



Automating Production of Cross Media Content for Multi-channel Distribution

www.AXMEDIS.org

DE5.2.1

AXMEDIS Framework and Validation

Version: 2.0

Date: 06/9/2005

Responsible: FUPF (silvia.llorente@upf.edu)

Project Number: IST-2-511299

Project Title: AXMEDIS

Deliverable Type: Public

Visible to User Groups: Yes

Visible to Affiliated: Yes

Visible to Public: Yes

Deliverable Number: DE5.2.1

Contractual Date of Delivery: Month 12

Actual Date of Delivery: 31/08/2005

Work-Package contributing to the Deliverable: WP5.2, WP5.3

Task contributing to the Deliverable:

Nature of the Deliverable: Report

Author(s): FUPF, DSI, EXITECH, UNIVLEEDS, CRS4, IRC, XIM

Abstract:

This document describes the guidelines for setting up and maintaining AXMEDIS framework. Repositories and validation and acceptance mechanisms will be used in order to provide support to the project partners, and other affiliated partners in the future. It is also described how some tools of the AXMEDIS framework will be tested, including some indications on how data can be created to do so.

Keyword List:

AXMEDIS Framework, validation, integration

Table of Contents

1	EXECUTIVE SUMMARY AND REPORT SCOPE (FUPF)	6
1.1	WP5.2 COMPONENT VALIDATION AND ACCEPTANCE.....	7
1.2	WP5.3 AXMEDIS FRAMEWORK INTEGRATION AND MAINTENANCE.....	7
2	GUIDELINES FOR CVS REPOSITORY SET UP AND MANAGEMENT (EXITECH)	9
2.1	GUIDELINES FOR THE SUBMISSION PROCESS	9
2.2	GUIDELINES FOR THE REPOSITORY STRUCTURE AND CONTENTS	10
2.2.1	Premises	10
2.2.2	Repository Structure	10
2.2.3	General Guidelines.....	14
2.2.3.1	Directory include (optional: only for C/C++).....	14
2.2.3.2	Directory source (mandatory)	14
2.2.3.3	Directory doc/specification	14
2.2.3.4	Directory doc/code (mandatory)	15
2.2.3.5	Directory doc/test (mandatory)	15
2.2.3.6	Directory doc/configuration-deployment (optional: only for modules requiring deployment).....	15
2.2.3.7	Directory doc/other (optional)	15
2.2.3.8	Directory lib (mandatory)	15
2.2.3.9	Directory bin (mandatory).....	16
2.2.3.10	Directory project (optional: only for modules requiring building)	16
2.2.4	C/C++ Application.....	16
2.2.4.1	Directory include.....	16
2.2.4.2	Directory source	16
2.2.4.3	Directory doc/specification	16
2.2.4.4	Directory doc/test.....	16
2.2.4.5	Directory doc/configuration-deployment	16
2.2.4.6	Directory doc/other	16
2.2.4.7	Directory lib	16
2.2.4.8	Directory bin.....	16
2.2.5	C/C++ Dynamic Library.....	16
2.2.5.1	Directory include.....	16
2.2.5.2	Directory source	16
2.2.5.3	Directory doc/specification	16
2.2.5.4	Directory doc/code	17
2.2.5.5	Directory doc/test	17
2.2.5.6	Directory doc/configuration-deployment	17
2.2.5.7	Directory doc/other	17
2.2.5.8	Directory lib	17
2.2.5.9	Directory bin.....	17
2.2.6	C/C++ Static Library	17
2.2.6.1	Directory include.....	17
2.2.6.2	Directory source	17
2.2.6.3	Directory doc/specification	17
2.2.6.4	Directory doc/test.....	17
2.2.6.5	Directory doc/configuration-deployment	17
2.2.6.6	Directory doc/other	17
2.2.6.7	Directory lib	17
2.2.6.8	Directory bin.....	17
2.2.7	C/C++ WebService	17
2.2.7.1	Directory include.....	17
2.2.7.2	Directory source	17
2.2.7.3	Directory doc/specification	18
2.2.7.4	Directory doc/code	18
2.2.7.5	Directory doc/test.....	18
2.2.7.6	Directory doc/configuration-deployment	18
2.2.7.7	Directory doc/other	18
2.2.7.8	Directory lib	18

2.2.7.9	Directory bin.....	18
2.2.8	Java Application.....	18
2.2.8.1	Directory include.....	18
2.2.8.2	Directory source.....	18
2.2.8.3	Directory doc/specification.....	18
2.2.8.4	Directory doc/code.....	18
2.2.8.5	Directory doc/test.....	18
2.2.8.6	Directory doc/configuration-deployment.....	18
2.2.8.7	Directory doc/other.....	18
2.2.8.8	Directory lib.....	18
2.2.8.9	Directory bin.....	19
2.2.9	Java Library (JAR).....	19
2.2.9.1	Directory include.....	19
2.2.9.2	Directory source.....	19
2.2.9.3	Directory doc/specification.....	19
2.2.9.4	Directory doc/code.....	19
2.2.9.5	Directory doc/test.....	19
2.2.9.6	Directory doc/configuration-deployment.....	19
2.2.9.7	Directory doc/other.....	19
2.2.9.8	Directory lib.....	19
2.2.9.9	Directory bin.....	19
2.2.10	Java WebService.....	19
2.2.10.1	Directory include.....	19
2.2.10.2	Directory source.....	19
2.2.10.3	Directory doc/specification.....	19
2.2.10.4	Directory doc/code.....	19
2.2.10.5	Directory doc/test.....	19
2.2.10.6	Directory doc/configuration-deployment.....	20
2.2.10.7	Directory doc/other.....	20
2.2.10.8	Directory lib.....	20
2.2.10.9	Directory bin.....	20
2.3	GUIDELINES FOR CHECKING-OUT, UPDATING COMMITTING.....	20
3	GUIDELINES FOR ADDING NEW EXTERNAL LIBRARIES TO THE AXFW (ALL).....	24
3.1	ADDITION OF EXTERNAL LIBRARIES.....	24
3.2	AUTHORISATION OF EXTERNAL LIBRARIES USE.....	24
4	GUIDELINES FOR COMPONENT VALIDATION AND ACCEPTANCE (ALL).....	25
4.1	COMPONENT VALIDATION.....	25
4.2	COMPONENT SUBMISSION.....	26
4.3	REVIEW REPORT FORM.....	26
4.4	VERIFICATION REPORT FORM.....	27
4.5	COMPONENT ACCEPTANCE.....	27
4.6	START UP OF COMPONENT VALIDATION AND ACCEPTANCE.....	28
4.7	PERIODIC VERIFICATION.....	28
4.8	ACCEPTANCE TESTING.....	29
5	AXMEDIS FRAMEWORK VALIDATION.....	30
5.1	AXMEDIS CONTENT PRODUCTION TOOLS.....	30
5.1.1	Composition Tools (DSI).....	30
5.1.1.1	Test case revision.....	30
5.1.1.2	Creation of data for test cases.....	30
5.1.1.3	Implementation of testing tool.....	30
5.1.1.4	Description of usage of the testing tool.....	30
5.1.2	Formatting Tools (DSI).....	30
5.1.2.1	Test case revision.....	30
5.1.2.2	Creation of data for test cases.....	30
5.1.2.3	Implementation of testing tool.....	31
5.1.2.4	Description of usage of the testing tool.....	31
5.1.3	AXMEDIS database, administrator tools and support (EXITECH).....	31
5.1.3.1	Test case revision.....	31
5.1.3.2	Implementation of testing tool.....	31
5.1.3.3	Description of usage of the testing tool.....	31

5.1.4	AXMEDIS Editor and Viewer, and verification on AXMEDIS terminals (DSI)	32
5.1.4.1	Test case revision	32
5.1.4.2	Creation of data for test cases	32
5.1.4.3	Implementation of testing tool	32
5.1.4.4	Description of usage of the testing tool	32
5.1.5	Programme and Publication engine for enabling the on demand (UNIVLEEDS, FHGIGD)	32
5.1.5.1	Test case revision	32
5.1.5.2	Creation of data for test cases	32
5.1.5.3	Implementation of testing tool	33
5.1.5.4	Description of usage of the testing tool	33
5.1.6	Content Workflow integration (IRC)	33
5.1.6.1	Test case revision	33
5.1.6.2	Creation of data for test cases	33
5.1.6.3	Implementation of testing tool	33
5.1.6.4	Description of usage of the testing tool	34
5.2	AXMEDIS P2P COOPERATIVE CONTENT SHARING AND PRODUCTION TOOL (CRS4)	34
5.2.1	Virtual Database	34
5.2.1.1	Test case revision	34
5.2.1.2	Creation of data for test cases	34
5.2.1.3	Implementation of testing tool	34
5.2.1.4	Description of usage of the testing tool	34
5.2.2	DownloadMonitor	34
5.2.2.1	Test case revision	34
5.2.2.2	Creation of data for test cases	35
5.2.2.3	Implementation of testing tool	35
5.2.2.4	Description of usage of the testing tool	36
5.2.3	Publishing And Monitoring Objects	36
5.2.3.1	Test case revision	36
5.2.3.2	Creation of data for test cases	36
5.2.3.3	Implementation of testing tool	36
5.2.3.4	Description of usage of the testing tool	37
5.2.4	Loading Module of AXEPTool	38
5.2.4.1	Test case revision	38
5.2.4.2	Creation of data for test cases	38
5.2.4.3	Implementation of testing tool	38
5.2.4.4	Description of usage of the testing tool	38
5.2.5	Publication Module of AXEPTool	38
5.2.5.1	Test case revision	38
5.2.5.2	Creation of data for test cases	39
5.2.5.3	Implementation of testing tool	39
5.2.5.4	Description of usage of the testing tool	39
5.2.6	Workflow management in AXEPTool (IRC)	39
5.2.6.1	Test case revision	39
5.2.6.2	Creation of data for test cases	40
5.2.6.3	Implementation of testing tool	40
5.2.6.4	Description of usage of the testing tool	40
5.3	AXMEDIS CERTIFIER AND SUPERVISOR	40
5.3.1	AXMEDIS Supervisor (FUPF)	40
5.3.1.1	Test case revision	40
5.3.1.2	Creation of data for test cases	40
5.3.1.3	Implementation of testing tool	40
5.3.1.4	Description of usage of the testing tool	41
5.3.2	AXMEDIS Registration (DSI)	41
5.3.2.1	Test case revision	41
5.3.2.2	Creation of data for test cases	41
5.3.2.3	Implementation of testing tool	41
5.3.2.4	Description of usage of the testing tool	41
5.3.3	AXMEDIS Certification and Verification (FUPF)	41
5.3.3.1	Test case revision	41
5.3.3.2	Creation of data for test cases	42
5.3.3.3	Implementation of testing tool	42
5.3.3.4	Description of usage of the testing tool	42
5.3.4	Trace, reporting and statistic analysis (EXITECH)	42
5.3.4.1	Test case revision	42

5.3.4.2	Implementation of testing tool	42
5.3.4.3	Description of usage of the testing tool	43
5.3.5	Accounting Managing and Reporting Tool (EXITECH).....	43
5.3.5.1	Test case revision	43
5.3.5.2	Implementation of testing tool	43
5.3.5.3	Description of usage of the testing tool	44
5.3.6	Protection Tool engine (FHGIGD).....	44
5.3.6.1	Test case revision	44
5.3.6.2	Creation of data for test cases	44
5.3.6.3	Implementation of testing tool	44
5.3.6.4	Description of usage of the testing tool	45
5.3.7	DRM tools (FUPF)	45
5.3.7.1	Test case revision	45
5.3.7.2	Creation of data for test cases	45
5.3.7.3	Implementation of testing tool	45
5.3.7.4	Description of usage of the testing tool	45
AXMEDIS FRAMEWORK INTEGRATION AND MAINTENANCE.....		46
5.4	SET UP OF THE AXMEDIS FRAMEWORK FOR CONTINUOUS INTEGRATION OF AXMEDIS COMPONENTS (EXITECH)	46
5.5	REGRESSION AND INTEGRATION TESTING (EPFL)	46
5.5.1	Regression testing	46
5.5.2	Integration testing	47
5.6	OPTIMISATION OF AXMEDIS COMPONENTS (DSI).....	48
6	BIBLIOGRAPHY	49
7	GLOSSARY	49

1 Executive Summary and Report Scope (FUPF)

Market and end-users are pressing content industry to reduce prices. This is presently the only solution to setup viable and sustainable business activities with e-content. Production costs have to be drastically reduced while maintaining product quality. Content providers, aggregators and distributors need innovative instruments to increase efficiency. A solution is automating, accelerating and restructuring the production process to make it faster and cheaper. The goals will be reached by: (i) accelerating and reducing costs for content production with artificial intelligence algorithms for content composition, formatting and workflow, (ii) reducing distribution and aggregation costs, increasing accessibility, with a P2P platform at B2B level integrating content management systems and workflows, (iii) providing algorithms and tools for innovative and flexible Digital Rights Management, exploiting MPEG-21 and overcoming its limits, supporting several business and transactions models. AXMEDIS consortium (producers, aggregators, distributors and researcher) will create the AXMEDIS framework with innovative methods and tools to speed up and optimise content production and distribution, for *production-on-demand*. The content model and manipulation will exploit and expand MPEG-4, MPEG-7 and MPEG-21 and others real and de-facto standards. AXMEDIS will realize demonstrators, validated by means of real activities with end-user by leading distributor partners: (i) tools for content production and B2B distribution; (ii) content production and distribution for i-TV-PC, PC, kiosks, mobiles, PDAs. The most relevant result will be to transform the demonstrators into sustainable business models for products and services during the last project year. Additional demonstrators will be 2-3 associated projects launched as take up actions. The project will be supported by activities of training, management, assessment and evaluation, dissemination and demonstration at conference and fairs.

