXnP"/>
  </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:string"/>
<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:complexType>
                <xs:simpleContent>
                  <xs:extension base="xs:integer">
                    <xs:attribute name="Unit" type="periodunit"/>
                  </xs:extension>
                </xs:simpleContent>
              </xs:complexType>
            </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:choice maxOccurs="unbounded">
                  <xs:element name="Parameter" minOccurs="0" maxOccurs="unbounded">
                    <xs:complexType>
                      <xs:simpleContent>
                        <xs:extension base="xs:string">
                          <xs:attribute name="Name" type="xs:string" use="required"/>
                          <xs:attribute name="Type" type="xs:string" use="required"/>
                        </xs:extension>
                      </xs:simpleContent>
                    </xs:complexType>
                  </xs:element>
                  <xs:element ref="selection" minOccurs="0" maxOccurs="unbounded"/>
                </xs:choice>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:choice>
  </xs:complexType>
</xs:element>

```

```

</xs:choice>
</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="Dependencies" minOccurs="0">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Dependency" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Plug_In_name" type="xs:string"/>
            <xs:element name="Version" type="xs:string"/>
          </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>

```

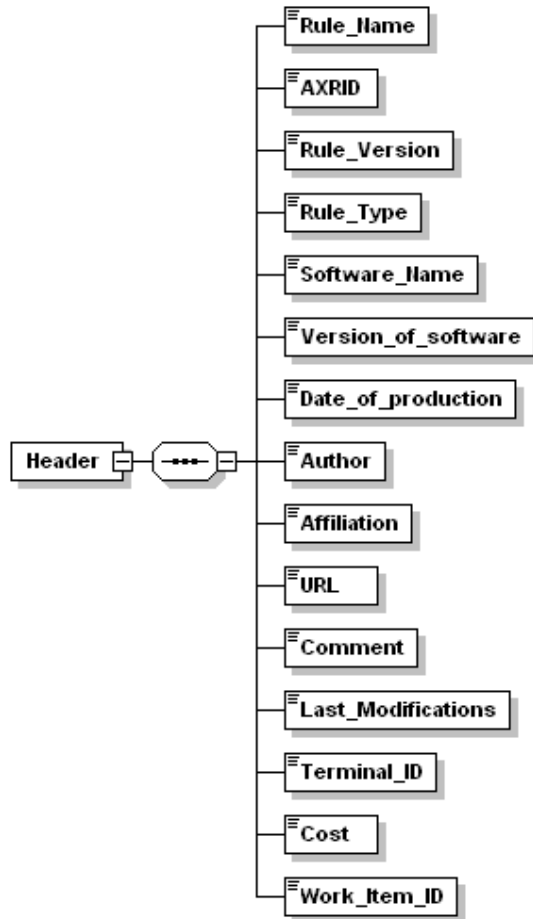
description

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

element **Rule/Header**

diagram



children

[Rule Name](#) [AXRID](#) [Rule Version](#) [Rule Type](#) [Software Name](#) [Version of software](#) [Date of production](#) [Author](#)
[Affiliation](#) [URL](#) [Comment](#) [Last Modifications](#) [Terminal ID](#) [Cost](#) [Work Item ID](#)

source

```
<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="AXCP"/>
            <xs:enumeration value="AXnP"/>
          </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:string"/>
      <xs:element name="Cost" type="xs:string"/>
      <xs:element name="Work_Item_ID" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

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

description This section contains metadata related to general information associated with a rule

element Rule/Header/Rule_Name

diagram



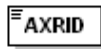
type **xs:string**

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

description It defines the name of the rule, e.g. "Audio Collection"

element Rule/Header/AXRID

diagram



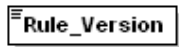
type **xs:string**

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

description It defines the AXMEDIS Rule ID

element Rule/Header/Rule_Version

diagram



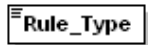
type **xs:string**

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

description It defines the version of the rule, e.g. "1.0"

element Rule/Header/Rule_Type

diagram



type restriction of **xs:string**

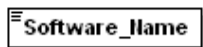
facets
enumeration AXCP
enumeration AXPnP

source `<xs:element name="Rule_Type">
 <xs:simpleType>
 <xs:restriction base="xs:string">
 <xs:enumeration value="AXCP"/>
 <xs:enumeration value="AXPNP"/>
 </xs:restriction>
 </xs:simpleType>
</xs:element>`

description 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

element Rule/Header/Software_Name

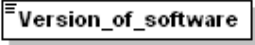
diagram



type **xs:string**

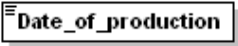
source `<xs:element name="Software_Name" type="xs:string"/>`
description It specifies the name of software used, e.g. "Axmedis Rule Editor"

element Rule/Header/Version_of_software

diagram A diagram showing a rectangular box with the text "Version_of_software" inside. To the left of the box is a small icon consisting of three horizontal lines of decreasing width, resembling a list or menu icon.

type **xs:string**
source `<xs:element name="Version_of_software" type="xs:string"/>`
description It defines the version of software used., e.g. "2.0"

element Rule/Header/Date_of_production

diagram A diagram showing a rectangular box with the text "Date_of_production" inside. To the left of the box is a small icon consisting of three horizontal lines of decreasing width, resembling a list or menu icon.

type **xs:date**
source `<xs:element name="Date_of_production" type="xs:date"/>`
description It defines when the rule has been created
Note This item now embeds the item Time_of_Production defined in the DE3-1-2C-AXFW_Spec-(content-production)-Part-C document.

:

element Rule/Header/Author

diagram A diagram showing a rectangular box with the text "Author" inside. To the left of the box is a small icon consisting of three horizontal lines of decreasing width, resembling a list or menu icon.

type **xs:string**
source `<xs:element name="Author" type="xs:string"/>`
description It defines the name of author who has created the rule

element Rule/Header/Affiliation

diagram A diagram showing a rectangular box with the text "Affiliation" inside. To the left of the box is a small icon consisting of three horizontal lines of decreasing width, resembling a list or menu icon.

type **xs:string**
source `<xs:element name="Affiliation" type="xs:string"/>`
description It defines the name of Affiliation

element Rule/Header/URL

diagram A diagram showing a rectangular box with the text "URL" inside. To the left of the box is a small icon consisting of three horizontal lines of decreasing width, resembling a list or menu icon.

type **xs:anyURI**
source `<xs:element name="URL" type="xs:anyURI"/>`
description It defines the Internet address/URL of the Affiliation

element Rule/Header/Comment

diagram



type **xs:string**

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

description It allows describing what the rule does

element Rule/Header/Last_Modifications

diagram



type **xs:date**

source `<xs:element name="Last_Modifications" type="xs:date"/>`

description It is used to report the last modified date

element Rule/Header/Terminal_ID

diagram



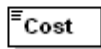
type **xs:string**

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

description The Id of the terminal used to write the rule.

element Rule/Header/Cost

diagram



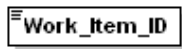
type **xs:string**

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

description Estimation of Cost

element Rule/Header/Work_Item_ID

diagram



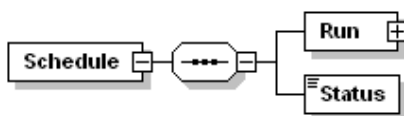
type **xs:string**

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

description External reference, for instance the commitment

element Rule/Schedule

diagram



children [Run](#) [Status](#)

```

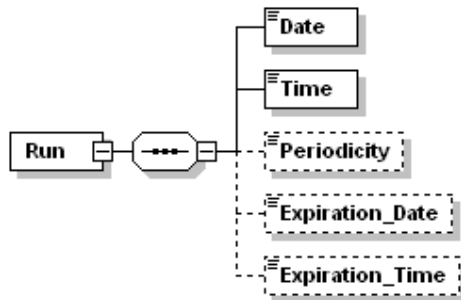
source
<xs:element name="Schedule">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Run">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Date" type="xs:date"/>
            <xs:element name="Time" type="xs:time"/>
            <xs:element name="Periodicity" minOccurs="0">
              <xs:complexType>
                <xs:simpleContent>
                  <xs:extension base="xs:integer">
                    <xs:attribute name="Unit" type="periodunit"/>
                  </xs:extension>
                </xs:simpleContent>
              </xs:complexType>
            </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>

```

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

element Rule/Schedule/Run

diagram



children [Date](#) [Time](#) [Periodicity](#) [Expiration_Date](#) [Expiration_Time](#)

```

source
<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:complexType>
          <xs:simpleContent>
            <xs:extension base="xs:integer">
              <xs:attribute name="Unit" type="periodunit"/>
            </xs:extension>
          </xs:simpleContent>
        </xs:complexType>
      </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>

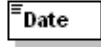
```

`</xs:element>`

description It defines a subsection of metadata that describes information needed for scheduling the execution of the rule.

element Rule/Schedule/Run/Date

diagram



type **xs:date**

source `<xs:element name="Date" type="xs:date"/>`

description It defines when the rule has to be executed by the engine in terms of day, month and year.

element Rule/Schedule/Run/Time

diagram



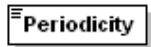
type **xs:time**

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

description It defines when the rule has to be executed by the engine in term of time clock.

element Rule/Schedule/Run/Periodicity

diagram



type extension of **xs:integer**

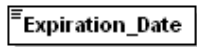
attributes	Name	Type	Use	Default	Fixed	Annotation
	Unit	periodunit				

source `<xs:element name="Periodicity" minOccurs="0">
 <xs:complexType>
 <xs:simpleContent>
 <xs:extension base="xs:integer">
 <xs:attribute name="Unit" type="periodunit"/>
 </xs:extension>
 </xs:simpleContent>
 </xs:complexType>
 </xs:element>`

description It defines if a rule has to be executed periodically, every "Unit" "periodunit" e.g. "2" "Week"

element Rule/Schedule/Run/Expiration_Date

diagram



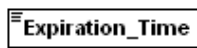
type **xs:date**

source `<xs:element name="Expiration_Date" type="xs:date" minOccurs="0"/>`

description The date to stop the periodicity

element Rule/Schedule/Run/Expiration_Time

diagram



type **xs:time**

source `<xs:element name="Expiration_Time" type="xs:time" minOccurs="0"/>`

description [The time to stop the periodicity](#)

element Rule/Schedule/Status

diagram



type restriction of **xs:string**

facets enumeration Active
 enumeration Inactive

source

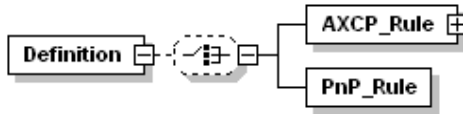
```
<xs:element name="Status">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Active"/>
      <xs:enumeration value="Inactive"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

description It defines if a rule is:

- "active" and can be executed
- "inactive"

element Rule/Definition

diagram



children [AXCP_Rule](#) [PnP_Rule](#)

source

```
<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:choice maxOccurs="unbounded">
                  <xs:element name="Parameter" minOccurs="0" maxOccurs="unbounded">
                    <xs:complexType>
                      <xs:simpleContent>
                        <xs:extension base="xs:string">
                          <xs:attribute name="Name" type="xs:string" use="required"/>
                          <xs:attribute name="Type" type="xs:string" use="required"/>
                        </xs:extension>
                      </xs:simpleContent>
                    </xs:complexType>
                  </xs:element>
                  <xs:element ref="selection" minOccurs="0" maxOccurs="unbounded"/>
                </xs:choice>
              </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="Dependencies" minOccurs="0">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="Dependency" minOccurs="0" maxOccurs="unbounded">
                    <xs:complexType>
```

```

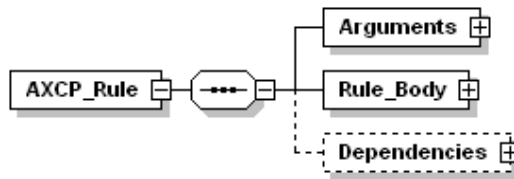
<xs:sequence>
  <xs:element name="Plug_In_name" type="xs:string"/>
  <xs:element name="Version" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="PnP_Rule"/>
</xs:choice>
</xs:complexType>
</xs:element>

```

description This section includes the section containing the procedural description of an AXCP or a PnP rule

element Rule/Definition/AXCP_Rule

diagram



children

[Arguments](#) [Rule](#) [Body](#) [Dependencies](#)

source

```

<xs:element name="AXCP_Rule">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Arguments">
        <xs:complexType>
          <xs:choice maxOccurs="unbounded">
            <xs:element name="Parameter" minOccurs="0" maxOccurs="unbounded">
              <xs:complexType>
                <xs:simpleContent>
                  <xs:extension base="xs:string">
                    <xs:attribute name="Name" type="xs:string" use="required"/>
                    <xs:attribute name="Type" type="xs:string" use="required"/>
                  </xs:extension>
                </xs:simpleContent>
              </xs:complexType>
            </xs:element>
            <xs:element ref="selection" minOccurs="0" maxOccurs="unbounded"/>
          </xs:choice>
        </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="Dependencies" minOccurs="0">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Dependency" minOccurs="0" maxOccurs="unbounded">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="Plug_In_name" type="xs:string"/>
                  <xs:element name="Version" type="xs:string"/>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

```

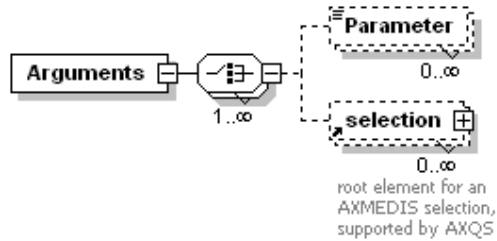
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>

```

description This section include the AXCP Rule section containing the procedural description of the rule

element Rule/Definition/AXCP_Rule/Arguments

diagram



children [Parameter selection](#)

```

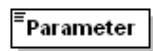
<xs:element name="Arguments">
  <xs:complexType>
    <xs:choice maxOccurs="unbounded">
      <xs:element name="Parameter" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:simpleContent>
            <xs:extension base="xs:string">
              <xs:attribute name="Name" type="xs:string" use="required"/>
              <xs:attribute name="Type" type="xs:string" use="required"/>
            </xs:extension>
          </xs:simpleContent>
        </xs:complexType>
      </xs:element>
      <xs:element ref="selection" minOccurs="0" maxOccurs="unbounded"/>
    </xs:choice>
  </xs:complexType>
</xs:element>

```

description It includes the set of selections and parameters that rule has in input.

element Rule/Definition/AXCP_Rule/Arguments/Parameter

diagram



type extension of **xs:string**

attributes	Name	Type	Use	Default	Fixed	Annotation
	Name	xs:string	required			
	Type	xs:string	required			

```

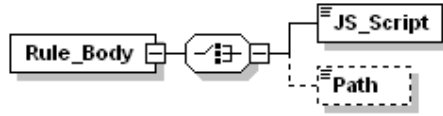
<xs:element name="Parameter" minOccurs="0" maxOccurs="unbounded">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="Name" type="xs:string" use="required"/>
        <xs:attribute name="Type" type="xs:string" use="required"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

```

description It defines a parameter in input to the rule by providing the name, the type and the actual value

element Rule/Definition/AXCP_Rule/Rule_Body

diagram



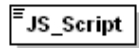
children [JS_Script Path](#)

```
<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>
```

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

element Rule/Definition/AXCP_Rule/Rule_Body/JS_Script

diagram



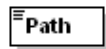
type **xs:string**

```
<xs:element name="JS_Script" type="xs:string"/>
```

description It is used to embed the whole script (JavaScript code) inside the XML rule format.

element Rule/Definition/AXCP_Rule/Rule_Body/Path

diagram



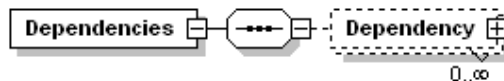
type **xs:anyURI**

```
<xs:element name="Path" type="xs:anyURI" minOccurs="0"/>
```

description It is used to specify a reference to a ".js" file that contains the source script code of the current rule (JavaScript code).

element Rule/Definition/AXCP_Rule/Dependencies

diagram



children [Dependency](#)

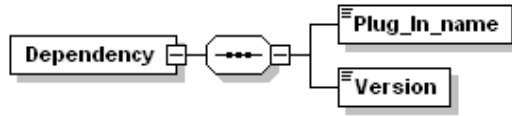
```
<xs:element name="Dependencies" minOccurs="0">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Dependency" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Plug_In_name" type="xs:string"/>
            <xs:element name="Version" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

description It contains a list of possible dependencies

Note [This item replaces the Precondition item defined in the schema reported in DE3-1-2C-AXFW_Spec-\(content-production\)-Part-C document.](#)

element **Rule/Definition/AXCP_Rule/Dependencies/Dependency**

diagram



children [Plug_In_name](#) [Version](#)

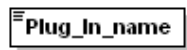
source

```
<xs:element name="Dependency" minOccurs="0" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Plug_In_name" type="xs:string"/>
      <xs:element name="Version" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

description It 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.

element **Rule/Definition/AXCP_Rule/Dependencies/Dependency/Plug_In_name**

diagram



type **xs:string**

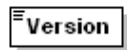
source

```
<xs:element name="Plug_In_name" type="xs:string"/>
```

description 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.

element **Rule/Definition/AXCP_Rule/Dependencies/Dependency/Version**

diagram



type **xs:string**

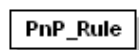
source

```
<xs:element name="Version" type="xs:string"/>
```

description Version of the Plug In. This information has to be matched with that provided by the DLL from its profile.

element **Rule/Definition/PnP_Rule**

diagram



source

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

simpleType **periodunit**

type restriction of **xs:string**

used by attribute [Rule/Schedule/Run/Periodicity/@Unit](#)

facets

- enumeration Day
- enumeration Month
- enumeration Week
- enumeration Year

source

```
<xs:simpleType name="periodunit">
```

```

<xs:restriction base="xs:string">
  <xs:enumeration value="Day"/>
  <xs:enumeration value="Month"/>
  <xs:enumeration value="Week"/>
  <xs:enumeration value="Year"/>
</xs:restriction>
</xs:simpleType>

```

4.2.1 AXCP Rule Model

According to the XML schema, the object oriented model of the rule is described by the class diagram reported in the Figure. Actually, the current status of the model is close to the final version. Implementation was realised in C++ MSVC7 and supported by the XERCES 2.6.0 libraries.

AxDOMImplementation Class

The *AxDOMImplementation* class specialises the *ErrorHandler* interface class of the XERCES library ver. 2.6.0. by redefining the pure virtual methods:

- virtual void **warning** (const [SAXParseException](#) &exc)=0
Receive notification of a warning. http://xml.apache.org/xerces-c/apiDocs/classErrorHandler.html - z819_0#z819_0
- virtual void **error** (const [SAXParseException](#) &exc)=0
Receive notification of a recoverable error. http://xml.apache.org/xerces-c/apiDocs/classErrorHandler.html - z819_1#z819_1
- virtual void **fatalError** (const [SAXParseException](#) &exc)=0
Receive notification of a non-recoverable error. http://xml.apache.org/xerces-c/apiDocs/classErrorHandler.html - z819_2#z819_2
- virtual void **resetErrors** ()=0
Reset the Error handler object on its reuse.

Such methods are called if the validation of the XML Rule file fails during the parsing. The *AxDOMImplementation* class does not consider warnings and puts the *fSawErrors* attribute at *true* when an error occurs. The *getSawErrors()* method returns the *fSawErrors* value and then allows evaluating the result of validation.

The class provides a pointer to the DOM tree (*DOMDoc*) of the all XML rule and is fixed by the *buildDOMDoc()* method. Such method was defined as virtual demanding its implementation to the *AxRule* class. It is equipped with a *XercesDOMParser (parser)* for loading the whole rule and *XercesDOMParser (parser)* for managing the XML string of a *Selection* in independent manner when saving the rule.

AxRuleHeader Class

The rule header class for storing the *AxRule* header information with access methods. It manages all properties of the *Header* section according to the XML schema. It provides getter and setter methods to access attributes.

AxRuleSchedule Class

The rule schedule class for storing the *AxRule* schedule information with access methods. It manages all properties of the *Schedule* section according to the XML schema. It provides getter and setter methods to access attributes.

AxRule Class

The *AxRule* is the main class for a rule. It specialises the *AxDOMImplementation* and redefines the *buildDOMDoc()* method for building the DOM tree during the load and save phases. It encapsulates the

AxRuleHeader and the *AxRuleSchedule* class. In this way, the *AxRule* class is the common class for the AXCP and AXpNP rules. Each of them has to define the *Definition* section according to Rule XML Schema in order to specialise the *AXRule*.

The *AxRule* class provides the *buildXMLString()* to build a string reporting the XML code of the rule, the *load()*, *save()* and *saveAs()* methods. The loading and saving methods instantiate an *AxRuleVisitor* object that is passed as argument to the *visit()* method. The *AxRuleVisitor* object will be an *AxRuleLoader* when loading and an *AxRuleSaver* when saving; in both case the polymorphic *visit()* method is called

AxCPRule Class

It specialises the *AxRule* class and inherits all methods. It adds methods and attributes to manage the *Definition* section of the XML schema. *AxRuleArgumentList* and *AxRuleDependencyList* contain respectively *AxRuleArgument* and *AxRuleDependency* object, they model arguments and dependencies of the rule. The class provides also direct access methods to the internal list (getter and setter, item counting, access and removal). The javascript code is stored in the *javascript* string attribute. Optionally, a file with the javascript code could be referred by means the *javascriptPath* string attribute (this alternative is not used at now).

AxRuleArgumentList

This class manages the list of arguments, It provides methods for accessing, reading, adding, removing a *AxRuleArgument* object.

AxRuleArgument

The *AxRuleArgument* models a generic argument of rule. The *argId* attributes specifies if the argument is a parameter (*axID_PARAMETER*) or a selection (*axID_SELECTION*). The class provides also the *name* attribute.

AxRuleParameter

It specialises the *AxRuleArgument* by defining the *type* and the *value* of the parameter. The *value* attribute is stored as string, whereas the *type* is associated with the enumerate values (*axINTEGER = 0*, *axREAL*, *axSTRING*, *axXMLSTRING*, *axDATE*, *axTIME*, *axBOOLEAN*, *axURL*, *axNULL*).

AxRuleSelection

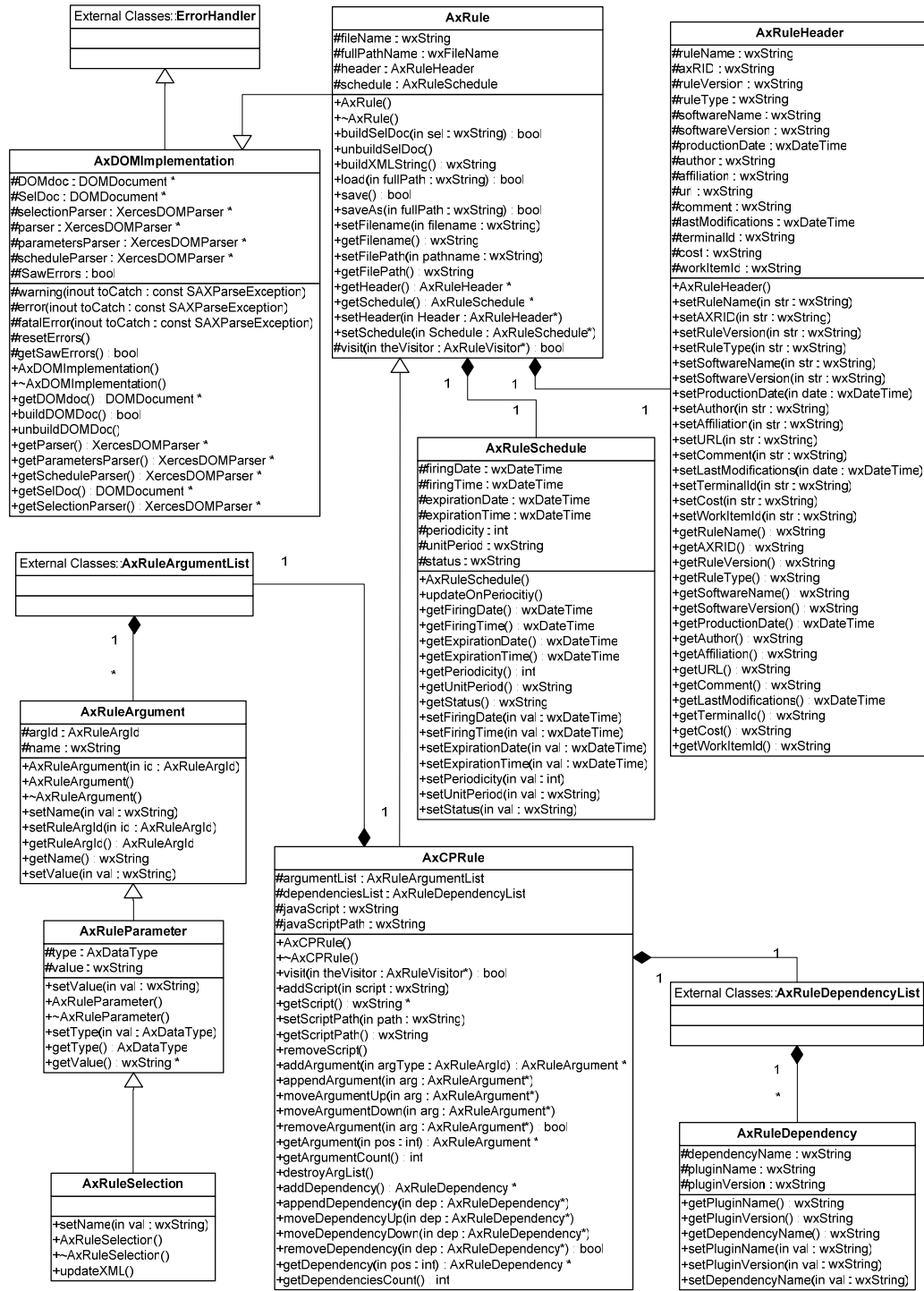
It is a particular type of parameter. A selection is stored as a full XML string according to the Selection schema. The *AxRuleSelection* class specialises the *AxRuleParameter* by adding a method (*updateXML()*) to manage the change of the selection name directly on the XML string when calling the *setName()* method.

AxRuleDependencyList

This class manages the list of dependencies, It provides methods for accessing, reading, adding, removing a *AxRuleDependency* object.

AxRuleDependency

This class manages the dependency of the javascript code to a specific Axmedis plugin. It provides getter and setter methods to access *dependencyName*, *pluginName* and *pluginVersion* attributes.



Rule Model Class Diagram

4.3 AXRule Loader and Saver Modules (DSI)

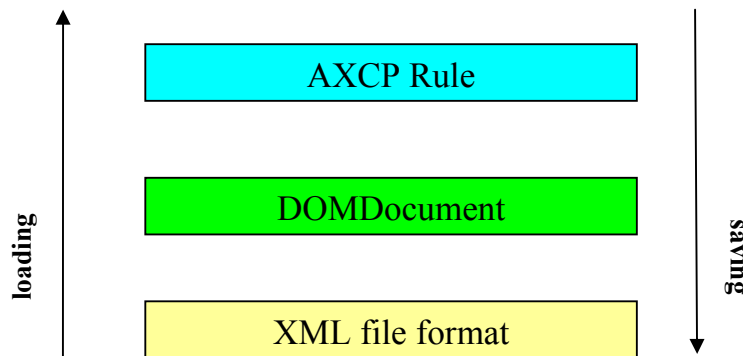
To manage the repository of rules it is necessary to load and save a rule in/from the system. The Rule Load and Save functionalities were designed and implemented by means of the following classes:

AXRule Loader - It is the module for loading an XML representation of the rule in the AXCP Rule Editor and AXCP Rule Engine. It works according to the XML rule specification and provides the following functionalities:

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

AXRule Saver – It is the module for saving an XML representation of the rule on disk. It works 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 was implemented by using an abstract class called *AXRuleVisitor* as reported in the UML diagram. This solution allows building an *AXRuleLoader* and an *AXRuleSaver* class that manage different types of rules by implementing different *Visit* methods (see the class diagram reported below). Both classes are related to a DOM Document 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 provides a *Load* and *Save* method and a virtual method *Visit* that have to be redefined in the *AXCP Rule* class. In this way, the *Visit* method of AXCP Rule calls the *Visit* method of *AXRuleLoader* on the AXCP Rule object by using the *this* pointer.

Implementation was realised in C++ MSVC7 and supported by wxWidgets ver. 2.4.2 and XERCES 2.6.0 libraries.

4.3.1 AxRuleVisitor Class

This class defines an abstract class for a Visitor. This class allows building a *Loader* and a *Saver* that manage different types of rules by implementing different *Visit* methods. The loader and saver visitor were implemented as specialized class of *AxRuleVisitor*.

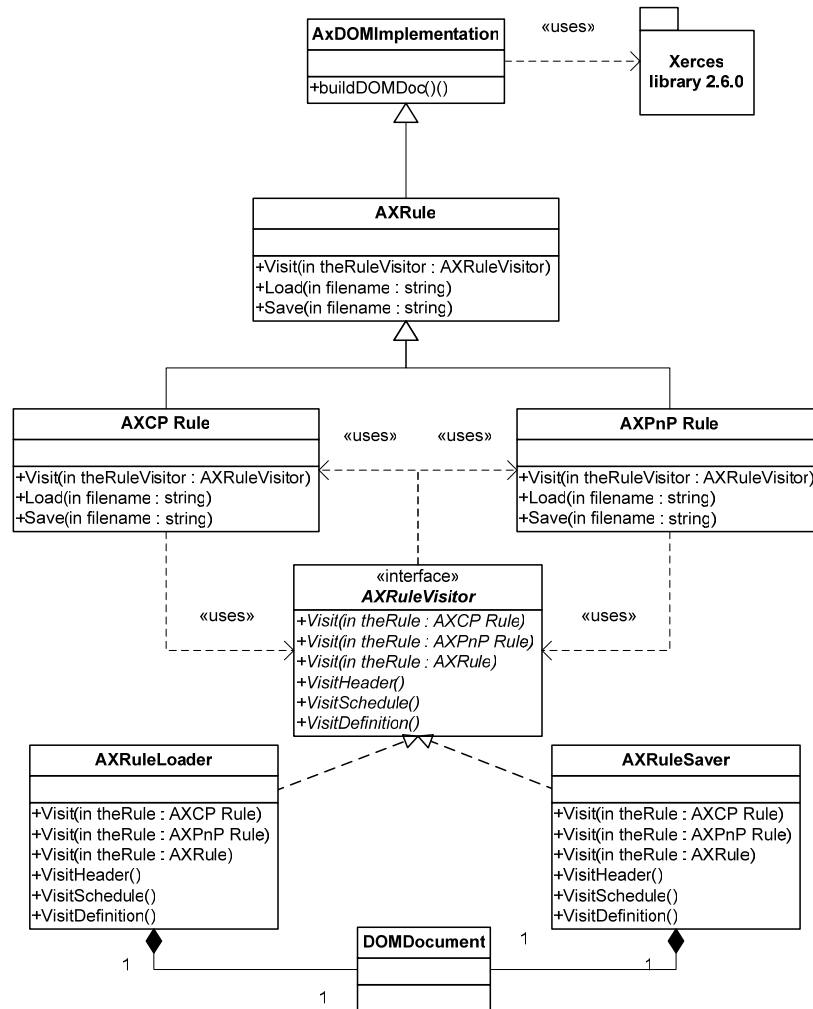
4.3.2 AxRuleLoader Class

This class is a specialisation of the **AxRuleVisitor** class and implements a visitor for loading both **AxRule** and **AxCPRule** XML file. Since **AxRule** contains only *Header* and *Schedule* section, the visitor loader for such rule will read only *Header* and *Scheduler* sections. The visitor loader for the **AxCPRule** will call the visitor loader of the **AxRule** and then will read the *Definition* section. Methods of this class work polymorphically.

4.3.3 AxRuleSaver Class

This class is a specialisation of the **AxRuleVisitor** class and implements a visitor for saving both **AxRule** and **AxCPRule** XML file. Since **AxRule** contains only *Header* and *Schedule* section, the visitor saver for such rule will write only *Header* and *Scheduler* sections. The visitor saver for the **AxCPRule** will call the

visitor saver of the **AxRule** and then will write the *Definition* section. Methods of this class work polymorphically.



