Automating Production of Cross Media Content for Multi-channel Distribution

www.AXMEDIS.org

DE9.2.4
Integrated Prototype of automating content production and formatting into CMSs of integrators

Version: 1.5
Date: 16/04/2007
Responsible: DSI (Ivan Bruno, Pierfrancesco Bellini) (revised and approved by coordinator)

Project Number: IST-2-511299 Project Title: AXMEDIS Deliverable Type: Public Visible to User Groups: Yes (only the document and the demonstrators) Visible to Affiliated: Yes (only the document and the demonstrators) Visible to Public: Yes (only the document and the demonstrators)

Deliverable Number: DE9.2.4 Contractual Date of Delivery: M30 Work-Package contributing to the Deliverable: WP 9.2 Nature of the Deliverable: Prototype and this short report Author(s): DSI plus all

Abstract:
This deliverable is related to the demonstrator of WP9.2 that has been completed in this period by integrating the connections with several CMS and content factories.
Keyword List: AXMEDIS content processing, integration with legacy CMS.

Table of Contents

1 EXECUTIVE SUMMARY AND REPORT SCOPE 3
2 INTRODUCTION 3
3 PROTOTYPE DESCRIPTION 4
4 AXCP AREA FOR CONTENT PRODUCTION AND PROCESSING 6
5 WEBSERVICE APPROACH FOR CONTENT MANAGEMENT 8
5.1 CONTENT PUBLICATION 9
5.2 CONTENT RETRIEVAL 10
6 ODBC APPROACH FOR CONTENT MANAGEMENT 10
6.1 CONTENT PUBLICATION 10
6.2 CONTENT RETRIEVAL 11
7 REAL CASES AND PROTOTYPES 12
7.1 THE ILABS LEARN EXACT CMS 12
7.1.1 Description of the CMS 12
7.1.2 Description of prototype and interfacing with AXCP Area 12
7.1.3 AXCP Scripts and their description 12
7.2 CONTENT GATHERING FROM FILE SYSTEM PROTOTYPE (ANSC) 20
7.2.1 Description of the CMS and scenario 20
7.2.2 AXCP Scripts and their description 22
7.3 CRAWLING FROM CMS USING SEARCHBOX TOOL (ANSC) 33
7.3.1 Description of the CMS and scenario 33
7.3.2 Content gathering prototype description 34
7.3.3 AXCP Scripts and their description 35
7.3.4 Content Formatting using SMIL (ANSC) 45
7.4 XAURA - TISCALI 53
7.4.1 Description of the CMS 53
7.4.2 Description of prototype and interfacing with AXCP Area 54
7.4.3 AXCP Scripts and their description 56
7.5 DIGITAL MEDIA PLATFORM (DMP) - HP 62
7.5.1 Description of the Digital Media Platform 62
7.5.2 Description of prototype and interfacing with AXCP Area 63
7.5.2.1 AXCP Interface and Results 66
7.5.2.2 Demonstrator fact sheet: 67
7.6 CONTENT AND FORMAT GATHERING FROM FILESYSTEM (XIM) 68
7.6.1 Description of the CMS and interfacing with AXCP 68
7.6.2 AXCP Scripts and their description 70
7.7 (WWW.LACENTRALDIGITAL-COM) - SDAE 70
7.7.1 Description of the CMS 70
8 BIBLIOGRAPHY 72

1 Executive Summary and Report Scope

This deliverable is the accompanying document for the Prototype having the name common to the deliverable.

2 Introduction

WP9.2 has been improved in this period by producing a first real prototype integrated with AXMEDIS Content Processing Tools.