This deliverable is devoted to the description of tools validation and integration done inside WP5.2 and WP5.3.

This activity is by no means finished with the completion of this deliverable, but it has to be revised during the development of the project.

Main deliverables in WP5 are:

- DE5.1.1 – AXMEDIS Framework Infrastructure (M12), report;
- DE5.2.1 – AXMEDIS Framework Validation and Integration (M12), report;
- DE5.4.1 – AXMEDIS Content Tools (M15), report and prototype;
- DE5.5.1 – AXMEDIS AXEPTool (M17), report and prototype;
- DE5.6.1 – AXMEDIS Certifier and Supervisor (M17) report and prototype.

The main activities that have supported the production of this deliverable are related to:

WP5.2 – Component Validation and Acceptance – This subWP is responsible to decide whether a component is valid or not for its use in the AXMEDIS framework. The guidelines for this task will be produced, describing the AXMEDIS validation steps for each kind of component and providing an acceptance methodology. The use of a semi-formal methodology in the certification process description will help in automating the acceptance process steps via core experiments. Moreover, tools and/or mechanisms for validation and acceptance could be developed taking into account the semi-formal description. In order to test this validation methodology some preliminary version of the AXMEDIS tools will be used. Skilled partners different from those that have created the software components will carry out these tests. Details: Definition of validation and acceptance guidelines for deciding if a component can be accepted and integrated into the AXMEDIS framework; Periodic validation and acceptance test; Design and Development of tools to check the above guidelines.

WP5.3 – AXMEDIS framework integration and maintenance – This subWP will study the components database structure for the integration and maintenance of the AXMEDIS framework. A first prototype of the components database will be developed and a CVS repository will be configured to enable source sharing. As soon as the prototypes of the AXMEDIS software components are available, they will be inserted in the component database for the preliminary integration. Details: AXMEDIS Framework integration guidelines, CVS rules, access to AXMEDIS framework database. Start up of the CVS for the AXMEDIS framework; Continuous integration and refinement of AXMEDIS components: regression testing, optimisation, etc.; Integration of DRM components into the framework.

1.1 WP5.2 Component Validation and Acceptance

This WP is coordinated by FUPF

Period: M8-M48

In order to correctly set up the AXMEDIS framework, each implemented and supplied component should be validated and certified before allowing its use by content creators, content providers and final users inside AXMEDIS. The validation content is produced and collected in WP8 and the related test cases are defined in WP2 as described in WP5.1. The validation will be performed by skilled partners but different to those have created the module. Several industrial partners are involved in this work. To perform validation and acceptance of software components and tools, we have split this WP into several tasks (the great part of this work will be done after the first 18 months).

T5.2.1: Start up of the Component Validation and Acceptance

Managed by FUPF. The work should be assigned on the basis of competencies to all the partners involved on this WP.

- Definition of guidelines for deciding if a component is valid or not for its use in the AXMEDIS framework.
- Definition of acceptance guidelines for components.
- Specification of requirements for constructing tools and / or mechanisms for acceptance and validation. At this point, it should be decided which mechanisms for component validation and acceptance will be based on automatic tools and which ones will be manual.
- Development of tools to check the above guidelines.

T5.2.2: Periodic verification and Acceptance Testing

Managed by FUPF. The work should be assigned on the basis of competencies to all the partners involved on this WP.

- Software components and tools Testing of the tools developed to check the guidelines.
- Software components and tools Verification and validation of interoperability, etc.

The scope of the validation covers not only that of assessing the quality in terms of research value, but also that of verifying the: reliability, the robustness with respect to the test cases, and the conformance to the AXMEDIS framework.

1.2 WP5.3 AXMEDIS framework integration and maintenance

This WP is coordinated by EXITECH

Period: M9-M48

This WP is devoted to the integration of the components mentioned in the WP5.1. The integration work will permit the building of several demonstrators, for instance, the demonstrators developed in WP9 and the trials developed by the Take up actions. The great part of this work will be done after the first 18 months of project and will consist of:

T5.3.1: set up of the AXMEDIS framework for integration

Managed by EXITECH

- Components database set up and data base management.
- CVS set up and management for source sharing.

T5.3.2: Continuous integration of AXMEDIS components

Managed by EXITECH, performed by all partners involved according to their skill and related tools of which they are responsible.

Activity of integrating into the CVS the components, resolving conflicts, supporting the other partners during the integration. This will lead to refine the guidelines and also the stubs of the modules.

T5.3.3: Regression and integration testing

Managed by EPFL, performed by all partners involved according to their skill and related tools of which they are responsible.

Regression and integration test of the software components to verify their global functionalities, by using the integration test cases defined in WP2 and the data set produced and collected in WP8.

T5.3.4: Optimisation of AXMEDIS components

Managed by DSI, performed by all partners involved according to their skill and related tools of which they are responsible.

Optimisation of AXMEDIS Components for improving the quality of their results and removing potential integration problems.

2 Guidelines for CVS repository set up and management (EXITECH)

2.1 Guidelines for the submission process

The objective of this process is to guarantee that each partner will “commit” the MODULES resulting from the development activities into a central repository (i.e.: the CVS tree) according to the defined standard format. The defined format takes into account all the technical components and documentation that is needed by other partners or external users to download the components and to adopt them into the applications.

The CVS is a complicated tool that is easy enough to use for normal users. A repository of files, arranged in a directory structure as described in the next paragraph, will be stored in the central server and all partners and future users could **check out** copies of this repository. This would create a copy of the (current version of the) files in the users local hard drive.

The management of the central repository is under the responsibility of Exitech and DSI.

Each partner could:

- Update the copy of the CVS tree to reflect the changes made since the last updated.
- Commit the changes to the CVS tree so that others can see them.

Each partner could only “add” a new file. A new version of a file will update the old one.

No “delete” operation should be done on the central CVS, unless it is necessary to guarantee the congruency at the system level.

All the partners will make changes to their own MODULES at their leisure (without connection to the server), and subsequently **commit** these changes to the main repository. It will not be possible for more than one user to work on (and commit changes in) the same MODULE. Each module has a Responsible.

Because CVS retains all previous versions of all files that were ever part of the repository, so it will be possible (though not easy) to recover from mistakes.

For creating new folders into the repository a request has to be send to the project coordinator with the repository administrator in cc. The email has to contain the names of the folders to be created, the content description and any other relevant information.

When a commit is performed an index.txt file has to be also updated (or created), for any folder, where the committed data is briefly explained.

The verification process

The verification process (Responsible: Exitech) will guarantee that the both technical contents and documentation will have been uploaded for each module. The Responsible will perform only a formal verification of the submitted data. Errors into the committed contribution will be communicated to the partner by the project coordinator.

Each partner will be responsible for the completeness and correctness of the modules.

The CVS process will not guarantee the integration of the components; anyway all the technical information is required to permit the “integrators” and “testers” to evaluate the components and to test according to the testing procedures.

The CVS system will guarantee the versioning functionalities; it will not provide configuration management functions.

Submission plan

Each partner has to submit the total contribution according to the development plan; anyway each partner could “commit” the new version of a component after any “relevant” upgrade (bug fix, new functionalities, updated documentation).

The maintainers of the CVS will verify after each “commit” the completeness of the documentation. Moreover, some statistics will be provided automatically for each component (i.e.: file age, number of submission per partners, etc).

The objective of this process is to guarantee that each partner will “commit” the MODULES resulting from the development activities into a central repository (i.e.: the CVS tree) according to the defined standard format. The defined format takes into account all the technical components and documentation that is needed by other partners or external users to download the components and to adopt them into the applications.

2.2 Guidelines for the repository structure and contents

2.2.1 Premises

In order to better understand the guidelines contained in this document, it is necessary to underline the premises under which they are valid and the context in which the repository for AXMEDIS framework, web service and application, hereinafter “the CVS repository” has been born, the name derives from the definition reported in Annex I. the structure of the CVS repository is oriented at who will use the AXMEDIS framework and in particular to partners involved in WP9 demonstrators and those that will contribute to AXMEDIS with take up actions and projects. It is for them that we are mainly creating this framework.

As stated by the project manager, that requested these guidelines according to the schedule of Annex I:

- The CVS repository is not a support for development and developers, since each team in each company has to create the environment for development inside the company;
- The CVS repository is not an instrument for integration, since it has no configuration management process inside, at least now, a task in WP5 will define the guidelines for the integration;
- The CVS repository is not an instrument for automated building and continuous integration, since it has no management of automatic building;
- The CVS repository is not an instrument for automated building and continuous integration, since it has no management of automatic building and no bound on submission due to a failure in compilation;
- The CVS repository is a support for management of the project, and for the versioning of the AXFW and its detailed components, and for the creation/storage of demonstrators;
- The CVS repository has to support versioning only at predefined time intervals (that can be also a time frame of months or weeks, this will depend on the project evolution and status);
- Each contribution at the CVS repository has to be self-contained in the sense that has to provide comprehensive support for compiling, executing, deploying the AXMEDIS part submitted;
- Each contribution at the CVS repository has to be completed with documentation according what has been formally defined in these guidelines, and revision of the specification document starting from the official version;
- Each contribution to the CVS repository has to be completed with test cases and test code for allowing regression testing, a more detailed description of the activity of testing will be described in specific guidelines on testing, regression and acceptance testing.

2.2.2 Repository Structure

After all these necessary premises the guidelines for the structure and the content of the repository can be detailed:

The repository is structured in three main parts:

- Framework
- WebServices
- Application

Each part will contain a set of subdirectory for putting all the necessary files. Not all the directories are mandatory for each kind of application types as detailed in the following. In any case the following set of directories will be present under each specified parts:

- include: directory only for C/C++ application that have include files;

- source: directory containing the source code;
- doc: directory containing the documentation as specification documentation, code documentation, test documentation, dependencies and configuration documentation, other documentation. For each kind of documentation a detailed description will be given;
- lib: directory containing all the necessary libraries (lib, dll, jar, etc) needed for the AXMEDIS part to work;
- bin: directory containing the result of building process that can be a jar, an executable, a library, a war or whatever.

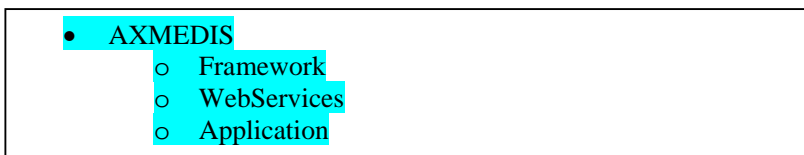
The doc directory has to be split in different directories according to the following structure:

- specification: this directory must contain the UML diagrams of all the classes and in order to avoid the problems due to incompatible format the diagrams have to be submitted in XMI format. In the same directory, if present, also the XML schemas and WSDL files have to be put. Any other format for UML with respect to XMI format will not be accepted;
- code: in this directory the documentation of code must be put. The documentation should be Javadoc, doc++ or other auto-generated documentation (preferred) or other documentation format. In any case documentation must be put in one of the following formats: HTML or TXT (preferred) and/or RTF or PDF (not preferred but acceptable). Any other proprietary format such as Microsoft Word docs, OpenOffice docs, or other will be refused;
- test: this directory must contain all the test cases necessary to test the part, the source code and building scripts for testing. Automatic testing is recommended such as JUnit (i.e. <http://www.junit.org/>) for Java, CppUnit (i.e. <http://sourceforge.net/projects/cppunit/>) for C++, PHPUnit (i.e. <http://www.phpunit.de/en/index.php>) for PHP and so on, and script for automating testing are kindly requested such as *Batch*, *Makefile*, *Ant* and so on.
- configuration-deployment
- other

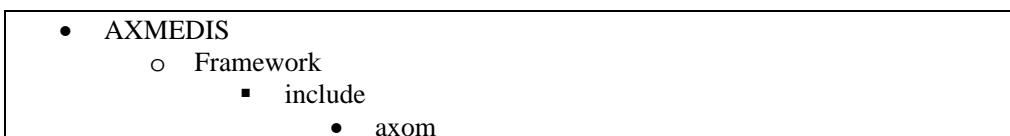
This structure has been selected:

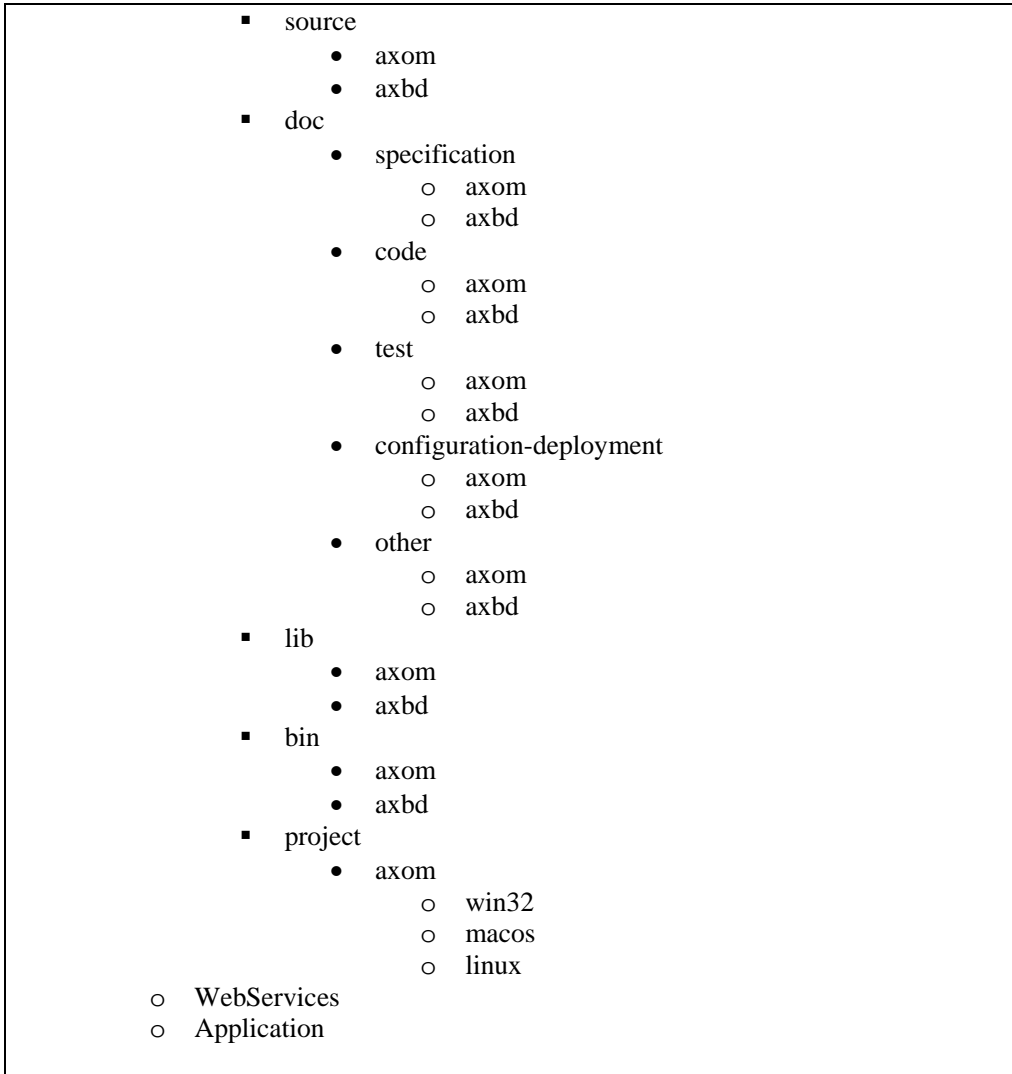
- To give the best integrated view to the future users of the AXFW. In this way we can give to them in a simple manner access to separate levels. For example, access to source code, only to applications, only to the includes, etc.
- To provide and collect the history of all the executable applications developed, this will permit to provide the history of development according to annex I of the contract and to provide demonstration without the need of recompiling and involving other partners
- To provide access in a simple manner at all the code and applications to be test for the partners that have to test and verify the tools even without the presence of who has produced the code.

Under each of the directories defined, a directory for each project part has to be created, so that the final structure will be something like:



The Framework Structure is described in the following picture:

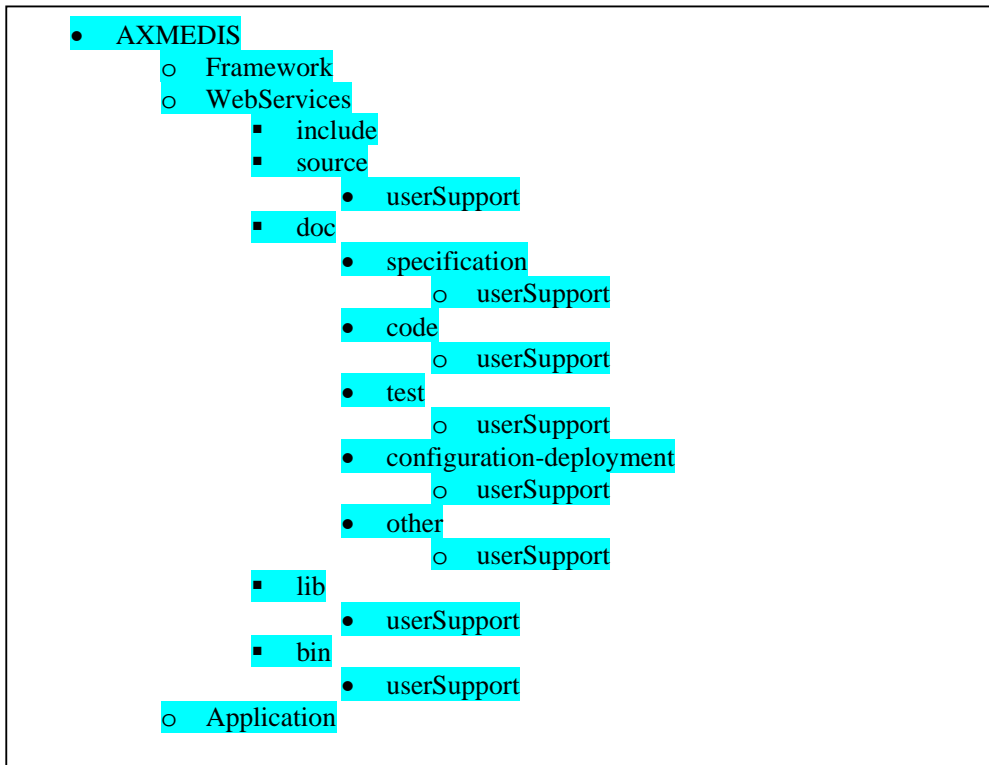




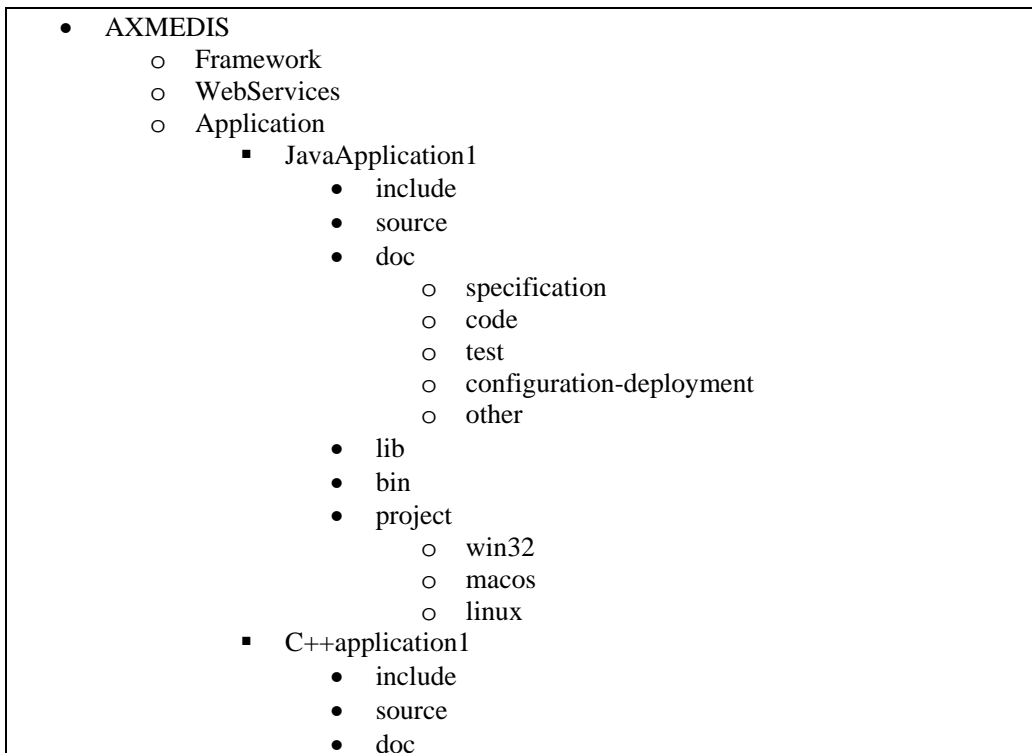
Into the “Framework” part of the repository it is supposed to have:

- object models
- loader and saver
- communication protocols
- DB access
- data transforming logic
- algorithms
- DLL
- plug ins
- libraries
- scripts
- web service support
- etc.

The Webservice Structure is described in the following picture:



And finally the Application structure can be depicted as follows:



- specification
- code
- test
- configuration-deployment
- other
- lib
- bin
- project
 - win32
 - macos
 - linux

As applications there are considered:

- User interfaces
- Skins
- Main classes that can generate an executable by using framework content
- all the test tools needed for other parts (protocols) verification.
- Etc.

It is possible, depending on the software that is realized, to define some guidelines for the different kind on file to be put in each directory as stated in the next paragraphs.

In the following the general guidelines will be reported and in the more specific subsection only the changes that applies to the general structure will be reported.

2.2.3 General Guidelines

2.2.3.1 Directory include (optional: only for C/C++)

All the include (**.h, .hpp, .hh, etc**) that are generated for the project by the developers.

Inside this directory, a sub directory for each component will be created.

2.2.3.2 Directory source (mandatory)

All the source code organized in subdirectory defined by each development team (**.c, .cxx, .cpp, .java, etc**) that are generated for the project by the developers.

Inside this directory, a sub directory for each component will be created. Inside this directory the partner can organize the source code in the best manner according to the project type; some example follows:

- A set of subdirectories with source code put inside
- The source code directly placed in the directory
- A hierarchy of directory differentiating for source type (i.e. Java, C++, WebPages, etc)

2.2.3.3 Directory doc/specification

In this directory must be placed:

- the *UML diagram* in **XMI or Visio formats** in order to guarantee maximum interoperability and independence by UML editor software vendors that are generated for the software under development – mandatory for modules having OO classes;
- the *XML schema* adopted for the XML test case or for other activities related to the software under examination in **XML format** – mandatory if the schema was used;
- the *WSDL* used by the application for accessing as a client to web services on the server side in **plain text format** – mandatory if the WSDL was used.

2.2.3.4 Directory doc/code (mandatory)

In this directory the documentation of the code must be present. No submission without documentation can be acceptable. It is suggested to use automatic documentation generation according for example to JavaDoc or Doc++ style. In any case the documentation can be submitted only in the following formats in order to guarantee the maximum interoperability and independence by software vendors:

- HTML (preferred)
- Plain TEXT (preferred)
- RTF
- DOC

No other format will be accepted.

2.2.3.5 Directory doc/test (mandatory)

In this directory all the test cases and the test code must be submitted together with the operational instruction to test the code. Automatic testing is recommended and scripts for automating testing are kindly requested as *Batch file*, *Makefile*, *Ant* and so on.

When in test cases you are referring to some objects or digital resource please provide reference according to the documentation reported in the official deliverable on Test Cases, see WP2 and WP8 deliverables.

The documentation on how to test software must be inserted in a **plain text** file named *HowToTest.txt* in the root directory of the doc/test. This file must contain a reference to additional libraries (to be put on the lib directory) that are necessary for testing and not for deployment.

2.2.3.6 Directory doc/configuration-deployment (optional: only for modules requiring deployment)

In this directory a **plain text** file named *configuration-dependencies.txt* must be placed in the root of the directory doc/configuration-deployment and it must be filled with the information on dependencies of this software piece from other software pieces that are present in the repository tree together with the related version that have been tested to correctly operates.

In addition a **plain text** file named *deployment.txt* must be placed in the root of the directory doc/configuration-deployment containing all the information that are needed to know how to deploy the software artifact (such as the library that are needed, environment variables, additional software needed, target operating system, etc).

Any other file can be added, but the only formats allowed are:

- HTML
- Plain TEXT
- RTF
- DOC

2.2.3.7 Directory doc/other (optional)

This directory can contain any additional document useful for the development community. The only bound is the format that must be chosen among:

- HTML
- Plain TEXT
- RTF
- PDF

2.2.3.8 Directory lib (mandatory)

This directory must contain all the libraries that are not statically linked to the application and that are needed for the application in order to work. In the case the no library is needed an empty file named *no-lib-needed.txt* must be placed in the lib root directory. In the case of multiplatform development a directory for each platform have to be created and the lib must be put in the correct directory.

2.2.3.9 Directory bin (mandatory)

This directory must contain the product of the module and all the components (configuration files, etc) that are necessary for a correct exploiting of the software. It is considered product: an executable, a library, etc. For application that do not have an executable, the jar, war, dll, lib or the result of the module that have to be deployed must be put here. In the case of multiplatform development a directory for each platform have to be created and the executable must be put in the correct directory.

2.2.3.10 Directory project (optional: only for modules requiring building)

This directory must contain the makefiles or IDE projects to build the corresponding library or application. This directory can contain also the minimal set of file (in a unique zip file for easy recompiling the module in a defined IDE of the user. This zip file must contain only the minimal set of file, i.e. those after a project clean up.

2.2.4 C/C++ Application

2.2.4.1 Directory include

Must be present

2.2.4.2 Directory source

No change

2.2.4.3 Directory doc/specification

No change

2.2.4.4 Directory doc/test

No change

2.2.4.5 Directory doc/configuration-deployment

No change

2.2.4.6 Directory doc/other

No change

2.2.4.7 Directory lib

No change

2.2.4.8 Directory bin

No change

2.2.5 C/C++ Dynamic Library

2.2.5.1 Directory include

Must be present

2.2.5.2 Directory source

No change

2.2.5.3 Directory doc/specification

No change

2.2.5.4 Directory doc/code

No change

2.2.5.5 Directory doc/test

No change

2.2.5.6 Directory doc/configuration-deployment

No change

2.2.5.7 Directory doc/other

No change

2.2.5.8 Directory lib

No change

2.2.5.9 Directory bin

No change

2.2.6 C/C++ Static Library

2.2.6.1 Directory include

Must be present

2.2.6.2 Directory source

No change

2.2.6.3 Directory doc/specification

No change

2.2.6.4 Directory doc/test

No change

2.2.6.5 Directory doc/configuration-deployment

No change

2.2.6.6 Directory doc/other

No change

2.2.6.7 Directory lib

No change

2.2.6.8 Directory bin

No change

2.2.7 C/C++ WebService

2.2.7.1 Directory include

Must be present

2.2.7.2 Directory source

No change

2.2.7.3 Directory doc/specification

The WSDL used by the application for accessing as a client to other web services and WSDL used for generating the server side path or the web service under development on the server side in **plain text format**.