UML Class Diagram of the AxRuleLoader and AxRuleSaver modules

5 AXMEDIS Rule Editor (DSI)

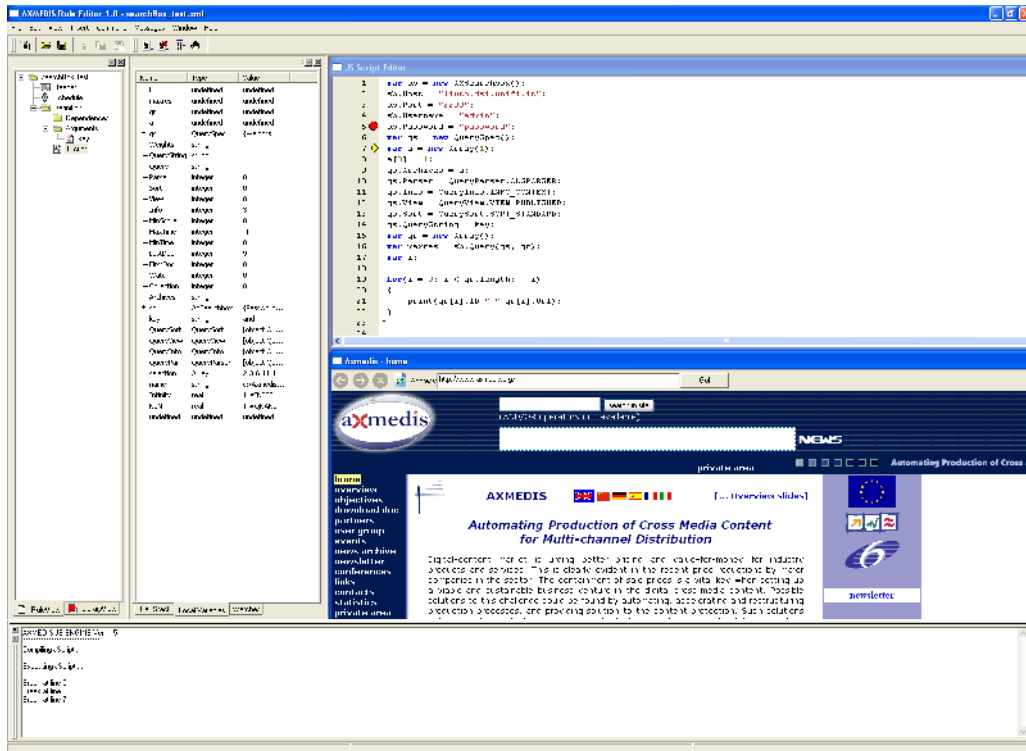
The AXCP Rule Editor is the editor that allows writing AXCP Rule. The following sections report the current status of the prototype under development and the list of functionalities that are available in the current version of the prototype.

5.1 Technical Details

reference to the AXFW location of the demonstrator	https://cvs.axmedis.org/repos/Framework/source/ruleeditor/
List of libraries used	wxWidgets ver. 2.4.2 Mozilla, STC (wxStyleTextControl classes embedding Scintilla Editor) and FL (wxWidgets contribution) libraries.
References to other major components needed	Rule Excutor, Rule Model, AXOM
Problems not solved	<ul style="list-style-type: none"> • Integration of the AXMDIS Plugin manager • Integration of communication support with the Workflow Manager
Configuration and execution context	
Programming language	C++ MSVC7

5.2 Description AXMEDIS Rule Editor (Work Done)

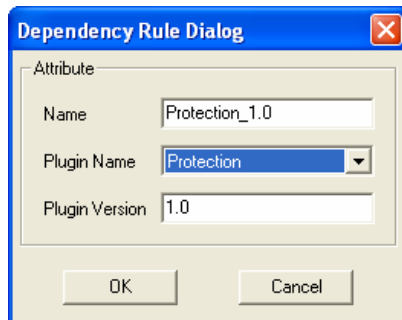
The AXCP Rule Editor GUI is a MDI window that manages a rule document at time. The editor provides a set of tools and views to help the user during the editing and building of rule, writing javascript code. It hosts an instance of the rule executor in order to provide functionalities for debugging, testing and validating the javascript code. The main architecture of GUI is reported in the following picture:



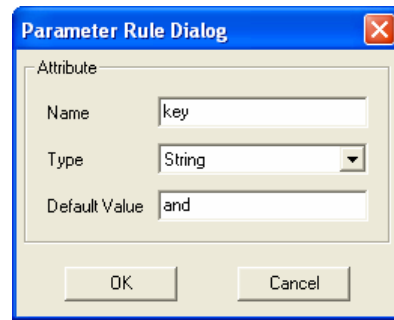
Main snapshot of the Axmedis Rule Editor GUI

Implemented features:

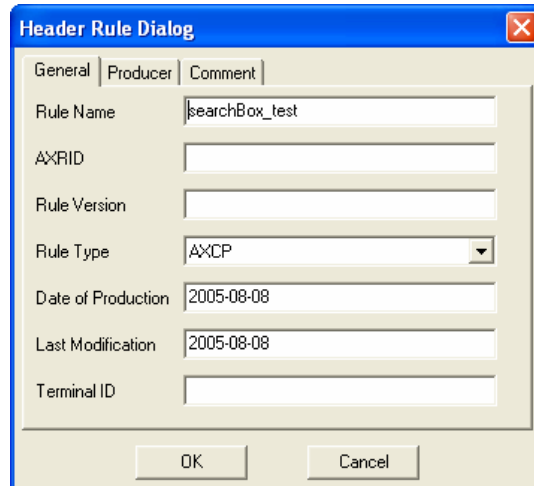
- Docking windows and toolbars
- Embedded the rule executor for running, compiling and debugging the rule (JavaScript code)
- Load & save of an AXCP rule
- Mozilla Browser based on wxMozilla contribution for browsing html help pages
- Dialogs/Tools for Edit Rule components
 - ♣ Header dialog for editing of rule header attributes
 - ♣ Schedule dialog for editing of schedule attributes
 - ♣ Parameter dialog for editing of attributes of a rule parameter
 - ♣ Dependency dialog for editing of attributes of a AXMEDIS PlugIn
 - ♣ XML Selection Editor (XML viewer/editor for the XML representation of selections) and JavaScript Editor based on Scintilla Editor for text/javascript code editing. It provides full editing capabilities (copy, cut, paste, redo, undo, syntax highlighting, etc...), print preview, page setup and print functionalities, syntax highlighting, brace highlighting, Folding/Hiding of lines, Breakpoint insertion/removal, Visualisation of line numbers



Dependency dialog



Parameter dialog



Header Rule Dialog

General | Producer | Comment

Rule Name:

AXRID:

Rule Version:

Rule Type:

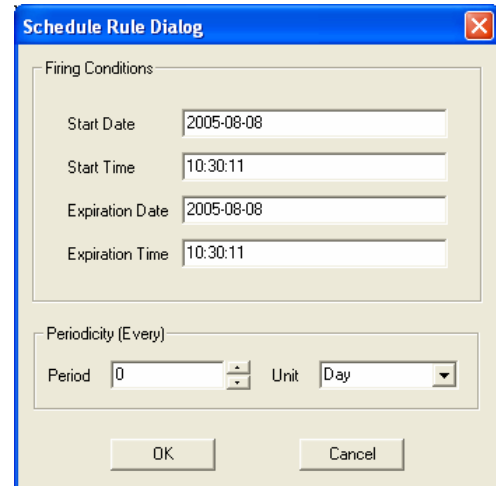
Date of Production:

Last Modification:

Terminal ID:

OK Cancel

Header dialog



Schedule Rule Dialog

Firing Conditions

Start Date:

Start Time:

Expiration Date:

Expiration Time:

Periodicity (Every)

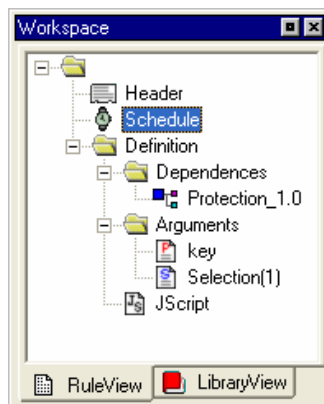
Period: Unit:

OK Cancel

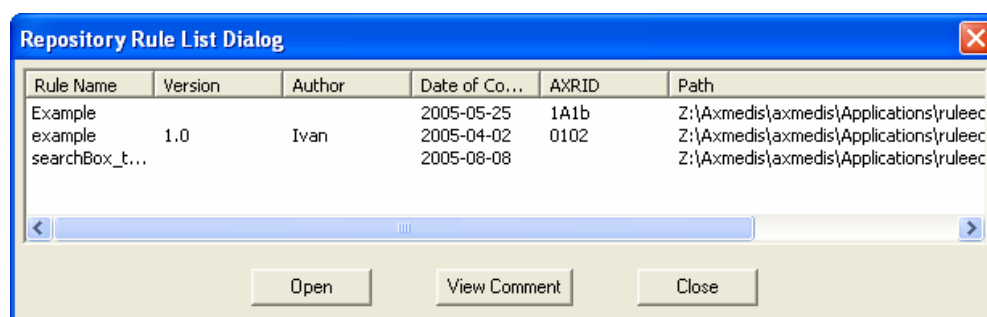
Schedule dialog

Editing dialogs

- Workspace Area is a resizable docking panel and includes a notebook control where the rule view is shown. Such view is a tree view. A dynamic popup menu is available for a quick access to functions that allows the quick management of items (edit and view metadata, delete, Insert, Cancel, Move up/Down, Rename, Open/Edit, ...). Appropriate icons allows identifying intuitively components of rule in view. In the following picture the actual structure of the Rule View area is depicted:



- Data interface to manage and synchronise tree items with rule items
- Drag and Drop of an XML AXCP Rule file
- Rule List: function for rule searching on the Rule Repository folder

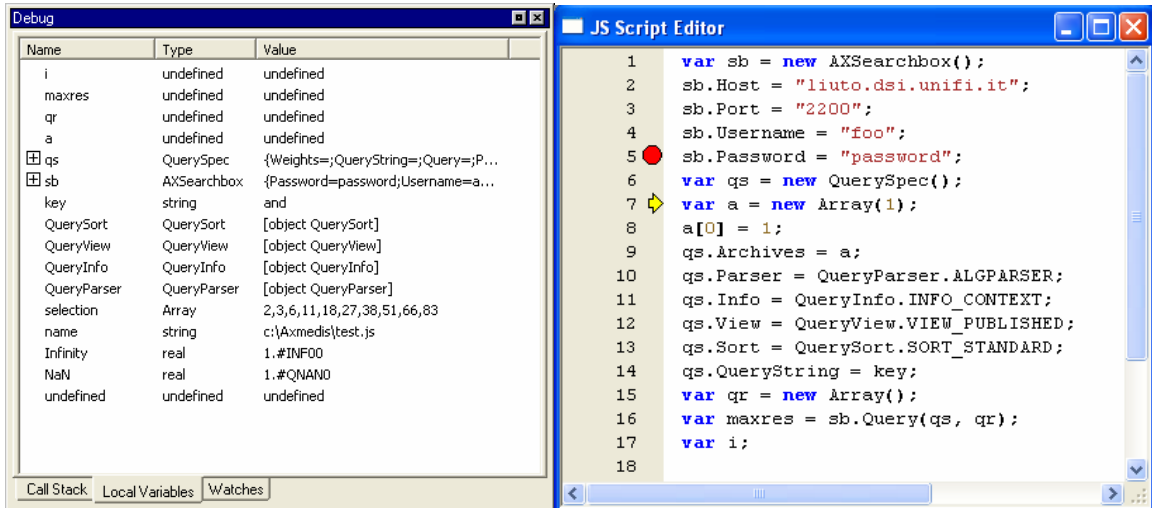


Repository Rule List Dialog

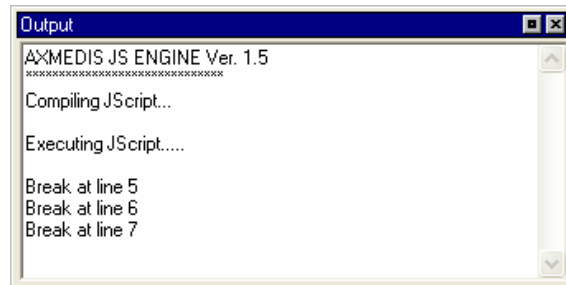
Rule Name	Version	Author	Date of Co...	AXRID	Path
Example			2005-05-25	1A1b	Z:\Axmedis\axmedis\Applications\ruleec
example	1.0	Ivan	2005-04-02	0102	Z:\Axmedis\axmedis\Applications\ruleec
searchBox_t...			2005-08-08		Z:\Axmedis\axmedis\Applications\ruleec

Open View Comment Close

- Debugging Rule functionalities: add/remove breakpoints, start debug, next breakpoint, step over, stack calls monitoring, local variables visualization.



- Print output window for visualising internal errors when script runs and monitoring the debug session



Missing Features that will be implemented:

- Integration of the AXMEDIS PlugIn manager and implementation of the Library view (Info View)
- Info View will be as an on-line book that will 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 the selected function by interacting with the corresponding item of the tree. To this end a contextual popup menu will be designed. The required documentation will be displayed in the Mozilla browser.
- Watching variables during debugging session
- Activation of Rule will send the current rule to the scheduler. A connection with the Rule Engine will allow the installation of rule in the Scheduler.
- Checking rule will test 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)
- Importing and Exporting external JS script file (*.js)
- Editing facilities for the script editor (intellisense, keywords suggestion, etc..)
- AXCP Rule Editor Configuration for managing gui properties and layout
- Command & reporting for communication with the AXMEDIS workflow

6 AXMEDIS Rule Scheduler (DSI)

As already discussed, the content processing activity (production, protection and publication on AXEPTool) is based on a unified and shared solution. In these terms, the AXCP Rule Engine plays 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 seemed to be not the best solution since the amount of work and the dimension of data that the engine have to manage is high. The main idea was 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.

The AXCP Rule engine was divided in two main components:

- **Rule Scheduler (Server Side)** – It consists of the an internal 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 runs the JavaScript code associated with rule.

A **Grid infrastructure** was realised by means P2P technology to support the distributed environment. For these reason both the Rule Scheduler and the Rule Executors has been equipped with a P2P communication support.

In this section, the status of the AXCP Rule Scheduler prototype is reported.

6.1 Technical Details

reference to the AXFW location of the demonstrator	https://cvs.axmedis.org/repos/Framework/source/rulescheduler/
List of libraries used	wxWidgets library ver. 2.4.2, xerces 2.6.0
References to other major components needed	AXMEDIS Rule Data Model for data rule memory representation and load & save functions.
Problems not solved	<ul style="list-style-type: none"> • •
Configuration and execution context	
Programming language	: C++ MSVC7

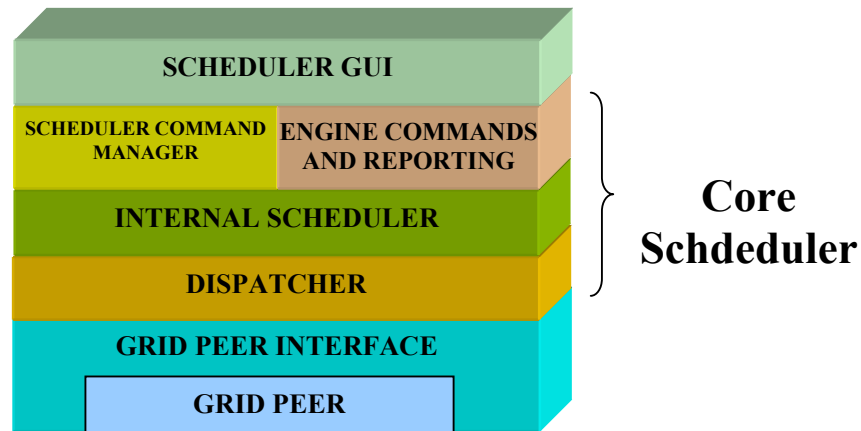
6.2 Description of Rule Scheduler (Work Done)

The Rule Scheduler is a multithread application involved in the rules executors management. It plays the role of controller in the distributed environment.

The AXCP Rule scheduler architecture is reported below and was divided into three main components:

- **Scheduler Gui** – the graphic user interface
- **Core Scheduler** – it is the internal set of modules that performs the scheduling activity

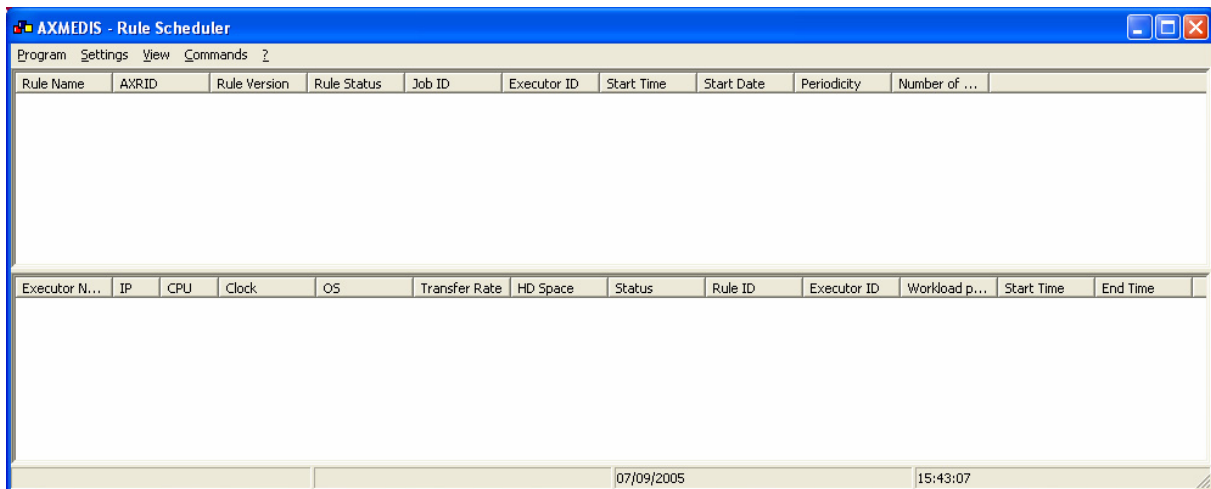
- **Grid Interface** – is the communication support based on P2P technology for communicating and monitoring the work performed by each Rule Executor (Peer)



6.2.1 Rule Scheduler Gui

The Scheduler GUI is the main window that allows interacting with the Scheduler. Referring to the snapshot, it is 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 provides the access to the following set of implemented functions:

1. Program

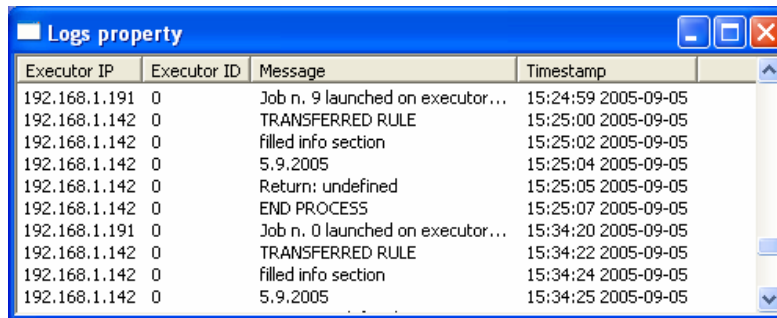
- a. *Add rule* – Load a rule in the scheduler
- b. *Launch scheduler* - Start the scheduler activity.
- c. *Stop scheduler* - Stop the scheduler activity.
- d. *Restore* - Backup Copy of the last jobs list.
- e. *Minimize* - It reduces at icon on the taskbar.
- f. *Exit* - Close the application.
- g. *Start Grid Peer functions* – It starts the grid peer network communication support

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.

Logs Dialog

This dialog allows viewing the logs of scheduler activity.



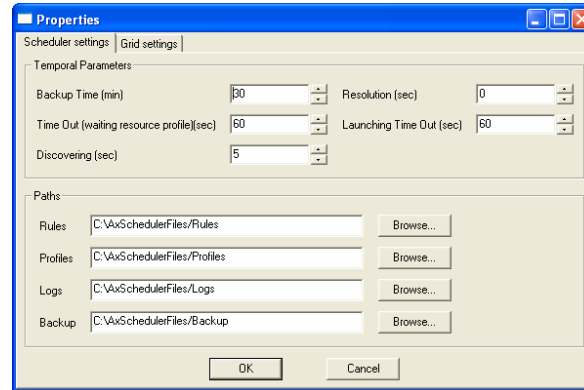
Executor IP	Executor ID	Message	Timestamp
192.168.1.191	0	Job n. 9 launched on executor...	15:24:59 2005-09-05
192.168.1.142	0	TRANSFERRED RULE	15:25:00 2005-09-05
192.168.1.142	0	filled info section	15:25:02 2005-09-05
192.168.1.142	0	5.9.2005	15:25:04 2005-09-05
192.168.1.142	0	Return: undefined	15:25:05 2005-09-05
192.168.1.142	0	END PROCESS	15:25:07 2005-09-05
192.168.1.191	0	Job n. 0 launched on executor...	15:34:20 2005-09-05
192.168.1.142	0	TRANSFERRED RULE	15:34:22 2005-09-05
192.168.1.142	0	filled info section	15:34:24 2005-09-05
192.168.1.142	0	5.9.2005	15:34:25 2005-09-05

Properties Dialog

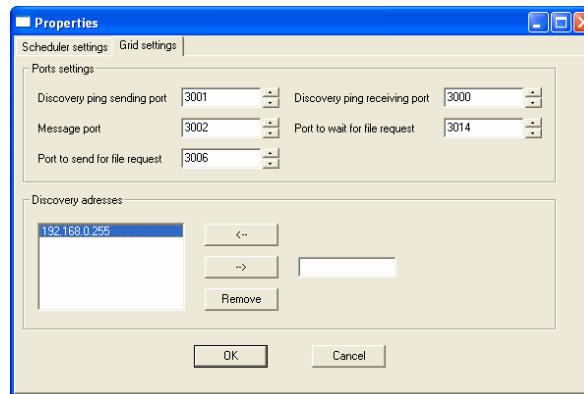
It is a tabbed dialog that allows editing settings parameters regarding the scheduler activity (*Scheduler settings*) and the GRID support (*GRID settings*).

Scheduler settings – It consists of a set of configuration parameters contains settings about:

- **Backup Time** - Backup interval for logging the set of submitted rule and tracing operations. It is expressed in minutes.
- **Time Out** - Time out on client activity. It is expressed in seconds.
- **Time Resolution** - Time Resolution of the scheduler. It is 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 saves the current rules list.



Grid settings – It provides a set of settings to setup the communication support. It allows to define the number of ports to use when receiving file, messages, sending files, responding to the discovering request. It allows also to define IPs of LANs to use when the scheduler performs the discovering of peers.



6.2.2 Core Scheduler

The core scheduler is the manager of active rules. It is a multithread module. It detects, fires, launches, manages and monitors the execution of a rule. During its activity, the core scheduler:

1. preserves the scheduled work from interruption of service (crash of the application) giving the possibility to restore the last status of activity
2. manages and update the list of rules to be scheduled and their status
3. manages and update the list of available rule executors
4. notifies 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)

The following list reports the actually implemented functionalities:

- Selecting 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
- Modifying and setting the time resolution for the control of rules execution
- Adding 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
- Removing a rule from the list of jobs
- Running a rule on demand
- Rescheduling a rule (by overriding the schedule information)
- Overriding rule arguments (by replacing the current arguments)
- Checking expiration conditions of a rule
- Providing the list of jobs/rules
- Updating firing conditions of a periodic rule
- Browsing the list of jobs/rules
- Modifying the status of rules
- Removing an executor from the list of executors
- Providing the list of executors
- Browsing the list of executors
- Saving periodically on disk a backup copy of the list of jobs
- Restoring the last status by loading the backup copy of the list of jobs
- Tracing all activity by means logs

6.2.3 Grid Interface and architecture

See section 7.2 of this DE

7 AXMEDIS Rule Executor (DSI)

The RuleExecutor is a console application embedding the GRID interface for network communication (peer) and the Spidermonkey JavaScript Engine extended with AxJSClasses that wrap AXMEDIS components and different Data Model. The following sections report the current status of the prototype under development and the list of functionalities that are available in the current version.

7.1 Technical Details

reference to the AXFW location of the demonstrator	https://cvs.axmedis.org/repos/Framework/source/ruleexecutor/
List of libraries used	.; wxWidgets library ver. 2.4.2. (wxBase.lib), Java Script Engine (Spidermonkey ver. 1.5 – JS32.dll).
References to other major components needed	Rule Data Model, Rule Scheduler, Grid Interface
Problems not solved	<ul style="list-style-type: none"> • •
Configuration and execution context	
Programming language	C++ MSVC7

7.2 Description of Rule Executor (Work Done)

The Rule executor consists of the following main components:

- **Grid Peer Interface** – the communication support with the AXCP rule scheduler
- **Rule Executor Manager** – the command interface to control the javascript engine
- **Script Executor/JSEngine** – The SpiderMonkey Javascript Engine (called JS Engine) extended with a set of JSClasses for AXMEDIS contents processing.

At now, the rule executor provides the following functionalities:

- Discovering: it responds when the Rule scheduler performs the discovering of peers on the network
- Sending profile: it generates the capabilities profile of peer machine and sends it to the rule scheduler
- Rule Receiving: it performs the download of a AXCP Rule from the rule scheduler
- Sending Messages: it sends different types of messages such as notification and errors,
- Commands parser for remote control messages

7.2.1 Rule ExecutorManager

The **Rule Executor Manager** is the interface between the JSEngine and the Grid Peer Interface. The actual prototype of the Rule Executor Manager provides functionalities for:

- Routing messages produced by internal components to the scheduler via the Grid Peer Interface
- Receiving control messages and commands from the Scheduler via the Grid Peer Interface
- Parsing and executing commands coming from the scheduler such as:
 - Launching the execution of rule
 - Killing the execution of rule
 - Pausing the execution of rule
 - Resuming the execution of rule
 - Requesting profile
 - Requesting status
- File Transferring to:
 - send the profile of the Rule Executor
 - receive the rule to be executed

- Sending messages and notification to the Scheduler via the Grid Peer Interface
- Creating the profile of the executor according to the XML schema
- Managing the status of the Executor
- Starting the execution of the rule calling the internal rule launcher

The role of the Rule Launcher is to start the execution of the script. The main steps that the Launcher performs, 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)
- Instantiating the Script Executor/JSEngine
- Calling the *Script Executor* for executing the script

7.2.2 Script Executor/JSEngine

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.
- Generating output

In the initialization phase, the JS Engine is initialised according to the guideline of Spidermonkey for defining the context, the runtime, the global variables, the definition of parameters of the rule.

The Engine Output for the JSEngine allows defining specialised display for output communication: GUI, network communication, etc.. It provides the way to:

- print internal error generated by the engine when compiling a script and at runtime during the execution/debugging
- print a message in the output interface
- visualize the line where the engine is stopped when a trap occurs
- clear the output display
- display the function name and the location (line of code) in the stack calls list
- reset the Stack call display
- reset the Local variables display
- print a variable in the Local variables display:
- print the properties of an object in the Local variables display

The Script Executor was developed to be used also in the AXCP Rule Editor in order to perform the debugging of rule.

Debugging of rule: This modality was developed using the debug function API provided by JSDebug API of SpiderMonkey and to be controlled by the AXCP Rule Editor. The Spidermonkey APIs permit to :

- put traps in the code corresponding to breakpoints (interrupting the execution)
- reporting of local variables
- manage the stack of functions
- realise the interface for debug functions and controls for the AXCP Rule Editor.

Missing features that will be implemented:

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 check mode, the rule will be executed in order to:

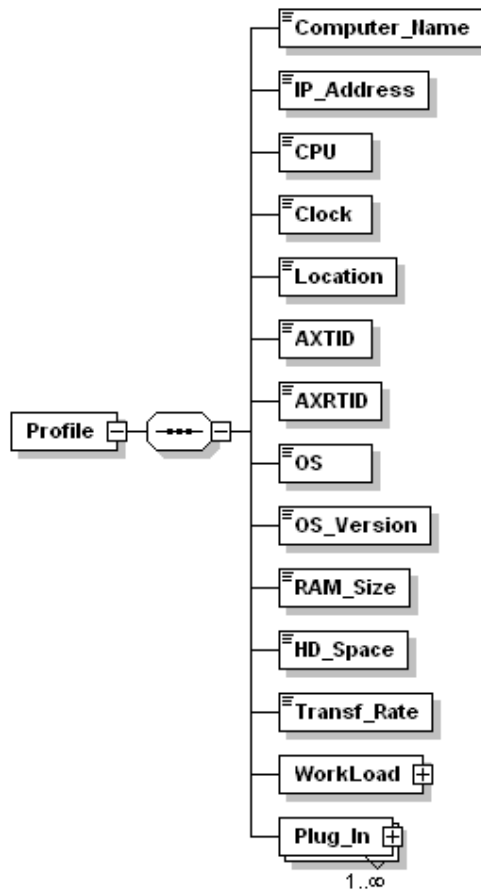
- verify the correctness of the rule before to send it to the AXCP Rule Engine
- estimate some parameters related to the complexity of the rule. Such parameters will be identified and defined during the project life. They will be used to define a complete profile of the rule in terms of required computational resources.

7.2.3 Profile of the Rule Executor (Peer)

The definition of profile for a rule executor peer was formalised by means the XML Schema as reported in this section. The profile describes the capability of the machine where the rule executor peer is running in terms of hardware, software and axmedis plugin configuration.

element Profile

diagram



children

[Computer Name](#) [IP_Address](#) [CPU](#) [Clock](#) [Location](#) [AXTID](#) [AXRTID](#) [OS](#) [OS_Version](#) [RAM_Size](#) [HD_Space](#) [Transf_Rate](#) [WorkLoad](#) [Plug_In](#)

source

```

<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>

```

element Profile/Computer_Name

diagram



type **xs:string**

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

description **Name of the computer hosting the rule executor**

element Profile/IP_Address

diagram



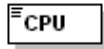
type **xs:anyURI**

source `<xs:element name="IP_Address" type="xs:anyURI"/>`

description **IP address**

element Profile/CPU

diagram



type **xs:string**

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

description **The type/family of CPU**

element Profile/Clock

diagram



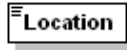
type **xs:float**

source `<xs:element name="Clock" type="xs:float"/>`

description [The clock of CPU](#)

element Profile/Location

diagram



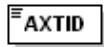
type **xs:string**

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

description [Where the computer is located](#)

element Profile/AXTID

diagram



type **xs:string**

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

description [The ID of a specific instance related to the AXTID](#)

element Profile/AXRTID

diagram



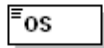
type **xs:string**

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

description [The AXMEDIS Registered Tool Id associated with the Rule Executor application](#)

element Profile/OS

diagram



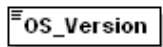
type **xs:string**

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

description [The Operating System](#)

element Profile/OS_Version

diagram



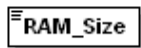
type **xs:string**

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

description [The OS version](#)

element Profile/RAM_Size

diagram



type **xs:string**

source `<xs:element name="RAM_Size" type="xs:string"/>`
 description [The amount of primary memory \(RAM\)](#)

element Profile/HD_Space

diagram 

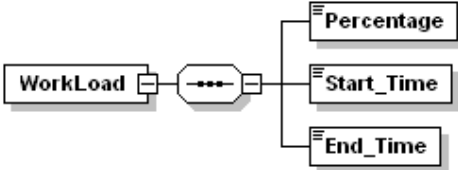
type **xs:string**
 source `<xs:element name="HD_Space" type="xs:string"/>`
 description [The amount of disk space available](#)

element Profile/Transf_Rate

diagram 

type **xs:unsignedInt**
 source `<xs:element name="Transf_Rate" type="xs:unsignedInt"/>`
 description [The network capability for transferring a file from the AXMEDIS Database to the rule executor machine.](#)

element Profile/WorkLoad

diagram 

children [Percentage Start_Time End_Time](#)
 source

```
<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>
```


 description [The percentage of availability during the time period](#)

element Profile/WorkLoad/Percentage

diagram 

type **xs:float**
 source `<xs:element name="Percentage" type="xs:float"/>`
 description [Availability to work expressed in percentage](#)

element Profile/WorkLoad/Start_Time

diagram 

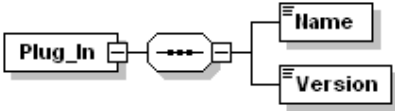
type **xs:time**
 source `<xs:element name="Start_Time" type="xs:time"/>`
 description [Start time of declared workload percentage](#)

element Profile/WorkLoad/End_Time

diagram A diagram showing a rectangular box labeled "End_Time" with a small icon of three horizontal lines in the top-left corner.

type **xs:time**
 source `<xs:element name="End_Time" type="xs:time"/>`
 description [End time of declared workload percentage](#)

element Profile/Plug_In

diagram A diagram showing a sequence of elements. It starts with a rectangular box labeled "Plug_In" with a small icon in the top-left corner. This is followed by a circle containing three dots, indicating a sequence. This is then followed by a branching structure with two rectangular boxes: "Name" and "Version", both with small icons in their top-left corners.

children [Name](#) [Version](#)
 source `<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>`
 description [Axmedis Plugin description mounted in the system](#)

element Profile/Plug_In/Name

diagram A diagram showing a rectangular box labeled "Name" with a small icon of three horizontal lines in the top-left corner.

type **xs:time**
 source `<xs:element name="Name" type="xs:time"/>`
 description [Name of plugin](#)

element Profile/Plug_In/Version

diagram A diagram showing a rectangular box labeled "Version" with a small icon of three horizontal lines in the top-left corner.