Content providers, content integrators, aggregator and distributors are using specific technology to compose and format content. These technologies, often coming from one big software solution, could be good in doing some task but poor or missing in other tasks.
For this reason the composition and formatting algorithms and tools conceived in the AXMEDIS framework could be reused inside the proprietary CMS solution to enhance the content workflow. For example a content integrator (content ingestion and composite objects production) can compose its own objects stored in the content databases, using algorithms of AXMEDIS or a content distributor can integrate inside its formatting process such algorithms.
Integrating the AXMEDIS Content Processing Area (see the Figure below) inside the proprietary CMS means having a set of digital content processing tools to:

- efficiently collect the components needed for producing composed objects, using a direct access and query to the database;
- find/produce alternatives/adaptations for the components that present potential distribution problems (too big files);
- structure the components, highlighting the relations among them;
- bind the structure of content and content collection to some presentation and formatting styles;
- format/adapt content for a variety of channels,
- protect digital content according to MPEG-21 models and related AXMEDIS tools, including license production;
- process and produce metadata and managing the estimation of fingerprint and descriptors;
- import from other sources (e.g., existing CMSs) content and metadata to be used for the production process.

The tools provided are also capable to automate the ingestion of content and the related metadata (images, video, audio, documents, etc.), already available in the Content Management Systems (CMS) of the factory.

Moreover the prototype presents the following additional functionalities with respect to the AXMEDIS Content Processing Tool:

- Connection to access in reading information (resources and metadata) from several different databases and protocols by means of Focuseek crawler, see DE3.1.2.2.8;
- Connection to access in reading information from Lobster database and eXact database of ILABS with an extension of Focuseek crawler;
- Connection to access in reading information from XAURA database in MySQL of TISCALI with Focuseek crawler;
- Connection to access in reading and writing information with HP database with XML interface developed for the AXCP in the prototype. HP DMP is well suited for video media, especially in a IPTV or WebTV environment, where the distribution is performed through IP Networks.
- Connection to access in reading and writing information XML database of ANSC with WSDL interface developed for Focuseek, and also with direct connection with WSDL from AXCP script;
- Connection to access in reading and writing information MySQL database of AFI with Focuseek, and also with direct connection with WSDL/FTP from AXCP script;

The following table summarises the technologies used in the CMS of the partners for content query, content access and content upload:

	Content Query		Content Access			Content Upload				
						upload control		file transfer		
CMS	**WS**	**DB**	**http**	**ftp**	**net fs**	**WS**	**http**	**ftp**	**http**	**net fs**
ILABS	X		X			X		X		
TISCALI	(X)	X	X		X	X				X
HP		X	X			X	X	X	X	
ANSC	X*[1]	X	X	X		X*	X	X	X	
AFI		X	X	X			X	X		
XIM			X	X			X	X	X	
SDAE	X		X				X		X	

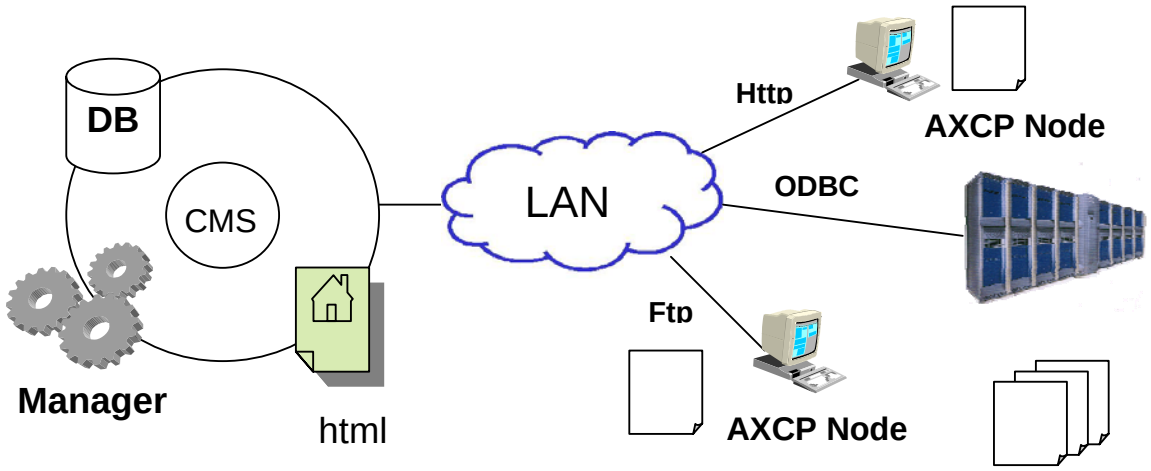
WS= Web Service, DB=Database, net fs=network file system