2.2.7.4 Directory doc/code

No change

2.2.7.5 Directory doc/test

No change

2.2.7.6 Directory doc/configuration-deployment

No change

2.2.7.7 Directory doc/other

No change

2.2.7.8 Directory lib

No change

2.2.7.9 Directory bin

No change

2.2.8 Java Application

2.2.8.1 Directory include

Not present

2.2.8.2 Directory source

No change

2.2.8.3 Directory doc/specification

No change

2.2.8.4 Directory doc/code

No change

2.2.8.5 Directory doc/test

No change

2.2.8.6 Directory doc/configuration-deployment

No change

2.2.8.7 Directory doc/other

No change

2.2.8.8 Directory lib

This directory must contain all the libraries in JAR format that are needed by the java application. In the case the no library is needed an empty file named *no-lib-needed.txt* must be placed in the lib root directory.

2.2.8.9 Directory bin

This directory must contain the application packaged in a JAR file with a batch file for executing it. In the case of multiplatform script for launching the application, a directory for each platform has to be created and the script must be put in the correct directory.

2.2.9 Java Library (JAR)

2.2.9.1 Directory include

Not present

2.2.9.2 Directory source

No change

2.2.9.3 Directory doc/specification

No change

2.2.9.4 Directory doc/code

No change

2.2.9.5 Directory doc/test

No change

2.2.9.6 Directory doc/configuration-deployment

No change

2.2.9.7 Directory doc/other

No change

2.2.9.8 Directory lib

This directory must contain all the libraries in JAR format that are needed by the java library. In the case the no library is needed an empty file named *no-lib-needed.txt* must be placed in the lib root directory.

2.2.9.9 Directory bin

This directory must contain the library packaged in a JAR file.

2.2.10 Java WebService

2.2.10.1 Directory include

Not present

2.2.10.2 Directory source

No change

2.2.10.3 Directory doc/specification

The *WSDL* used by the application for accessing as a client to other web services and *WSDL* used for generating the server side path or the web service under development on the server side in **plain text format**.

2.2.10.4 Directory doc/code

No change

2.2.10.5 Directory doc/test

No change

2.2.10.6 Directory doc/configuration-deployment

No change

2.2.10.7 Directory doc/other

No change

2.2.10.8 Directory lib

No change

2.2.10.9 Directory bin

No change

2.3 Guidelines for checking-out, updating committing

For managing the AXMEDIS repository the Subversion (<http://subversion.tigris.org/>) platform has been chosen.

A complete Subversion documentation and also a Subversion book are available on the project website <http://subversion.tigris.org/servlets/ProjectDocumentList>.

Subversion is a free/open-source version control system. That is, Subversion manages files and directories over time. A tree of files is placed into a central repository. The repository is something like an ordinary file server, except that it remembers every change ever made to your files and directories. This allows recovering older versions of data, or examining the history of how data changed.

Subversion can access its repository across networks, which allows it to be used by people on different computers.

Some of Subversion capabilities are:

Directory versioning

CVS only tracks the history of individual files, but Subversion implements a “virtual” versioned file system that tracks changes to whole directory trees over time. Files and directories are versioned.

True version history

Atomic commits

A collection of modifications either goes into the repository completely, or not at all. This allows developers to construct and commit changes as logical chunks, and prevents problems that can occur when only a portion of a set of changes is successfully sent to the repository.

Versioned metadata

Each file and directory has a set of properties—keys and their values— associated with it. You can create and store any arbitrary key/value pairs you wish. Properties are versioned over time, just like file contents.

Choice of network layers

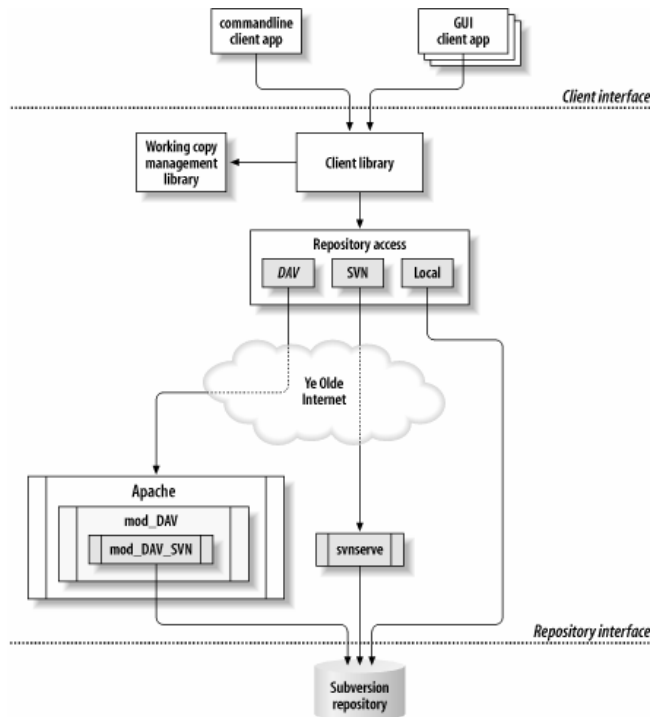
Consistent data handling

Subversion expresses file differences using a binary differencing algorithm, which works identically on both text (human-readable) and binary (human-unreadable) files. Both types of files are stored equally compressed in the repository, and differences are transmitted in both directions across the network.

Efficient branching and tagging

Subversion creates branches and tags by simply copying the project, using a mechanism similar to a hard-link. Thus these operations take only a very small, constant amount of time.

Subversion architecture:



Subversion architecture (extracted from: <http://svnbook.red-bean.com/en/1.0/svn-book.html>)

The AXMEDIS repository will be configured using the Apache “mod_DAV” & the Subversion “mod_DAV_SVN” since the “svnserve” seems to give some limits to the user configuration.

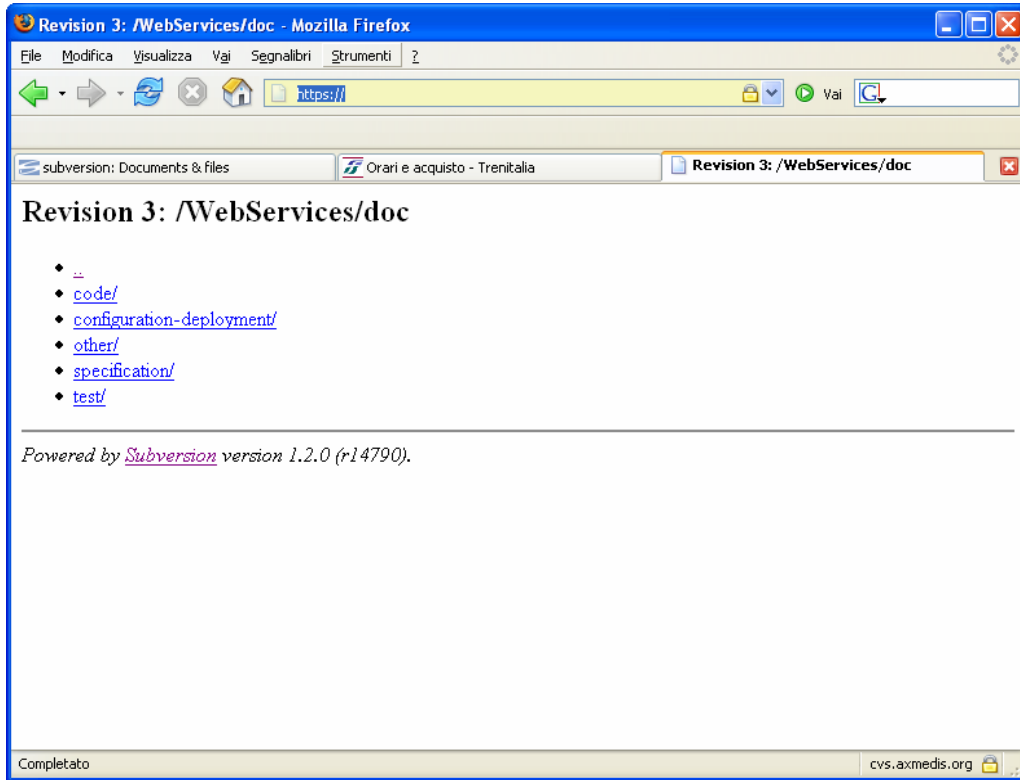
The AXMEDIS repository will be available on a SSL protected site. The repository will be protected with an account and a password and the internal folder will be configured in order to allow the access only to the selected users.

The repository can be accessed for reading purposes via browser, command line and using Subversion compatible GUI clients.

Web access

It will be possible to access and browse the (full or part of) repository according to the user rights. It is proposed to give one account for each partner.

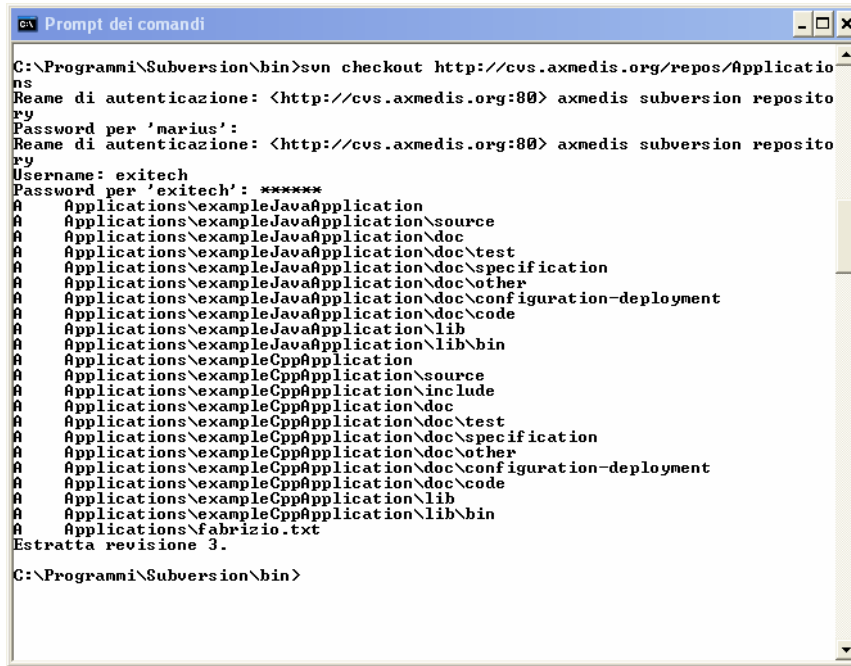
A possible view of the repository could be the one shown in the next figure.



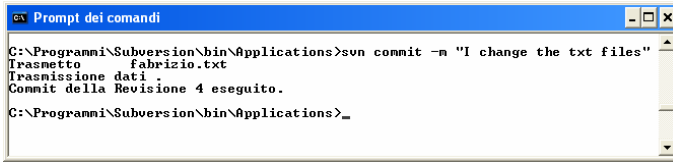
Command line client access

Command line clients are available for all the operating systems (MAC OS X, Linux, Solaris, BSD, and Windows).

For getting a working copy of the Application Folder it is enough to run the “svn checkout” command. The following figure shows the Application folder checkout from a windows command line subversion client.

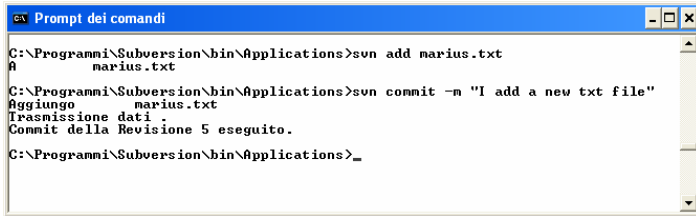


Let suppose that the fabrizio.txt file was changed and a new file is added to the working directory. The changes can be committed by using the “svn commit” as follow:



```
sv Prompt dei comandi
C:\Programmi\Subversion\bin\Applications>svn commit -m "I change the txt files"
Trasmetto      fabrizio.txt
Trasmissione dati .
Commit della Revisione 4 eseguito.
C:\Programmi\Subversion\bin\Applications>_
```

If the new file has to be added to the repository it has to be first added to the working directory:



```
sv Prompt dei comandi
C:\Programmi\Subversion\bin\Applications>svn add marius.txt
A
marius.txt
C:\Programmi\Subversion\bin\Applications>svn commit -m "I add a new txt file"
Aggiungo      marius.txt
Trasmissione dati .
Commit della Revisione 5 eseguito.
C:\Programmi\Subversion\bin\Applications>_
```

A list of the other Subversion command line is available on the Subversion book (<http://svnbook.red-bean.com/en/1.0/svn-book.html>) at Chapter 9 (<http://svnbook.red-bean.com/en/1.0/svn-book.html#svn-ch-9>).

Using a GUI Subversion client

Several subversion clients are available. For a list you can see: http://subversion.tigris.org/project_links.html at the clients and plug-in section.

Also some development environments include GUI subversion clients.

3 Guidelines for adding new external libraries to the AXFW (ALL)

This section describes the process for adding new external libraries to be used by the tools inside the AXFW, apart from the ones declared in the specification documents.

3.1 Addition of external libraries

In order to add a new library, the following directives have to be followed:

- 1) Any library that is being used for the development of any tool to be included in the AXFW has to be authorised before its usage.
- 2) An AXMEDIS partner cannot decide autonomously to make some GPL or LGPL if it comes from the project results. The coding of any part is IPR of the project and thus it has to be authorised and any implication has to be verified.