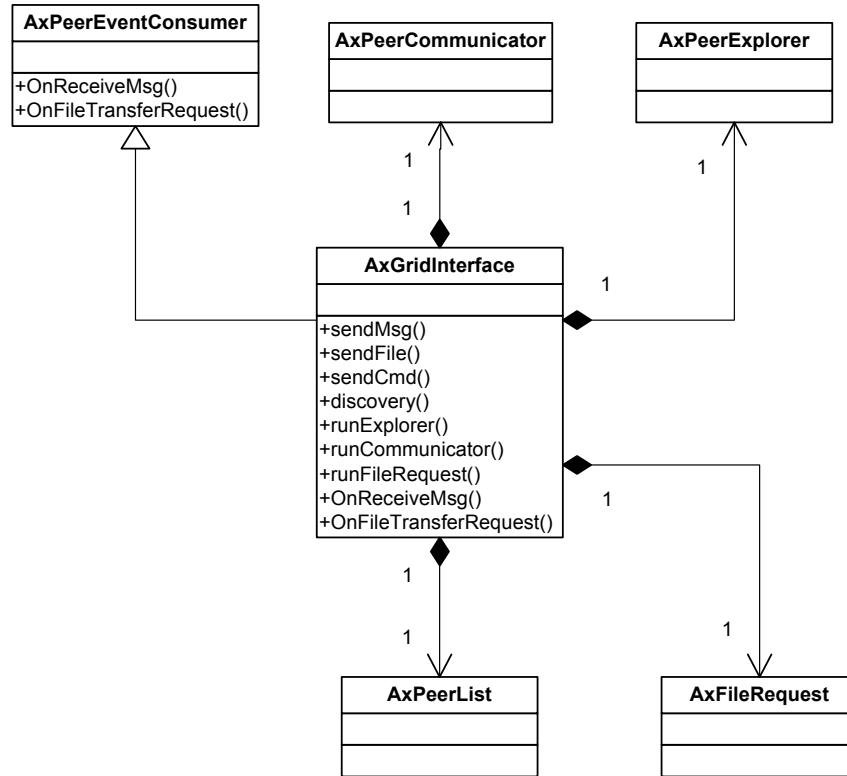
type **xs:string**
 source `<xs:element name="Version" type="xs:string"/>`
 description [Version of plugin](#)

7.3 AXMEDIS Grid architecture

Both the Rule Scheduler and the Rule Remote Executor are based on a **GRID infrastructure**. This infrastructure is realized by means a P2P technology based on TCP/UDP protocol.

Each node of the P2P network is called **GRID Peer** and provides the communication and the file transfer support to components of the distributed system. A GRID Peer provides four different and independent components:

- **Peer Explorer** – to provide functionalities for discovering the presence of other peers based on UDP broadcast messages.
- **Peer Communicator** – to provide communication functionalities and support for data exchanging with available peers.
- **Peer File Transfer** – to provide functionalities and support for file transfer among selected peers.
- **Peer Event Consumer** – to provide functionalities and support for handling events of communication, file transfer and discovering.



7.3.1 Structure of messages exchanged between Scheduler and Remote Executor

Messages exchanged between the Scheduler and the Remote Executor are 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 was to have a common message structure that allows covering all these types. In addition, to guarantee a fast delivery on the network, messages are light. To this end, they are 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 is 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

8 JSAXClasses Status (DSI plus ALL)

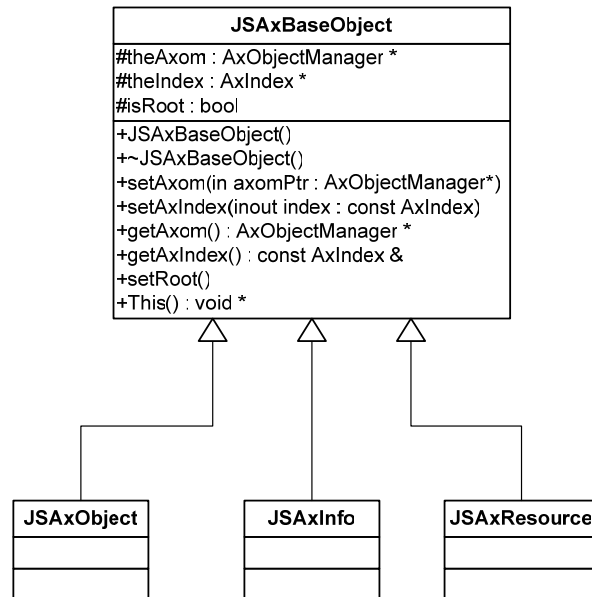
The composition and formatting process by means javascript required the extension of the object data type inside the javascript language. For this reason the following set of JSClasses was defined:

- JS_AXOM
- JS_FUNCTIONS FROM AXOM_CONTENT_PROCESSING
- JS_AXINFO
- JS_DUBLIN_CORE
- JS_SELECTION
- JS_RESOURCE
- JS_CRAWLERDB ACCESS (DSI WITH SUBCONTRACT) (see DE4.2.1)
- JS_FORMATTING (Style and Optimisation)
- JS_PROTECTION (FHGIGD, see DE4.5.1)
- JS_DRM (FHGIGD, see DE4.5.1)
- JS_PUBLISHER (CRS4, see DE4.4.1)
- JS_DOWNLOADER (CRS4, see DE4.4.1)
- JS_LOADER (INCLUDING METADATA MAPPER) (CRS4, see DE4.4.1)
- JS_FUNCTIONS
- JS_TRANSCODING (IRC, see DE4.7.1)
- JS_CLIENT_PROFILE (IRC, see DE4.7.1)

Some of them are at prototype level, the other will be implemented during the life of the project since most of these depend from AXMEDIS components and modules.

8.1 JS_AXOM: AXMEDIS Data Model JS wrapping (DSI)

In this section the current status of Axmedis Data Model wrapping for the Javascript and the current class diagram are reported.



UML Class Diagram of JSAXClasses for the AXMEDIS Data Model in Javascript

JSAxBaseObject Class – This class is the base class for *JSAxObject*, *JSAxInfo* and *JSAxResource* classes. It allows linking them to the current instance of the Axmedis Object Manager (*theAxom*) and accessing to its methods and commands. It allows also to associate a reference to *theIndex* (*AxIndex*) with the current instance of a child class when it is inserted into the Axmedis Object Data Model. The *isRoot* attribute is set at TRUE when the instance refers to the root object of the Axmedis Object Data Model.

JSAxObject Class – It is the mapping of an AXMEDIS OBJECT for JavaScript. According to the specification of the AXMEDIS OBJECT MANAGER and Axmedis Data Model, it provides and wraps methods to:

- Create an empty AXMEDIS object with own AXOM by instantiating a new Axmedis Object.
- Create and fill an AXMEDIS object with own AXOM by loading content from an URL.
- Add/Remove an Element/Object to the AXMEDIS object. The addition of an element returns the new object reference inside the Axmedis Object.
- Get all Elements/Objects. It returns a Javascript array of Element/Objects.
- Add Resource, it adds a digital resource (audio, video, text, etc...) to a specific Element/Object. It returns the new resource object reference inside the Axmedis Object.
- Remove a Resource (audio, video, text, etc...) by using the object Resource reference.
- Get Resources. It returns a Javascript array of Resource objects.
- Add an AXInfo, Dublin Core or generic metadata object. It returns the new metadata object reference inside the Axmedis Object.
- Get the AXInfo, the Dublin Core metadata
- Remove any metadata object.

JSAxInfo Class – It maps and allows managing the metadata of the AXINFO in the JavaScript. This class manages the access to individual elements and fields in AXINFO metadata, this class map all the functionalities provided by AxInfo class exposing setter and getter methods for accessing to data. It allows to manage:

- ObjectCreator information
- Owner information
- Distributor information
- Object Status information
- PromoOf information
- Workflow information
- Fingerprints information
- PAR information

JSAxResource Class – This class allows managing whatever digital resource type: Image, video, animation, etc... They derive from classes that model the single resource in AXMEDIS object model. This class wraps the AxResource class and provides functionalities exposing setter and getter methods to:

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

8.2 JS_Crawler: Crawler JS wrapping (Focuseek subcontract, DSI)

The JS_Crawler classes 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 was developed.

8.3 JS_Selection and query and data base access (DSI, EXITECH, CRS4)

This JS class allows managing the access to the AXEMDIS Database in order to resolve queries and retrieve the list of AXOID related to the set of AXMEDIS Objects matching the query. It exposes methods for:

- Managing the XML description of a selection
- Making query and then actualize the selection
- Obtaining the array of AXOID as result of the query

8.4 JS_FUNCTION: Integration with external resources and libraries (DSI)

It is a set of functions that provides utilities for different purpose such as:

- Print message on out device (console, gui, network, etc...)
- MimeType manager
- Basic network communication functions for managing http, https, ftp and other network connection and protocol.
- Disk and OS utilities

Other functions will be added during the life of the project according to emerging needs and requirements.

8.5 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 i.e. access to the DC Elements

- 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 field related to the specified element

Saving the new DC object

- JSUpdateDC()
 - Update the object to the originator

8.6 JS_FORMATTING (DSI)

The JS Formatting functionalities and data types are accessible in JavaScript through *JS_Template*, *JS_Style* and *JS_Format* classes. They will wrap the main classes of the C++ format engine (see paragraph 10.5 for more details).

8.6.1 JS_Template (DSI)

JS_Template provides all functionalities regarding templates accessible through the C++ Format_Manager:

- templates filtering;
- templates selection;
- templates application.

For instance:

- filterTemplates(): returns a list of templates suitable for the given resources and preferences;
- filterCriteria(): returns the ordered list of best templates for the given resources and preferences;
- getResourcesCount(): returns the number of resources that may take place into a given template;
- getResourcesTypes(): returns the types of resources that may take place into a given template.

8.6.2 JS_Style (DSI)

JS_Style provides all functionalities regarding style-sheets exposed by the C++ Format_Manager:

- style-sheets selection;
- style-sheets application.

For instance:

- filterStyles(): returns the ordered list of best style-sheets for the given template and preferences;
- getParams(): returns the list of parameters defined in a given style-sheet.

8.6.3 JS_Format (DSI)

JS_Format provides all functionalities exposed by the C++ Format_Manager to optimize style-sheets and produce the final SMIL description of the document. For instance:

- setResources(): creates the descriptors for resources contained within the given AXMEDIS object;
- setCriteria(): sets criteria used by the template selection logic;
- optimizeStyle(): gets sub-optimal values for parameters defined in the given style-sheet;
- createSMIL(): processes the given template and style-sheet to produce a SMIL document.

9 Content Composition, examples (DSI)

The Compositional Process aims to discover and specify the relationships (semantic, logical or spatial) between the resources returned from the Selection Process, and to create new digital items (AXMEDIS objects) that include the resources and their metadata and relationships.

Different types of relationships are possible:

- spatial: resources belong to the same area of the layout (e.g.: header or footer elements);
- logical: resources are linked together (e.g.: the image associated with a menu item and the resource related to the linked item);
- semantic: resources are connected by their “meaning” within the presentation (e.g.: a painting and its description).

Relationships could be defined through the hierarchical structure of the AXMEDIS object (which may include other AXMEDIS objects), or specified within metadata.

Relationships may be obtained from the author (who could define them explicitly) or from the Selection Process. In the latter case, the queries executed on the database could be used to get informations about the relationships between the resources that were returned.

In this section two examples of script for content processing are reported.

9.1 Composition process, example

In the following an example of composition script. A new object is created by adding an image resource and defining some metadata.

```
var obj = new AxmedisObject();

var info = new AxInfo();
resource = new AxResource();
var obj_metadata = obj.getMetadata();

// definition of a resource, the resourcePath variable defines where the resource is located
resource.mimetype="image/gif";
resource.ref=resourcePath;
resource.contentID="gif";
// mCon is the reference to the resource embedded in the Axmedis object "obj"
mCon = obj.addContent(resource);

// Adding an empty AxInfo inside the Axmedis Object
var mInfo = obj.addMetadata(info);

mInfo.isProtected = true;
mInfo.setOwnerID("abcdefghi", "Ivan");
mInfo.ownerNationality = "Italy";

//Setting the Date of Creation
var e = new Date();
mInfo.objectCreationDate = e.getDate()+"."+e.getMonth()+1+"."+e.getFullYear();

obj.save("c:\\axobj.xml");
```

9.2 Content Processing, example

In this portion of javascript code the conversion/adaptation of digital resources for mobile is reported.

```
// Loading Axmedis Object via URI
var b = new AxmedisObject(URI);
//Retrieving the array of Resources inside the object
var resource = b.getResource();

var obj = new AxmedisObject();

// For each resource the following lines perform the coversion for the mobile specified in the “profile”
// parameter and put the newResource in the empty Axmedis object “obj”
for( res in resource)
{
    newResource = AdaptForMobile(resource[res], mobileProfile)
    mCon = obj.addContent(newResource);
    // mCon is the reference to the resource embedded in the Axmedis object “obj”
}

[.....]
```

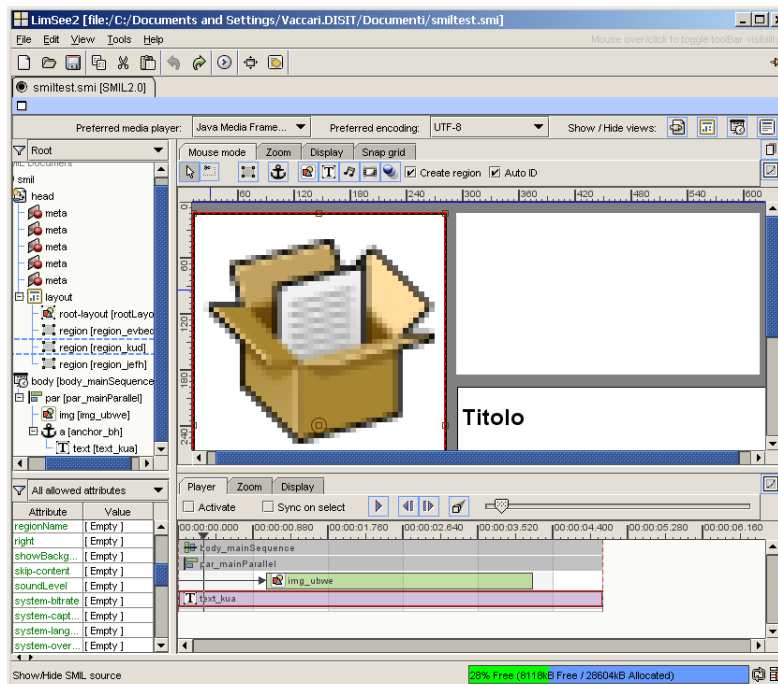
10 Content Composition and Formatting (DSI)

10.1 State of the art

As the number of the devices which can exploit the potentialities of multimedia is more and more increasing, a great variety of approaches for the description and the adaptation of the multimedia presentations has been proposed in the last years.

Some of them are based on the graph grammars: the graph structure is governed by rules that guide the automatic adaptation of the presentation to satisfy input and output constraints. Furthermore, the graph gives the author a visual representation of the components in the presentation and their logical positioning. Relational Grammars [Wei94] and others grammatical specifications has been proposed to describe the structure of multimedia presentations; the graphs that express the document structure may also be represented graphically and managed with visual tools [Zha05]. Reserved Graph Grammars (RGG) have been used for the transformation of XML documents that define multimedia structures [Zha02]. Between the main lacks of these approaches we remember the absence of the time dimension.

Several document models have been developed specifically to describe multimedia formats. The SGML-based HyTime language has been associated with DSSSL (Document Style Semantics and Specification Language, a transformation language for SGML) to produce adaptable presentations [Rut98]. ZYX [Bol01] is a sophisticated model created to provide a great support for reuse and adaptation: it supports both static and dynamic adaptation, and introduces the concept of “augmentation”, a semi-automatic selection of alternative contents which performs a “cross-media adaptation” [Bol99]. Madeus [Jou98] is a constraint-based authoring environment with temporal and spatial specifications: the timing of elements is calculated from the constraints specified among them, rather than as absolute position on the timeline. Madeus, unfortunately, doesn't offer any support for adaptation than constraints specification.



LimSee2

The models above have been used in the implementation of different framework for multimedia authoring. MM4U [Bol03], starting from the ZYX model, propose a generic framework for creating personalized

multimedia presentations, staying independent from the underlying data model and the adopted compositional algorithm. With the same objective of realizing a generic system for the automatic cross-platform adaptation, Cuypers [Van01] exploits the semantic description of the document offered by the Rethorical Structures. LimSee and LimSee2 [Roi03] are SMIL editors that rely on the Madeus model and offer a multi-view solution to render the structure of the SMIL document at different levels during the authoring process; the timeline is the main tool for synchronization in LimSee2, and supports both direct and constraint-based temporal scheduling.

Recently, SMIL [W3C05] – an XML-based language for declarative definition of multimedia presentations – is considered the most significant standard in this field. The strength of SMIL relies on two main features: a strong definition of the temporal relations between the elements of the multimedia presentation [Har99], and an automatic user-level adaptation of the content [Bul98].

SMIL has become very popular for the description of structured presentations, and many extensions for its model have been proposed: for instance, a finer granularity in the references to the media and a more generic expression of duration properties may enrich the language [Thu02].

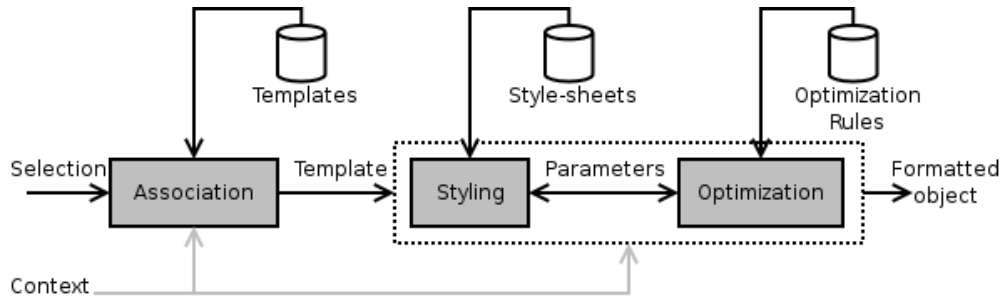
Moreover, the capabilities of adaptation offered by SMIL may be enhanced from the association with XML transformation languages: among them, the XSL has often been preferred. The use of XSL style-sheets may transfer a part of the adaptation work, normally requested to the SMIL clients, on the server-side [Pih03]; furthermore, a double transformation system has been proposed to adapt both the document source and its style-sheet [Lem03]. A refined adaptivity may be obtained introducing additional constraints that have to be solved after the style-sheet application [Bes01].



The MM4U web demonstrator

10.2 System outline

In our design, the formatting system is composed of two main logical blocks:



The main blocks of the formatting system

1. the **Association Block** receives as input the media obtained from the composition process (see Chapter 9) and some context informations from the author (or from an user profile). Context informations provide basic knowledge about the type of the presentation, its output format and the platform for which the presentation is targeted. The output of this block is the indication of a template that matches the input and describes the basic structure of the presentation.

To detect the “type” of the presentation and choose a template that matches it, it is very relevant for this block to have a precise context definition. At least, the author should specify:

- the platform for which the presentation is targeted, i.e.: PC, PDA, SmartMobile, Mobile, iTV, etc.;
- the output format, i.e.: MPEG2, MPEG4, AXMEDIS/MPEG21, SMIL, HTML, etc.;
- a general category for the presentation, i.e.: slide show, electronic book, kiosk, interactive music, interactive video, training tutorial, etc.

The system will perform a mapping of context, selected media, and compositional properties, and it will choose the template nearest to these criteria.

2. the **Styling/Optimization Block** select a style-sheet for the given template and adjusts its parameters to cope with the context, managing the adaptation/transcoding of the media involved. The output of this block is the formatted presentation.

The optimization process is a very critical and complex phase, that may involve many aspects. Layout determination and media encoding are part of the optimization: to fit the context, the layout specified in the template may be radically modified and the media may necessitate transcoding or resampling; moreover, media have to be transformed (scaled, rotated, etc.) to fit the adapted layout.

The choice of the optimization algorithm is very important: the problem of finding the best combination of a potentially large number of layout parameters is NP-complete, therefore the computational time to determine an exact solution is not reasonable. A more practical approach is to search approximate solutions, with methods such as Tabu Search or Genetic Algorithms. By the way, these algorithms are still very time consuming, and they take advantage from a parallel execution. For that reason the chosen algorithm has to be easily distributed on the AXMEDIS GRID.

The system is supposed to work in two modalities:

1. an interactive modality, which allows the author to choose or create templates and style-sheets and control the results of the adaptation. This modality is based on the C++ AXMEDIS Formatting Engine;
2. an automatic modality, that manages the whole process following a set of rules and the specified context; this modality uses the JavaScript modules that wrap the Formatting Engine.

10.3 Formatting process

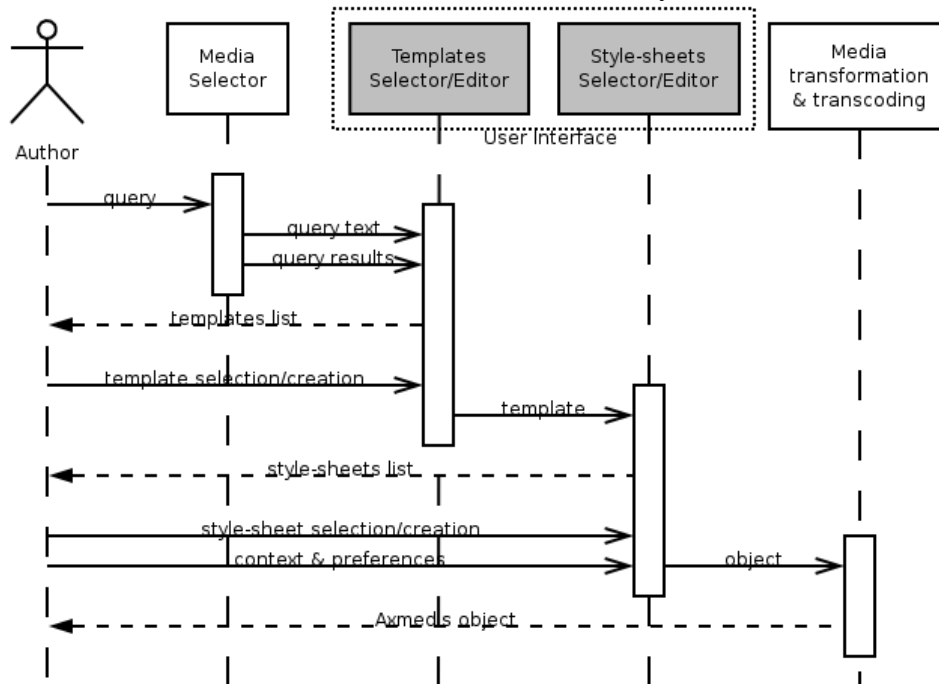
In the interactive modality (described in the pseudo-sequence diagram below), the author of the document (which is either the owner of the media, or the distributor of the document) starts the creation process with a query to select some media. The query string and its results are used by the Association Block to associate the document with an existing template. The Association Block can also interact with the user: the system

either requires a choice between different options (if several templates match the query results) or allows the creation of a new template.

To correctly associate the input set with a suitable template, the query results have to be preliminary submitted to the Compositional Process (see Chapter 9). During the Compositional Process, multimedia resources are ordered or logically grouped, according to their semantic or spatial relationships. Such relationships may also reflect the logic within the querying process. The Association Block uses these relationships to map the input set onto the space of templates:

1. resources are logically organized in a tree structure, which reflects compositional relationships between them;
2. tree properties are compared to those provided by templates stored in the Templates Database;
3. the closest template can be automatically selected, otherwise the list of closest templates will be submitted to the user.

After that, the style-sheet selection is performed by the Styling/Optimization Block: in the automatic modality one of the style-sheets created for the given template is selected following context indications; otherwise, a list of style-sheets is proposed to the author, who can choose one of them or create a new one. The final stage is the optimization: the parameters of the presentation are optimized following the context specification, and all media are transcoded and transformed to fit the layout.

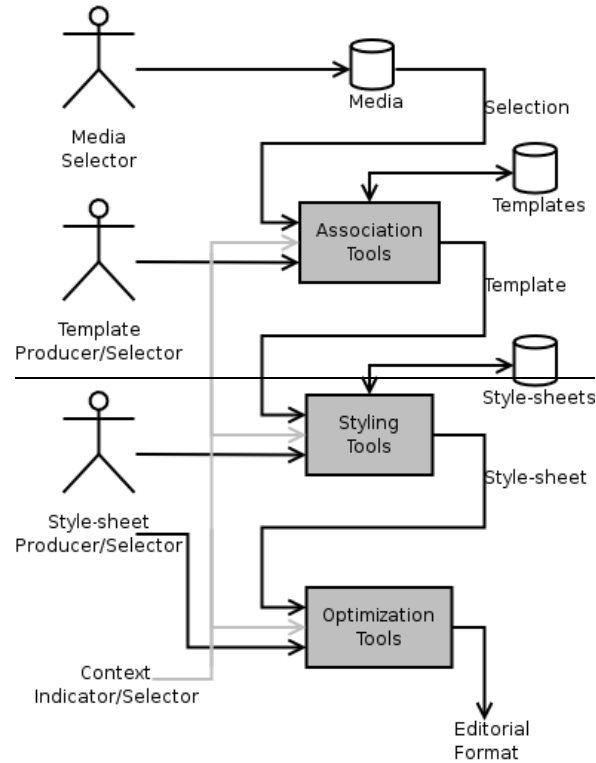


The formatting process

10.4 System architecture overview

The system architecture is depicted in the figure below.

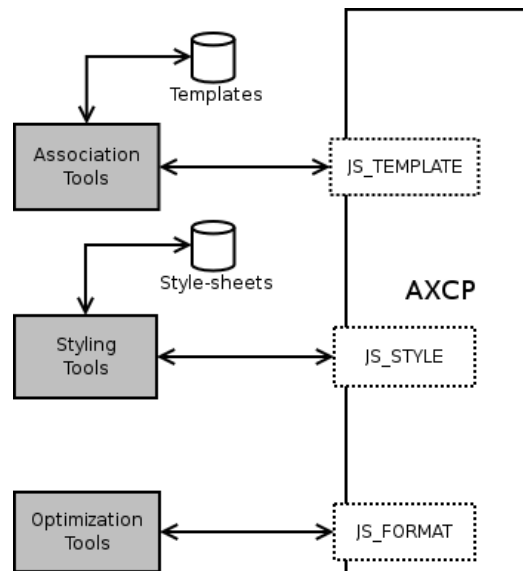
The system is provided with automatic tools and an user interface. The automatic tools have to find the better solution in terms of templates, style-sheets and optimization; the user interface allows the author either to preview and evaluate the result (using an integrated viewer), or create new templates and style-sheets (using the editors).



Architecture of the formatting system

The automatic tools are wrapped by the JavaScript modules of the AXMEDIS Content Processor. These are:

- JS_TEMPLATE, the interface to templates database and templates selection logic;
- JS_STYLE, the interface to style-sheets database and style-sheets selection logic;
- JS_FORMAT, the interface to adaptation logic and style-sheet processor.



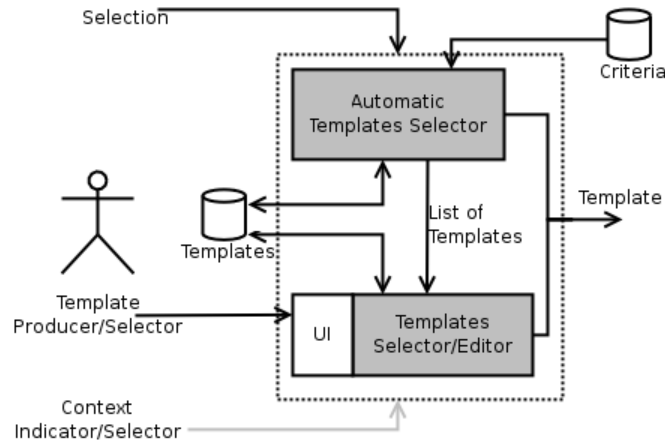
The JS Modules

The Association Block is made of two tools:

1. an Automatic Templates Selector, which uses some criteria for assign the input to an existing

template;

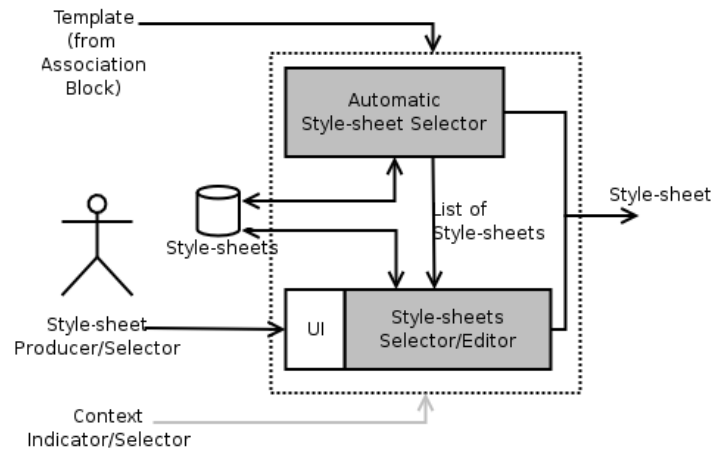
2. a Templates Editor, which allows the author to choose between the existing templates (eventually proposed by the automatic tool) or create a new one.



The Association Block

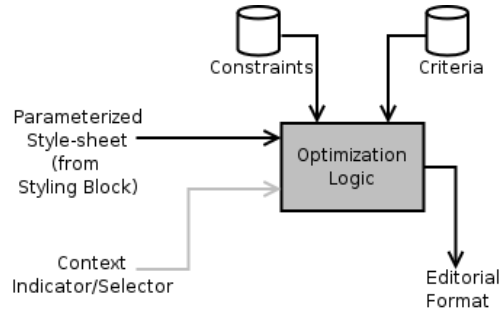
The Styling Block is made of two tools:

1. an Automatic Style-sheet Selector, which chooses the best style-sheet for given template and context;
2. a Style-sheet Editor, which allows the author to choose between the existing style-sheets (eventually proposed by the automatic tool) or create a new one.