3 Prototype Description

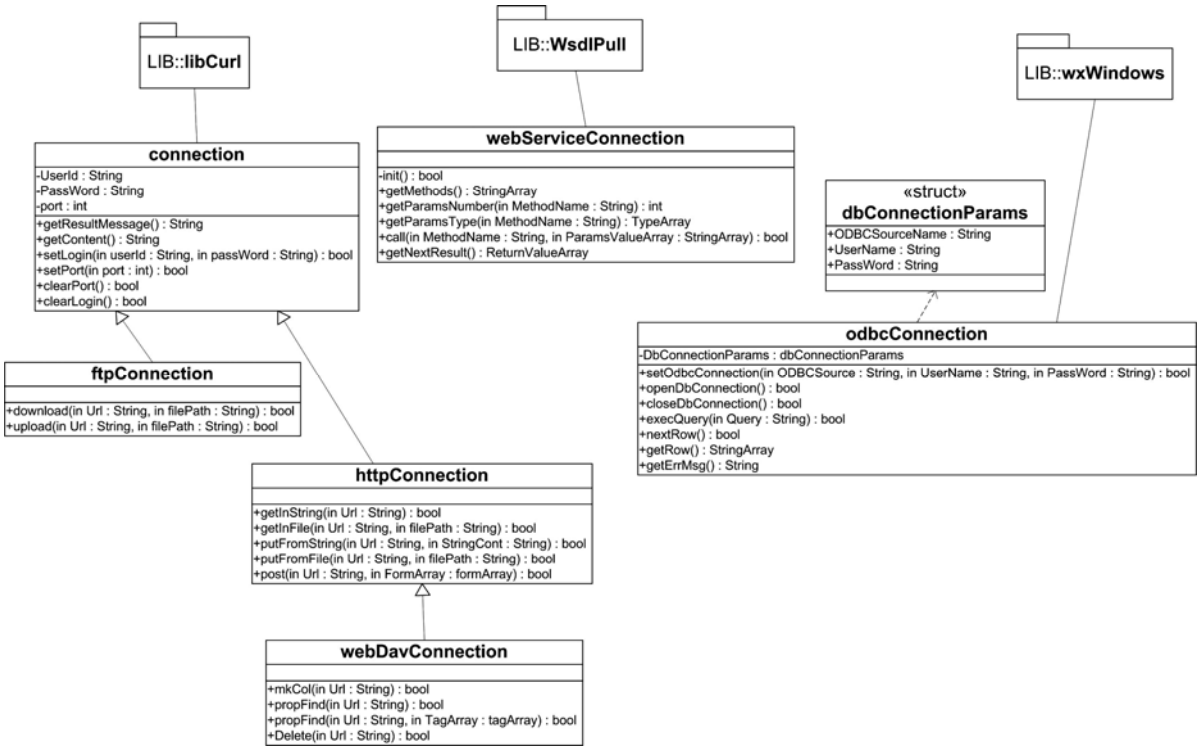
The added technologies in the prototype are:

- Possibility of using into the AXMEDIS script language FTP commands to read and write files;
- Possibility of using into the AXMEDIS script language WSDL actions to access to Web Services in general. It can be used for accessing to database or for other means;
- Possibility of using into the AXMEDIS script language HTTP commands to read and write information according to that protocol;
- Possibility of using into the AXMEDIS script language access to the file system resources, copy, move, delete, etc. of files;
- Possibility of using into the AXMEDIS script language access to the operating system API, such as those related to put in execution processes;
- Possibility of using into the AXMEDIS script language access to XML files for reading and modifying them. Loading/saving XML files in simple and automatic manner.
- Possibility of using into the AXMEDIS script language commands related to compress/ZIP uncompress/unzip files.
- Possibility of using into the AXMEDIS script language SQL commands for ODBC databases. This allows to load and save content in that manner.
- Possibility of using into the AXMEDIS script language to access in reading and writing information of databases via WEBDAV protocol.