The violation of these directives implies may lead at the exclusion of a partner for bad behaviour. The CA reports most of the details on this issue.

3.2 Authorisation of external libraries use

The inclusion of any library has to be authorised by sending a request to the project coordinator and the general reflector indicating:

- what would you like to use
- which OS
- why
- license model of the library

For instance, the authorisation will not be given if the module is GPL. It could be discussed depending on the point in which the module is used if it should be GPL. In any case has to be separately discussed.

Finally, the delivery of code in GPL or LGPL is not authorised without an explicit authorisation of the AXMEDIS consortium.

The list of accepted libraries is reported in the AXMEDIS portal (www.axmedis.org).

Each accepted library has to be posted on the portal in the corresponding folder with:

- library in binary and source code
- documentation of the library for its compilation and usage
- license

4 Guidelines for Component Validation and Acceptance (ALL)

In order to correctly set up the AXMEDIS framework, each implemented and supplied component should be validated and certified before allowing its use by content creators, content providers and final users inside AXMEDIS. This will help in guaranteeing that the implemented component meets the requirements.

The validation will depend on several issues and will seek to establish the degree to which a given component fulfils the following performance criteria:

- It does what it is supposed to do
- It does its job in a reasonable time
- It can be integrated with other components
- It is reliable
- It is robust
- It accomplishes test cases
- It is conformant with the AXMEDIS framework

In the following sections we will describe the general guidelines for component validation and acceptance in the AXMEDIS framework.

4.1 Component validation

The component validation process should involve the checking and analysis of the component in order to verify that the component meets the requirements demanded of it.

The main validation activities are the revision and testing of the component.

Review phase main steps:

- Review phase involves the manual review of the component, directly evaluating it. It will help in determining that the requirements, design concepts and specifications have been met. The revision of a component can include several activities, like peer reviewing or code inspection.
- The result of the review should be a report where the reviewing team explains the revision performed, the errors found and other.

Testing phase involves the testing of the component at different levels, from unit test through to system test.

The main steps in the testing phase are the following ones:

- During development phase of the component, unit tests should be done in order to check that the functionality being implemented is the correct one.
 - This task should be done by the development team of the component, as they have the complete knowledge over it.
 - Modifications over the specification should be reported
- When the development of a component has been completed, an initial unit testing phase and an integration test is needed to evaluate how the unit performs and how it interacts with other units.
 - This task should be done with the cooperation of the development teams of the integrated components. The result of this task should be reported in order to perform the needed actions for solving the problems found.
 - This task also needs to evaluate the following aspects, and report on them:
 - Documentation provided
 - Differences with the specification, if any
 - Interfaces among components

- The rest of tests should involve the whole system and it will be different depending on the demonstrator. So system testing should be demonstrator testing for the AXMEDIS framework, at least in a first phase.

In order to perform the above tests, the test cases identified and described in WP2 and the content for test cases created and described in WP8 should be taken into account.

Apart from tests and reviews, component validation should also include the examination of the degree to which the component is properly documented, including updates in the specification documents, documentation of the programming interface, usage manual (specially for end-user or business-user applications) or installation manual. This information should be included in the AXMEDIS framework part associated to this component.

The major requirements and functionalities of the component need to be mapped to the component under validation, in order to check that the component accomplishes them.

4.2 Component Submission

This section describes what has to be reported when a component is submitted into the AXMEDIS framework.

Module name	Name of the module being reviewed.
Module description	General purpose of the module.
Major requirements	To be provided by the component owner
Major use cases	To be provided by the component owner
Major related components	To be provided by the component owner
List of Test Cases	<ul style="list-style-type: none"> • To be provided by the component owner • Accepted by the validators Additional test cases may be provided.
Used Libraries	To be provided by the component owner Versions and library related information should be provided
Languages	Programming languages
Operating system(s)	A list and information to compile for different OS
Author	Organisation that has the responsibility of the implementation of the module. The name of the person involved in the implementation can also be given.

4.3 Review report form

This section gives an example of what has to be reported when a module is being reviewed. The main information to be reported is described in the following table.

Review ID	Identifier of the review.
Module name	Name of the module being reviewed.
Module description	General purpose of the module.
Major requirements	Comments about conformance on requirements
Major use cases	Comments about conformance on use cases
Major related components	Comments on the integration aspects with other AXMEDIS tools or components
List of Test Cases	Comments on the results about the usage of those test cases
Used Libraries	Comments on the usage of those external libraries
Author	Organisation that has the responsibility of the implementation of the module.

		The name of the person involved in the implementation can also be given.
Date of review		Date when the review is done.
Participants		Name, organisation and role of people involved in the review. This field can be repeated as many times as needed.
Issues	Location	Part of the module where the issue is found. For instance, this could be the source code file, web page, etc; it will depend on the type of application.
	Who	Person who finds the issue.
	Description	Description of the issues found in the module during revision.

4.4 Verification report form

This section gives an example of what has to be reported when a module is verified, after it has been reviewed and some issues have been found. The information to be reported is described in the following table.

Verification ID		Identifier of the verification.
Review ID		Identifier of the review to which this verification refers.
Module name		Name of the module being verified.
Module description		General purpose of the module.
Major requirements		Comments about conformance on requirements
Major use cases		Comments about conformance on use cases
Major related components		Comments on the integration aspects with other AXMEDIS tools or components
List of Test Cases		Comments on the results about the usage of those test cases
Used Libraries		Comments on the usage of those external libraries
Author		Organisation that has the responsibility of the implementation of the module. The name of the person involved in the implementation can also be given.
Date of verification		Date when the verification is done.
Participants		Name, organisation and role of people involved in the review. This field can be repeated as many times as needed.
Issues	Location	Part of the module where the issue is found. For instance, this could be the source code file, web page, etc; it will depend on the type of application.
	Who	Person who finds the issue.
	Description	Description of the issues found in the module during revision.
	Resolution	How the issue found during the revision process has to be solved.

4.5 Component acceptance

Component acceptance mostly involves those components, which are addressed to application users, either business or final users.

Acceptance testing allows users (or project partners which can assume this role), to test the functionality of the system against the requirements and use cases. Each kind of tool should be tested by a key user on this area. For instance, the component acceptance test of a content production tool should be done by a user that is skilled in the area together with part of the development team, in order to get the comments on the tool behaviour.

It will be very useful to follow the test cases that involve components to be accepted in order to check if they are correct. A report on if test cases are accomplished or not should be done by test participants.

4.6 Start up of component validation and acceptance

In order to start up the component validation and acceptance in the AXMEDIS framework, the responsible of each module should:

- Follow the CVS repository guidelines
- Prepare unit tests
- Perform unit tests and give a brief report on them
- Prepare integration tests
- Perform integration tests. They may be done by the partners participating in the integration together or not. In the latter case, some communication mechanisms will be established in order to solve possible integration problems as soon as possible. For instance, if one partner is performing an integration test of one of his components with other partner's component, a videoconference or audio-conference might be set up before, during or after tests are done
- Give a brief report on the integration test, outlining problems found, categorising them (application error, bad parameter value, inconsistent API, etc)

These reports should be available in the portal to the rest of the partners in order to improve the component knowledge and maintenance. A new directory / portal section should be created to contain them. They could also be included into the CVS repository, together with the corresponding application, web service, library, etc.

4.7 Periodic verification

During project development, components will be improved / updated to solve problems found during the different testing phases or to meet new requirements found during the acceptance tests or the evolution of commercial software products and standards affecting the AXMEDIS framework components.

In this way, there will be the need to inform the rest of the partners of any components having been updated, so that new integration tests are performed if needed.

Reports on the unit tests performed over the new or updated components have to be given together with the new version of the component.

The periodic verification steps should be as follows:

- 1 Update component into the corresponding directory of the CVS repository, indicating that it is a new version.
- 2 Send e-mail to the reflector and / or developers mailing list to inform that an update on a component has been done and including the results of the related regression test(s) when needed.
- 3 Partners using the updated component should:
 - 3.1 Perform integration tests with the new version of the component. The participation / support of the component responsible may be requested.
 - 3.2 Report errors / problems detected during the integration test, if any. The report has to be uploaded in the portal, in order that partners are informed of the results of the test.
 - 3.3 If needed, after integration errors / problems reported in 3.2 are solved, components using the initially updated component should be also updated in the CVS repository.
 - 3.4 Go to verification step 2 for the component(s) updated in step 3.3.

As a modification in one component may involve the modification of many other components, regression and integration tests should be systematically done. In some cases, it could happen that a component will no longer work with lower versions of libraries and components. This should be specified in the documentation of the component.

4.8 Acceptance testing

Acceptance of a component involves that it has previously been validated and verified, making the corresponding unit and integration tests. Once they have been passed, acceptance tests should be done. They may involve other partners, final users, user group experts, etc.

Acceptance test results could be reported using the review form described in section 4.3.

Once the component has been accepted, it can be made publicly available in the AXMEDIS framework. Moreover, it may be further optimised.

5 AXMEDIS Framework validation

This section describes how the AXMEDIS framework should be validated. It is organised by the groups of AXMEDIS Framework tools that have to be implemented in WP5.4, WP5.5 and WP5.6.

The following steps could be followed in order to validate tools:

- Revise test cases corresponding to each tool inside the framework.
- Collection and/or creation of the data needed for checking the revised test cases.
- Implementation of tools that check test cases against the corresponding tools using the defined content.
- Description of usage of the testing tools, in order that partners different from the implementer one could test the tool.

5.1 AXMEDIS Content production Tools

5.1.1 Composition Tools (DSI)

5.1.1.1 Test case revision

The test cases that can be revised for Composition Tools are from TC5.1.1.1 to TC5.1.2.5. They were described in DE2.2.1 Test cases and Content Description.

During AXMEDIS Composition Tools implementation, these test cases should be considered for revision, as specification and implementation details could make them change. Moreover, some more test cases could appear, not initially considered on the test cases deliverable.

5.1.1.2 Creation of data for test cases

Utilising the content produced in WP8, the test cases will use a number of selected AXMEDIS objects in order to show and demonstrate the composition functionalities in AXMEDIS. The content used for tests will try to cover a wide set of content types (video, images, text, audio) and formats.

5.1.1.3 Implementation of testing tool

The rule editor prototype will be used to test scripts for composition. A batch test script executor will be used for batch testing.

5.1.1.4 Description of usage of the testing tool

Description of usage of the testing tool, to allow other partners and external users to test the tool or to test equivalent tools.

5.1.2 Formatting Tools (DSI)

5.1.2.1 Test case revision

The test cases that can be revised for Formatting Tools are from TC5.2.1.1 to TC5.2.2.5. They were described in DE2.2.1 Test cases and Content Description.

During AXMEDIS Formatting Tools implementation, these test cases should be considered for revision, as specification and implementation details could make them change. Moreover, some more test cases could appear, not initially considered on the test cases deliverable.

5.1.2.2 Creation of data for test cases

Utilising the content produced in WP8, the test cases will use of a number of selected AXMEDIS objects in order to show and demonstrate the formatting functionalities in AXMEDIS. The content used for tests will try to cover a wide set of content types (video, images, text, audio) and formats.

5.1.2.3 Implementation of testing tool

The rule editor prototype will be used to test scripts for formatting. A batch test script executor will be used for batch testing.

5.1.2.4 Description of usage of the testing tool

Description of usage of the testing tool, to allow other partners and external users to test the tool or to test equivalent tools.

5.1.3 AXMEDIS database, administrator tools and support (EXITECH)

5.1.3.1 Test case revision

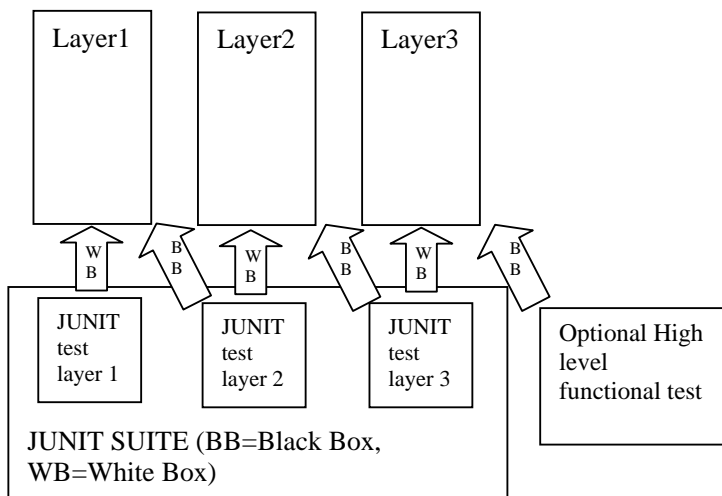
The test cases that can be revised for AXMEDIS database, administrator tools and support are from TC8.1.1 to TC8.1.7 and from TC 8.2.1 to 8.2.6. They were described in DE2.2.1 Test cases and Content Description. During AXMEDIS database, administrator tools and support implementation, these test cases should be considered for revision, as specification and implementation details could make them change. Moreover, some more test cases could appear, not initially considered on the test cases deliverable. Creation of data for test cases

5.1.3.2 Implementation of testing tool

Instead of a testing tool, regression testing has been implemented by the adoption of JUNIT suite for automating testing.

At each level of the hierarchy in the logical diagram of the system, Unit tests are adopted for testing the layer in a white box manner. The Unit test of the higher level is used to test in a black box manner the underlying layer. A functional test to check in black box manner the last layer will be implemented if needed.

The following schema details this assumption:



5.1.3.3 Description of usage of the testing tool

The testing is completely automated by using JUNIT or the ANT scripts provided with the source code.