The Styling Block

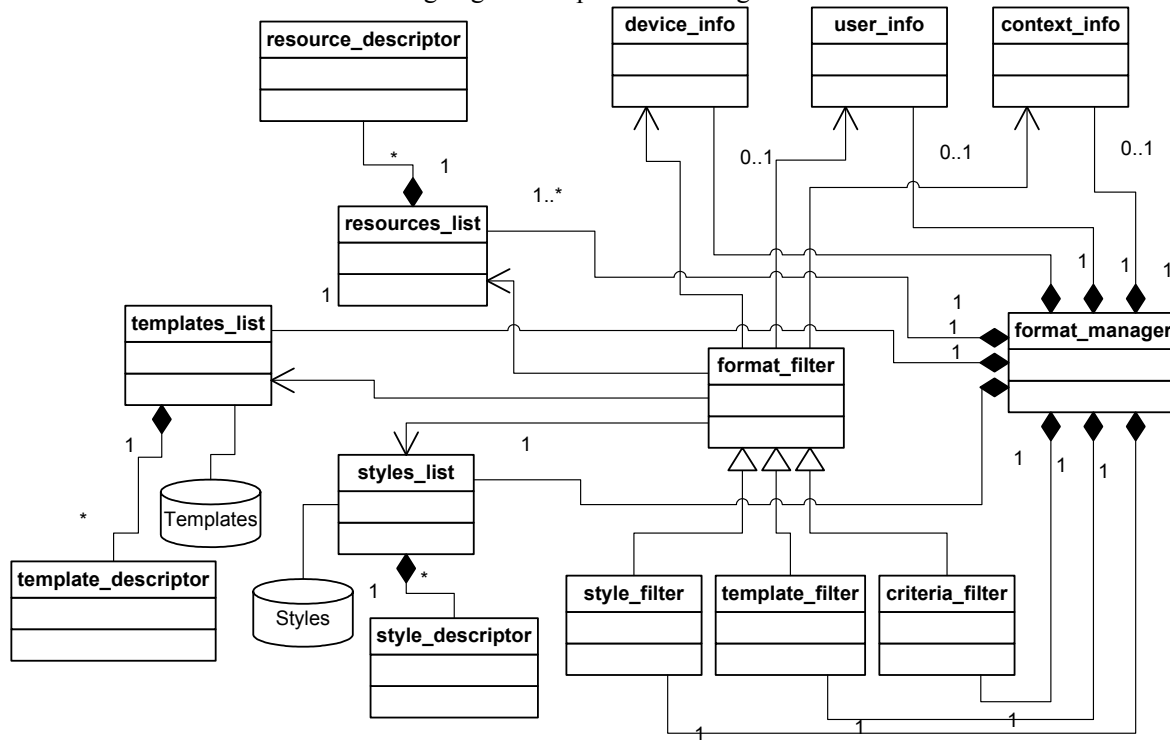
The Optimization Tool execute some algorithms to choose the best values for the parameters specified in the style-sheet. The algorithms have to respect some constraints for the allowed values and some criteria (functional or aesthetic) for choose between alternative solutions.



The Optimization Block

10.5 Formatting engine architecture

The architecture of the C++ formatting engine is depicted in the figure below:



Formatting engine architecture

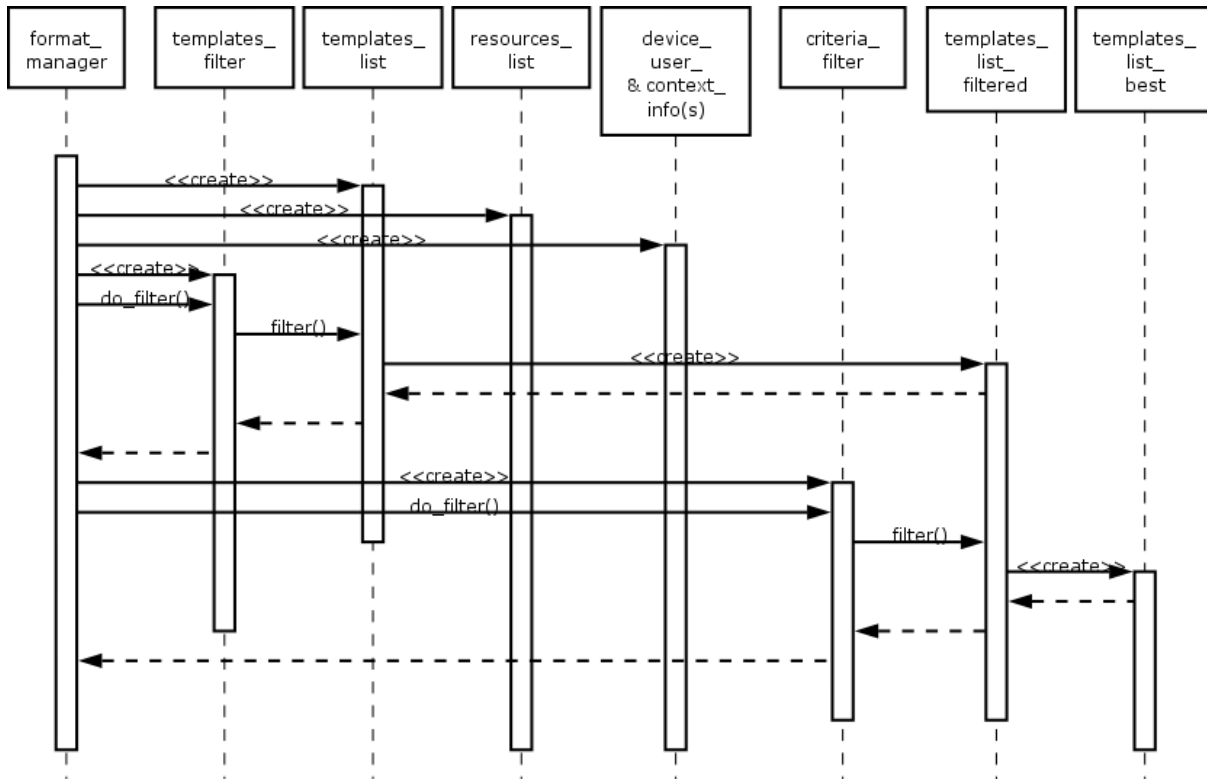
The *Format_Manager* drives the whole process: it collects information about resources (*Resources_List*), preferences (*User_Info*, *Device_Info* and *Context_Info*), templates (*Templates_List*) and style-sheets (*Style_List*), and makes them available to other components.

The *Templates_Filter* creates an unordered *Templates_List* of templates that suit to the given resources and preferences; the *Criteria_Filter* creates a second (ordered) *Templates_List* that contains better templates.

The *Styles_Filter* creates the *Styles_List* that contains better style-sheets for the selected template.

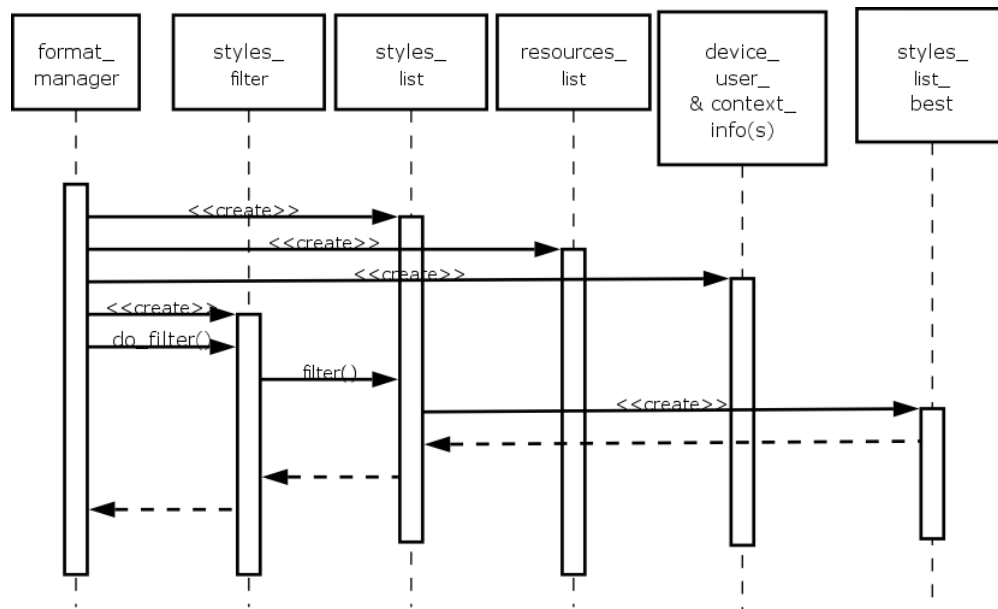
The *Format_Manager* also performs optimization and processes template and style-sheet to get the final SMIL description of the formatted document.

The sequence diagram for template selection is depicted below:



Template selection sequence

The sequence diagram for style-sheet selection follows the same logic.



Style-sheet selection sequence

10.6 Template language

The choice of template language is the first step in the system definition. Templates have to describe correctly multimedia layout and synchronization (including user interaction), define media adaptation (if needed), be adaptable at different user profiles, be suitable for use of style-sheets; thus the choice of the description language is a delicate issue.

Many advanced systems for multimedia authoring (e.g.: MM4U [Bol03], GRiNS [Bul98], LimSee2 [Roi03]) use their own internal model to describe the structure of the document; this way they get the maximum degree of expressivity and flexibility. On the other hand, standard languages (e.g.: SMIL, HyTime) may have some lacks or limitations, but they allow using off-the-shelf software and they don't require long definition and verification activities. Last, the chosen language has to provide an easy way to convert the description into required output formats (MPEG, HTML, SMIL, etc.).

Foremost standard languages for describing multimedia are:

- HyTime (ISO/IEC 10744) is an SGML-based standard that seeks to provide ways of addressing and linking to any kind of information, anywhere in time and space. Very hard to learn, it has been substantially replaced by SMIL;
- MPEG-4 (ISO/IEC 14496) provides a standardized representation of interactive multimedia content in terms of media objects, hierarchically organized within a scene. The XMT (Extensible MPEG-4 Textual) format (and specifically the XMT-O) provides interoperability between MPEG-4 and SMIL;
- MPEG-7 is a standard (ISO/IEC 15938) for describing the multimedia content data that supports some degree of interpretation of the information meaning. It includes a Description Definition Language (DDL) based on XML Schema language (XSD);
- SMIL (Synchronized Multimedia Integration Language) is an open XML-based language proposed by the World Wide Web Consortium [W3C05]. SMIL allows to create a well structured document, with the focus at final result and not at rendering mechanism. As stated by its name, SMIL is an "integration language": media files (images, audio, video, animations, text) are only referenced, but they stay independent from the SMIL file. SMIL has been designed to be used over Internet and may

be integrated with others W3C standards: XML, XSL, CSS, XHTML, SVG, etc.

SMIL project has been started by W3C in 1995; its first version (SMIL 1.0) has been released in 1998, the second (SMIL 2.0) in January 2005. Currently (August 2005) the last version is the "Candidate Recommendation" for SMIL 2.1.

SMIL 2.0 is composed by 45 modules divided in 10 groups:

- timing (19 modules);
- time manipulations (1);
- animation (2);
- content control (4);
- layout (4);
- linking (3);
- media objects (7);
- metainformation (1);
- structure (1);
- transitions (3).

Modules are designed to be reusable as parts of other XML vocabularies, so vendors or other standards initiatives may decide to implement only parts of SMIL. On the other hand, it is possible enhance SMIL with proprietary extensions: for instance, Apple QuickTime provides extensions to play automatically the presentation, to show a time slider, to allow full screen playback, to get informations about the bandwidth a media object needs in order to play back in real time.

SMIL has some interesting features that make it our best choice as description language for multimedia documents:

- human-readable format;
- "open" standard (in opposition to proprietary standards);
- designed for the web (in opposition to media-oriented MPEG);
- separation between spatial relationships (layout) and time relationships (synchronization);
- CSS support;
- independent multiple windows support (via the <topLayout> element);
- switching between different layout types and media formats depending on the client or the user preferences (via the <switch> element);
- synchronization with absolute time, relative time or asynchronous events;
- support of complex transitions between media;
- metadata support;
- extensibility (proprietary extensions allow adding of specific attributes to SMIL presentations);
- interaction with the user.

For the reasons above, we decided to use SMIL language for definition of structure and synchronization. Moreover, it is important to remark that a SMIL editor will be provided within the AXMEDIS framework: this is a plus for the adoption of SMIL in the formatting system.

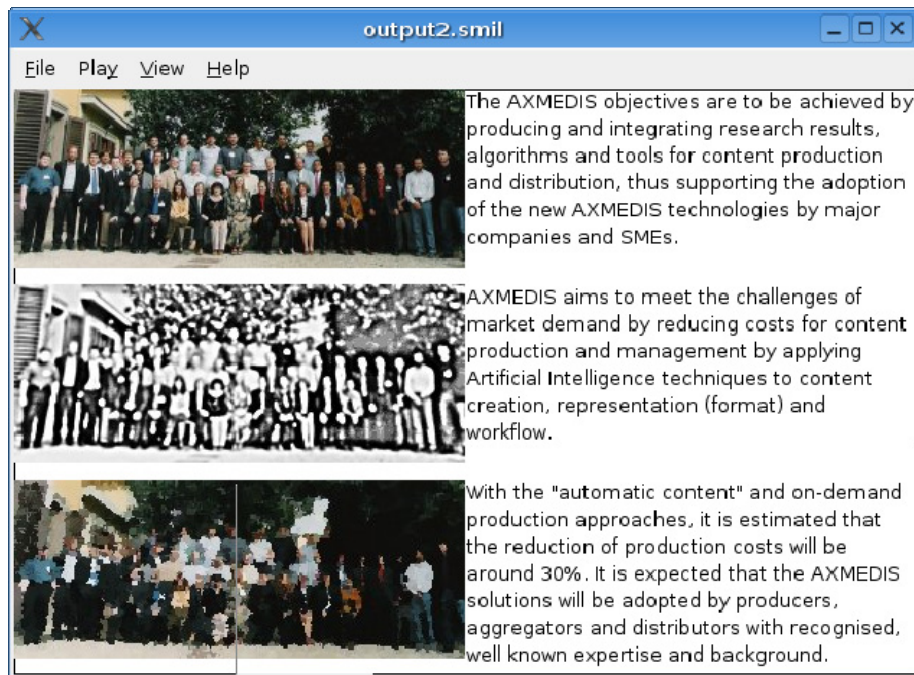
Templates will describe synchronization in detail, but layout properties may be very generic (i.e.: "auto" and "indefinite" values will be used often); spatial relationships will be defined with precision in the style-sheets. This way our template is "real SMIL" also if visual rendering may be inconsistent; more important, there's no formal difference between a template and a styled presentation: thus style-sheets may be applied indifferently to templates and to already styled presentations, enhancing possibilities of reuse.

To make more general the document description, we plan to define (with the “ax:” prefix for its namespace) some extensions for the SMIL language used in templates. The most important is the `<ax:repeat>` element, that aims to describe homogeneous groups of elements, without regard for their number: it is very useful during Composition and Association. It could be used in the following simple way:

```
<par>
  <ax:repeat xmlns:ax="http://www.axmedis.org/extensions">
    <ax:items element="text" type="text/plain" dur="2s">
      <ax:item src="axm1.txt" region="txt_region1"/>
      <ax:item src="axm2.txt" region="txt_region2"/>
      <ax:item src="axm3.txt" region="txt_region3"/>
    </ax:items>
    <ax:items element="img" type="image/jpeg" dur="indefinite">
      <ax:item src="gruppo1.jpg" region="img_region1"/>
      <ax:item src="gruppo2.jpg" region="img_region2"/>
      <ax:item src="gruppo3.jpg" region="img_region3"/>
    </ax:items>
  </ax:repeat>
</par>
```

The new elements could be processed by the generic part of style-sheets; the XML produced for the above example should be:

```
<par>
  <text type="text/plain" src="media/axm1.txt" region="txt_region1" dur="2s"/>
  
  <text type="text/plain" src="axm2.txt" region="txt_region2" dur="2s"/>
  
  <text type="text/plain" src="axm3.txt" region="txt_region3" dur="2s"/>
  
</par>
```



An use case for the `<ax:repeat>` extension

Ideally, every graphical SMIL editor could be used as Templates Editor. Anyway, it must be pointed that, in the SMIL language, media may be synchronized using an absolute timing (the presentation time), a relative

ordering, or a set of constraints based on the temporal properties of other media; therefore, to manage such temporal and behavioral properties, the editor has to be very refined. It would also be good if the editor could check the syntax against an arbitrary DTD, to allow modifications and extensions to SMIL.

10.7 Style-sheet language

The main goal of style-sheets systems is separation between document content and its visual representation. The most widely diffused style-sheets standards are CSS and XSL, both defined by the W3Consortium. Cascading Style Sheets (CSS) is a simple mechanism for adding style to XML documents. A CSS engine visits each node of the XML document node hierarchy and tries to match this node to a CSS rule. Therefore, CSS cannot be used to modify the XML document structure and the rendered document is strictly dependent on the XML document structure. Nevertheless CSS is human readable, very easy to learn, efficient.

XSL (eXtensible Stylesheet Language) is the more advanced style-sheet language for XML. It is made of three parts:

1. XSL transformations (XSLT), used to transform XML documents;
2. XML Path Language (XPath), used by XSLT to refer to parts of an XML document;
3. XSL Formatting Objects (XSL FO), used to describe formatting semantics.

The main advantage of XSL over CSS is its capability of transform the XML structure. Price to pay is a more difficult syntax and a less immediate readability. Furthermore, for its tree-shaped nature and its dependence from the XML source, creating visual tools for XSL editing it's not a simple task: market offers several IDE for XSL editing, but rarely they offer sufficient flexibility, since they are targeted to a particular type of output (generally HTML).

Anyway, XSL is the more advanced, powerful and thus preferred language for styling XML (and so SMIL); and CSS, eventually, may still be used in association with it.

In our system, the XSL processor receives as inputs an XSLT style-sheet and a template written in SMIL; its output is a styled SMIL document. XSLT style-sheets may radically change the final aspect of the presentation, and a style-sheet may be applied to an already styled document with no problem.

The XSLT style-sheet will be divided in two parts: a generic redefinition of the SMIL format structure (valid for every style-sheet) and a style-sheet specific layout description. This way, style-sheet creation will be simpler.

The Style-sheets Editor will be a visual XSLT editor: it operates on the entities included in the template, managing their spatial properties and their temporal parameters.

10.8 Document optimization

A single multimedia document (described by its template) has to be formatted in different ways and its components have to be adapted to different users that access the document over different channels.

Complexity degree introduced by the number of different channels, users platforms and users preferences, may be very high. Therefore, the automatic formatting system has to offer an easy but powerful way to manage this complexity.

We plan to provide these functionalities at three levels:

1. at template selection level: only templates that match with context (and input) have to be selected. For this reason each template should include a list of output options and context informations. Context informations will be stored in the Templates Database. For output options, could be used the SMIL <switch> statement: a single SMIL template may adapt its structure to cope with several targets. In this way we can change layout and synchronization and even disable some features to get the best results on each client. The <switch> statement may also drive the input adaptation: depending on the targeted channel, the template chooses the right format for the input media. The style-sheet will process only the appropriate <switch> cases and it will remove the whole <switch> statement.

The <switch> statement, defined in the SMIL 2.0 Basic Content Control Module, expresses that a set of document parts are alternatives, and the first one fulfilling certain conditions should be chosen. One or

more test attributes may appear on media object references or timing structure elements; the system test attributes are:

- systemBitrate: allows a choice based on the user connection to the network;
 - systemCaptions: specifies a redundant text equivalent of the audio portion of the presentation;
 - systemLanguage: allows a choice based on the languages indicated by user preferences;
 - systemRequired: provides an extension mechanism for new elements or attributes;
 - systemScreenDepth: specifies the depth of the screen color palette in bits;
 - systemScreenSize: specifies if the playback engine is capable of displaying a presentation of the given size;
 - systemAudioDesc: specifies whether or not closed audio descriptions should be rendered;
 - systemCPU: specifies the CPU on which a user agent may be running;
 - systemComponent: identifies the components of the playback system (e.g.: user agent component/feature, number of audio channels, codec, HW MPEG decoder, etc.);
 - systemOperatingSystem: specifies the operating system on which a user agent may be running;
 - systemOverdubOrSubtitle: specifies whether subtitles or overdub is rendered;
2. at style-sheet selection level: several style-sheets may be provided for each template, to get the maximum flexibility. In the Style-sheets Database every style-sheet has a descriptor that specifies output and context informations. Moreover, every style-sheet may be parameterized, and the value of its parameters will be chosen according an optimization algorithm. Finally, templates may be applied in cascade: the application of a style-sheet to a previously adapted document is very useful in terms of adaptability and reuse;
 3. at user level (eventually): if the output is in SMIL format, style-sheets may provide an additional <switch> statement that the SMIL player will process at runtime to fit specific user preferences.

The system we have designed allows to manage these different levels of adaptation:

- SMIL templates permit to specify, within the <switch> statement, alternatives for layout and media to enclose;
- the generic part of the XSL style-sheet process the <switch> statements, selecting only the appropriate <switch> cases;
- the specific part of the XSL style-sheet allows the fine-grained specification of parameters, whose values will be chosen by the optimization algorithm. Such parameters can also define values of attributes that fulfill the conditions of the <switch> statement: in doing so, editing the generic part of the style-sheet will not be necessary at all.

10.9 Optimization algorithm

The optimization of the style-sheet is the process that aims to get the best looking layout for the document, according certain criteria and the range for values of parameters. The number of parameters is potentially very high, and their values may have a large range: this makes the optimization a critical step in the automated creation of the document.

The optimization problem may be defined as follows. Many parameters allow to tune the style-sheet, and each of them may assume a given range of values; every combination of parameters generates a possible layout that has to be evaluated according some general criteria, based on aesthetic or functional considerations. For example: a cost functional could evaluate results in terms of proportions, visibility and use of the screen offered by the considered layout, adopting a different weight for each criteria.

The optimization algorithm should provide the optimal solution, that is the best layout for the document. Actually, the complexity of the problem is very high (it may be considered almost equivalent to the “knapsack problem”, which is NP-complete), thus should be used algorithms that search sub-optimal solutions, as the Tabu Search.

Tabu Search (TS) is an optimization iterative algorithm that explores some possible solutions of the problem making repeated moves from a solution to another belonging to its “neighborhood”. The neighborhood of the current solution is individuated by the group of solutions that can be reached with a single variation of the parameters. The procedure stops when a fixed number of iterations have been performed.

The main characteristic of the TS is the concept of memory: a list of moves, called Tabu list, that cannot be executed at the current iteration to move towards the next solution. The Tabu list includes moves performed recently or that were performed frequently in the last iterations. The memory structure impedes to the algorithm to remain trapped in local minimums, leading it to the space of solutions still unexplored, permitting the performing of moves characterized by values of the cost functional major than the current solution. The Tabu state of a move can be changed to an allowed state if the value of the cost functional associated with it is the minimum between all determined until the current solution (standard criteria).

The Tabu Search, in a comparison with other techniques, is robust and rapid. Moreover, this methodology is relatively simple to implement and its execution can be parallelized with the AXMEDIS GRID.

10.10 Format example

A simple SMIL template looks like this:

```
<smil>
<head>
  <!-- metadata -->
  <meta name="title" content="Axmedis Formatting Test" />
  <meta name="author" content="P. V." />
  <meta name="copyright" content="nocopyright" />
  <!-- regions -->
  <layout type="text/smil-basic-layout">
    <root-layout
      width="auto" height="auto"
      title="Smil test"
    />
    <region id="text_region"
      width="auto" height="auto"
      top="auto" left="auto"
    />
    <region id="axm_img_region"
      width="auto" height="auto"
      top="auto" left="auto"
    />
    <region id="eu_img_region"
      width="auto" height="auto"
      top="auto" left="auto"
    />
  </layout>
</head>
<body>
  <!-- layout -->
  <par>
    <switch>
      <text id="text1_it"
        systemLanguage="it"
        src="media/axm_it.txt"
        type="text/plain"
        region="text_region"
      />
      <text id="text1_en"
        systemLanguage="en"
        src="media/axm_en.txt"
        type="text/plain"
        region="text_region"
      />
    </switch>
  </par>
</body>
</smil>
```



```

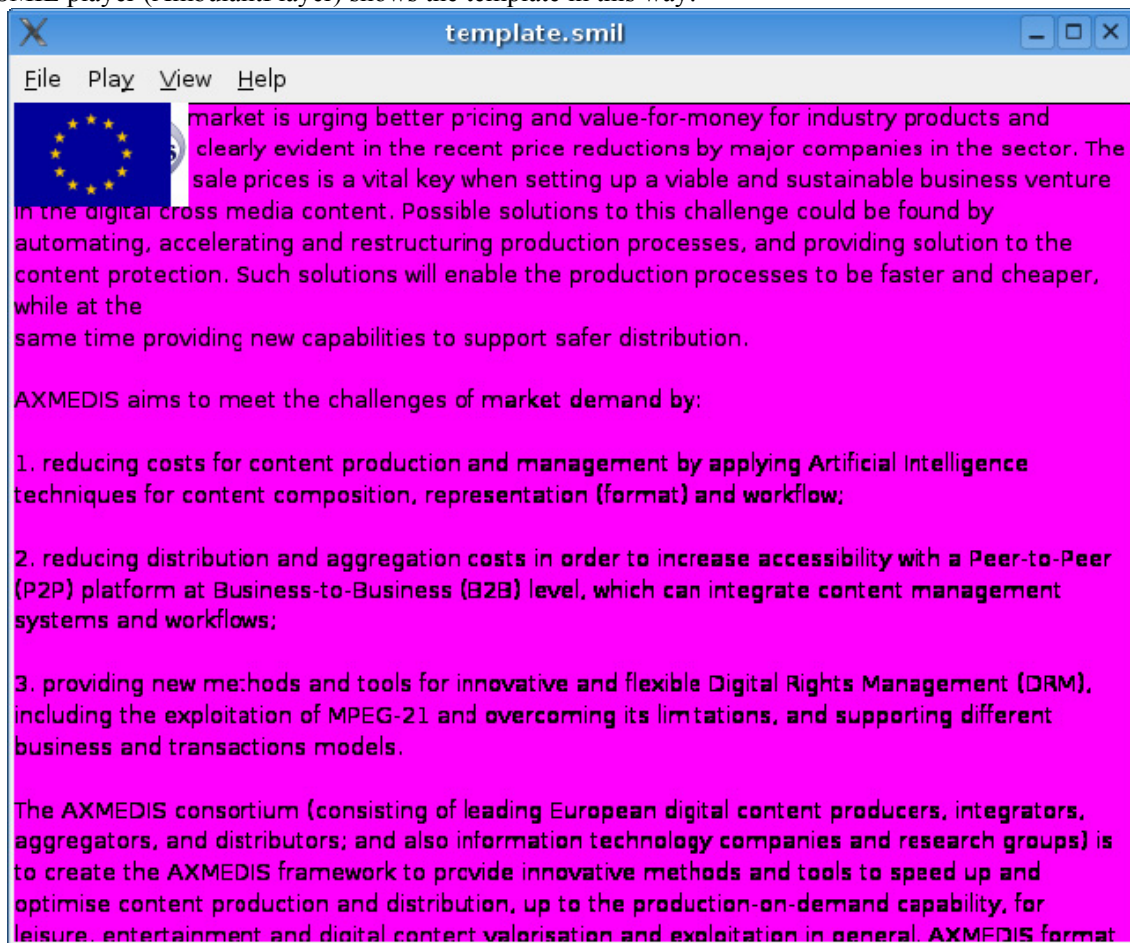


</par>
</body>
</smil>

```

The layout section describes all the regions that will be displayed in the “root-layout” window; the body section associates a media file to each region. The <switch> element in the body section is the most interesting thing in this simple template: two different text files are associated to the same region and the system is supposed to choose the right one on the basis of the systemLanguage attribute.

A SMIL player (AmbulantPlayer) shows the template in this way:



A SMIL template

The document is correctly opened by the player, since it is correct SMIL, but its structure it's not consistent and its aspect it's really poor (two images are overlapped and both overlap the text).

A basic style-sheet is the following:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

```

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <!-- parameters -->
  <xsl:param name="screenWidth"/>
  <xsl:param name="screenHeight"/>
  <xsl:param name="backGround"/>
  <xsl:param name="foreGround"/>
  <xsl:param name="menuWidth"/>
  <xsl:param name="textWidth"/>

  <!-- SMIL structure -->
  <xsl:include href="smilstruct.xml" />

  <!-- attributes for root-layout -->
  <xsl:template match="root-layout">
    <root-layout
      width="{ $screenWidth }" height="{ $screenHeight }"
      backgroundColor="{ $foreGround }"
      title="Smil+XSL test"
    />
  </xsl:template>

  <!-- attributes for regions -->
  <xsl:template match="region">
    <xsl:choose>
      <xsl:when test="@id='axm_img_region'">
        <region
          id="{ @id }"
          backgroundColor="{ $backGround }"
          fit="hidden"
          regPoint="center"
          regAlign="center"
          width="110" height="70"
          top="{ $screenHeight * 0.05 }"
          left="0"
          z-index="1"
        />
      </xsl:when>
      <xsl:when test="@id='eu_img_region'">
        <region
          id="{ @id }"
          backgroundColor="{ $backGround }"
          fit="hidden"
          regPoint="center"
          regAlign="center"
          width="100" height="70"
          bottom="{ $screenHeight * 0.05 }"
          left="0"
          z-index="1"
        />
      </xsl:when>
      <xsl:when test="@id='text_region'">
        <region
          id="{ @id }"
          backgroundColor="{ $backGround }"
          fit="scroll"
          width="{ $textWidth }"
          left="{ $menuWidth }"
          height="{ $screenHeight * 0.9 }"
          top="{ $screenHeight * 0.05 }"
        />
      </xsl:when>
    </xsl:choose>
  </xsl:template>

```

```

        z-index="2"
      />
    </xsl:when>
    <xsl:otherwise>
      <!-- include the region as is -->
      <xsl:copy-of select="."/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

```

```
</xsl:stylesheet>
```

The style-sheet starts with the declaration of the parameters: their values will be determined by the optimization algorithm. The main part of the style-sheet is for region attributes: their values are redefined to customize the presentation; most of them are based on the parameters. The style-sheet include an external section (smilstruct.xml):

```

<?xml version="1.0" encoding="ISO-8859-1"?>

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <!-- build the basic SMIL structure -->
  <xsl:template match="smil">
    <smil>
      <head>
        <xsl:apply-templates select="head"/>
      </head>
      <body>
        <xsl:apply-templates select="body"/>
      </body>
    </smil>
  </xsl:template>

  <!-- apply templates for head -->
  <xsl:template match="head">
    <xsl:apply-templates/>
  </xsl:template>

  <!-- apply templates for body -->
  <xsl:template match="body">
    <xsl:apply-templates/>
  </xsl:template>

  <!-- apply templates for layout -->
  <xsl:template match="layout">
    <layout type="text/smil-basic-layout">
      <xsl:apply-templates/>
    </layout>
  </xsl:template>

  <!-- apply templates for switch, using my preferences -->
  <xsl:template match="switch">
    <xsl:apply-templates select="*[@systemLanguage='en']|
      *[@systemScreenDepth='16']|
      *[@systemCaptions='on']|
      *[@systemOperatingSystem='linux']|
      *[@systemScreenSize='800x600']
    "/>
  </xsl:template>

  <!-- apply templates for par -->
  <xsl:template match="par">
    <par>
      <xsl:if test="@id">

```

```

        <xsl:attribute name="id">
          <xsl:value-of select="@id"/>
        </xsl:attribute>
      </xsl:if>
    <xsl:apply-templates/>
  </par>
</xsl:template>

<!-- apply templates for seq -->
<xsl:template match="seq">
  <seq>
    <xsl:if test="@id">
      <xsl:attribute name="id">
        <xsl:value-of select="@id"/>
      </xsl:attribute>
    </xsl:if>
    <xsl:apply-templates/>
  </seq>
</xsl:template>

<!-- apply templates for excl -->
<xsl:template match="excl">
  <seq>
    <xsl:if test="@id">
      <xsl:attribute name="id">
        <xsl:value-of select="@id"/>
      </xsl:attribute>
    </xsl:if>
    <xsl:apply-templates/>
  </seq>
</xsl:template>

<!-- media -->

<!-- apply templates for img -->
<xsl:template match="img">
  
    <xsl:if test="@dur">
      <xsl:attribute name="dur">
        <xsl:value-of select="@dur"/>
      </xsl:attribute>
    </xsl:if>
    <xsl:apply-templates/>
  </img>
</xsl:template>

<!-- apply templates for text -->
<xsl:template match="text">
  <text
    id="{@id}"
    src="{@src}"
    type="{@type}"
    region="{@region}"
  />
</xsl:template>

<!-- apply templates for audio -->

```

```
<xsl:template match="audio">
  <audio
    id="{@id}"
    src="{@src}"
    type="{@type}"
  >
    <xsl:if test="@dur">
      <xsl:attribute name="dur">
        <xsl:value-of select="@dur"/>
      </xsl:attribute>
    </xsl:if>
  </audio>
</xsl:template>

<!-- apply templates for area -->
<xsl:template match="area">
  <area
    href="{@href}"
  />
</xsl:template>

</xsl:stylesheet>
```

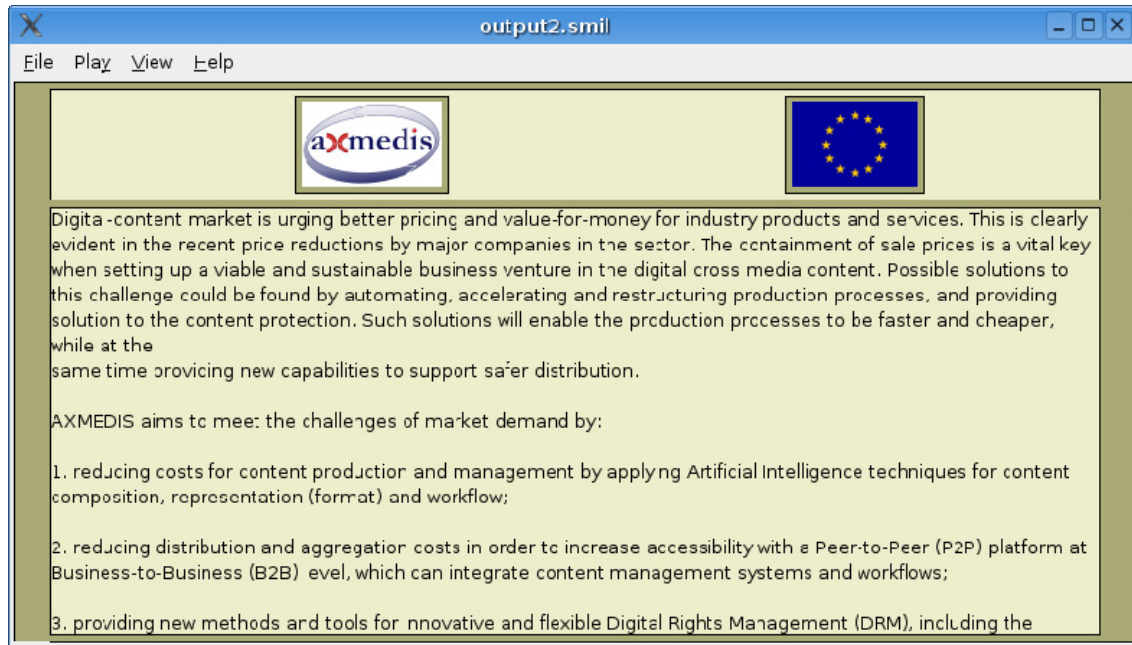
This section is standard for every style-sheet: its goal is simply recreating the basic structure of a SMIL document, based on the elements that appear in the template. The most interesting part is the <switch> processing: here we are default values for selecting options in the <switch> statement.