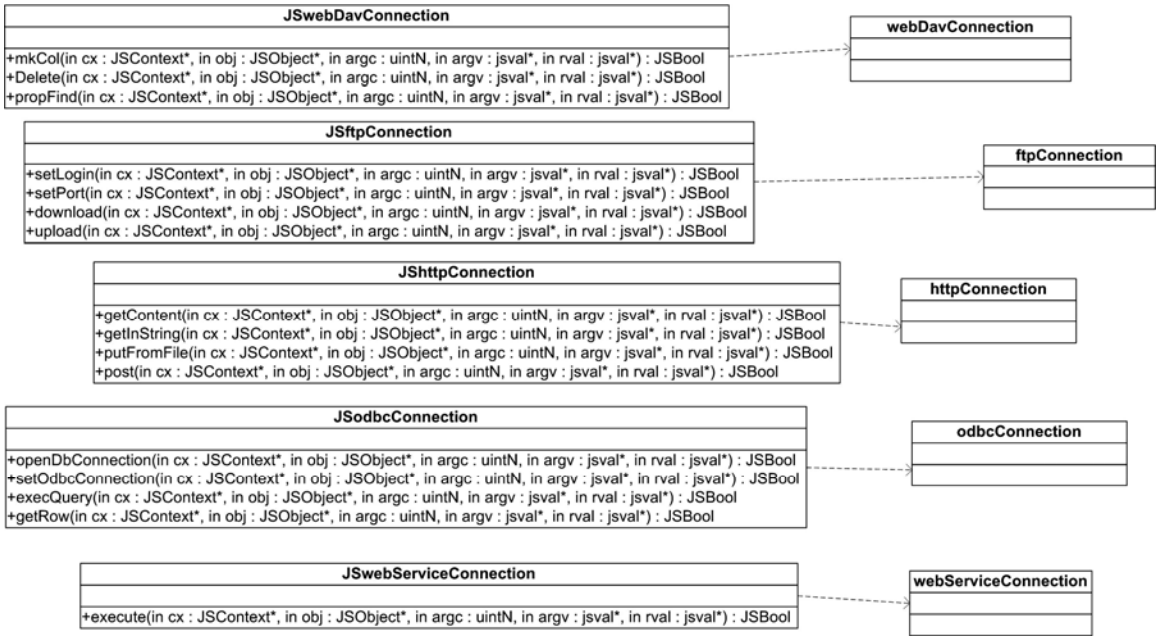
[1] Still under development



All mentioned functionalities are provided by the following set of classes that model different protocols and database access: FTP and HTTP protocols, WebService, ODBC and WebDAV.



They were wrapped by the corresponding classes in the AXMEDIS Script Language based on Javascript. In this way, they can be mixed to implement different mechanisms and models for content publication and retrieval.



In this terms, the support to the content publication and retrieval to/from CMS allows using the AXCP Area to perform content composition, formatting, migration of content to AXMEDIS and more in general content processing using a script based approach and a GRID infrastructure to speed content production..

4 AXCP Area for content production and processing

The AXCP area allows processing both raw digital resource and AXMEDIS objects. The first example of script shows the adaptation of a raw resource (the “axmedis_logo.png” file). It is resized in several way and converted in the jpeg format.
The class AxResource is used to manage the digital resource, whereas the ImageProcessing class allows accessing to the functionality for image processing (Resize and Conversion functions).

```
print("Creating adapted image");
var imgResource = new AxResource();
var outResource = new AxResource();
imgResource.load("C:\\axcptest\\axmedis_logo.png");
for(i = 0; i<n; i++)
{
    ImageProcessing.Resize(imgResource,32+i,32+i,false,outResource);
    print("Resized to" + (32+i) + " x "+ (32+i));
    ImageProcessing.Conversion(outResource,"image/jpeg",outResource);
    outResource.save("C:\\axcptest\\Example100_"+i);
}
print("End of job");
```