5.1.4 AXMEDIS Editor and Viewer, and verification on AXMEDIS terminals (DSI)

5.1.4.1 Test case revision

The test cases that can be revised for AXMEDIS Editor and Viewer are from TC4.1.1 to TC4.3.3. They were described in DE2.2.1 Test cases and Content Description.

During AXMEDIS Editor and Viewer implementation, these test cases should be considered for revision, as specification and implementation details could make them change. Moreover, some more test cases could appear, not initially considered on the test cases deliverable.

5.1.4.2 Creation of data for test cases

Utilising the content produced in WP8, the test cases will use a number of selected AXMEDIS objects in order to show and demonstrate the editing and viewing functionalities in AXMEDIS. The data used for test will try to cover a wide set of content types (video, images, text, audio) and formats.

5.1.4.3 Implementation of testing tool

Unit testing will be used to test the functionalities of AXOM, which is used by the AXMEDIS Editor & Viewer and many other tools.

The AXMEDIS editor/viewer prototype will be used for testing user interface and content use.

5.1.4.4 Description of usage of the testing tool

Description of usage of the testing tool, to allow other partners and external users to test the tool or to test equivalent tools.

5.1.5 Programme and Publication engine for enabling the on demand (UNIVLEEDS, FHGIGD)

5.1.5.1 Test case revision

To prepare for a mock up demo test case for AXMEDIS 2005 conference the P&P section will communicate with on demand via socket. The steps for the test case were revised from TC10.8 (DE2-2-1-TestCases) for the push scenario for content on demand. The following is the basic three steps for the revised test case and demonstration:

1. The on demand <send> socket request to P&P engine (without the formal P&P programme specification) and with the AXMEDIS test content to be requested.
2. The engine processes the request
3. P&P Reply by transmitting the prepared test content to the on-demand.

5.1.5.2 Creation of data for test cases

Utilising the content produced in accordance to WP8, the test cases will make use of a number of selected AXMEDIS objects in order to show and demonstrate the push and on-demand functionalities in AXMEDIS.

For the test case, a selection of the various identified set of basic components will be used as samples for testing the operations (i.e. requesting and sending of AXMEDIS test object) for on demand. The components will be selected from the rich media interactive content developed by the content integrator, XIM, in accordance to WP8.5 and can be deployed across all of the target platforms (internet, mobile, PDA, i-TV, PC) and a range of audio, text, multimedia and visual content provided by the consortium partners which is suitable for multi-channel delivery. We will create or locate eight appropriate AXMEDIS test data in accordance with the content selection guidelines (DE3-1-3) for demonstration activities. The criteria for test selection are components that are not time consuming in terms of download and upload time and the amount of processing time required.

5.1.5.3 Implementation of testing tool

The first prototype of the Programme and Publication Engine will be used for testing the on-demand publications.

This application will be implemented in C++ using wxWindows and xerces libraries as specified in DE3-1-2. This prototype will be developed for as a win32 application only. The engine will be tested using the XML P&P programmes saved using the first prototype of the P&P Editor using the Programme and Publication libraries developed to create new programmes and editing existing programmes (AxPnPRule.lib).

The testing tool will accept a XML programme string from on-demand created utilising the programme libraries. Using sockets to send the string, the P&P engine will validate the string. A failed process returns and error message to on demand. A successful process will return the prepared test content to the on-demand.

5.1.5.4 Description of usage of the testing tool

The test Programme and Publication (P&P) Engine will be available for downloading from the CVS tree and will be developed to execute locally. From the initial CVS tree specification, the engine will be located at AXMEDIS/framework/bin/win32.

A client application representing on-demand will also be downloaded and executed. Using the GUI of the client application, users will make an on-demand request. Results will be returned to the client application to demonstrate the creation of a P&P programme by the client application using the P&P Rule libraries; the sending of the request and the reply to the request.

5.1.6 Content Workflow integration (IRC)

5.1.6.1 Test case revision

Of all the 23 Test Cases involved in integration of external workflow systems with the relevant AXMEDIS Tools, the test cases that will involve exchanges between the native workflow environment and the AXMEDIS tools and thus an opportunity for testing out the integrating plug-ins would seem to be as follows:

- TC 6.1.3 (Edit)
- TC 6.1.8 (Search),
- TC 6.1.9 (Track component)
- TC6.1.22 (Check-in)
- TC6.1.23 (Check-out)

The above can be revised for Content Workflow Integration. These Test Cases were amongst those described in DE2.2.1, Test cases and Content Description.

During AXMEDIS Content Workflow implementation for the demonstrator, these test cases should be considered for revision, in light of any new refinements that might have become necessary resulting from the specification and implementation process. Additionally some new test cases could emerge that will have to be added to the test cases deliverable.

5.1.6.2 Creation of data for test cases

The content repository produced in WP8 will be deployed for this. The test cases will use a number of selected AXMEDIS objects in order to demonstrate the Content Workflow Integration in AXMEDIS. The content used for tests will cover a wide set of content types (video, images, text, audio) and formats.

5.1.6.3 Implementation of testing tool

The first prototype of the AXMEDIS Workflow Integration Plug-ins will be used for testing Content Workflow in full integration with the AXMEDIS tools environment.

The AXMEDIS editor/viewer prototype will be used for testing user interface and content workflow integration, which will also involve scenarios for invoking the AXMEDIS Query support and AXDB.

5.1.6.4 Description of usage of the testing tool

The way to use of the testing process will be described so as to allow other partners and external users to test the AXMEDIS Content Workflow Integration by invoking the above testing scenarios and thus test the workflow integration in AXMEDIS Content production.

5.2 AXMEDIS P2P Cooperative Content Sharing and Production Tool (CRS4)

5.2.1 Virtual Database

5.2.1.1 Test case revision

- TC9.1.1 is confirmed as is.
- TC9.2.5 The Manual publication of an AXMEDIS Object is now performed by the AXEPTool Console.

The steps are the following:

Open the console

Type `publish file://path/file.ext`

The expected results are: the object is committed into the AXDBOUT, the file `file.ext` is copied in the public directory of the AXEPTa PublicationEvent is sent in the `http://hostname/RSS/file.xml` RSS channel.

5.2.1.2 Creation of data for test cases

Collection and/or creation of the data needed for checking the revised test cases.

5.2.1.3 Implementation of testing tool

The AXEPTool is provided with a tool called AXEPTool console, which is used to test the main functionalities with a text-based user interface. All details about the console are described in DE4.4.1

5.2.1.4 Description of usage of the testing tool

The AXEPTool is provided with a tool called AXEPTool console, which is used to test the main functionalities with a text-based user interface. All details about the console are described in DE4.4.1.

5.2.2 DownloadMonitor

5.2.2.1 Test case revision

This test case integrates the TC9.2.9 in DE2.2.1 Test Cases and Content Description, focusing on the DownloadMonitor and related web service testing; In details: the test involves two AXEPTool instances running on different PC (or in one PC) and an automated testing tool.

The user wants to download an object in AXEPToolA from P2P known the object URI (in AXEPToolB).

The testing tool launches the download and allows testers to monitor the download progress. At the end of the process the object is downloaded and copied, according to TC9.2.9.

Download Monitor test is also involved in TC 9.2.12, as part of the test for automatic loading of objects; in particular its web service module is involved in the downloading step.

5.2.2.2 Creation of data for test cases

Data needed for DownloadMonitor testing tool is represented by files made available for http download in an AXEPTool instance.

5.2.2.3 Implementation of testing tool

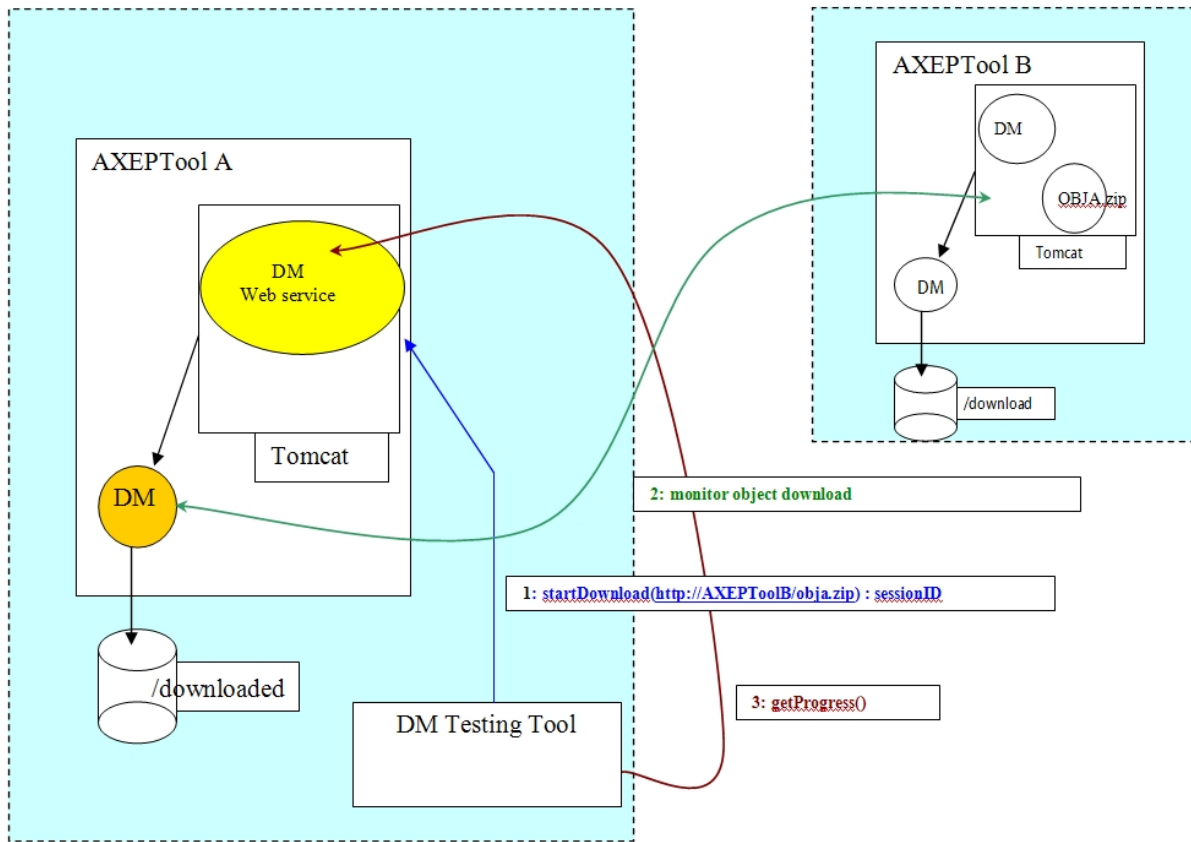
First of all two running instances of AXEPTool are needed named AXEPToolA and AXEPToolB;

Some object files are made available, for example, in AXEPToolB.

So, initial conditions are that some object files are available for http download on AXEPToolB.

Post conditions are that the file is downloaded on AXEPToolA after a time t and during time t the testing tool has monitored the download session showing progress, status, suspending and resuming it: this is made using the web service interface exposed by DownloadMonitor.

Next picture shows the main implemented functionalities of the testing tool.



Testing steps:

1. The testing tool invokes a new object download available at an AXEPToolB related URL; the startDownload () DownloadMonitorWebService method is invoked.
2. The DownloadMonitor starts the download
3. The testing tool performs:
 - i. Repeatedly prints download session info, i.e. download progress status invoking DownloadMonitor WebService methods.
 - ii. Suspend the session showing related status change
 - iii. Resumes download showing status change and progress
 - iv. Reports download final status

5.2.2.4 Description of usage of the testing tool

In order to run this test next requirements are needed:

1. two AXEPTool instances must be running
2. at least one file must be available for download on one AXEPTool instance
3. tester configures the testing tool passing it the URL of the object to download from AXEPToolA to AXEPTool B.
4. check info reported by tool
5. check the downloaded object.

5.2.3 Publishing And Monitoring Objects

5.2.3.1 Test case revision

This test case integrates the TC9.2.3 in DE2.2.1 “Test Cases and Content Description”, focusing the attention in notification tasks of AXMEDIS Objects updates events in the P2P network, in order to perform appropriate operations such as automatic updating actions for these objects. Testing is performed through Publishing And Monitoring Objects Web Service usage, in this way also web service is properly tested.

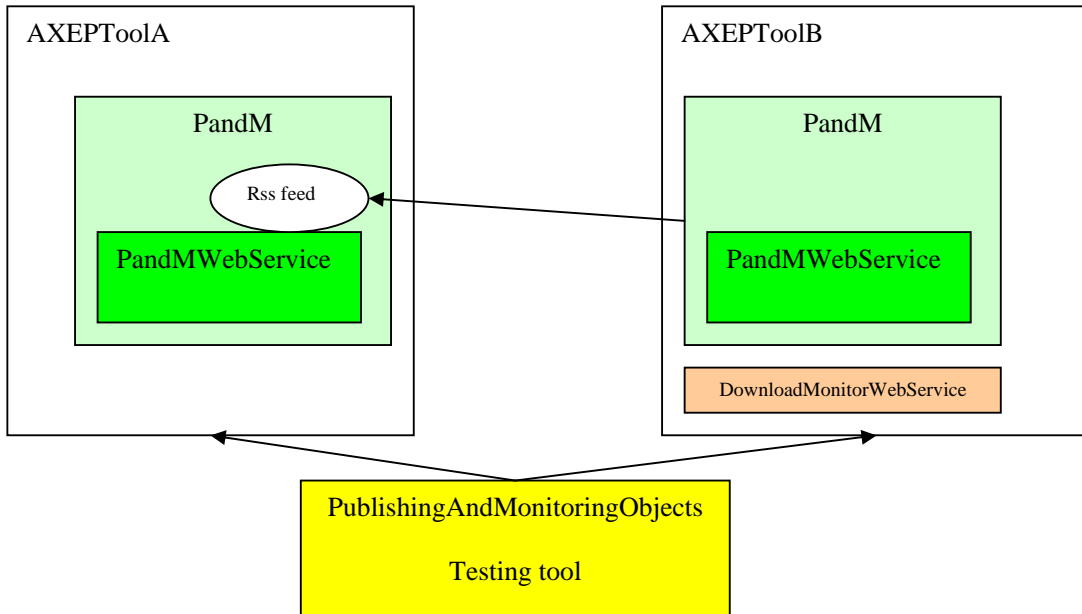
5.2.3.2 Creation of data for test cases

Testers must have two AXEPTool instances running on different machines and a file available for http download on AXEPToolA.