The result of the style-sheet application to the template, depending on values assigned to parameters, could be the following:



A styled presentation

Other style-sheets may be applied at the document we have just produced, modifying its aspect; for instance:



Another style-sheet

11 Content Formatting Tools (DSI)

These tools allow manual or automatic formatting of multimedia presentations. The “presentation” is intended here as the most generic type of multimedia content that may be diffused through a network, which includes all kinds of single and composed media and the interactions among them and with the user. The format of the presentation has to be automatically selected between the previously created models and has to be adaptable to a wide range of platforms and user preferences.

The authoring of such a multimedia presentation is a complex process that requires the specification of several types of information: the media items to use, their spatial layout, links, interactions, and temporal relationships between them. Moreover, the increased number of platforms and network connections that are used to obtain multimedia contents, requires an additional step for the final optimization.

Content Formatting Tools are at blueprint level.

11.1 Technical Details

reference to the AXFW location of the demonstrator	A path in the CVS for example: https://cvs.axmedis.org/repos/Framework/source/rulemodel/
List of libraries used	Xalan-C++ Xerces-C++
References to other major components needed	
Problems not solved	•
Configuration and execution context	
Programming language	

11.2 Template Editor

The Template Editor allows to create the basic structure for a new type of document. In the Formatting System, templates are written in SMIL: they have to define the spatial regions of the presentation, their associations with the media, their temporal relationships and their interactions. Although the spatial properties have not to be defined in the template, a good SMIL editor is required to accomplish the task of defining timing and synchronization.

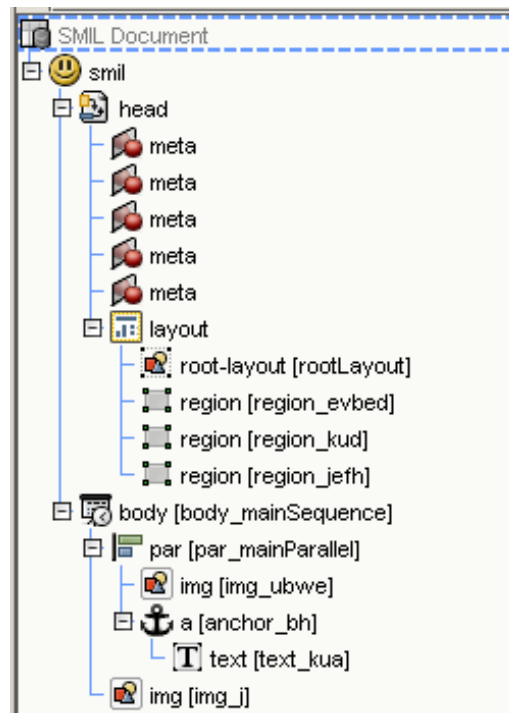
The Template Editor GUI allows the user to:

- create and edit (with mouse operations) rectangular regions on the screen;
- associate the rectangles with the <region> elements of the SMIL language;
- associate media file to the rectangular regions;
- access a context menu to set regions attributes (e.g.: identifier, position, dimensions, colors) that will be immediately reflected in the drawing;

Attribute	Value
background-color	[Empty]
backgroundColor	white
bottom	[Empty]
fit	scroll
height	269
id	region_evbed
left	294
right	[Empty]
showBackground	[Empty]
top	198
width	329
z-index	[Empty]

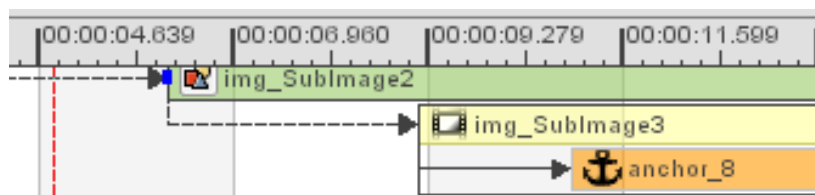
Example of attributes menu

- switch by one touch tabs among different views such as text view (XML text view), composite media scene view (WYSIWYG), tree view (XML elements), single media view (video, audio, images, etc.);



Example of tree view

- edit timing and synchronization properties with a timeline view.



Example of timeline view

- syntax coloring and automatic indentation in XML text view;
- code completion in XML text view.

Examples of SMIL editors with these capabilities are:

- LimSee2 (<http://wam.inrialpes.fr/software/limsee2/>) a free (for not commercial purposes) and open-source Java application that features a powerful WYSIWYG graphical user interface designed to ease the manipulation of time-based multimedia;
- GRiNS (<http://www.oratrix.com/Products/G2E>), a commercial product (with a free trial version) that features a high level of control over presentation making, multiple exports, complete control for the all SMIL 2.0 Content Control constructs.

We planned to use as Template Editor the SMIL Editor included within the AXMEDIS Framework, which will provide such functionalities. Additionally, the editor should check the syntax against an arbitrary DTD, to allow extensions to the SMIL language.

11.3 Automatic Template Selector

The Automatic Template Selector performs an input mapping, based on selected media and other informations provided by the author.

The input media set, that varies for type and number of elements, may be considered as composed of subsets to get more informations about content destination: these subsets should represent the "multimedia primitives" (scenes, clips, slides, etc.). Such subsets may eventually derive from querying process or be specified by the author and they are defined in the Compositional Process (see Chapter 9). The author also specifies if interactions with the user are needed or not, and other general preferences.

11.4 Style-sheet Editor

The Style-sheet Editor allows to create and modify the XSL style-sheets for the templates. XSLT is a powerful general-purpose transformation language and purely visual editors can not fully exploit its expressivity. Thus the Style-sheet Editor has to offer both visual and text tools.

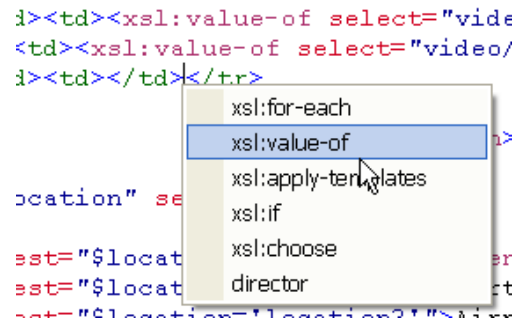
Text tools provide:

- syntax coloring and automatic indentation;
- code completion;



Example of code completion and syntax highlighting

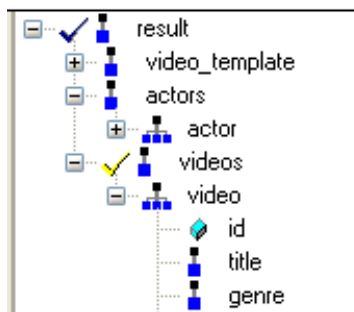
- available functions listing and their prototypes.



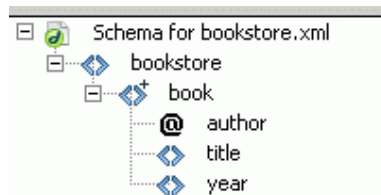
Example of functions listing

Visual tools provide:

- a source tree view for the input XML (SMIL) document, featuring the canonized (simplified and symbolic) representation of the tree;



Example of symbolic XML tree view



Example of canonized XML tree view

- an `xsl:template` view;
- association between `xsl:template` and source tree elements with mouse operations;
- result preview.

A good examples of XSL editor is the Stylus Studio's suite (<http://www.stylusstudio.com/xslt.html>), that offers an XSLT Editor, an XSLT Mapper and WYSIWYG XSLT Designer.

Text tools may be almost the same used in the Template Editor. Probably, visual tools have to be created specifically for the Style-sheet Editor.

11.5 Automatic Style-sheet Selector

The Automatic Style-sheet Selector search for style-sheets that may be applied to the given template. Data needed to associate a style-sheet to a template are stored in the style-sheet database, thus the Automatic Style-sheet Selector has to perform a simple query.

If query returns many style-sheets, the Automatic Style-sheet Selector chooses the best one on the basis of the preferences specified by the author.

11.6 Style-sheet Optimizer

The Style-sheet Optimizer is based on iterative algorithms: they have to determine (sub-) optimal values for the parameters defined in the style-sheet. For instance, an XSL style-sheet may define parameters for dimensions and colours:

```
<!-- parameters -->
<xsl:param name="screenWidth"/>
<xsl:param name="screenHeight"/>
<xsl:param name="backGround"/>
<xsl:param name="foreGround"/>
<xsl:param name="menuWidth"/>
<xsl:param name="textWidth"/>
```

Each parameter may assume a range of values; every combination of parameters generates a possible layout that has to be evaluated according a cost functional. For example, we could take into account criteria as the white space minimization or the overlapping minimization, adopting a different weight for each criteria.

Tabu Search and Genetic Algorithms are best candidates to be adopted in the Style-sheet Optimizer.

11.7 Style-sheet Processor

The Style-sheet Processor applies the style-sheets onto the templates, transforming SMIL documents into other SMIL documents.

Many XSLT processors are freely available: we plan to use the Xalan-C++ from The Apache Software Foundation (<http://xml.apache.org/xalan-c/index.html>). It is a robust implementation of the W3C Recommendations for XSL Transformations (XSLT) and the XML Path Language (XPath). The input may appear in the form of a file or URL, a stream, or a DOM; Xalan-C++ performs the transformations specified in the XSL style-sheet and produces a file, a stream, or a DOM as specified by the transformation. Along with a complete API for performing transformations in C++ applications, Xalan-C++ provides a command line utility for prototyping convenience. Furthermore, Xalan-C++ allows to call custom C++ functions from a style-sheet. Xalan-C++ is available under the terms of The Apache Software License.

11.8 Template and Document Viewer

The Template and Document Viewer has to show:

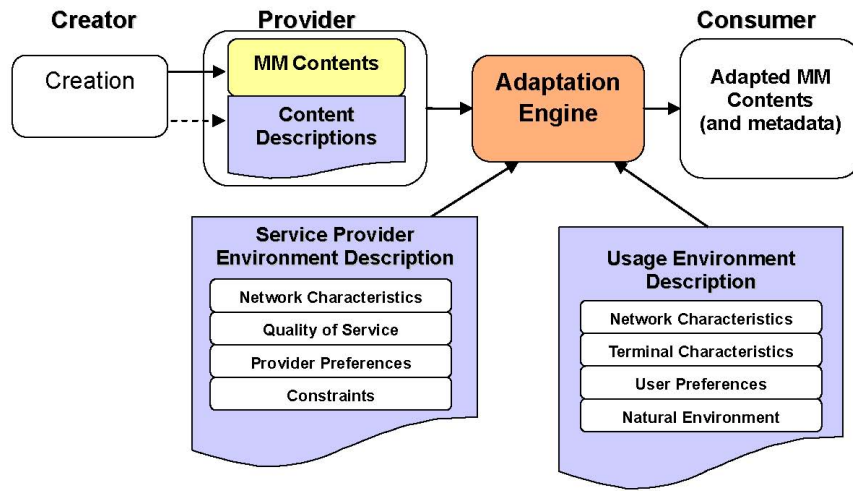
- the SMIL document that integrates media files and template;
- the SMIL document created by an XSLT processor that applies the style-sheet onto the template;
- other multimedia document formats that are admitted for the distribution (i.e.: MPEG4).

These tasks can be accomplished by the viewer included within the AXMEDIS Framework: the Internal SMIL Player and the Internal MPEG4 Player for multimedia formats, the Document Viewer for HTML, the other players for single media formats (the Internal Audio Player for MP3 and WAV, the Internal Image Viewer for JPG and PNG, the Internal Video Player for AVI and MPG, etc.). For more informations about internal players and viewers, see DE3.1.2 part B.

12 Transcoding and Adaptation (FHGIGD)

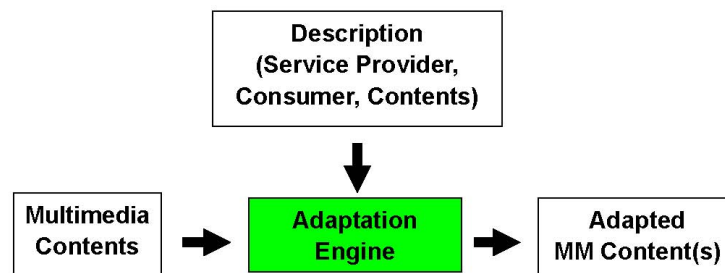
Nowadays the users demand is developing into one direction: users want to access any multimedia content over any network type with any device from anywhere at anytime. This requirement is considered by the Universal Multimedia Access (UMA, cf. [1], [2] and [3]). As described in [1], “UMA is a key framework for multimedia content delivery service using metadata.”

In [1] the adaptation between provider and consumers are identified as a major problem: for the end users, the quality of service as well as user experience has to be maximized. The task of the adaptation engine is to “bridge the gap between media format and terminal, network, user and provider characteristics” [1]. The central role of the adaptation engine in the UMA framework is shown in the next figure.



Central role of the Adaptation Engine within UMA (source: [1]).

As shown in the next figure, the adaptation engine according to UMA adapts content depending on the input content and the available description of the service providers, of the consumers and of the content.



Content adaptation framework in UMA systems. (source [1])

Examples for the different parameters for content adaptation are:

Content	Service Provider	Usage Environment
Media formats, Bitrate	Quality of Service	Access Network (Bandwidth)
Spatial resolution	Available Bandwidth	Display resolution / color
Temporal resolution	Error rate	Memory / CPU / Decoders
Number of Colors	Constraints	User preference
Limitations, rights	Delay	Access location, time

According to [1], the adaptation engine can have different functionalities:

- Transformation engine
- Variation selection engine
- Content selection engine
- Content and variation selection engine

This approach is considered within AXMEDIS as seen in the previous sections. MPEG-21 Part 7 Digital Item Adaptation (DIA), tools in MPEG-7 Part 5 Multimedia Description Scheme (MDS) are relevant for adaptation. These standards address a major part of the multimedia content adaptation process. Details of the AXMEDIS architecture are described in chapter 3 (“AXMEDIS Architecture for Content Processing”).

In this section, the focus is on transcoding and adaptation, which is the “transformation engine” according to the UMA terminology.

General issues which are relevant for the transcoding and adaptation of content include (see [1]):

- Adaptation system architecture: adaptation on server-side vs. adaptation on client-side vs. proxy-based adaptation.
- Storage location of usage environment descriptions: storage on server side, storage on client side, storage on intermediate server
- Privacy of personal data, of personal communication, of the person and of personal behaviour.

[1] E, K. & T, E. New Frontiers in Universal Multimedia Access EPFL, 2004

[2] Sanjuan, D.M.; Steiger, O. & Ebrahimi, T. Design and Implementation of a Universal Multimedia Access Environment 2002

[3] Steiger, O.; Sanjuán, D.M. & Ebrahimi, T. MPEG-based personalized content delivery. 2003, 45-48

13 Transcoding Audio (EPFL)

13.1 Technical Details

reference to the AXFW location of the demonstrator	
List of libraries used	
References to other major components needed	
Problems not solved	<ul style="list-style-type: none"> • •
Configuration and execution context	
Programming language	

13.2 Audio: State of the art

Content is generally created in a single format and can thus only be consumed by a limited set of users. Content adaptation provides mechanisms that allow formats to be interchanged such that the content may be delivered in other formats than that in which it was originally created. For example, it is common practice for radio stations to adapt the audio signal to be transmitted according to the channel characteristics.

With the explosion in availability of digital audio content and the advent of audio transmission over the Internet, adaptation, and particularly techniques that maximize quality output given a set of constraints, has gained a new dimension. Multimedia applications are increasingly used in mobile networks as well as in cable and fixed networks. Small devices with low performance have to be supported, as well as home entertainment equipment with high quality and multi channel audio.

To allow for transparent audio distribution, an adaptation tool needs to be able to perform the following three transformations:

1. Format and codec conversion
2. Sampling rate conversion
3. Channel mix-down

These transformations, though they may alter the quality of the signal, do not aim at modifying the content explicitly. On the contrary, digital audio effects aim at transforming the audio signal more radically. Transformations such as equalization or reverberation are widely used by music producers but also by music listeners since many hi-fi's propose such settings. More complex effects such as time-stretching are used to synchronise easily a soundtrack or dialogues with a movie. Fading effects are used when one needs to shorten a sound excerpts (to build a small sample clip from a complete song for example).

13.3 Audio: The problems

The main problems encountered concern the integration of libraries coming from different sources so that special care has to be taken to allow producing a multi-platform tool for transcoding. Other libraries will be integrated to allow manipulating more file formats (libfaac and libfaad to use AAC files and libvorbis and libogg to use ogg files),

13.4 Audio: Work performed

The FFMPEG library and the LIBSNDFILE library have been integrated in a single command line application to allow for multi-platform transcoding between most compressed and uncompressed audio formats. Sampling rate conversion and channel mixing is also supported as well as the selection of some specific codec for changing the resolution of audio samples (from 16 to 8 bits for example). The audio adaptation tool has been successfully compiled on Mac Os X and Win32 (MinGW) and should compile as well on Linux.

We describe here the command line interface to the audio adaptation tool so that all of its functionalities can be presented.

The generic syntax is:

```
AudioAdaptation [input file] [options] [output file]
```

By default, the conversion is done as losslessly as possible by using the same parameters for the output as the one of the input (i.e. if the input as a sampling rate of 44100 Hz and 2 channels, the output by default will have the same settings). File formats for both input and expected output are detected according to the extension of the files. The following table lists the recognized file extensions and the corresponding file formats.

Supported Extension	Correponding format
.aif	Apple/SGI AIFF
.wav	Microsoft WAV
.au	Sun/Next AU
.snd	Sun/Next AU
.svx	Amiga IFF/SVX8/SV16 IFF
.paf	Ensoniq PARIS PAF
.fap	Ensoniq PARIS PAF
.nist	NIST Sphere WAV
.ircam	Berkeley/IRCAM/CARL SF
.sf	Berkeley/IRCAM/CARL SF
.voc	Creative Labs VOC
.w64	SoundFoundry WAVE 64 W64
.raw	Header-less RAW
.mat4	GNU Octave 2.0 / Matlab 4.2 MAT4
.mat5	GNU Octave 2.1 / Matlab 5.0 MAT5
.mat	GNU Octave 2.0 / Matlab 4.2 MAT4
.pvf	Portable Voice Format PVF
.sds	Midi Sample Dump Standard SDS
.xi	FastTracker 2 XI
.ac3	Raw AC3
.asf	Advanced Systems Format
.mp2	MPEG audio layer 2
.mp3	MPEG audio layer 3
.alaw	PCM A law format
.mulaw	PCM Mu law format
.s16be	PCM signed 16 bit big endian format
.s16le	PCM signed 16 bit little endian format

.s8	PCM signed 8 bit format
.u16be	PCM unsigned 16 bit big endian format
.u16le	PCM unsigned 16 bit little endian format
.u8	PCM unsigned 8 bit format
.ra	Audio Real Media format
.wma	Windows Media Audio format

The following options are supported:

- ‘-c’ : set the number of audio channels of the output (default: number of channels of input)
- ‘-r’ : set the sampling rate of the output (default: sampling rate of the input)
- ‘-b’ : set the bit rate of the output (default: 64 kb/s)
- ‘-codec’ : set the encoding codec
- ‘-y’ : overwrite input file (if input and output file have the same name and extension)
- ‘-st’ : start time position in the input file (in seconds)
- ‘-d’ : duration to read from the input file (in seconds)

The supported codecs are listed in the table below. Notice however that some codecs may not be useable with some specific format.

Codecs	Decoding	Encoding
wmav1	Yes	No
wmav2	Yes	No
mace3	Yes	No
mace6	Yes	No
vmdataudio	Yes	No
real_144	Yes	No
real_288	Yes	No
roq_dpcm	Yes	No
interplay_dpcm	Yes	No
xan_dpcm	Yes	No
sol_dpcm	Yes	No
Flac	Yes	No
shorten	Yes	No
alac	Yes	No
ws_snd1	Yes	No
vorbis	Yes	No
ac3	No	Yes
mp2	Yes	Yes
mp3	Yes	Yes
mp3adu	Yes	No
mp3on4	Yes	No
sonic	Yes	Yes
sonicls	No	Yes
float_s32	Yes	Yes
pcm_s32	Yes	Yes
pcm_s24	Yes	Yes
pcm_s16le	Yes	Yes
pcm_s16be	Yes	Yes
pcm_u16le	Yes	Yes
pcm_u16be	Yes	Yes

pcm_s8	Yes	Yes
pcm_u8	Yes	Yes
pcm_alaw	Yes	Yes
pcm_mulaw	Yes	Yes
adpcm_ima_qt	Yes	Yes
adpcm_ima_wav	Yes	Yes
adpcm_ima_dk3	Yes	Yes
adpcm_ima_dk4	Yes	Yes
adpcm_ima_ws	Yes	Yes
adpcm_ima_smjpeg	Yes	Yes
adpcm_ms	Yes	Yes
adpcm_4xm	Yes	Yes
adpcm_xa	Yes	Yes
adpcm_adx	Yes	Yes
adpcm_ea	Yes	Yes
adpcm_ct	Yes	Yes
adpcm_swf	Yes	Yes
g726	Yes	Yes
gsm610	Yes	Yes
dwvw12	Yes	Yes
dwvw16	Yes	Yes
dwvw24	Yes	Yes

Here are a few examples of typical command line:

- Convert a WAV file into a AIFF file with 1 channel and a sampling rate of 22050 Hz:

```
AudioAdaptation example_in.wav -c 1 -r 22050 example_out.aif
```
- Transform the WAV file `example.wav` into a mono file with overwriting:

```
AudioAdaptation example.wav -y -c 1 example.wav
```
- Convert a WAV file into an MP3 file with 128kb/s bit rate:

```
AudioAdaptation example.wav -b 128 -codec mp3 example.mp3
```
- Convert a WAV file into a WAV file with same encoding parameters but with duration 30 seconds starting 90 seconds after the beginning of the original file:

```
AudioAdaptation example.wav -d 30 -st 90 example.wav
```

14 Transcoding Video (FHGIGD)

14.1 Technical Details

reference to the AXFW location of the demonstrator	A path in the CVS for example: https://cvs.axmedis.org/repos/Framework/source/rulemodel/
List of libraries used	
References to other major components needed	
Problems not solved	<ul style="list-style-type: none"> • •
Configuration and execution context	
Programming language	

14.2 Video: State of the art

Chang et al. [Chang05] identified active research areas of video adaptation. These include:

- Semantic event-based adaptation: Semantic events in the video are the basis for this kind of adaptation. Depending on the video content and the target audience and the usage of the video content the adaptation varies. For example, in a soccer game only the key (scoring) scenes might be extracted in a video. The basic required semantic information can also be extracted from other source than from the video, e.g. from the corresponding audio [Chang01, Zhang02, Zhong01].
- Structural-Level Adaptation is a simpler approach than semantic event-based adaptation, which is based on the structure of video. An example is the extraction of key-frames that summarize the video content best [Irani98].
- Transcoding is on a signal level. For example, spatial resolution, precision or temporal properties (e.g. frame rate) are adapted according to the possibilities of the transmission channel and the receiving device [Vetro03].
- Rapid Fast-Forward can be compared with the fast-forward feature of a VCR player. The content is temporally “condensed” [Smith95, Smith97, Lou04].

14.3 Video: The problems

Different open issues have been identified in the literature, e.g. in [Vetro05] and [Chang05]. Among these open issues are:

- Permissible and secure adaptation: The two aspects which are comprised here are the permission for specific adaptations and the secure adaptation of encrypted objects.
- Adaptation in Constrained and Streaming Environment: For some devices or scenarios limited hardware capabilities have to be considered. This influences the adaptation, e.g. the conversion of adaptation parameters or the adapted content.
- Transport, Negotiation and Exchange of Descriptors: The adaptation parameters have to be transported, negotiated or exchanged. This has to be handled in a standardized manner.
- Semantic Clues for Adaptation: Semantic information of the content that should be adapted can improve the result of the adaptation.

- Modality Conversion: Sometimes a modality conversion (e.g. from a video format to a image sequence) might be better suited. E.g. for certain devices/transmission channel there might be strong hardware limitations.
- Maximum User Experience: The previous aspects address technical issues. The ultimate criteria, however, is the user experience. This user experience has to be maximised.

These general aspects are also relevant for video adaptation. In addition to the previously described active research areas, which can also be considered as open issues, in [Chang05] specific open issues for video adaptation are also described:

- Definition of utility measure and user preferences: There is a general lack of a definition of adequate measures or methods for estimating the quality or the user's satisfaction of the video content ("utility").
- Resolving ambiguities in specifying adaptation operation: A verbal description of an adaptation operation is often ambiguous.
- Relations among adaptation, utility, and resource. The relationship between adaptation, utility, and resource is often complex (high dimensional space). The representation of this high dimensional space strongly depends on the application.
- Optimal solutions in large spaces are required to address the previously described problem of high-dimensional spaces. From a mathematical point of view, this is a constraint optimization problem.
- Design end-to-end integrated systems

14.4 Video: Work performed

The work performed focused on the (integration of) transcoding algorithms. To show AXMEDIS' capabilities in terms of content adaptation as well as extensibility the FFMPEG library has been selected for the implementation/integration of the firstly available video adaptation functionality.

The via command line version of FFMPEG available video options are:

-b bitrate	set video bitrate (in kbit/s)
-r rate	set frame rate (Hz value, fraction or abbreviation)
-s size	set frame size (WxH or abbreviation)
-aspect aspect	set aspect ratio (4:3, 16:9 or 1.3333, 1.7777)
-croptop size	set top crop band size (in pixels)
-cropbottom size	set bottom crop band size (in pixels)
-cropleft size	set left crop band size (in pixels)
-cropright size	set right crop band size (in pixels)
-vn	disable video
-bt tolerance	set video bitrate tolerance (in kbit/s)
-maxrate bitrate	set max video bitrate tolerance (in kbit/s)
-minrate bitrate	set min video bitrate tolerance (in kbit/s)
-bufsize size	set ratecontrol buffere size (in kByte)
-vcodec codec	force video codec ('copy' to copy stream)
-sameq	use same video quality as source (implies VBR)
-pass n	select the pass number (1 or 2)
-passlogfile file	select two pass log file name

Besides the basic functionality so called "advanced options" are available:

-pix_fmt format	set pixel format
-g gop_size	set the group of picture size

-intra	use only intra frames
-qscale q	use fixed video quantiser scale (VBR)
-qmin q	min video quantiser scale (VBR)
-qmax q	max video quantiser scale (VBR)
-mbqmin q	min macroblock quantiser scale (VBR)
-mbqmax q	max macroblock quantiser scale (VBR)
-qdiff q	max difference between the quantiser scale (VBR)
-qblur blur	video quantiser scale blur (VBR)
-qcomp compression	video quantiser scale compression (VBR)
-rc_init_cplx	complexity initial complexity for 1-pass encoding
-b_qfactor factor	qp factor between p and b frames
-i_qfactor factor	qp factor between p and i frames
-b_qoffset offset	qp offset between p and b frames
-i_qoffset offset	qp offset between p and i frames
-rc_eq equation	set rate control equation
-rc_override	override rate control override for specific intervals
-me method	set motion estimation method
-dct_algo algo	set dct algo
-idct_algo algo	set idct algo
-er n	set error resilience
-ec bit_mask	set error concealment
-bf frames	use 'frames' B frames
-mbd mode	macroblock decision
-mbcmp	cmp function macroblock compare function
-ildctcmp	cmp function ildct compare function
-subcmp	cmp function subpel compare function
-cmp	cmp function fullpel compare function
-4mv	use four motion vector by macroblock (MPEG4)
-obmc	use overlapped block motion compensation (h263+)
-part	use data partitioning (MPEG4)
-bug param	workaround not auto detected encoder bugs
-strict strictness	how strictly to follow the standards
-deinterlace	deinterlace pictures
-ildct	force interlaced dct support in encoder (MPEG2/MPEG4)
-ilme	force interlaced me support in encoder MPEG2
-psnr	calculate PSNR of compressed frames
-vstats	dump video coding statistics to file
-vhook module	insert video processing module
-aic	enable Advanced intra coding (h263+)
-aiv	enable Alternative inter vlc (h263+)
-umv	enable Unlimited Motion Vector (h263+)
-alt	enable alternate scantable (mpeg2)
-trell	enable trellis quantization
-scan_offset	enable SVCD Scan Offset placeholder
-intra_matrix	matrix specify intra matrix coeffs
-inter_matrix	matrix specify inter matrix coeffs
-top	top=1/bottom=0/auto=-1 field first
-nr	noise reduction

The functionality available within FFMPEG addresses two categories of video adaptation:

- Considering the (fixed) needs of the content creator or service providers: The content can be adapted according to the requirements of the content creator or service providers. The simplest case is the

adaptation of the spatial resolution as supported by FFMPEG to a fixed resolution without changing the aspect ratio.

- Dynamic content creation: FFMPEG also allows changing the aspect ratio of the video content. Thus, a basic functionality required to dynamically adapt content according to the receiving device is available.

14.5 References

- [Chang01] Chang, S.; Zhong, D. & Kumar, R. Real-Time Content-Based Adaptive Streaming of Sports Videos IEEE Computer Society, 2001, 139
- [Chang05] Shih-Fu Chang Vetro, A. Video adaptation: concepts, technologies, and open issues 2005, 93, 148 – 158
- [Irani98] Irani, M. & Anandan, P. Video indexing based on mosaic representation 1998
- [Lou04] Luo, H. & Fan, J. Concept-oriented video skimming via semantic video classification ACM Press, 2004, 760-761
- [Smith95] Smith, M. & Kanade, T. Video Skimming for Quick Browsing based on Audio and Image Characterization Computer Science Department, Carnegie Mellon University, 1995
- [Smith97] Smith, M.A. Video Skimming and Characterization through the Combination of Image and Language Understanding Techniques IEEE Computer Society, 1997, 775
- [Zhang02] Zhang, D. & Chang, S. Event detection in baseball video using superimposed caption recognition ACM Press, 2002, 315-318
- [Zhong01] Zhong, D.; Kumar, R. & Chang, S. Real-time personalized sports video filtering and summarization ACM Press, 2001, 623-625
- [Vetro03] Vetro, A.; Haga, T.; Sumi, K. & Sun, H. Object-based coding for long-term archive of surveillance video 2003, 2
- [Vetro05] Vetro, A. & Timmer, C., Digital Item Adaptation: Overview of Standardization and Research Activities IEEE Transaction On Multimedia, 2005, 7, 418-426

15 Transcoding Documents and text (DIPITA)

The prototype for textual format conversion tool has to gather and integrate the different standard libraries for the conversion of the required formats. According to the AXMEDIS FW specifications (DE 3.1.2d), the formats treated within the conversion procedures are: plain text (TXT), Rich Text Format (RTF), Hyper Text Mark-up Language (HMTL), portable document Format (PDF), Post Script (PS).