The next script shows a way to create an AXMEDIS object embedding a single resource. This example shows a possible basic step for migrating into AXMEDIS format. The raw resource locate at resourceURI is embedded into a new AXMEDIS object (by the addContent method), successively the title is set as Dublin Core metadata. Finally, the new AXMEDIS object is uploaded into the AXMEDIS Database

```
var resource = new AxResource();
resource.load(resourceURI);
var axObject = new AxmedisObject();
axObject.addContent(resource);
var title = resource.contentID;
axObject.getDublinCore().addElement("dc:title", title);
axObject.uploadToDB();
```

In the following a script for composition and adaptation of AXMEDIS objects is reported. The script produces different AXMEDIS objects constituted of a video and at most `maxDoc` documents. The list of videos and documents are provided by two *AxSelection* objects: `videoSel` and `docSel`. They are used to retrieve the array of AXOIDs from the database creating instances of *AxmedisObject*. Each instance is able to retrieve the corresponding video or document. The `maxDoc` parameter is a global variable and it is used to store and fix the maximum number of documents to retrieve from the documents selection and associate with a video resource. The Dublin Core metadata are generated by the `createDublinCoreTitle()` function for each new composite AXMEDIS object associated with the `axObj` instance of *AxmedisObject* by adding an *AxDublinCore* object with the title. The video resource adaptation is performed by the *VideoProcessing* plugin object by means of the `Converting` method: each video is converted in the format specified by the `outFormat` rule parameter. The *VideoProcessing* plugin is a dependency for the script and has to be included in the XML description in order to instantiate the *AxCPPlugin* object in charge to load the plugin and provide its functions. All new