5.2.3.3 Implementation of testing tool

First of all two running instances of AXEPTool are needed named AXEPToolA and AXEPToolB; One object file is made available in AXEPToolA.

Next picture shows the testing scenario:



Testing steps:

1. Testing tool produces a PublicationEvent for a published object in AXEPToolA invoking the producePublicationEvent() method of AXEPToolA PublishingAndMonitoringWebService. These steps involve the creation of a RSS feed file in AXEPToolA related to the event generated.
2. Testing tool invokes an object download on AXEPToolB for the published object available in AXEPToolA. This is made invoking the startDownload() method of DownloadMonitorWebService of AXEPToolB. Successfully finishing the download involves that PublishingAndMonitoringObjects module on AXEPToolB automatically starts polling the RSS feed generated in step 1, going to “listen” for updates on this feed.
3. calling the listEventsFromDate() method on AXEPToolB PublishingAndMonitoringWebService, the testing tool must check for only one event in list.
4. The testing tool produces another PublicationEvent on AXEPToolA for the same object published in step 1. In this way the related RSS file is modified and a new item is added. PublishingAndMonitoringObjects module in AXEPToolB must register this change and saves the event in its internal list.
5. The testing tool invokes the listEventsFromDate() on AXEPToolB checking for two events in list.

5.2.3.4 Description of usage of the testing tool

Testers run two instances of AXEPTool on two PCs configure testing tool with their URIs and run the testing tool, observing reported logs.

5.2.4 Loading Module of AXEPTool

5.2.4.1 Test case revision

Loading activity is supposed to be an iteration through steps of creation of rule, actualization and performing of publication itself. TC9.2 (in DE2.2.1 Test Cases and Content Description) reports several possible cases on loading.

- TC9.2.13 Automatic loading new AXMEDIS Objects with the AXEPTool
 - Loading is driven by rules. There is no interaction between user and module. A specific rule has to be created in order to start the loading.
- TC9.2.14 Manual loading of AXMEDIS Objects with AXEPTool
 - No more applicable. The only way to start loading is through rules.
- TC9.2.15 Creation of a loading rule for the AXEPTool
 - No more concerning Loading Module. Rules are created through the Rule Editor.

Starting from them is reasonable to revise the process in the following steps:

1. create a rule: the rule is created using the rule editor.
2. invoke module: the Rule Engine actualizes the rule and invokes the Loading Module.
3. perform loading: the Loading Module receives data from the Rule Engine and performs the loading.

5.2.4.2 Creation of data for test cases

For the test cases is enough to suppose the rule already actualized and assume the script (managed by Spider Monkey) with data becoming from the assumed rule.

The main interest is on loading new objects and loading new version ones. So are first necessary rules able to configure such situations:

- new objects
- new objects + updated

Also, in order to test the mapping module of the loading module is necessary to create data set with different custom metadata.

5.2.4.3 Implementation of testing tool

Unit tests are implemented using JUnit utility.

Integration tests of the module are composed if a self-contained class driven by a script.

5.2.4.4 Description of usage of the testing tool

Testing tool description should form part of the document DE4.4.1 “Content Sharing and Production on P2P”.

5.2.5 Publication Module of AXEPTool

5.2.5.1 Test case revision

Publication activity is supposed to be an iteration through steps of creation of rule, actualization and performing of publication itself. TC9.2 (in DE2.2.1 Test Cases and Content Description) reports several possible cases on publication.

- TC9.2.1 Creation of a publishing rule for the AXEPTool
 - No more concerning Publication Module. Rules are created through the Rule Editor.
- TC9.2.2 Automatic publication of a selection of objects on the AXEPTool

- Publication Engine has been renamed Publication Module. There are no more editors for the module, so no user interaction will be performed. Objects to publish are selected through rules.
- TC9.2.3 Automatic updating of a modified object on the AXEPTool
 - No more applicable, because already included in standard publication.
- TC9.2.4 Automatic publication of a non protected object on the AXEPTool
 - Confirmed (only renamed Engine in Module). Dataset contains unprotected objects.

Starting from them is reasonable to revise the process in the following steps:

1. create a rule: the rule is created using the rule editor.
 - a. Standard rule, every kind of object
 - b. Updating rule, objects updated; the module will discard old objects.
 - c. Rule for unprotected objects.
2. invoke module: the Rule Engine actualizes the rule and invokes the Publication Module.
3. perform publication: the Publication Module receives data from the Rule Engine and performs the publication.

5.2.5.2 Creation of data for test cases

For the test case is enough to suppose the rule already actualized and assume the script (managed by Spider Monkey) with data becoming from the assumed rule.

The main interest is on publication of new objects and publication of new version ones. So are first necessary rules able to configure such situations:

1. new objects
2. new objects + updated

5.2.5.3 Implementation of testing tool

Unit tests are implemented using JUnit utility.

Integration tests of the module are composed if a self-contained class driven by a script.

5.2.5.4 Description of usage of the testing tool

Testing tool description should form part of the document DE4.4.1 Content Sharing and Production on P2P.

5.2.6 Workflow management in AXEPTool (IRC)

5.2.6.1 Test case revision

Of all the 23 Test Cases involved in the integration of external workflow systems with the relevant AXMEDIS Tools (as described in DE2.2.1), the test cases that will involve Workflow Management in AXEPTool are those that can change the activation list used by the AXEPTool Scheduler. These test cases thus provide an opportunity for testing out the integrating plug-ins that allow workflow management in AXEPTool.

These test cases are as follows:

- TC 6.1.3 (Edit)
- TC 6.1.17 (Change State)
- TC 6.1.1.22 (Check – in)

The above can be revised for the Workflow Management in AXEPTool. These Test Cases were amongst those described in DE2.2.1, Test cases and Content Description.

During AXMEDIS Content Workflow implementation for the demonstrator, these test cases should be considered for revision, in light of any new refinements that might have become necessary resulting from the specification and implementation process. Additionally some new test cases could emerge that will have to be added to the test cases deliverable.

5.2.6.2 Creation of data for test cases

The content repository produced in WP8 will be deployed for this. The test cases will use a number of selected AXMEDIS objects in order to demonstrate the Workflow management in AXEPTool in AXMEDIS. The content used for tests will cover a wide set of content types (video, images, text, audio) and formats.

5.2.6.3 Implementation of testing tool

The first prototype of the AXMEDIS Workflow Integration Plug-ins will be used for testing Workflow Management in AXEPTool in full integration with the AXMEDIS tools environment.

The AXMEDIS editor/viewer prototype will be used for testing the user interface and AXEPTool Workflow Management, which will also involve scenarios for invoking the AXMEDIS Query support and AXDB through the invocation of the TC 6.1.1.22 (Check – in) as a functionality that may in turn be called by TC 6.1.1.17 (Change State).

5.2.6.4 Description of usage of the testing tool

The way to use of the testing process will be described so as to allow other partners and external users to test the AXEPTOOL Workflow Management integration by invoking the above test cases or any others that will emerge to be relevant.

5.3 AXMEDIS Certifier and Supervisor

5.3.1 AXMEDIS Supervisor (FUPF)

5.3.1.1 Test case revision

The test cases that can be revised for AXMEDIS Supervisor are from TC12.2.3.1 to TC12.2.3.7. They were described in DE2.2.1 Test cases and Content Description.

During AXMEDIS Supervisor implementation, these test cases should be considered for revision, as specification and implementation details could make them change. Moreover, some more test cases could appear, not initially considered on the test cases deliverable.

5.3.1.2 Creation of data for test cases

The creation of data sets mainly involves inserting information in the AXCS database related to AXMEDIS users, tools and objects.

The insertion of this information can be done in different ways: manual creation and insertion of SQL statements into a local test database, call to the AXCS database interface classes or call to the AXCS database WS. In any kind of insertion, the information will be equivalent, and should involve user information, tool information and action log information.

5.3.1.3 Implementation of testing tool

AXMEDIS Supervisor testing tools will consist on:

- Unit tests of the operations implemented. JUnit utility can be used for this purpose, calling the corresponding methods.
- Integration tests of the module. In this case, it is needed an initial implementation of other modules to be jointly tested.

Functionality to be tested includes the management of action logs, those received from users and those stored in the database, in order to check that the history of actions is consistent.

Also generation of Supervisor Input Data after some events occur has to be tested.

Integration between AXMEDIS Supervisor and AXMEDIS Certification and Verification also has to be tested, as blocking of users and tools will depend on the communication performed between these two modules.

5.3.1.4 Description of usage of the testing tool

Testing tool description should form part of the documentation associated to AXMEDIS Supervisor. It should test all the functionality described, including integration with other modules.

5.3.2 AXMEDIS Registration (DSI)

5.3.2.1 Test case revision

The test cases that can be revised for AXMEDIS Registration are from TC12.2.1.1 to TC12.2.1.8. They were described in DE2.2.1 Test cases and Content Description.

During AXMEDIS Registration implementation, these test cases should be considered for revision, as specification and implementation details could make them change. Moreover, some more test cases could appear, not initially considered on the test cases deliverable. The registration process is composed by some elements: the Registration Web Service and the client parts that use the related web service. Every single part is involved only in some specific tests, and not necessary in all tests.

5.3.2.2 Creation of data for test cases

The creation of data sets is mainly composed by insertions of information in the AXCS database related to AXMEDIS users, tools and their pertinent data.

The insertion of this information can be done in different ways, each specific for the component to be tested. In order to test the database interface, a set of data have to be prepared and inserted manually (using SQL scripts) and with a simple code that uses methods provided by database interface. The test is successful if tests results are the same all times.

In order to test the web service it has to be written a small client application that uses the web services and returns the expected results.

5.3.2.3 Implementation of testing tool

AXMEDIS Registration testing tools will consist on:

- Unit tests of the operations implemented. JUnit utility can be used for this purpose, calling the corresponding methods.
- Integration tests of the module. In this case, it is needed an initial implementation of other modules to be jointly tested.

5.3.2.4 Description of usage of the testing tool

Testing tool description should form part of the documentation associated to Registration Web Service and AXCS-DB Interface API.

5.3.3 AXMEDIS Certification and Verification (FUPF)

5.3.3.1 Test case revision

The test cases that can be revised for AXMEDIS Certification and Verification are from TC12.2.2.1 to TC12.2.2.3. They were described in DE2.2.1 Test cases and Content Description.

During AXMEDIS Certification and Verification implementation, these test cases should be considered for revision, as specification and implementation details could make them change. Moreover, some more test cases could appear, not initially considered on the test cases deliverable.

5.3.3.2 Creation of data for test cases

The creation of data sets mainly involves inserting information in the AXCS database related to AXMEDIS users and tools.

The insertion of this information can be done in different ways: manual creation and insertion of SQL statements into a local test database, call to the AXCS database interface classes or call to the AXCS database WS. In any kind of insertion, the information will be equivalent, and should involve user information and tool information. Different sets of information are needed to check different functionality, so valid and invalid information has to be used to perform tests.

5.3.3.3 Implementation of testing tool

AXMEDIS Certification and Verification testing tools will consist on:

- Unit tests of the operations implemented. JUnit utility can be used for this purpose, calling the corresponding methods.
- Integration tests of the module. In this case, it is needed an initial implementation of other modules to be jointly tested.

Functionality to be tested includes the management of user and tool information in order to check if user can be certified and his actions verified. Also tools have to be certified and verified.

Integration between AXMEDIS Supervisor and AXMEDIS Certification and Verification also has to be tested, as blocking of users and tools will depend on the communication performed between these two modules.

5.3.3.4 Description of usage of the testing tool

Testing tool description should form part of the documentation associated to AXMEDIS Certification and Verification. It should test all the functionality described, including integration with other modules.

The rest of modules communicating with AXMEDIS Certification and Verification module are needed in order to test the integration.

5.3.4 Trace, reporting and statistic analysis (EXITECH)

5.3.4.1 Test case revision

The test cases that can be revised for Trace, reporting and statistic analysis are from TC12.2.5.1 to TC12.2.5.6. They were described in DE2.2.1 Test cases and Content Description.

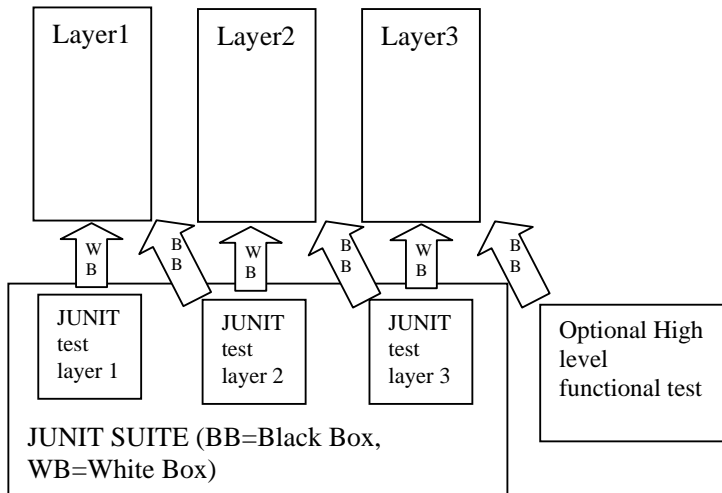
During trace, reporting and statistic analysis implementation, these test cases should be considered for revision, as specification and implementation details could make them change. Moreover, some more test cases could appear, not initially considered on the test cases deliverable.