Textual conversion is needed for:

- format adaptation tasks;
- perform keyword extraction on different format input files (see Humphreys 2002);
- perform content fingerprint estimation (independently from the specific format of input file);
- guessing of the language of a document.

The demonstrator

The demonstrator is a simple command line interface program, that takes four parameters as input: the name of the file to be transcoded, the name of the file to be created, the formats of the file to be transcoded and of the file to be created. The demonstrator shows how two of the libraries (XPDF and HTMLDOC) that will be exploited (listed in section 16.4) are used.

Two transcoding are allowed in the demonstrator: pdf to txt (using XPDF's pdftotext.exe executable) and html to txt (using HTMLDOC's htmldoc.exe and XPDF's pdftotext.exe executables).

Below the snapshot of the execution of the pdf to txt transcoding:

```

C:\> Command Prompt
C:\Documents and Settings\Fabbri.LABLITA_DOM2.001\My Documents\axmedis\DocumentAdaptation\DocumentAdaptationExternalModule\Debug>DocumentAdaptationExternalModule.exe c:\trattato_EU.pdf c:\trattato_EU.txt pdf txt
C:\Documents and Settings\Fabbri.LABLITA_DOM2.001\My Documents\axmedis\DocumentAdaptation\DocumentAdaptationExternalModule\Debug>_

```

The next snapshot shows the execution of the html to txt transcoding:

```

C:\> Command Prompt
C:\Documents and Settings\Fabbri.LABLITA_DOM2.001\My Documents\axmedis\DocumentAdaptation\DocumentAdaptationExternalModule\Debug>DocumentAdaptationExternalModule.exe c:\htmldoc.htm c:\htmldoc.txt html txt
PAGES: 19
BYTES: 382350
C:\Documents and Settings\Fabbri.LABLITA_DOM2.001\My Documents\axmedis\DocumentAdaptation\DocumentAdaptationExternalModule\Debug>_

```

15.1 Technical Details

reference to the AXFW location of the demonstrator	https://cvs.axmedis.org/repos/Framework/source/adaptation/document/
List of libraries used	XPDF, HTMLDOC, DOCFRAC, GNU Ghostscript
References to other major components needed	
Problems not solved	<ul style="list-style-type: none"> • Input format guessing (so far, input format is an input parameter). • Not all possible conversion are covered.
Configuration and execution context	<p>To exexute the program the following dll, exe files and directories must be in the same directory as the program one:</p> <p>doxlib.dll libeay32.dll ssleay32.dll pdftotext.exe htmldoc.exe data/</p>

	doc/ fonts/ (all under Framework\lib\adaptation\document directory in the repository) The directory in which the program is runned must have write privileges.
Programming language	C++

15.2 Documents and text: State of the art

Since textual conversion is a useful task for a lot of text processing applications, there has been developed a huge variety of conversion libraries and software in order to get a plain text file from the most common formats (see Anjewierden, 2003). As a matter of fact, the conversion of RTF and HTML formats into plain text is a task that is completely satisfied by the modern technologies, and it represents a quite trivial issue. More problems emerge from the conversion procedures starting from PDR or PS format (see the next paragraphs for further details). Format adaptation is needed to pre-process document in order to deal with the main procedures of the AXMEDIS linguistic interface (keyword extraction and language guessing) and of the content protection and fingerprint. In the following section (7.1.3) there are presented the libraries to be exploited.

Particular issue: conversion from PDF/PS document to plain text format

Most of the textual files associated to multimedia content production and documentation are in PDF format, because of it is a platform-independent format, and because of the PDF files are relatively small in file size, allowing space saving on servers and repositories.

In different environment dealing with text pre-processing for information extraction issues, PDF format is used for storage and start point of the further conversion procedures. It is the case of CERN's solution for the automatic extraction of references from High Energy Physics documents and papers (to the aim of make them easy to access to physics researchers). The extraction procedure involve the whole repository of up to 170,000 electronic documents (available via the CERN Document Server; see Claivaz et al. 2001). For this task, PDF was deliberately chosen as a format from which extract the plain text document, due to its stability over other file formats such as PostScript or MSWord.

In Robinson (2001) there is a comparison study of these different tools with their advantages and drawbacks (pdftotext, ps2ascii.ps, pdf2html, pdf2html(.bin), pstotext, prescript). A lot of conversion tools were tested to find the most suitable one.

The primary flaw discovered was that none of the tools were able to convert PostScript documents that were produced by MSWord. Due to the increasing number of PostScript files submitted to the document archives and repositories that were actually produced from MSWord files, this is a major problem, as it means that none of these could be converted to text. This means that it is really necessary to perform the conversion from a PDF document, as it seems to be more stable. Tests were conducted for the conversion of PDF documents to text that were produced from PostScripts that were in turn produced from MSWord files. Problems were not encountered with these PDF file conversions.

Therefore, the best strategy for conversion of PS format into plain text seems to be a two stage conversion:

- first, convert the PS to PDF;
- second, covert the PDF obtained to a plain text.

As a result from the comparison among pds2txt converters, the *pdftotext* tool from Foolabs (within XPDF tool-suite) was stated as the more reliable among the others (Robinson 2001).

First prototype to be posted in the AXMEDIS CVS repository within M12

15.3 Documents and text: The problems

Libraries to be exploited: general public license.

Because some of the used libraries are licensed under GPL, the conversion tool (pool of document converters) will be developed under GPL too. So the conversion tool will be accessed by the plug-in as a separated executable program, not as a library. The communication mechanism between the plug-in and the converter has to be defined.

Different output of format conversions

It is necessary to specify the particular features of the txt output files, depending on the different tasks in which textual format conversion are involved. The output textual files have to deal with:

1. Content fingerprint adaptation (within WP 4.2.2)

The main requirement to be satisfied by a fingerprint extractor is the independence from the file format. To solve this problem, the file of the original document is converted to a plain text file. As totally reliable format-conversion procedures are not available, to obtain the most similar fingerprint values from different format files it is needed, for example, to reduce the text file to a more stable one, by removing white space and punctuation, removing unusual characters (which can be easily mismatched by the converters), converting all letters to lower case, replacing variable names (in script files) by one string.

2. Language guessing adaptation (within WP 4.2.2)

Language guessing rely on statistics of character n-grams, extracted from documents of the languages to be treated. For this purpose, sequences of spaces and alphabetic character have to be exactly the same, even after document format conversion. Adaptation strategies such as the ones adopted in the fingerprint adaptation of text (which point to erase the uncertain converted characters) are unsuitable for this task.

3. Keyword extraction adaptation (within WP 4.2.2)

The most important unit for this task is not the text character (as in the previous ones), but the word. The conversion of text format for keyword extraction purposes has to respect exactly the word combinations in the context within they occurs. For this task, it is important to have a certain degree of reliability even in the treatment of punctuation.

15.4 Documents and text: Work performed

The following conversion libraries are the ones that will be used for text document conversion (see AXMEDIS FW Specifications in DE 3.1.2d), as they obtain (according to the literature) the best performances compared to the other similar ones:

1. DOCFRAC (<http://docfrac.sourceforge.net/>), that allows conversions from RTF to HTML, from RTF to TXT, from HTML to RTF, from HTML to TXT, from TXT to RTF and from TXT to HTML.

The exploitation of the DOCFRAC features library will be particularly useful for converting active web pages, converting many documents at a time and converting output from Microsoft's Internet Explorer RTF control to HTML.

The supported platforms are Windows, Linux (command line) and programming kit (ActiveX and DLL). DocFrac is free. It is released under the [LGPL](#).

2. GNU Ghostscript (<http://www.cs.wisc.edu/~ghost/>)

Ghostscript is the name of a set of software that contains 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.

3. 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.

4. 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.

The adaptation tool will recognize the format of the input file, to call the right conversion library and obtain the plain text file as output.

To test the first results of the adaptation tool, it has been created a test corpus for content adaptation. It contains several documents, each one in different textual format, and precisely:

- a) Digital music report 2005 (a state of the art on digital music; formats: pdf, txt)
- b) European constitution (the EU constitution in Italian, formats: pdf, rtf, txt)
- c) Hibernate documentation (documentation on an object/relational mapping tool for Java environments; formats: pdf, html, txt)
- d) Html specification (specification from the w3c on html formatting; formats: pdf, ps, html, txt).

References

Anjewierden, A. (2003) Document Analysis Component Library. University of Amsterdam

Claivaz, J.B., Le Meur, J.-Y., Robinson, N. (2001). From Fulltext Documents to Structured Citations. CERN-ETT-2001-003. Geneva. CERN.

Heintze, N. (1996), Scalable document fingerprinting. In *1996 USENIX Workshop on Electronic Commerce*.

Humphreys, J.-B. K. (2002) PhraseRate: An HTML Keyphrase Extractor. University of California

Robinson, N. (2001). A Comparison of Utilities for Converting from PostScript or Portable Document Format to Text. [CERN-OPEN-2001-065](#). Geneva. CERN.

Websites

DOCFRAC (<http://docfrac.sourceforge.net/>),

GNU Ghostscript (<http://www.cs.wisc.edu/~ghost/>)

XPDF (<http://www.foolabs.com/xpdf/>)

HTMLDOC (<http://www.easysw.com/htmldoc/>)

16 Transcoding Images (DSI, IRC)

16.1 Technical Details

reference to the AXFW location of the demonstrator	
List of libraries used	
References to other major components needed	
Problems not solved	<ul style="list-style-type: none"> • •
Configuration and execution context	
Programming language	

16.2 Images : State of the art

Image adaptation tools provide functions which can be used for scaling, resolution improvements, text drawing, and image decomposition etc. of an image file. In the case of Image Objects, the following libraries can be used for implementing the functions described above.

a. ImageMagick Library

ImageMagick, version 6.2.3, is a free software suite designed to create, edit, and compose 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 Bezier curves can be added to images and stretched and rotated.

Here are just a few examples of what ImageMagick can do:

- i. Convert an image from one format to another (e.g. PNG to JPEG)
- ii. Resize, rotate, sharpen, colour reduce, or add special effects to an image
- iii. Create a montage of image thumbnails
- iv. Create a transparent image suitable for use on the Web
- v. Turn a group of images into a GIF animation sequence
- vi. Create a composite image by combining several images
- vii. Draw shapes or text on an image
- viii. Decorate an image with a border or frame
- ix. Describe the format and characteristics of an image

A list of supported formats is given below. The documentation of the ImageMagick is available on <http://www.imagemagick.org/>.

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 http://www.jmc	R	Microsoft Audio/Visual Interleaved	

gowan.com/avi.html			
AVS	RW	AVS X image	
BMP http://msdn.microsoft.com/library/sdkdoc/gdi/bitmaps_9c6r.htm	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	8-bit RGB PseudoColor with up to 256 palette entries. Specify

		Interchange Format	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 <code>-size</code> and <code>-depth</code> 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 Ghostscript 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 $(5 * \text{QuantumDepth} * \text{Rows} * \text{Columns}) / 8$.
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 <code>-draw</code> 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 <i>ra_ppm</i> 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 <code>-size</code> 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 <code>-density</code> option to adjust the output resolution, and thereby adjust the output size. The default output resolution is 72DPI so <code>"-density 144"</code> results in an image twice as large as the default. Use <code>-background color</code> to specify the WMF background color (default white) or <code>-texture filename</code> 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 <code>-size</code> and <code>-depth</code> to specify the image width, height, and depth.
YCbCrA	RW	Raw Y, Cb, Cr, and alpha samples	Use <code>-size</code> and <code>-depth</code> to specify the image width, height, and depth.
YUV	RW	CCIR 601 4:1:1	

17 Transcoding Multimedia (EPFL)

The goal of content adaptation tools is to adapt the original modality, resolution, and format of multimedia content to potential terminal and network capabilities. A prerequisite for efficient adaptation of multimedia information is a careful analysis of the properties of different media types. Video, audio, images and text require different adaptation algorithms. Consequently, the AXMEDIS framework will provide a number of algorithms to adapt these media types (see sections 13, 15, 16, 17).

17.1 Technical Details

reference to the AXFW location of the demonstrator	A path in the CVS for example: https://cvs.axmedis.org/repos/Framework/source/rulemodel/
List of libraries used	
References to other major components needed	
Problems not solved	<ul style="list-style-type: none"> • •
Configuration and execution context	
Programming language	

17.2 Multimedia: State of the art

The complex nature of multimedia makes the adaptation difficult to design and implement. Indeed, 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 so that tools dealing with such objects shall be able to parse the corresponding textual descriptions. Moreover, objects we refer to as multimedia or rich media may also be composed of synthetic audio (MIDI, MPEG-4 Structured Audio), 2D and 3D graphics, web content, etc... We will now enumerate and describe a number of file formats considered as multimedia.

MPEG-4: The primary uses for the MPEG-4 standard are web ([streaming media](#)) and [CD distribution](#), conversational ([videophone](#)), and [broadcast television](#). MPEG-4 absorbs many of the features of [MPEG-1](#) and [MPEG-2](#) and other related standards, adding new features such as (extended) [VRML](#) support for 3D rendering, [object-oriented](#) composite files (including audio, video and VRML (Virtual Reality Modeling Language) objects), support for externally-specified [Digital Rights Management](#) and various types of interactivity.

Most of the features included in MPEG-4 are left to individual [developers](#) to decide whether to implement them. This means that there are probably no complete implementations of the entire MPEG-4 set of standards. To deal with this, the standard includes the concept of "profiles" and "levels", allowing a specific set of capabilities to be defined in a manner appropriate for a subset of applications.

Scalable Vector Graphics (SVG): It is an [XML markup language](#) for describing two-dimensional [vector graphics](#), both [static](#) and [animated](#). It is an [open standard](#) created by the [World Wide Web Consortium](#), which is also responsible for standards like [HTML](#) and [XHTML](#).

SVG allows three types of graphic objects:

1. vector graphic [shapes](#) (e.g. paths consisting of straight [lines](#) and [curves](#), and areas bounded by them)

2. [raster graphics](#) images / [digital images](#)
3. text

Graphical objects can be grouped, styled, transformed and composited into previously [rendered](#) objects. Text can be in any XML [namespace](#) suitable to the application, which enhances searchability and [accessibility](#) of the SVG graphics. The feature set includes nested [transformations](#), [clipping paths](#), [alpha masks](#), [filter effects](#), [template objects](#) and [extensibility](#).

SVG drawings can be dynamic and interactive. The [Document Object Model](#) (DOM) for SVG, which includes the full XML DOM, allows straightforward and efficient vector graphics animation via [ECMAScript](#) or [SMIL](#). A rich set of [event handlers](#) such as *onmouseover* and *onclick* can be assigned to any SVG graphical object. Because of its compatibility and leveraging of other Web standards, features like [scripting](#) can be done on SVG elements and other XML elements from different [namespaces](#) simultaneously within the same web page.

If storage space is an issue, SVG images are sometimes saved with [gzip](#) compression, in which case they may be called "SVGZ files". Because XML contains a lot of redundant data, XML tends to [compress](#) very well and these files can be much smaller.

SWF: it is the file format used by [Macromedia Flash](#) to describe movies built of mainly two elements: vector based objects and images. The newest versions also allow audio, video and many different possible forms of interaction with the end user. Once created, SWF files can be played by the [Macromedia Flash Player](#), working either as a browser plugin or as an standalone (executable) player. Most of the times, SWF files can also be encapsulated with the player, creating a self-running SWF movie called projector.

VRML: VRML is a [text file](#) format where, e.g., [vertices](#) and [edges](#) for a 3D [polygon](#) can be specified along with the [surface color](#), image-mapped [textures](#), [shininess](#), [transparency](#), and so on. [URLs](#) can be associated with [graphical components](#) so that a [web browser](#) might fetch a web-page or a new VRML file from the [Internet](#) when the [user](#) clicks on the specific graphical [component](#). [Animations](#), [sounds](#), [lighting](#), and other [aspects](#) of the [virtual world](#) can interact with the user or may be triggered by external [events](#) such as [timers](#). A special [Script Node](#) allows to add [program code](#) (e.g., written in [Java](#) or [JavaScript](#) (ECMAScript)) to a VRML file.

VRML files are commonly called [worlds](#) and have the [.wrl extension](#) (for example island.wrl). Although VRML worlds use a text format they may often be compressed using [gzip](#) so that they transfer over the internet more quickly. Most 3D [modeling programs](#) can save [objects](#) and [scenes](#) in VRML format.

X3D: X3D is the ISO standard for real time 3D graphics, the successor to [Virtual Reality Modelling Language](#). X3D features [extensions](#) to VRML (e.g. [Humanoid Animation](#), [Nurbs](#), [GeoVRML](#) etc.), the ability to encode the scene using an [XML syntax](#) as well as the [Open Inventor](#)-like syntax of VRML97, and enhanced application programmer interfaces (APIs).

SMIL: SMIL is an abbreviation for the **Synchronized Multimedia Integration Language**. It is a [W3C](#) Recommendation for describing [multimedia](#) presentations using the Extensible Markup Language ([XML](#)). It defines timing markup, layout markup, animations, visual transitions, and media embedding, among other things.

SMIL 1.0 became an official recommendation of the World Wide Web Consortium W3C in June 1998. SMIL 2.0 became an official recommendation in August 2001. A [W3C](#) working group is currently working on SMIL 2.1, which will include a small number of extensions based on practical experience gathered using SMIL in the [Multimedia Messaging System](#) on mobile phones.

A SMIL document is similar in structure to an [HTML](#) document in that they are typically divided between a <head> section and a <body> section. The <head> section contains layout and metadata information. The <body> section contains the timing information, and is generally comprised of combinations of two main tags: parallel ("<par>") and sequential ("<seq>"). It refers to media objects by [URLs](#), allowing them to be shared between presentations and stored on different servers for [load balancing](#). The language can also associate different media objects with different [bandwidths](#).

SMIL enables people without programming or scripting backgrounds to author multimedia presentations in a simple [text editor](#). For example, a developer can write SMIL to display an [image](#) after an [audio](#) track ends.

3GPP: The 3rd Generation Partnership Project (3GPP) is a collaboration agreement that was established in December 1998. The collaboration agreement brings together a number of telecommunications standards bodies which are known as “Organizational Partners”. The current Organizational Partners are ARIB, CCSA, ETSI, ATIS, TTA, and TTC. 3GPP is the new worldwide standard for sharing multimedia content between cell phones, PDAs and computers. It features high quality at extremely low bitrates and is perfect for encoding speech.

As it was seen in AXMEDIS specification DE3-1-2C, there are only a few libraries available to manipulate multimedia files. Among them is GPAC, a multimedia framework based on the MPEG-4 Systems standard (ISO/IEC 14496-1) developed from scratch in ANSI C. As of version 0.4.0 GPAC is released under the GNU Lesser General Public License. At the time of the specification of AXMEDIS (see DE3-1-2C), GPAC was released under a GNU General Public Licensing scheme, which made it unsuitable for integration into the AXMEDIS framework. Furthermore, the GPAC project was at that time in a frozen state and has been restarted since June 2005, so that it is now one of the most advanced projects dedicated to advanced content. Consequently, we propose to use the functionalities of the GPAC library to perform multimedia files adaptation in the AXMEDIS framework. The following is summary of the functionalities provided by the GPAC library.

GPAC aims at integrating the recent multimedia standards (SVG/SMIL, VRML, X3D, SWF, 3GPP(2) tools, etc...) into a single framework. VRML97 and a good amount of the X3D standard have already been integrated into GPAC, as well as some SVG support and experimental Macromedia Flash support. The current GPAC release (0.4.0) already covers a very large part of the MPEG-4 standard, and has some good support for 3GPP and VRML/X3D. It is currently running under Windows, Linux platforms and Windows CE/PocketPC 2002 (2D rendering only, not tested on SmartPhones).

In terms of adaptation capabilities, the MP4Box developed with GPAC should be integrated in the context of AXMEDIS. MP4Box aims at providing all tools needed to produce and distribute MPEG-4, 3GP and 3GP2 content in a single command-line application. Here are is a list of some of the functionalities offered by MP4Box:

- File splitting by size or time, chunk extraction from file and file concatenation, including all supported input files concatenation (e.g. cat a set of AVIs to a single 3GPP/MP4),

- MP4/3GP conversion from MP3, AVI, MPEG-PS, AAC, H263, H264, AMR, QCP, EVRC, SMV, SUB, SRT, TTXT, TeXML...
- Media track extractions.
- MPEG-4 BIFS codec and scene conversion from and to MP4, BT and XMT-A formats.
- Conversion of simple Macromedia Flash (SWF) to MPEG-4 Systems (BT/XMT/MP4).
- Conversion to and from BT, XMT-A, WRL, X3D, X3DV formats (GZipped or not).

18 Transcoding/adaptation PAR and Licenses (FUPF)

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 following sections describe the research work performed in the adaptation of PAR and Licenses. Currently, we do not have a prototype that demonstrates the work described in this section, but isolated modules that provide a brief part of the functionality. Nevertheless, the implementation of this prototype was not planned for this period and it is expected that the corresponding prototype will be ready for the planned date.

18.1 PAR and Licenses: State of the art

18.1.1 MPEG-21 Rights Expression Language (REL)

The different parties involved in the online distribution and consumption of multimedia resources need to exchange information about the rights, terms, and conditions associated with each resource at each step in the multimedia resource lifecycle. For example in distribution and super distribution business models, the information related to the rights and the terms and conditions under which the rights may be exercised needs to be communicated to each participant in the distribution chain.

In an end-to-end system, other considerations such as authenticity and integrity of Rights Expressions become important. For example, any content provider or distributor who issues rights to use or distribute resources must be identified and authorized. In addition, different participants may access a Rights Expression, which requires mechanisms and semantics for validating the authenticity and integrity of the Rights Expression. A common Rights Expression Language that can be shared among all participants in this digital workflow is required.

Part 5 of the MPEG-21 standard specifies the syntax and semantics of a Rights Expression Language. MPEG chose XrML [1] as the basis for the development of the MPEG-21 Rights expression language.

MPEG-21 Rights Expression Language (REL) [2] specifies the syntax and semantics of the language for issuing rights for Users to act on Digital Items, their Components, Fragments, and Containers.

The most important concept in REL is the license that conceptually is a container of grants, each one of which conveys to a principal the sanction to exercise a right against a resource. A license is formed by the following elements:

- Title: this element provides a descriptive phrase about the License that is intended for human consumption in user interfaces. Automated processors must not interpret semantically the contents of such title elements.
- Inventory: this element is used for defining variables within a License. In the Inventory element of a license can be defined LicensePart elements that in turn can have licensePartId attributes that can be referenced from elsewhere in the license.

Therefore, REL provides a syntactic mechanism for reducing redundancy and verbosity in Licenses that can be used throughout a License.

- Grant or GrantGroup: The Grants and GrantGroups contained in a license are the means by which authorization policies are conveyed in the REL architecture. Each Grant or GrantGroup that is an immediate child of a license exists independently within that license, no collective semantic (having to do with their particular ordering or otherwise) is intrinsically associated with the presence of two or more of them within a certain license.
- Other information: Using the wildcard construct from XML Schema, a License provides an extensibility hook within which license issuers may place additional content as they find appropriate and convenient. This can be useful for conveying information that is peripherally related to, for example, authentication and authorization, but is not part of the REL core infrastructure. It should, however, be carefully understood that not all processors of REL licenses will understand the semantics intended by any particular use of this extensibility hook. Processors of the license may choose wholly at their own discretion to completely ignore any such content that might be present therein.

Next figure shows the structure of a REL License.

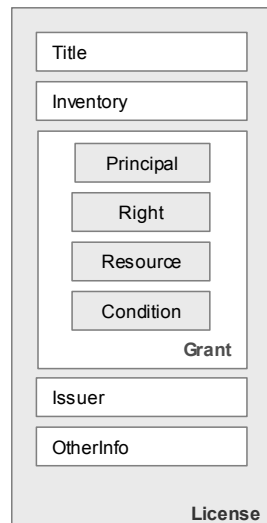


Figure REL License Structure

The most important concept within a license is the grant that conveys to a particular principal the sanction to exercise some identified right against some identified resource, possibly subject to the need for some condition to be first fulfilled. A Grant is an XML structure that is at the heart of the rights management and authorization policy semantics that REL is designed to express.

A grant is formed by four elements, a Principal that represents the unique identification of an entity involved in the granting or exercising of Rights. A Right that specifies an action or activity that a Principal may perform on, or using, some associated target Resource. A Resource that represents the object against which the Principal of a Grant has the Right to perform. The use of a digital resource in a Grant provides a means by which a sequence of digital bits can be identified within the Grant. The Condition element represents grammatical terms, conditions and obligations that a Principal must satisfy before it may take advantage of an authorization conveyed to it in a Grant. The issuer element that may contain two pieces of information, a set of issuer-specific details about the circumstances under which he issues the license, and an identification of the issuer, possibly coupled with a digital signature for the license. The optional issuer-specific details are found in the details element of the issuer. These details optionally include any of the following information the specific date and time at which this issuer claims to have effected his issuance of the license and an

indication of the mechanism or mechanisms by which the Issuer of the license will, if he later Revokes it, post notice of such revocation. When checking for revocation, REL processing systems may choose to use any one of the identified mechanisms, that is, they are all considered equally authoritative as to the revocation status of the issuance of the License.

The structure of a REL license is the one described if it is in clear text, but a REL license can contain only an encryptedLicense element if the license is encrypted. The encryptedLicense element provides a mechanism by which the contents of a License may be encrypted and so hidden from view from inappropriate parties. This mechanism makes straightforward use of XML Encryption Syntax and Processing (XML Encryption). Specifically, the XML content model of a License is a choice between a sequence containing the elements previously described in this section and an encryptedLicense element that represents the encryption of the contents of the License element.

The principals, rights, resources and conditions of the REL are organized in three main groups. The first one, the Core specifies structural elements and types and how are they related. The standard extension and the multimedia extension specifies standard or multimedia principals, rights, resources and conditions.

At the heart of REL is the REL Core Schema whose elements and types define the core structural and validation semantics that comprises the essence of the specification. The REL Core Schema includes different elements and types organised in four main groups:

- Principals: Within REL, instances of the type Principal represent the unique identification of an entity involved in the granting or exercising of rights. They represent the subject that is permitted to carry out the action involved in exercising the Right. The principal element and its type are conceptually abstracts. Then, it does not indicate how a particular principal is actually identified and authenticated. Rather, this is carried out in types that are derivations of Principal. Such derived types may be defined in extensions to REL in order to provide, for example, a means by which Principals who are authenticated using some proprietary logon mechanism may be granted certain Rights using the REL License mechanism.

There are derivations that are important and central enough to be defined within the REL core itself:

- allPrincipals: Structurally, an AllPrincipals Principal is a simple container of Principals. Semantically, an AllPrincipals represents the logical conjunct of the Principals represented by all of its children.
- keyHolder: Instances of a KeyHolder Principal represent entities which are identified by their possession of a certain cryptographic key. For example, using a KeyHolder, a Principal that uses public-key cryptography may be conceptually identified as that Principal which possesses the private key that corresponds to this-here public key.
- Rights: Within REL, instances of the type Right represent a verb that a Principal may be authorized to carry out under the authority conveyed by some authorized Grant. Typically, a Right specifies an action or activity that a Principal may perform on or using some associated target Resource. The semantic specification of each different particular kind of Right should indicate which kinds of Resource if any may be legally used in authorized Grants containing that Right. The element right and its type are conceptually abstract. Therefore, the type Right itself does not indicate any actual action or activity that may be carried out. Rather, such actions or activities are to be defined in types that are derivations of Right. Such derived types will commonly be defined in extensions to REL. However, the following rights are related to the domain of the REL core itself:
- issue: When an Issue element is used as the right in an authorized grant, it is required that resource against which the right is applied in fact be a grant or grantGroup. The grant then conveys the authorization for the principal to issue the resource.

At the instant a License is issued, the issue right must be held by the issuer of the License with respect to all the grants and grantGroups directly authorized therein.

- obtain: When an obtain element is used as the right in an authorized grant, the resource must be present and be a grant or a grantGroup. The use of the obtain right can be conceptualized as an offer or advertisement for the sale of the contained grant
- possessProperty: The possessProperty right represents the right for the associated principal to claim ownership of a particular characteristic, which is listed as the resource associated with this Right.
- revoke: The authorized act of exercising the revoke right by a principal effects a retraction of a dsig:Signature that was previously issued and thus accomplishes a withdrawal of any authorization conveyed by that dsig:Signature.
- Resources: An instance of type resource represents the direct object against which the subject principal of a grant has the right to perform some verb. The actual element resource and its type are conceptually abstracts. That is, the type resource itself does not indicate any actual object against which a Right may be carried out. Rather, such target objects are to be defined in types that are derivations of Resource. Such derived types will commonly be defined in extensions to REL. The relevant resources defined within the REL core:
- digitalResource: Use of a digitalResource resource in a grant provides a means by which an arbitrary sequence of digital bits can be identified as being the target object of relevance within the grant. Specifically, such bits are not required to be character strings that conform to the XML specification, but may be arbitrary binary data. The means by which this is accomplished breaks down into several cases. For example, the bits are to be physically present within the digitalResource or the bits are to be physically located at some external location (e.g. in a Web site).
- propertyAbstract: An instance of type propertyAbstract represents some sort of property that can be possessed by principals via possessProperty right.
- Conditions: Within REL, instances of the type Condition represent grammatical terms and conditions that a Principal must satisfy before it may take advantage of an authorization conveyed to it in a grant containing the condition instance. The semantic specification of each different particular kind of condition must indicate the details of the terms, conditions, and obligations that use of the Condition actually imposes. When these requirements are fulfilled, the Condition is said to be satisfied.

The actual element condition and its type are conceptually abstracts. That is, the type Condition itself does not indicate the imposition of any actual term or condition. Rather, such terms and conditions are to be defined in types that are derivations of Condition. Such derived types will commonly be defined in extensions to REL. The conditions defined within the REL core that we consider relevant to detail:

 - AllConditions: Structurally, an allConditions is a simple container of conditions. Semantically, the allConditions represents a logical conjunct of the conditions represented by all of its children.
 - validityInterval: A ValidityInterval condition indicates a contiguous, unbroken interval of time. The semantics of the condition expressed is that the interval of the exercise of a right to which a validityInterval is applied must lie wholly within this interval. The delineation of the interval is expressed by the presence, as children of the condition, of up to two specific fixed time instants:
 - notBefore element, of type xsd:dateTime, indicates the inclusive instant in time at which the interval begins. If absent, the interval is considered to begin at an instant infinitely distant in the past
 - notAfter element, also of type xsd:dateTime, indicates the inclusive instant in time at which the interval ends. If absent, the interval is considered to end at an instant infinitely distant in the future.

The Standard Extension schema defines terms to extend the usability of the Core Schema, some of them are:

- Right Extensions: Right Uri.
- Resource Extensions: Property Extensions and Revocable.
- Condition Extensions: Stateful Condition, State Reference Value Pattern, Exercise Limit Condition, Transfer Control Condition, Seek Approval Condition, Track Report Condition, Track Query Condition, Validity Interval Floating Condition, Validity Time Metered Condition, Validity Time Periodic Condition, Fee Condition and Territory Condition.
- Payment Abstract and its Extensions: Payment Abstract, Rate, Payment Flat, Payment Metered, Payment per Interval, Payment per Use, Best Price Under, Call for Price and Markup.
- Service Description: WSDL and UDDI
- Country, Region and Currency Qualified Names: Namespace URI Structure, Country Qualified Names, Region Qualified Names and Currency Qualified Names.
- Matches XPath Function: Regular Expression Syntax and Flags.

The REL Multimedia Extension expands the Core Schema by specifying terms that relate to digital works. Specifically describes rights, conditions and metadata for digital works, that includes:

- Rights: Modify, Enlarge, Reduce, Move, Adapt, Extract, Embed, Play, Print, Execute, Install, Uninstall and Delete.
- Resources: Digital Item Resources.
- Conditions: Resource Attribute Conditions, Digital Item Conditions, Marking Conditions, Security Conditions and Transactional Conditions.
- Resource Attribute Set Definitions: Complement, Intersection, Set and Union.