AXMEDIS objects are successively formatted using the *SMILTemplate* and *SMILStyle* objects to find the best template and style to be used as input to the *SMILFormat* object. Finally, the SMIL description is added as resource to the AXMEDIS objects and then they are stored in the AXMEDIS Database by the `uploadToDB()` method of *AxmedisObject* class.

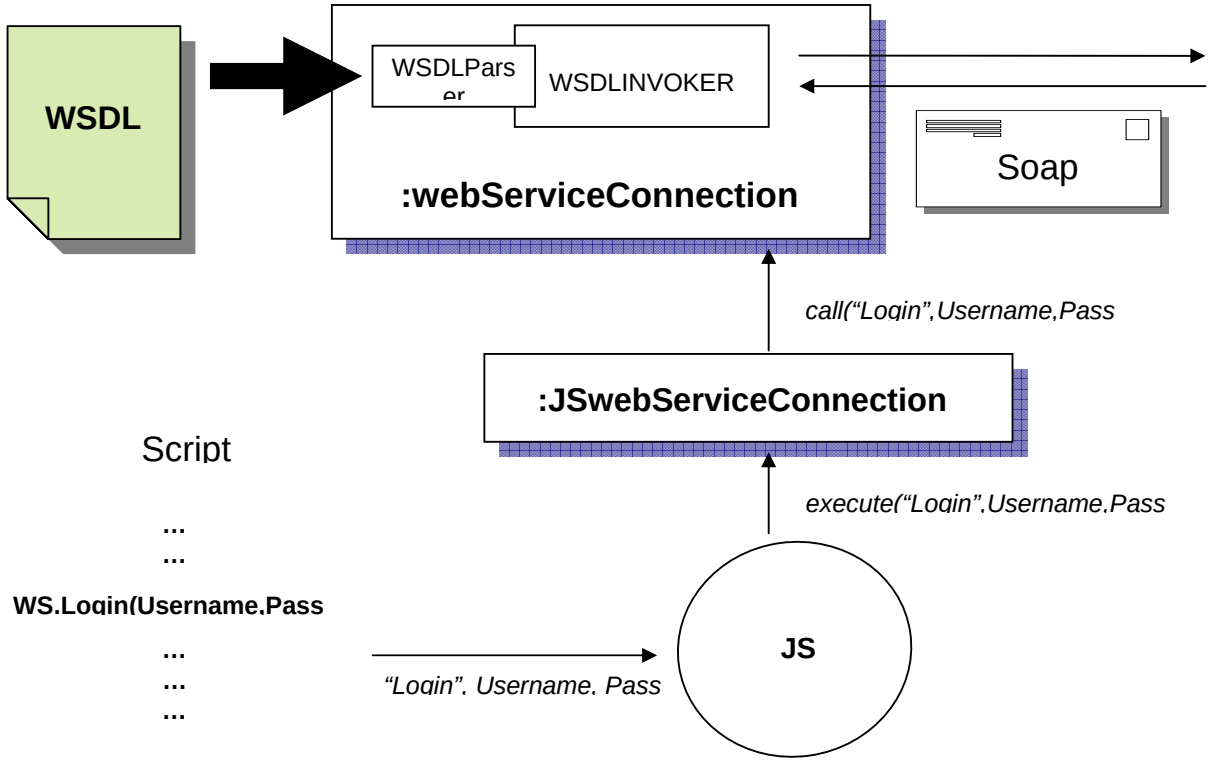
```
function getTitle(axObj)  {
      var dc = axObj.getDC();
      return dc.getElement("dc.title");
}

function createDublinCoreTitle(axObj, title) {
      var dc = new AxDublinCore();
      dc.addElement("dc:title", title);
      axObj.addMetadata(dc);
}
/* Start of script: videos and docs arrays collect results of two selections
objects. They provide AXMEDIS Objects to use in the composition process */

var video_axoids = videoSel.resolve();
var doc_axoids = docSel.resolve();
var template = new SMILTemplate();
var style = new SMILStyle();
var formatter = new SMILFormat();
for(v in video_ axoids) {
      var axObj = new AxmedisObject();
      var obj = new AxmedisObject (video_ axoids [v]);
      var nDoc = 0;
      var title= GetTitle(obj);
// extraction of the video resource reference
      var vRes = obj.getContent();
// adaptation of the resource
      VideoProcessing.Converting(vRes,outFormat, vRes);
// embed the adapted object
      axObj.addContent(obj);
      for (d in doc_ axoids) {   // adding maxDoc documents
            axObj.addContent(doc_ axoids [d]);
```

```
            nDoc++;
            if(nDoc>maxDoc)
                  break;
      }
      createDublinCoreTitle(axObj, "Collection of  "+title):
      var template = new SMILTemplate(axObj);
      var style = new SMILStyle(axObj);
// the formatter return an AxResource with the SMIL
      smilRes = formatter.createSMIL(template,style);
      axObj.addContent(smilRes);
// the Axmedis Object is stored into the DB
      axObj.uploadToDB();
}
```