5.3.4.2 Implementation of testing tool

Instead of a testing tool, regression testing has been implemented by the adoption of JUNIT suite for automating testing.

At each level of the hierarchy in the logical diagram of the system, Unit tests are adopted for testing the layer in a white box manner. The Unit test of the higher level is used to test in a black box manner the underlying layer. A functional test to check in black box manner the last layer will be implemented if needed.

The following schema details this assumption:



5.3.4.3 Description of usage of the testing tool

The testing is completely automated by using JUNIT or the ANT scripts provided with the code.

5.3.5 Accounting Managing and Reporting Tool (EXITECH)

5.3.5.1 Test case revision

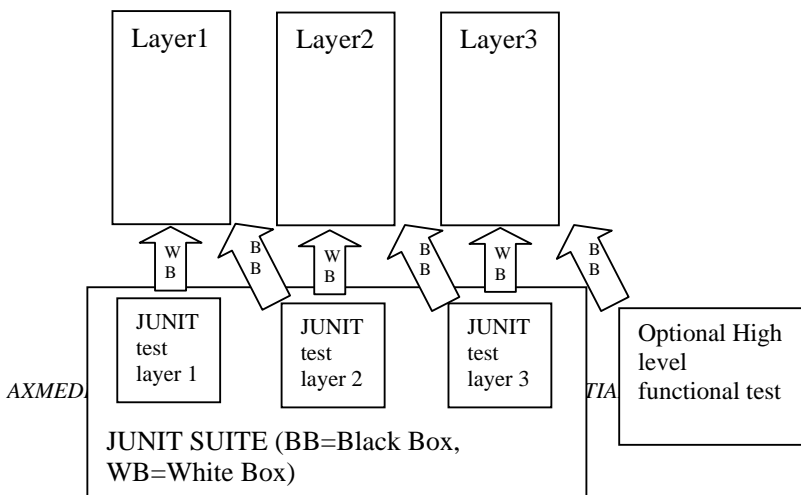
The test cases that can be revised for Accounting Managing and Reporting Tool are from TC12.2.5.7 to TC12.2.5.8 and the 12.4.1. They were described in DE2.2.1 Test cases and Content Description. During Accounting Managing and Reporting Tool implementation, these test cases should be considered for revision, as specification and implementation details could make them change. Moreover, some more test cases could appear, not initially considered on the test cases deliverable.

5.3.5.2 Implementation of testing tool

Instead of a testing tool, regression testing has been implemented by the adoption of JUNIT suite for automating testing.

At each level of the hierarchy in the logical diagram of the system, Unit tests are adopted for testing the layer in a white box manner. The Unit test of the higher level is used to test in a black box manner the underlying layer. A functional test to check in black box manner the last layer will be implemented if needed.

The following schema details this assumption:



5.3.5.3 Description of usage of the testing tool

The testing is completely automated by using JUNIT or the ANT scripts provided with the code.

5.3.6 Protection Tool engine (FHGIGD)

The terminology Protection Tool engine refers to the protection functionality of the AXMEDIS Content Processing (AXCP) engine. At the beginning of the AXMEDIS projects multiple engines for different purposes were considered. The design phase revealed a functional overlap between these individual engines. Hence, the functionality of these different engines is now integrated in the AXCP engine.

For simplicity we also use the terminology Protection Tool engine in some parts of this document. But the reader should be aware that this is just a functional extension of the core AXCP engine.

5.3.6.1 Test case revision

During the implementation of the protection extensions of the AXMEDIS Content Processing (AXCP) Engine, the test cases are continuously revised, as specification and implementation details might change the test cases.

At the implementation stage new insights and experiences might lead to new test cases that have not been considered in the design phase of the AXMEDIS project. The initial set of test cases for the Protection Tool engine were described in DE2.2.1 Test cases and Content Description are TC12.3.1.1 to TC12.3.3.

The current status of the implementation is not finished. Thus, the ongoing implementation of the Protection Tool engine does not allow a thorough revision at this project stage. A final revision of the Protection Tool engine will be possible when the implementation reaches a project status that provides the necessary functionality for the analysis and adjustment of the test cases.

5.3.6.2 Creation of data for test cases

In the specification documents test data was described according to the requirements of the AXMEDIS partners. During the implementation the Protection Tool engine is first tested with this sample data that is available in the specification documents.

A collection of sample data was created according to this specification document. This collection will be the basis for the first tests of the Protection Tool engine prototype during its implementation. More test data will be created and used for testing as the implementation evolves: This allows the previously described thorough revision of the test cases.

Utilising the content produced in WP8, the test cases will use a number of selected AXMEDIS objects in order to show and demonstrate the protection editing and license creation functionalities in AXMEDIS. The data used for test will try to cover a wide set of content types, encryption formats, and license types.

5.3.6.3 Implementation of testing tool

First implementations were performed to test and validate the functionality of the AXCP Rule Editor and Executor. As the complete protection functionality was not available functional stubs were implemented to perform a first series of tests. These tests focused on the passing of parameters. Furthermore, the possibilities and limitations of the integration of additional components and their functionalities into the AXCP Java Script engine was tested and validated.

Summarized, the implementation validated the processing of rules and the passing of parameters into the Java Script engine with dummy extensions.

However, the further implementation of the Protection Tool engine relies on the availability of the other components that it provide the protection and license handling functionality.

5.3.6.4 Description of usage of the testing tool

The Protection Tool engine will be available in the AXMEDIS subversion repository for testing and will be developed to be executed locally. The AXCP Rule Editor and engine and the protection extensions will be located at AXMEDIS/Framework/ruleeditor and AXMEDIS/Framework/ruleexecutor.

The Rule Editor and Executor will be used to test the functionalities of the Protection Rule engine. Using the GUI of the Rule Editor a user will be able to create, edit, and debug Protection rules. The Rule Executor can be used to execute and test existing rules and the protection functionalities of the other components involved.

5.3.7 DRM tools (FUPF)

5.3.7.1 Test case revision

The test cases that can be revised for DRM tools are from TC12.5.1 and subsections to TC12.5.2 and subsections. They were described in DE2.2.1 Test cases and Content Description.

During the implementation of these tools, test cases should be considered for revision, as specification and implementation details could make them change. Moreover, some more test cases could appear, not initially considered on the test cases deliverable.

5.3.7.2 Creation of data for test cases

The data needed for the test cases will depend on the kind of tool, but we will use:

- Licenses expressed in MPEG-21 REL language. They will be provided by:
 - The implemented tools
 - XML editors
 - Other license editors, that follow MPEG-21 REL XML schemas
- Licenses expressed in OMA REL
- Protection information regarding AXMEDIS objects
- Information stored in the AXMEDIS databases, like user information, tool information, object information and action logs.

5.3.7.3 Implementation of testing tool

As DRM tools are a set of tools, different kinds of test have to be performed:

- Unit tests of the operations implemented. These tests will depend on the implementation language.
- Integration tests of the DRM tools. In this case, it is needed an initial implementation of the tools.
- Integration tests with other modules. In this case, it is needed an initial implementation of other modules to be jointly tested.

5.3.7.4 Description of usage of the testing tool

Testing tool description should form part of the documentation associated to the different DRM tools.

Features that will be tested by these tools:

- Creation and management of licenses
- Caching and management of protection information
- Integration and communication between the different modules

AXMEDIS Framework Integration and Maintenance

5.4 Set up of the AXMEDIS framework for continuous integration of AXMEDIS components (EXITECH)

This task will be managed by EXITECH, performed by all partners involved according to their skill and related tools of which they are responsible. The integration work will permit to the building of several demonstrators and of the trials developed by the Take up actions.

In order to allow an integration of the AXMEDIS framework a central repository will be created. By using a VCS (Version Controlling System) the partners involved into the Framework development will submit their work periodically. The AXMEDIS framework repository should be set up on a dedicated hardware.

The VCS system will be chosen in order to allow the version control, the folder access control and the possibility to revert to a previous configuration. The submission process has to be executed by following specific rules in order to guarantee the validation and the integration of the repository content.

The VCS responsible will also performs the activities of resolving conflicts and supporting the other partners during the submission process. This will lead to refine the guidelines and also the stubs of the modules.

As soon as the prototypes of the AXMEDIS software components are available, they will be inserted in the component database for the integration. The submitted components will be verified periodically from the formal and functional point of view.

5.5 Regression and integration testing (EPFL)

These two test phases are rather distinct and will be described in two separate subsections.

5.5.1 Regression testing

Regression testing is a process which tests a system or a tool to ensure that it still functions as expected / as per specification. The reason for this renewed testing activity is a modification of an implementation within the program. For example; a major upgrade in the code or an important bug fix; a new hardware platform or a major release of the operating system. In addition, when a development team releases a new version of its database, a comprehensive regression test plan needs to be developed and completed to ensure that the reports, scripts, (remote) procedure calls and user options, are all functioning as expected.

Regression testing should also test the revised software by simulating its operational environment to be sure that all systems and interfaces still operate properly.

Regression testing will be conducted in AXMEDIS according to a Test Plan. Testing of the different units or components to verify their global functionality will be done by using the test cases defined in WP2 and the data set initially produced and collected in WP8. Major factors and strategies that will be followed to perform regression testing are the following:

- An as much as possible automated process will be created (with the help of each module author) based on test cases and data set mentioned above. For instance in section 5 some components are described for which the JUNIT suite for automating testing is adopted.
- Adequate coverage without wasting time will be considered when defining and conducting regression tests.
- A regression test for concerned modules will be run each time a major bug fix is handled. The bug itself might be fixed but the fix might create other bugs.
- A regression test for concerned modules will be run each time modifications are made following integration tests or periodic verifications (see section 4.7).

- If the bug fix is not tested by one of the tests in the process, a new regression test covering the functionality of this bug should be established (to be sure that this is fixed in general or not coming back).
- If two or more tests are similar, the less effective will be identified and eliminated (duplications are quite common when more than one person is writing code).
- Effects of changes in the program on memory and CPU usage should be traced.
- ...

As said, with the help of all development teams, a plan of tests will be developed made up of WP2 test cases that can be run every time a new version of the program is built. Automated tests, as well as test cases involving boundary conditions and timing will be defined in the plan.

The regression test library will be periodically reviewed to eliminate redundant or unnecessary tests. This could be done for instance every third testing cycle.

Regression test plan and reports for each unit or component has to be maintained by each responsible and periodically updated on the portal. In particular the plan should include the test cases concerning the component, if the regression test is automated (how) or a short description of the testing procedure, functionality covered by the test. Reports should include results, test cases that have been added or deprecated, major observed changes in memory and CPU usage.

5.5.2 Integration testing

Integration testing is the phase of software testing in which individual software modules are combined and tested as a group. Integration testing takes as input modules that have been checked already by individual unit testing, groups them in larger aggregates, and applies tests defined in the integration test plan to those groups.

In integration testing simplest form, two units that have already been tested are combined into a component and the interface between them is tested. A component is an integrated group of more than one unit. In a more realistic scenario, many units are combined into components. Eventually all the modules making up a process are tested together.

Integration testing identifies problems that occur when units are combined. By using a test plan that requires testing each unit before combining units, any errors discovered when combining units are likely related to the interface between units.

Integration testing can proceed in a number of different ways, which can be identified as top down or bottom up. In top down integration testing the high-level control routines are tested first, possibly with the middle level control structures present only as stubs. Subprogram stubs essentially are incomplete subprograms, which are only present to allow the higher-level control routines to be tested.

The other major way for integration testing is bottom up integration testing where an individual module is tested from a test set. Once a set of individual modules has been tested they are then combined into a collection of modules, or builds, which are then tested by a second test set. This process can continue until the build consists of the entire application.

In AXMEDIS a combination of top-down and bottom-up testing would be used. Since AXMEDIS is a large software project being developed by a number of teams at different levels (from signal processing to applications), the teams or individuals would define and conduct bottom-up testing of the modules which they were implementing before releasing them to teams developing more high level or “integration” modules (larger applications) which would assemble them together for top-down testing.

A description of suitable integration test procedure should be included with the description of each module and application.

5.6 Optimisation of AXMEDIS components (DSI)

Optimisation of AXMEDIS Components will be done to improve the quality of their results and to remove the potential integration problems.

The optimization phase will analyse the whole production process to see which are the components to be optimized to reduce the overall production time taking into account the use scenarios.

An evaluation of when and how many times some basic functions are used will help to identify the points where an optimization is needed.

The optimization will not consider only the execution speed but it will take into account also the time used by humans to write scripts for the production process. In this case recurring activities will be identified and shortcuts will be introduced.

6 Bibliography

Testing utilities

- JUnit for Java: <http://www.junit.org/>
- CppUnit for C++: <http://sourceforge.net/projects/cppunit/>
- PHPUnit for PHP: <http://www.phpunit.de/en/index.php>

Versioning control system

- Subversion: <http://subversion.tigris.org/>
- Subversion book: <http://svnbook.red-bean.com/en/1.0/svn-book.html>

Verification and validation

- Software Engineering, 6th Edition, Ian Sommerville, Addison Wesley.
<http://www.comp.lancs.ac.uk/computing/resources/IanS/SE6/index.html>

7 Glossary

Concept	Description
Peer review	Validation technique which comprises the revision of software in order to quickly find errors
Inspection	Detailed examination of component step-by-step