18.1.2 ODRL

The ODRL [3] is a proposed language for the DRM community for the standardisation of expressing rights information over content. The ODRL is intended to provide flexible and interoperable mechanisms to support transparent and innovative use of digital resources in publishing, distributing and consuming of electronic publications, music, audio, movies, digital images, learning objects, computer software and other creations in digital form.

Using ODRL it is possible to specify, for a digital resource (music work, content, service, or software application), which is allowed to use that resource, the rights available to them and the terms, conditions or restrictions necessary to exercise those rights on the resource. The ODRL function is to express rights granted by some parties for specific resources and the conditions under which those rights apply.

ODRL is based on an extensible model for rights expressions, which involves three core entities and their relationships: **Party** (identifies an entity such as the person, organisation, or device to whom rights are granted), **Right** (includes permissions, which can then contain constraints, requirements, and conditions) and **Asset** (includes any physical or digital content).

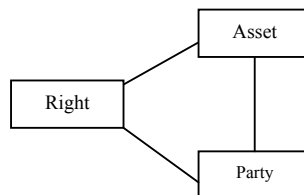


Figure Core elements of ODRL

18.1.3 OMA DRM Rights Expression Language

OMA (Open Mobile Alliance) has developed the OMA DRM Rights Expression Language versions [4] based on ODRL [3].

Rights are the collection of permissions and constraints defining under which circumstances access is granted to DRM Content. The structure of the rights expression language enables the following functionality:

1. Metadata such as version and content ID
2. The actual rights specification consisting of
 - a. Linking to and providing protection information for the content, and
 - b. Specification of usage rights and constraints

Models are used to group rights elements according to their functionality, and thus enable concise definition of elements and their semantics. The following models are used throughout this specification:

- Foundation model: constitutes the basis for rights. It contains the rights element bringing together meta information and agreement information. The foundation model serves as the starting point for incorporating the agreement model and the context model.
- Agreement model: expresses the Rights that are granted over an DRM Content. It consists of the agreement element connecting a set of Rights with the corresponding DRM Content specified with the asset element. The agreement model incorporates the permission model and the security model
- Context model: provides Meta information about the rights. It augments the foundation model, the agreement model, and the constraint model by expressing additional information.
- Permission model: augments the agreement model. It facilitates the expression of permissions over assets by specifying the access granted to a device. The permission model incorporates the constraint model allowing fine-grained consumption control of DRM Content. The set of permissions comprises play, display, execute, print, and export. Usage of the DRM Content **MUST** only be granted according to the permissions explicitly specified by the corresponding Rights Object(s). A permission that does not contain a constraint child element is unconstrained and access according to the respective permission element(s) **MUST** be granted. Note that the REL only specifies consumption and export rights and not management rights, e.g., install, uninstall, delete, or distribution rights. This is made possible by the separation of DRM Content and Rights Objects (although DRM Content and Rights Objects may be delivered together) freeing the REL from unnecessary complexity and overhead. Content can be stored; however, it can only be accessed if a corresponding Rights Object is available. Similarly, encrypted content can be super-distributed without unnecessarily complicating the REL; no separate distribution permissions are necessary, since DRM Content without the decryption key is of no value. The DRM Agent **MUST** ignore unknown or unsupported permission elements. The DRM Agent **MUST NOT** grant alternative, not explicitly specified rights to access Content instead. Known and supported permission elements defined by the same Rights Object **MUST** remain unaffected and the DRM Agent **MUST** grant access according to those. A Permission that is not granted due to unknown or unsupported constraints (section 5.5) **MUST NOT** affect the granting of other permissions.
- Constraint model: enhances the permission model by providing fine-grained consumption control of content. Constraints are associated with one permission element at a time. For a permission to be granted all its constraints **MUST** be fulfilled. If a constraint is not understood or cannot be enforced by the consuming device the parent permission is invalid and must not be granted. If present, a constraint element should contain at least one of its child elements. If a constraint element does not contain any constraints such as count, datetime, etc. it is unconstrained, and a DRM Agent must grant unconstrained access according to the permission containing such an unconstrained constraint element.
- Inheritance model: describes how a parent Rights Object can specify Permissions and Constraints for one or more pieces of DRM Content each governed by a child Rights Object, using a limited subset of the ODRL inheritance model. The DRM Agent must not accept parent child Rights Objects constellations with more than one level of inheritance (i.e. parent-child). In other words, a parent Rights Object must not inherit Permissions and Constraints from another Rights Object.
- Security model: Security constitutes an important part of a DRM system. OMA DRM 2.0 provides confidentiality for the CEK of Rights Objects, integrity of the association between Rights Objects

and DRM Content and Rights Object integrity and authenticity. The ODRL security model, which forms the basis for the security model of this specification, is based on XMLENC [5] and XMLSIG [6].

18.1.4 PAR and Licenses: The problems

Different systems define different rights expression languages. If we want to make them interoperable, or simply work with different kinds of devices we may be obliged to adapt rights expressions.

Moreover, changing versions of specifications make difficult to implement any kind of software to solve this issue.

18.1.5 PAR and Licenses: Work performed

In order to be able to transform rights expressions expressed in different rights expression languages, the first step to take is to study their common points. We have studied in detail the relationship between MPEG-21 REL and ODRL, based on the OMA DRM REL, as described next.

18.1.5.1 OMA-based MPEG-21 REL DTD

In order to perform adaptation of rights between OMA REL and MPEG-21 REL, we propose an equivalent structure of the rights expression language of OMA DRM v1.0, but defined as a subset of MPEG-21 REL, and not as a subset of ODRL. This research work was presented in [7], [8].

The main concept in this equivalent structure is the `r:license` element that conceptually is a container of a `r:grant` or a `r:grantgroup` elements. A `r:grant` element conveys to someone the sanction to exercise a right (`mx:play`, `mx:execute` or `mx:print` are considered) against a resource. In this case, a resource is represented with the `r:digitalResource` element.

A `r:license` element also contains a `r:otherinfo` element that it is useful to include OMA DRM REL v1.0 information not considered by MPEG-21 REL, and it provides meta information about the rights.

The conditions considered are: `sx:exerciseLimit` that specifies the number of allowed exercises of a certain right, `validityInterval` that specifies an interval of time within which a right can be exercised and `validityIntervalDurationPattern` that specifies a period of time within which a right can be exercised.

```
<!ELEMENT r:license ( (r:grantgroup|r:grant), r:otherinfo? )>
<!ATTLIST r:license
  xmlns:r CDATA #FIXED
    "urn:mpeg:mpeg21:2003:01-REL-R-NS"
  xmlns:dsig CDATA #FIXED
    "http://www.w3.org/2000/09/xmldsig#"
  xmlns:mx CDATA #FIXED
    "urn:mpeg:mpeg21:2003:01-REL-MX-NS"
  xmlns:sx CDATA #FIXED
    "urn:mpeg:mpeg21:2003:01-REL-SX-NS">

<!ELEMENT r:grantgroup (r:grant+)>
<!ELEMENT r:grant
  ((mx:play|mx:execute|mx:print)?,
   r:digitalResource,
   r:allConditions?)>

<!ELEMENT mx:play EMPTY>
<!ELEMENT mx:execute EMPTY>
<!ELEMENT mx:print EMPTY>

<!ELEMENT r:digitalResource (r:nonSecureIndirect) >
<!ELEMENT r:nonSecureIndirect EMPTY>
```

```

<!ATTLIST r:nonSecureIndirect URI CDATA #IMPLIED>

<!ELEMENT r:allConditions
  (sx:exerciseLimit?,
   validityInterval?,
   validityIntervalDurationPattern?)>
<!ELEMENT sx:exerciseLimit (sx:count)>
<!ELEMENT sx:count (#PCDATA)>
<!ELEMENT r:validityInterval (r:notBefore?, r:notAfter?)>
<!ELEMENT r:notBefore (#PCDATA)>
<!ELEMENT r:notAfter (#PCDATA)>
<!ELEMENT sx:validityIntervalDurationPattern (sx:duration)>
<!ELEMENT sx:duration (#PCDATA)>
<!ELEMENT r:otherinfo (version?, KeyValue?)>
<!ELEMENT version (#PCDATA)>
<!ELEMENT KeyValue (#PCDATA)>

```

Figure OMA-based MPEG-21 REL DTD

18.1.5.2 Interoperability between MPEG-21 REL and OMA DRM REL v2.0

In this section, we introduce different tables with XML equivalences, between the OMA DRM REL v2.0 and the MPEG-21 REL subset that will lead us to achieve interoperability between the MPEG-21 REL subset for the mobile domain and OMA DRM REL v2.0 specification. This work has been presented in [9], [10].

We use different models to group the XML equivalences according to their functionality and license structure. The models described in this section are: Basic equivalences, Rights and Conditions. They define the elements that form the subset of MPEG-21 REL that fulfils OMA DRM REL v2.0 specification.

Basic equivalences

The basic equivalences constitute the basis for licenses and include the necessary elements in any license. The OMA DRM REL <rights> and <asset> elements are represented with the MPEG 21 REL <license> and <digitalResource> elements. The OMA <context> element provides meta information about the rights, and is represented with the MPEG-21 <otherinfo> element.

Table Basic model

OMA DRM REL v2.0	OMA-based MPEG-21 REL
<o-ex:rights>	<r:license>
<o-ex:context> <o-dd:version>2.0</o-dd:version> <o-dd:uid>RightsObjectID</o-dd:uid> </o-ex:context>	<r:otherinfo> <version>1.0</version> <uid>RightsObjectID</uid> </r:otherinfo>
<o-ex:asset> <o-ex:context> <o-dd:uid>ContentID</o-dd:uid> </o-ex:context> </o-ex:asset>	<r:digitalResource> <r:nonSecureIndirect URI='ContentID' /> </r:digitalResource>

Rights

In the following table are introduced the MPEG-21 REL rights equivalent to the rights specified in OMA DRM REL. The <display> and <play> elements are represented with the <play> element, the <export - move> element with the <move> element and the <export - copy> element with the <adapt> element and <prohibitedAttributeChanges> elements.

Table Rights model

OMA DRM REL v2.0	OMA-based MPEG-21 REL
<o-dd:display/>	<mx:play />

<o-dd:play/>	<mx:play />
<o-dd:execute/>	<mx:execute />
<o-dd:print/>	<mx:print />
<oma-dd:export oma-dd:mode="move"> <o-ex:constraint> <oma-dd:system> <o-ex:context> <o-dd:version> 1.0 </o-dd:version> <o-dd:uid> XYZ </o-dd:uid> </o-ex:context> </oma-dd:system> </o-ex:constraint> </oma-dd:export>	<mx:move/> <r:digitalResource> <r:nonSecureIndirect URI="ContentID"/> </r:digitalResource> <r:allConditions> <mx:destination> <r:keyHolder> <r:info> <version>1.0</version> <uid>XYZ</uid> </r:info> </r:keyHolder> </mx:destination> </r:allConditions>
<oma-dd:export oma-dd:mode="copy"> <o-ex:constraint> <oma-dd:system> <o-ex:context> <o-dd:version> 1.0 </o-dd:version> <o-dd:uid> XYZ </o-dd:uid> </o-ex:context> </oma-dd:system> </o-ex:constraint> </oma-dd:export>	<mx:adapt/> <r:digitalResource> <r:nonSecureIndirect URI="ContentID1"/> </r:digitalResource> <mx:prohibitedAttributeChanges> <set definition= "urn:mpeg:mpeg21:2003:01- RDD-NS:2346"/> <set definition= "urn:mpeg:mpeg21:2003:01- RDD-NS:2347"/> </mx:prohibitedAttributeChanges> <r:keyHolder> <version>1.0</version> <uid>XYZ</uid> </r:keyHolder>

Conditions

The following tables introduce different kinds of MPEG-21 REL conditions equivalent to the ones specified in OMA DRM REL.

One kind of conditions represented are time conditions. The <datetime> element represented in MPEG-21 REL with the <validityInterval> element specifies an interval of time within which a right can be exercised. The <interval> represented in MPEG-21 REL with the <validityIntervalDurationPattern> element specifies a period of time within which a right can be exercised. Finally, the <accumulated> element represented in MPEG-21 REL with the <validityTimeMetered> specifies the maximum period of metered usage time during which the rights can be exercised.

Table Time conditions model

OMA DRM REL v2.0	OMA-based MPEG-21 REL
<o-ex:constraint> <o-dd:datetime> <o-dd:start>... </o-dd:start> <o-dd:end>... </o-dd:end> </o-dd:datetime>	<r:allConditions> <r:validityInterval> <r:notBefore>...</r:notBefore> <r:notAfter>...</r:notAfter> </r:validityInterval> </r:allConditions>

</o-ex:constraint>	
<o-ex:constraint> <o-dd:interval> </o-dd:interval> </o-ex:constraint>	<r:allConditions> <sx:validityIntervalDurationPattern> <sx:duration>...</sx:duration> </sx:validityIntervalDurationPattern> </r:allConditions>
<o-ex:constraint> <o-dd:accumulated> PT10H </o-dd:accumulated> </o-ex:constraint>	<r:allConditions> <sx:validityTimeMetered> <sx:duration>PT10H</sx:duration> </sx:validityTimeMetered> </r:allConditions>

The next table introduces the rest of MPEG-21 REL conditions considered in the mobile subset we are defining equivalent to the ones specified in OMA DRM REL. The <count> element represented in MPEG-21 REL with the <exerciseLimit> element specifies the number of allowed exercises. The <timed-count> element specify the number of times a permission may be granted over an asset or resource, with the addition of an optional timer attribute. This timer attribute specifies the number of seconds after which the count state can be reduced. As the timer attribute is not specified in MPEG-21 REL, we have defined the <exerciseLimitTime>, that consist of <count> and <duration> elements. The <individual> represented in MPEG-21 REL with the <keyHolder> element specifies the individual to which content is bound. The <system> represented in MPEG-21 REL with the <renderer> element specifies the target system to which DRM Content and Rights Objects can be exported.

Table Other conditions model

OMA DRM REL v2.0	OMA-based MPEG-21 REL
<o-ex:constraint> <o-dd:count> 1 </o-dd:count> </o-ex:constraint>	<sx:exerciseLimit> <sx:count> 1</sx:count> </sx:exerciseLimit>
<o-ex:constraint> <o-dd:timed-count timer="30"> 1 </o-dd:timed-count> </o-ex:constraint>	<r:otherinfo> <exerciseLimitTime> <sx:count> 1</sx:count> <sx:duration> 30 </sx:duration> </exerciseLimit> </r:otherinfo>
	<r:grant licensePartId="Asset-1"> <r:allConditions> <sx:exerciseLimit> <sx:count> 1</sx:count> </sx:exerciseLimit> </r:allConditions> </r:grant licensePartId="Asset-1"> <r:otherinfo> <grant licensePartIdRef="Asset-1"> <exerciseLimitDuration> 30 </exerciseLimitDuration> </grant> </r:otherinfo>
	<sx:exerciseLimit> <r:serviceReference licensePartIdRef="externalService"/> <sx:count> 1</sx:count> </sx:exerciseLimit>
<o-ex:constraint> <o-dd:individual> <o-ex:context> <odd:uid> XYZ	<r:grant> <r:keyHolder> <r:info> <uid>XYZ</uid>

</odd:uid> </o-ex:context> </o-dd: individual> </o-ex:constraint>	</r:info> </r:keyHolder> </r:grant>
<o-ex:constraint> <oma-dd:system> <o-ex:context> <odd:uid> XYZ </odd:uid> </o-ex:context> </oma-dd system> </o-ex:constraint>	<mx:renderer> <r:keyHolder> <r:info> <uid>XYZ</uid> </r:info> </r:keyHolder> </mx:renderer>

19 Transcoding Metadata (UNIVLEEDS)

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 and processing the adaptation
- 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* are used to parse a given piece of XML data.

The AxMetadata Model has been developed to provide the functionality for the transcoding of AXMEDIS metadata. Current functionalities include:

- Loading, saving, writing to string and parsing an XML document using the Xerces SAX2 implementation
- Changing the Element name and Uri without validation
- Changing the Element values (currently there is no validation with respect to date and time elements)
- Changing the Element Attribute values name, Uri and values without validation

Demonstrating the current functionality can be viewed in the Metadata Editor (see deliverable DE4.1.1 content modelling and managing).

19.1 Technical Details

reference to the AXFW location of the demonstrator	https://cvs.axmedis.org/repos/Framework/source/metamodel/	https://cvs.axmedis.org/repos/Framework/source/metadatamodel/
List of libraries used	wxWindows v-2.4.2 Xerces-c++ v-2.6.0 Xalan-c++ v-1.9	wxWindows v-2.4.2 Xerces-c++ v-2.6.0 Xalan-c++ v-1.9
References to other major components needed	xerces-c_2_6D.dll	
Problems not solved	<ul style="list-style-type: none"> • Adding and deleting elements with validation with schemas • Adding and deleting attributes with validation with schemas 	<ul style="list-style-type: none"> • •
Configuration and execution context		
Programming language	Visual C++	

19.2 Metadata: State of the art

The transcoding of metadata has two levels of adaptation that need to be considered. The first is transcoding the metadata to represent the adaptation applied to the object such as scaling an object, reformatting etc. This requires methods to adapt the metadata to represent the object in its new state. The second transcoding depends on the devices to use the object. In this scenario, for small memory devices, the metadata may need to be scaled and tables for this process are required to be devised.

There are no specific tools for the adaptation of metadata in this manner, however libraries are being developed such as Xerces, Xalan, and Expat for the parsing and handling of XML documents. This is an ongoing research and development process. Xerces and Xalan provide libraries in both C++ and Java and initial support of Schemas. The Xerces libraries provide methods for the programmatic generation and validation of XML and customisable error handlers (see AXMEDIS FW Specifications in DE 3.1.2d). Xalan libraries are an XSLT processor for transforming XML documents to HTML or another XML document types. These methods can be utilized in the adaptation of the metadata for not only transcoding the format changes info in the AxInfo elements but in the scaling of the metadata for different clients.

Validation is an important issue that needs to be considered where adapted metadata is valid after transcoding. Due to the complex nature of AXMEDIS object, there may be one or more metadata sections with different schemas, including Dublin Core, AXInfo and Mpeg7 and the next development method is to devise methods to solve these problems.

19.3 Metadata: The problems

Some of the desired functionality has not been resolved:

- Adding and deletion of elements with validation to keep the metadata as valid XML using the schemas.
- Adding and deletion of element attributes with schema validation.
- Moving elements to another part of the XML structure, this may or may not need to reallocate children elements.

19.4 Metadata: Work performed

The following classes have been developed for the adaptation of XML AXMEDIS Metadata providing functionalities for loading, saving, parsing, manipulating and writing XML files.

The implementation was performed using C++ MSVC7 and supported by wxWidgets ver. 2.4.2 and XERCES 2.6.0 libraries. For the transcoding of metadata, the following classes were developed for the first prototype to adapt generic XML metadata elements.

19.4.1 AxMetadata Class

This class is the main functionality for loading of an XML file or XML string, parsing the XML into a list of elements, saving to an XML file and writing to an XML string.

19.4.2 AxmetadataElement Class

This class is for the XML Element data including the setting and retrieving of element name, value and Uri. This class also allows the definition of parent Element and the child Elements to manipulate the structure of the XML. The last functionality is the adding, retrieving and deletion of the Attributes for the XML Element.

19.4.3 AxMetadataAttribute Class

This class is for the attribute data including the setting and retrieving of an attribute name, value and Uri.

19.4.4 AxMetadataSAXImplementation Class

This class provides the functionality for defining the parser. The parsing rules are specified by the startElement(...) and endElement(..) functions on how to process the AXMEDIS metadata XML Elements. This class also processes the error handling using the Xerces error handling interface. During the parsing if an error is found, an error value is returned and subject to the level of error i.e. warning, error, fatal error, the appropriate message with the type of error and error information such as element or a pointer to the line position if the XML was opened in a text reader.

Future implementation includes increasing the adaptation functionalities. This is achieved by adding the following functionality:

- Adding and deleting XML elements,
- Adding and deleting attributes to an element

- Reposition the element to restructure the XML tree structure by defining a new parent element.
- Copying elements to a new position in the XML structure

20 Workflow Management and database (IRC)

20.1 Technical Details

reference to the AXFW location of the demonstrator	https://cvs.axmedis.org/repos/Framework/source/Workflow
List of libraries used	Pythonlib & Zope server
References to other major components needed	Openflow
Problems not solved	<ul style="list-style-type: none"> • •
Configuration and execution context	
Programming language	Python

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

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

It is possible for the AXMEDIS workflow management system to support inter-factory workflow. An example of this would be collaborating content producers who work jointly on common objects. Content Factory A would create an object, then Factory B would perform some activities to add value to the object, then returning it to Factory A for completion. Conceptually, this process is identical to the normal, intra-factory scenario where activities are carried out in one content factory. In the inter-factory scenario, the collaborating factories needs to establish an agreed workflow in order to manage their division of work productively and efficiently. This workflow agreement can be modelled in the same manner as a conventional intra-factory workflow.

We have not proposed for centralised single server architecture; rather each partner will have their workflow running with their part of the project workflow definition. The transitions resulting into change of partner is defined in the workflow to reflect the collaborative workflow logic as agreed between the collaborators. The waiting period for the factories can be defined as “Idle/wait” activities within the workflow which are completed upon receiving the workitem from the external factory. For example when the workitem is handed to Factory B from Factory A, as defined in the workflow, Factory A will then start an “Idle/wait” activity which will end upon receiving the workitem back from Factory B.

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

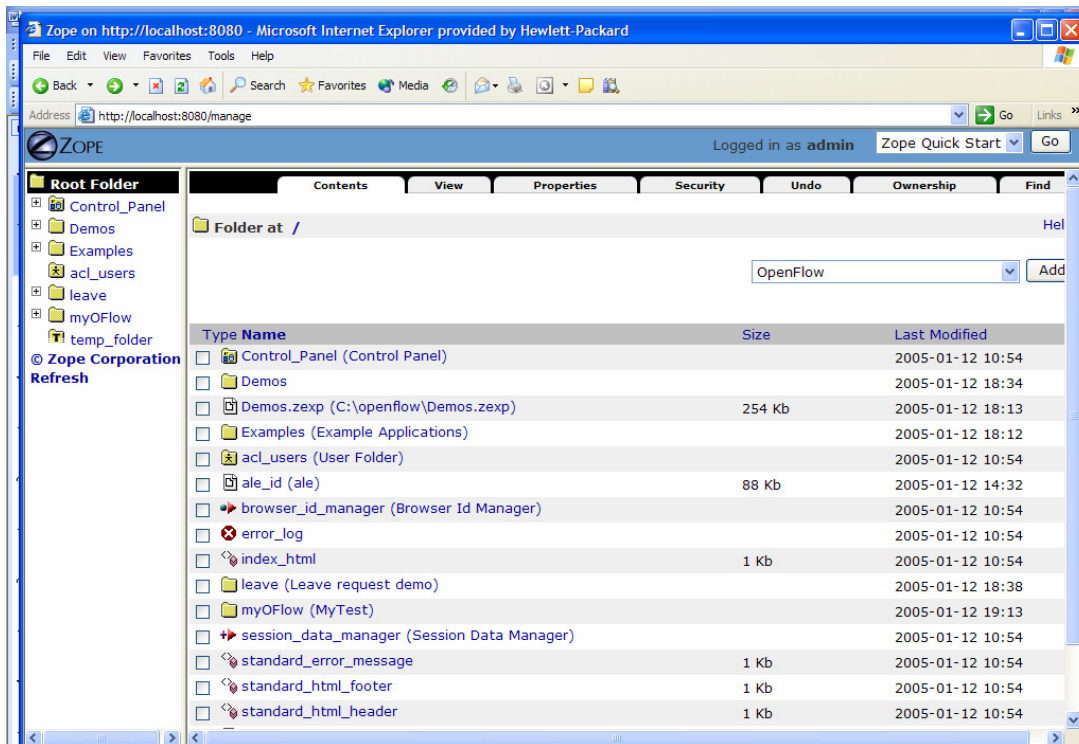
This dynamic visibility would not be possible if separate WFMS tools were employed in each factory and the only communication available were some embedded historic metadata within objects passed between factories.

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

20.2 Writing and describing workflow, harmonising AXMEDIS tools

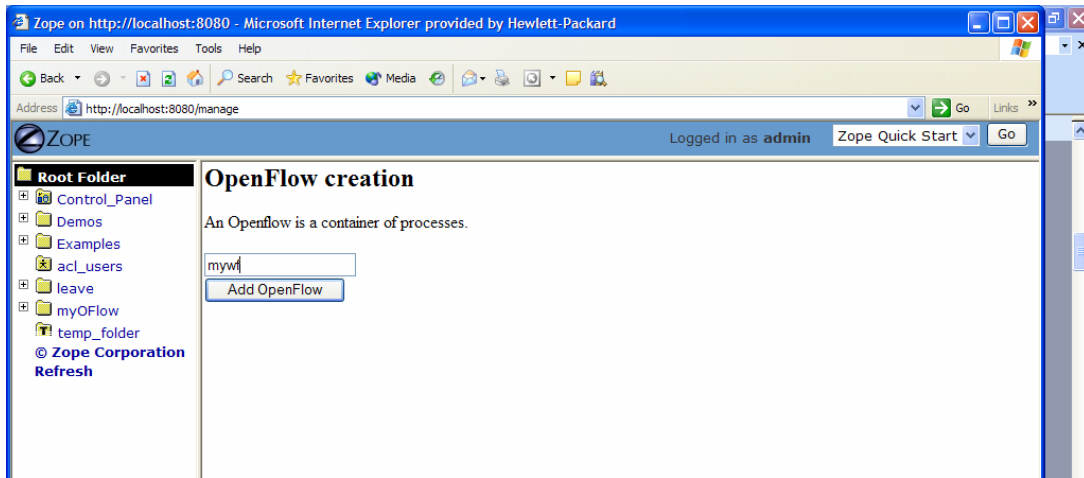
Openflow runs on the Zope platform which is managed through the “Zope Management Interface” using industry standard browsers, typically by logging on as the administrator (admin) at URL <http://localhost:8080/manage>. The screen shot below shows an example of this management interface.

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



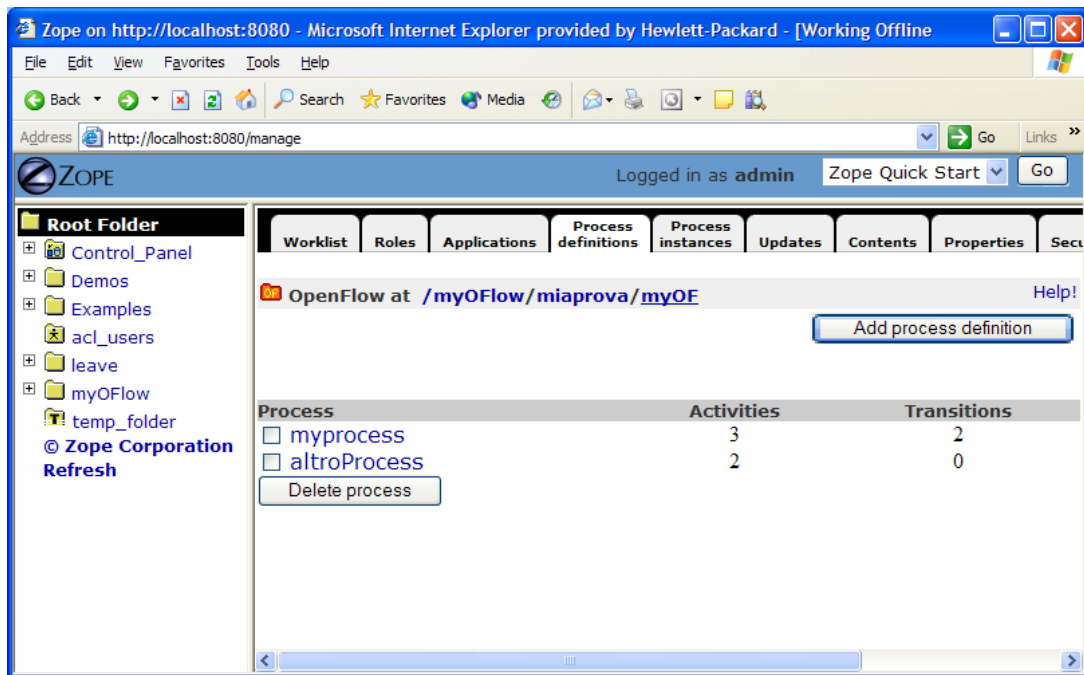
Adding an OpenFlow container through the Zope Management Interface

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



Creating the OpenFlow container

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



The process definition tab

Process creation

Create a new process:

Id:

Title:

Description:

☒ Create standard Begin and End activities

Priority: (0 = lowest priority)

Creating a new Process definition

Activities

Activity	Kind	JoinSplit	Application name	Push Application	Start mode	Finish mode	Subflow process
<input type="checkbox"/> Begin	standard	and			Manual	Manual	
<input type="checkbox"/> End	standard	and			Manual	Manual	
<input type="checkbox"/> myactiv	standard	and		myapp	Automatic	Manual	

Transitions

Transition	Condition	From	To
<input type="checkbox"/> Begin_myactiv	python:instance.some_property=='value'	Begin	myactiv
<input type="checkbox"/> myactiv_End	python:instance.some_property=='value'	myactiv	End

Management of activity and transitions of a process

ZOPE on http://localhost:8080 - Microsoft Internet Explorer provided by Hewlett-Packard - [Working Offline]

File Edit View Favorites Tools Help

Address http://localhost:8080/manage Go Links

ZOPE Logged in as admin ZOPE Quick Start Go

Root Folder

- Control_Pane
- Demos
- Examples
- acl_users
- leave
- myOFlow
- temp_folder

© ZOPE Corporat Refresh

Edit activity

Activity id myactiv

Title: test activity

Description: prova

General settings

Dummy ☐

Name:

Application ☒

Pushing application: myapp

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

☒ Automatic start

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

☐ Automatic finish

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

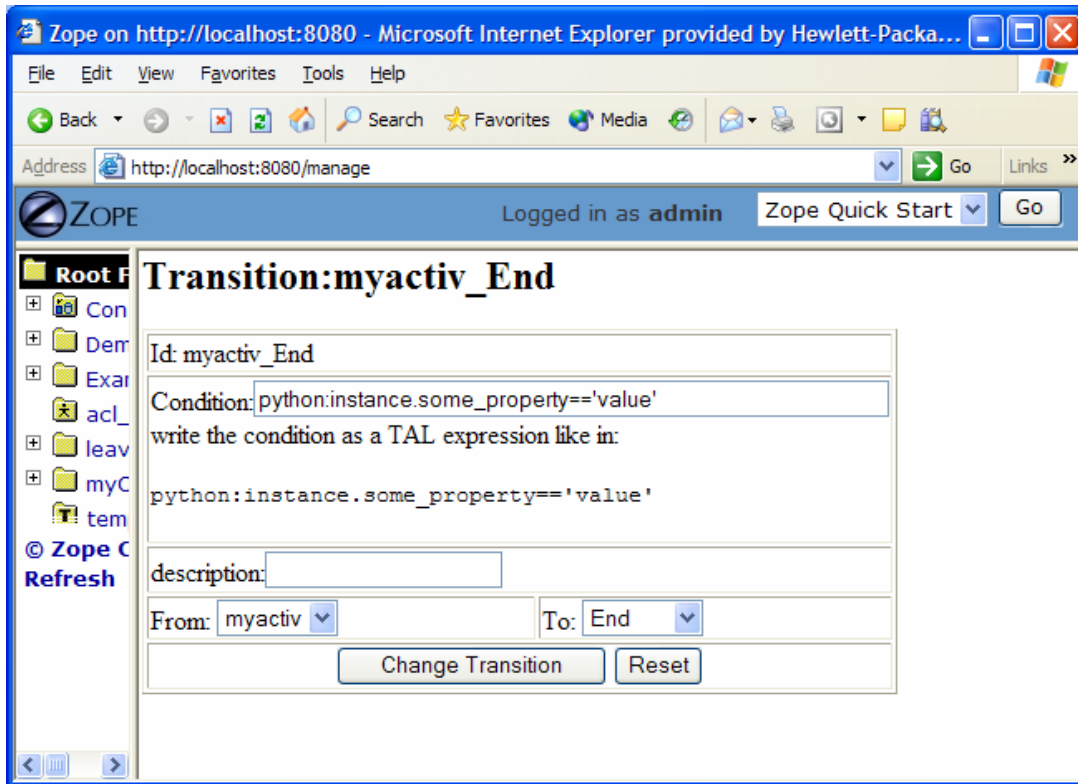
Subprocess ☐

Subflow:

Workitem handling Join kind: and Split kind: and

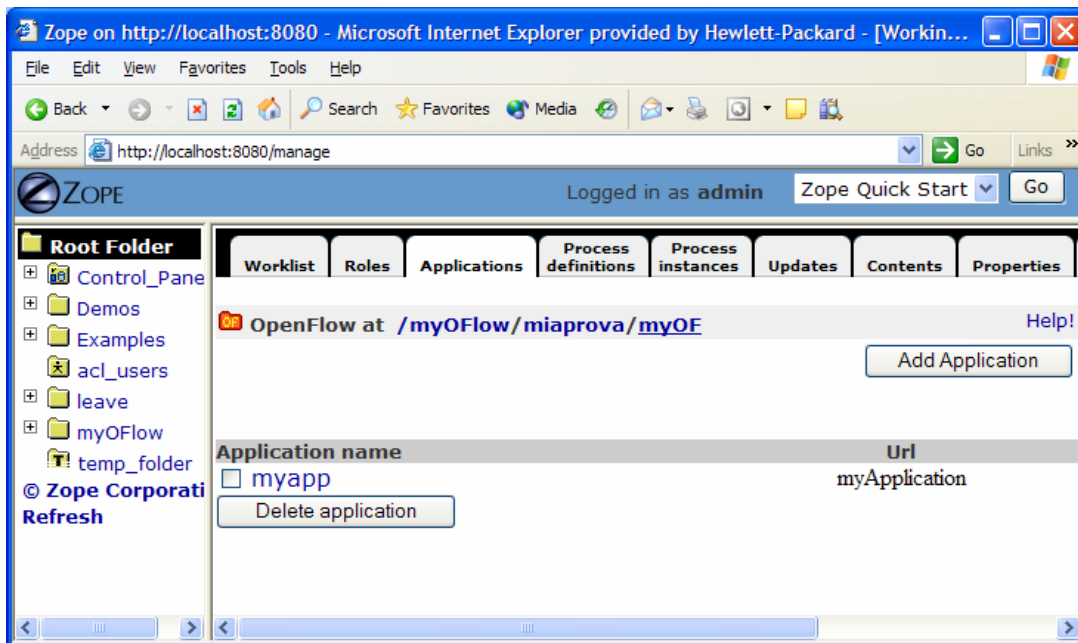
Change

Editing a process activity



Defining process transition and related conditions

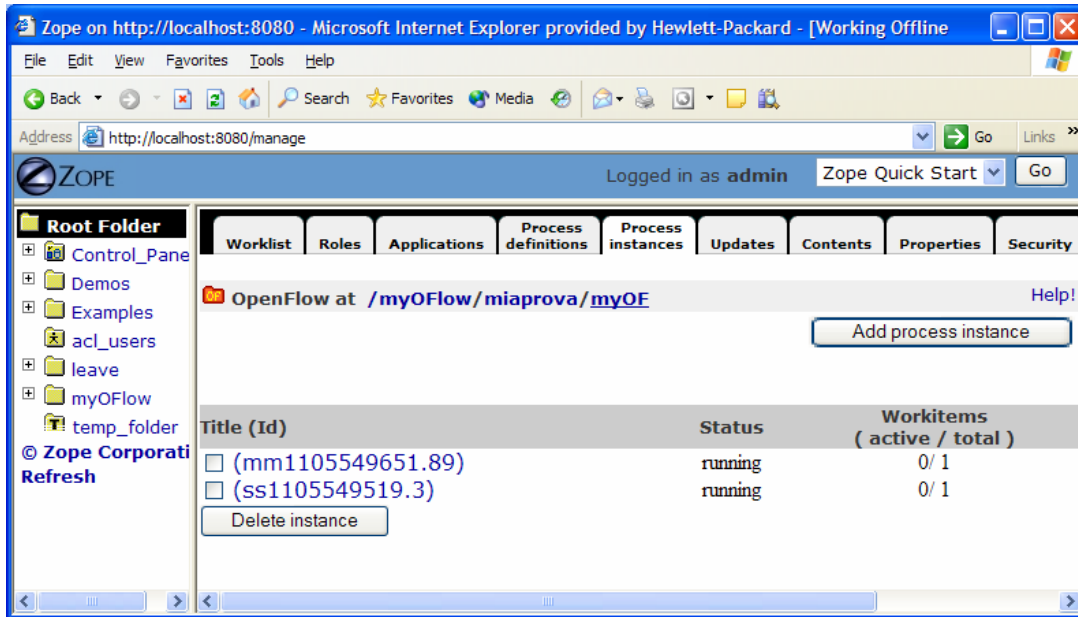
Applications associated to the activities are then specified selecting the Applications Tab.



: Defining process applications

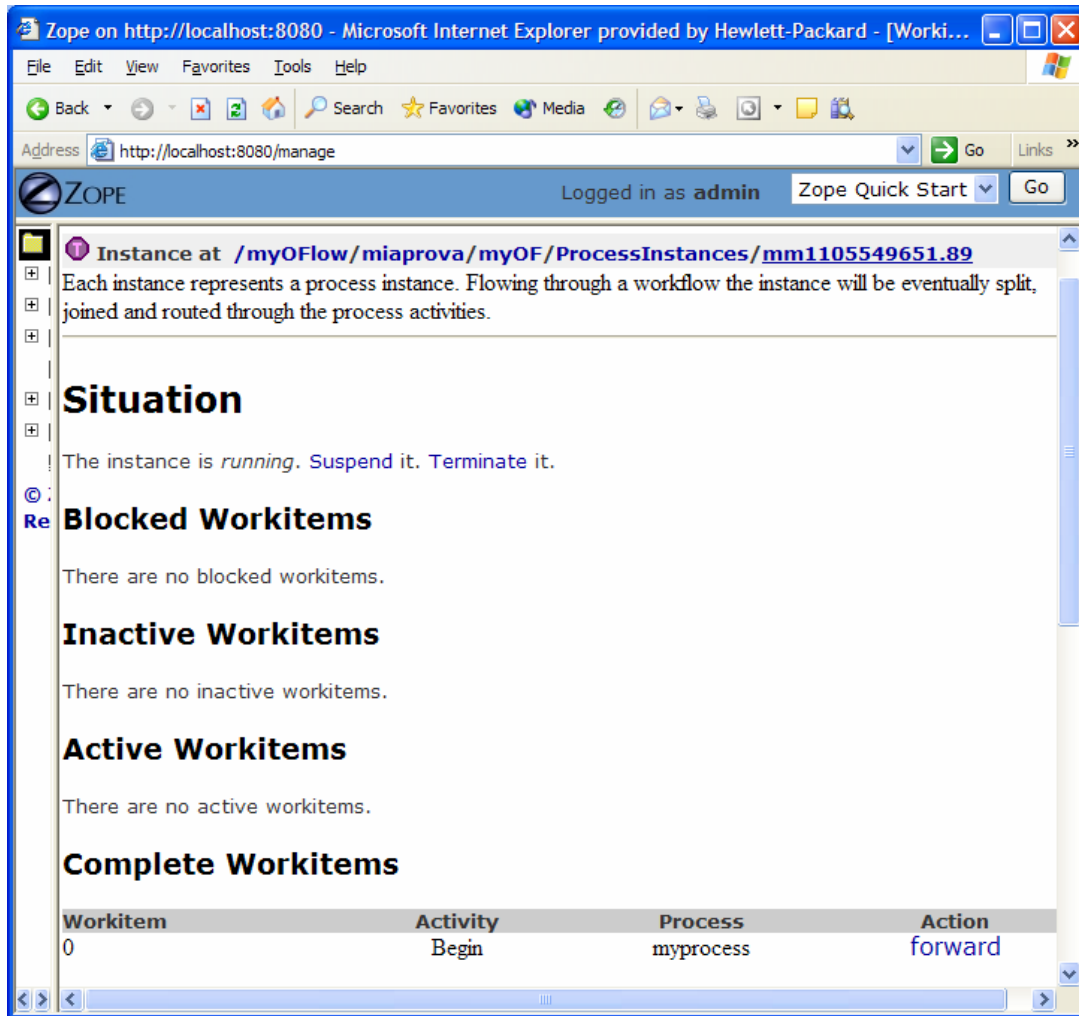
The users and roles are configured as Zope users and roles as access control list (acl_users).

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



Process instance management tab

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



Monitoring and management of a specific process instance

Process Example:

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

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

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

Leave request demo - Microsoft Internet Explorer provided by Hewlett-Packard

Address: http://localhost:8080/leave/leave_startform

Request for AXMEDIS object manipulation

Start date: 2005-01-19 End date: 2005-01-19

Type: Create new AXMEDIS object

Reason:

Tertius' AXMEDIS object manipulation form

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

Leave request demo - Microsoft Internet Explorer provided by Hewlett-Packard

Address: http://localhost:8080/leave/mywork

Worklist for socius

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

[To frontpage](#)

Socius' worklist and workitem activation

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

The screenshot shows a web browser window titled "Leave request demo - Microsoft Internet Explorer provided by He...". The address bar shows the URL "http://localhost:8080/leave/leave_checkstatus?instance_id=tertiu". The main content area is titled "Check request" and contains the following text:

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

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

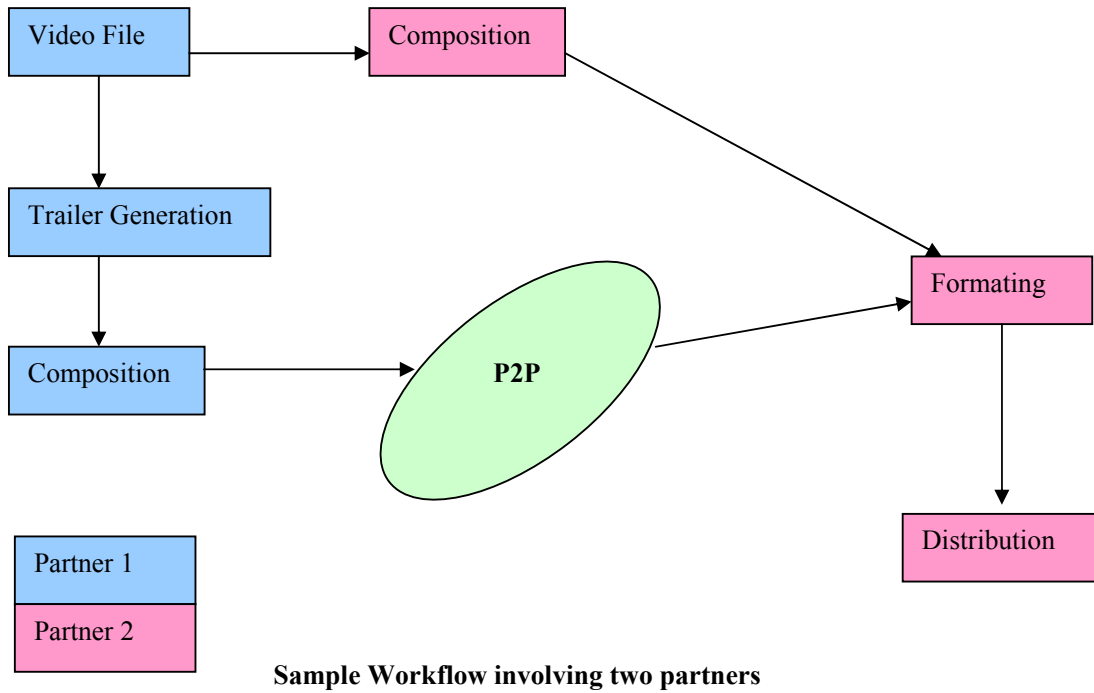
Start date 2005-01-19
End date 2005-01-19
Reason
Type Create new AXMEDIS object
Reason for denial:

Below the text is a large empty text area for input. At the bottom of the form are two buttons: "OK: Forward to next actor" and "Denied: Back to requester".

Socius' workitem execution and forwarding

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

The sample workflow that is being defined for demonstration purpose, will be a inter factory workflow utilising the important Axmedis tools like AXEPTTool, PnP Engine, Axmedis Editor, etc. The following diagram is the schematic of the sample workflow to be deployed for the demonstration. It includes two factories sharing some content using AXEPTTool. This workflow will be further refined to map to the actual low level atomic activities that are available within the Axmedis Framework.



The above workflow involves two partners. The partner 1 is responsible for generating the trailer from a selected video file and then publishes over P2P network. The partner 2 downloads this trailer video from the P2P network along with the original video file from the database. These two files are then formatted to be distributed to the user.

21 Workflow Integration of tools (IRC)

21.1 Technical Details

reference to the AXFW location of the demonstrator	A path in the CVS for example: https://cvs.axmedis.org/repos/Framework/source/Workflow/
List of libraries used	gSOAP
References to other major components needed	
Problems not solved	<ul style="list-style-type: none"> • Multithreaded Plugins • Complex Datatype mismatch amongst webservice clients and services.
Configuration and execution context	
Programming language	C++

The Axmedis Workflow integration involves developing of following modules as per the specifications:

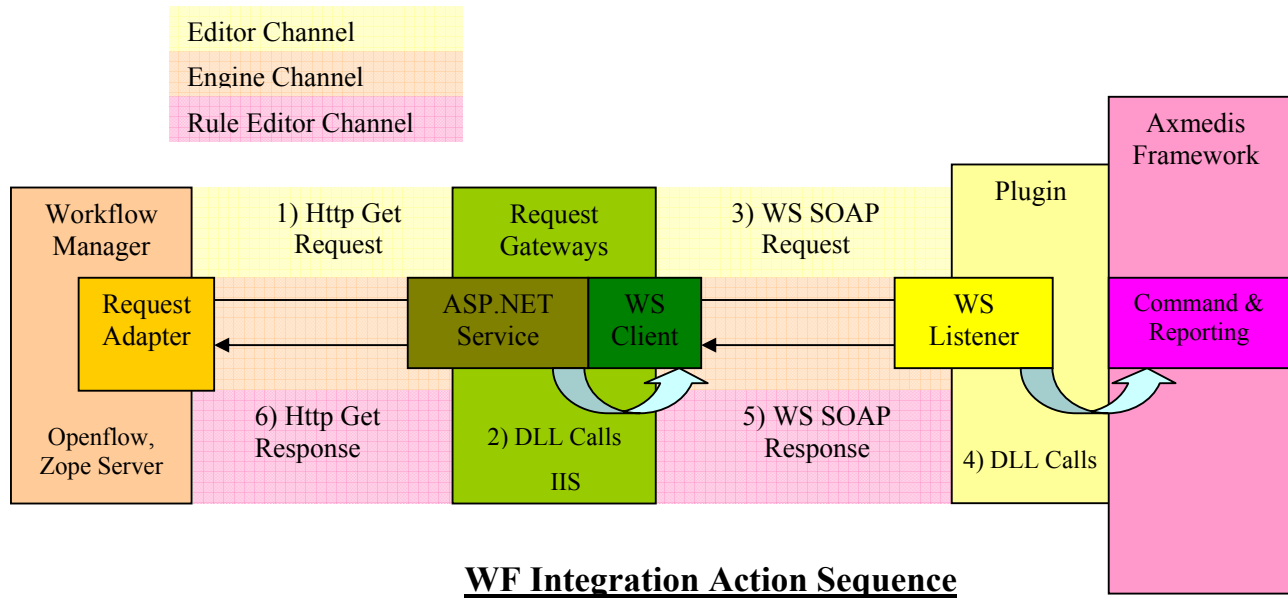
The AXMEDIS WorkFlow Area includes:

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

The overall integration is divided into four channels in order to group the common functionalities together as follows:

- Workflow Editors Interfaces
- Workflow Engines Interfaces
- Workflow Rule Editors/Viewers Interfaces
- Workflow Query Support Interfaces

As part of the first prototype development, we have developed the communication path which is the backbone for the overall integration. The following diagram shows the modules that were developed for the first prototype and the protocols used for laying the foundation.



The following modules were delivered for the first prototype integration of Axmedis workflow:

1. Editor Channel

- i) Request Adaptors
 - a) Source Code
 - b) Documentation
- ii) Request Gateways
 - a) Source Code
 - b) Documentation
- iii) Plugin
 - a) Source Code
 - b) Documentation

2. Rule Editor Channel

- i) Request Adaptors
 - a) Source Code
 - b) Documentation
- ii) Request Gateways
 - a) Source Code
 - b) Documentation
- iii) Plugin
 - c) Source Code
 - d) Documentation

3. Engine Channel

- i) Request Adaptors
 - a) Source Code
 - b) Documentation
- ii) Request Gateways
 - a) Source Code
 - b) Documentation
- iii) Plugin
 - a) Source Code
 - b) Documentation

4. Openflow

- i) User Interface
 - a) Source Code
 - b) Documentation

For the next phase of project, we have planned to complete following task:

- Multithreading for all the Plug-ins.
- Removing any inconsistencies in the data structures being transferred.
- Replicating the database channel from the other channels.
- Second version of the openflow UI.
- Writing workflow for content production.
- Black-box testing for the overall integration using the workflow defined.
- White-box testing to eliminate any inaccurate results.

21.2 Integration Support with content processing tools (AXCP processing tools: engine and scheduler)

The workflow engine interacts with the following content processing tools:

- Axmedis Compositional/Formatting Engine
- Axmedis Program and Publication Engine
- The Protection Tool Engine

The integration is realised as per the specification document. The integration involves two distinct channels for workflow communication: Request Channel and Response Channel.

The Request Channel consists of Workflow Request Adaptors, Workflow Request Gateways and Webservice Listeners, while the Response channel consists of Response Gateways and Input Queue Adaptors.

The Axmedis Engine WorkFlow channel passes through the WF Engine Request Gateway where the Engine Command and Reporting exposes the following methods, via WebServices:

- `Install_and_activate` for installing a XML rule in the scheduler and activate it. This method is valid for the Compositional/Formatting engine, for the AxepTool Loading and Publication Engine and for the Protection Engine.
- `Run_rule` for immediately run a rule already loaded inside the scheduler. This method is valid for the Compositional/Formatting engine, for the AxepTool Loading and Publication Engine and for the Protection Engine.
- `Activate_rule` for activating a rule already loaded inside the scheduler. This method is valid for the Compositional/Formatting engine, for the AxepTool Loading and Publication Engine and for the Protection Engine.
- `deactivate_rule` for disabling a not-running rule. This method is valid for the Compositional/Formatting engine, for the AxepTool Loading and Publication Engine and for the Protection Engine.
- `Suspend_rule` for suspending a rule for a specified time interval. This method is valid for the Compositional/Formatting engine, for the AxepTool Loading and Publication Engine and for the Protection Engine.

- `Pause_rule`, for suspending a rule until it will be restarted. This method is valid for the Compositional/Formatting engine, for the AsepTool Loading and Publication Engine and for the Protection Engine.
- `Kill_rule` for stopping the execution of a rule. This method is valid for the Compositional/Formatting engine, for the AsepTool Loading and Publication Engine and for the Protection Engine.
- `Remove_rule` for removing a rule from the scheduler. This method is valid for the Compositional/Formatting engine, for the AsepTool Loading and Publication Engine and for the Protection Engine.
- `Resume_rule` for resuming a paused rule. This method is valid for the Compositional/Formatting engine, for the AsepTool Loading and Publication Engine and for the Protection Engine.
- `Get_rule_status` for getting the status of a rule inside the scheduler. This method is valid for the Compositional/Formatting engine, for the AsepTool Loading and Publication Engine and for the Protection Engine.
- `Get_rule_logs` for getting history log of a rule. This method is valid for the Compositional/Formatting engine, for the AsepTool Loading and Publication Engine and for the Protection Engine.
- `Get_list_of_rules` for getting the list of the rules of a certain user inside the scheduler. This method is valid for the Compositional/Formatting engine, for the AsepTool Loading and Publication Engine and for the Protection Engine.
- `Get_rule` for getting the XML definition of a rule inside the scheduler. This method is valid for the Compositional/Formatting engine, for the AsepTool Loading and Publication Engine and for the Protection Engine.
- `Status_request_to_PnP` for getting the status of a Program of the Program and Publication Engine
- `Suspend_PnP_Program` for suspending a Program of the Program and Publication Engine
- `Abort_PnP_Program` for aborting a Program of the Program and Publication Engine
- `Resume_PnP_Program` for resuming a suspended Program of the Program and Publication Engine
- `Activate_PnP_Program` for activating a Program of the Program and Publication Engine
- `Workflow_Notification` is used to return back to the requesting engine (basically PnP) the status about the requested execution of a Workflow process

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

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

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

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

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

The response to the invoked method has the same contents listed in the preceeding Paragraph and is sent via an http GET response. The response is XML coded, following the schema:

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

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

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

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

21.3 Integration Support with Editors (AXCP rule editor and AXMEDIS editor)

The workflow engine interacts with the following Editor:

- Axmedis Editors (Object, DRM, Metadata, ...)
- Axmedis Rule Editors (PnP, Content Processing, AXEPTool, etc).

The integration is realised as per the specification document. The integration involves two distinct channels for workflow communication: Request Channel and Response Channel.

The Request Channel consists of Workflow Request Adaptors, Workflow Request Gateways and Webservice Listeners, while the Response channel consists of Response Gateways and Input Queue Adaptors.

The Axmedis Workflow Manager communicate to AXOM's Command and Reporting through WF Editor Request Adapter. The WF Request Adapter sends the requests via an http GET call. This http GET call is received by a Web Server running Microsoft IIS and directed to an ASP process called WF Request Gateway. This ASP process decodes the GET requests and formats a WebService request towards the proper Axmedis module, Axmedis Editor in this case. As AXOM, along with Command and Reporting, is a static library, a listener service is required to listen to incoming Request from Workflow and invoke Axmedis Editor accordingly. We call this listener service as AXOM_WebServices_Listener which will be resident on

client's machine. AXOM WebServices Listener is a multithreading process written in C++ which exposes methods through WebServices, listens to them and forwards the requests to the AXOM Command and Reporting module which is a C++ library. So the interface between AXOM WebServices Listener and AXOM Command and Reporting are C++ library calls.

As the AXOM and Commands and Reporting are a set of libraries, we define a new function Workflow_Editor_Launcher within the AXOM_WebService_Listener Module to launch the Editor. Upon receipt of a request to launch the Axmedis Editor, the Workflow_Editor_Launcher will launch the editor using an ActiveX Call invoking the appropriate editor. This method will not return until the Editor is terminated and the ActiveX control comes back, after which it can notify the workflow manager for the completion of the activity.

As described before, the Axmedis Editor WorkFlow channel passes through the WF Editor Request Gateway where the AXOM_WebService_Listener exposes the following methods, via WebServices:

- Edit_Object, for launching the Axmedis Object Editor and all its Plug-ins used for editing and viewing Axmedis Objects, Object Behaviours, DRMs, Hierarchies and Metadata
- Add_Object to request the Axmedis Object Manager to have a new object created
- Compose_Object for creating a subobject inside an Axmedis Object
- Delete_Object to request the Axmedis Object Manager to have a specific object deleted
- Modify_Object to request the Axmedis Object Manager to have certain object attributes modified
- View_Object_Attribute to request the Axmedis Object Manager to allow the viewing of the object attributes
- Add_History_Info to request the Axmedis Object Manager to have a history information added to an object AXINFO
- View_History_Info to request the Axmedis Object Manager to retrieve history information of a given object AXINFO
- Edit_Composition_Formatting_Rule for launching the Composition/Formatting Rule Editor
- Program_Publication_User_Interface for launching the Program/Publication User Interface
- Activate_Program_Publication for requesting the activation of a Program
- List_of_Programs for requesting the list of current programs in PnP
- Edit_AXEPTool_Rule for launching the AXEPTool Rule Editor
- Edit_Protection_Rule for launching the Protection Rule Editor

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

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

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

GET/Control_Panel/Products/OpenFlow/AXWF/**editor_name**_request_editor?AXOID="Object ID
string"&Credentials="Credential string"&AXRQID="Request ID

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

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

The response to the invoked method has the same contents listed in the preceeding Paragraph and is sent via an http GET response. The response is XML coded, following the schema:

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

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

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

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

21.4 Integration Support with AXEPTools

The workflow engine interacts with the following engines of AXEPTool:

- AXEPTool Loading Tool Engine
- AXEPTool Publication Tool Engine

As part of refinements, all the Axmedis engines were unified to form a single interface towards Axmedis Engines, the integration with AXEPTool is exactly same as that of content processing engines.

The integration is realised as per the specification document. The integration involves two distinct channels for workflow communication: Request Channel and Response Channel.

The Request Channel consists of Workflow Request Adaptors, Workflow Request Gateways and Webservice Listeners, while the Response channel consists of Response Gateways and Input Queue Adaptors.

The Axmedis Engine WorkFlow channel passes through the WF Engine Request Gateway where the Engine Command and Reporting exposes the following methods, via WebServices:

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

- Resume_PnP_Program for resuming a suspended Program of the Program and Publication Engine
- Activate_PnP_Program for activating a Program of the Program and Publication Engine
- WorFlow_Notification is used to return back to the requesting engine (basically PnP) the status about the requested execution of a WorFlow process

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

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

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

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

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

The response to the invoked method has the same contents listed in the preceeding Paragraph and is sent via an http GET response. The response is XML coded, following the schema:

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

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

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

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

21.5 Integration Support with Query Support

The integration is realised as per the specification document. The integration involves two distinct channels for workflow communication: Request Channel and Response Channel.

The Request Channel consists of Workflow Request Adaptors, Workflow Request Gateways and Webservice Listeners, while the Response channel consists of Response Gateways and Input Queue Adaptors.

The Axmedis DB Workflow channel passes through the WF DB Request Gateway where the Loader/Saver and the Query Support WebServices Interface modules will expose the following methods, via WebServices:

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

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

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

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

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

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

The response is XML coded, following the schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <xs:element name="Database_Response">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="result" type="xs:boolean"/>
        <xs:element name="errmsg" type="xs:string" nillable="true"
minOccurs="0"/>
        <xs:element name="errorcode" type="xs:int"/>
        <xs:element name="AXOID" type="xs:string" minOccurs="0"
maxOccurs="unbounded"/>
        <xs:element name="XML_Selection" minOccurs="0">
          <xs:complexType>
            <xs:simpleContent>
              <xs:extension base="xs:string"/>
            </xs:simpleContent>
          </xs:complexType>
        </xs:element>
        <xs:element name="Selection_ID" type="xs:string" minOccurs="0"
maxOccurs="unbounded"/>
        <xs:element name="Version" type="xs:string" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

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

- Edit_Query for notifying the termination of the editor:
 - NOTIFICATION: AXRQID, Completion_Result (OK, EXCEPTION)

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

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

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

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

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

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

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

21.6 Integration Support with AXMEDIS P&P Editor

The integration is realised as per the specification document. The integration involves two distinct channels for workflow communication: Request Channel and Response Channel. This integration is exactly same as for the Axmedis Editors as mentioned above.

The Request Channel consists of Workflow Request Adaptors, Workflow Request Gateways and Webservice Listeners, while the Response channel consists of Response Gateways and Input Queue Adaptors.

The following functions are available for P&P Editor.

- Axmedis Program and Publication User Interface
- AXEPTool Publication/Loading Rule Editor

The Axmedis Rule Editor WorkFlow channel passes through the WF Rule Editor Request Gateway where the User Command and Reporting will expose the following methods, via WebServices:

- Program_Publication_User_Interface for launching the Program/Publication User Interface
- Activate_Program_Publication for requesting the activation of a Program
- List_of_Programs for requesting the list of current programs in PnP

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

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

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

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

The response to the invoked method has the same contents listed in the preceeding Paragraph and is sent via an http GET response. The response is XML coded, following the schema:

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

```

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

```

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

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

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

22 Bibliography

- [Bes01] F. Bes, M. Jourdan, F. Khantache “A Generic Architecture for Automated Construction of Multimedia Presentation”, Amsterdam, The Netherlands, 2001.
- [Bol99] S. Boll, W. Klas, J. Wandel “A Cross-Media Adaptation Strategy for Multimedia Presentations”, Ulm, Germany, 1999.
- [Bol01] S. Boll, W. Klas “ZYX - a multimedia document model for reuse and adaptation of multimedia content”, Wien, Austria, 2001.
- [Bol03] S. Boll “MM4U - A framework for creating personalized multimedia content”, 2003.
- [Bul98] D.C.A. Bulterman "User-centered abstractions for adaptive hypermedia presentations", Bristol, United Kingdom, 1998.
- [Bul05] D.C.A. Bulterman, L. Hardman "Structured Multimedia Authoring", Amsterdam, The Netherlands, 1993-2005.
- [Har99] L. Hardman, J. van Ossenbruggen, K. Sjoerd Mullender, L. Rutledge, D.C.A. Bulterman "Do you have the time? Composition and linking in time-based hypermedia", Darmstadt, Germany, 1999.
- [Jou98] M. Jourdan, N. Layaïda, C. Roisin, L. Sabry-Ismail, L. Tardif “Madeus, an authoring environment for interactive multimedia documents”, Bristol, UK, 1998.
- [Lem03] T. Lemlouma, N. Layaida “Adapted content delivery for different contexts”, 2003.
- [Pih03] K. Pihkala "Extensions to the SMIL Multimedia Language", Helsinki, Finland, 2003.
- [Roi03] C. Roisin, V. Kober, V. Quint, P. Genevès, P. Navarro “Editing SMIL with Timelines”, 2003.
- [Rut98] L. Rutledge, L. Hardman, J. van Ossenbruggen, D.C.A. Bulterman "Structural Distinctions Between Hypermedia Storage and Presentation", Bristol, United Kingdom, 1998.
- [Thu02] T.T. Thuong, C. Roisin “A Multimedia Model Based on Structured Media and Sub-elements for Complex Multimedia Authoring And Presentation”, 2002.
- [Van01] J. van Ossenbruggen, J. Geurts, F. Cornelissen, L. Hardman, L. Rutledge “Towards second and third generation web-based multimedia”, Hong Kong, 2001.
- [Vil01] L. Villard “Authoring transformations by direct manipulation for adaptable multimedia presentations”, Atlanta, Georgia, USA, 2001.
- [W3C05] W3 Consortium (Bulterman et al.) "Synchronized Multimedia Integration Language

(SMIL 2.0)", 2005.

- [Wei94] L. Weitzman, K. Wittenburg "Automatic presentation of multimedia documents using relational grammars", San Francisco, CA, USA, 1994.
- [Zha02] K. Zhang, D.Q. Zhang, Y. Deng "Graphical Transformation of Multimedia XML Documents" Red Bank, NJ, USA, 2002.
- [Zha05] K. Zhang, J. Kong, M. Qiu, G. Song "Multimedia layout adaptation through grammatical specifications", Dallas, TX, USA, 2005.

23 Other reference

- [1] ISO/IEC, ISO/IEC IS 21000-5 – Rights Expression Language.
- [2] XrML, [http:// www.xrml.org/](http://www.xrml.org/).
- [3] Open Digital Rights Language (ODRL). <http://odrl.net>.
- [4] OMA DRM Rights Expression Language, OMA-Download-DRMREL-V2_0-20041210-C. 10 December 2004.
- [5] XML Encryption Syntax and Processing, W3C Candidate Recommendation 10 December 2002, <http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/>
- [6] XML Signature Syntax and Processing, W3C Recommendation 12 February 2002, <http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/>
- [7] Prados, J., Rodríguez, E., Delgado, J., Profiles for interoperability between MPEG-21 REL and OMA DRM, CEC 2005, Munich (Germany), 19 – 22 July 2005, ISBN 0-7695-2277-7.
- [8] Delgado, J., Prados, J., Rodríguez, E., Interoperability between MPEG-21 REL and OMA DRM: A profile?, ISO/IEC JTC1/SC29/WG11 MPEG2005/M11580, January 2005.
- [9] Delgado, J., Prados, J., Rodríguez, E., Interoperability between different Rights Expression Languages and Protection Mechanisms, AXMEDIS 2005, Florence (Italy), 30 November – 2 December 2005, To be published.
- [10] Delgado, J., Prados, J., Rodríguez, E., A subset of MPEG-21 REL for interoperability with OMA DRM v2.0, ISO/IEC JTC 1/SC 29/WG 11/ M11893, April 2005.