These scripts can be combined to the following examples and that allow accessing to contents in the own CMS. In this way, such scripts can be used to build a scripted procedure for content adaptation or migration to AXMEDIS.

5 WebService Approach for content management

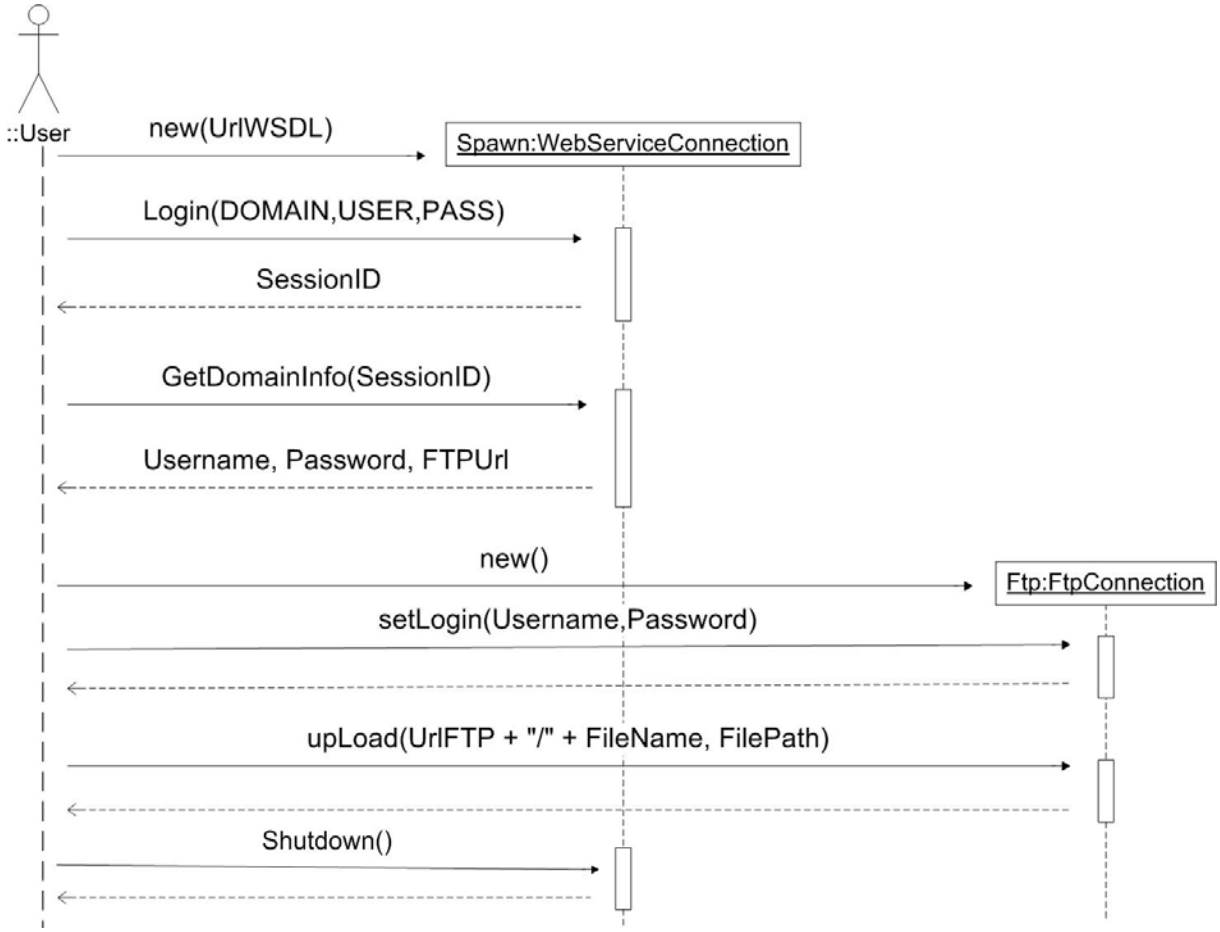
In this section, examples of scripts for content management based on WebService are reported. The WebServiceConnection class in the AXMEDIS Script Language allows implementing several accesses to own CMS based on WSDL (Web Service Description Language). In this way, content owners can manage the migration of content from their CMS to AXMEDIS.
The following picture details the approach based on a WSDL description. The WSDL is used to initialize an instance of the webServiceConnection class. The internal wsdl parser allows extracting names and signatures of methods. In this way the webServiceConnection object is populated when is built automatically and dynamically by means the information provided by the wsdl description. The communication is based on SOAP messages.



In the following some scripts have been reported. When possible, the sequence diagram is associated with the corresponding implementation in Javascript.

5.1 Content Publication

The example shows the publication of content. The Javascript code implements the following Sequence diagram, that we suppose being associated to the WebSevices provided by the wsdl defined at the SpawnWdlUrl
The WebServices functions are described by the wsdl (SpawnWdlUrl). All methods of the Spawn instance, their names and signatures are automatically and dynamically generated by means the information inside the wsdl description.




```
var Spawn = new WebServiceConnection(SpawnWsdlUrl);

var SessionID = Spawn.Login(DOMAIN,USER,PASS);

var GetDomainInfoResponse = Spawn.GetDomainInfo(SessionID);

var Parser = new XMLparser();
Parser.Set(GetDomainInfoResponse);
var UserName = Parser.GetValue("UserName");
```

```
var PassWord = Parser.GetValue("PassWord");
var FTPDestinationUrl = Parser.GetValue("Url");

var FileName = Spawn.InsertPackageRequest(SessionID,PackID,ObjType);

var Zipper = new ZipArchiver();
var PIFPath = "C:/H725PIF.zip";
Zipper.createNewZipArchive(PIFPath);
Zipper.addFileToZip("C:/manifest.xml");
Zipper.addFileToZip("C:/H725.bin");

var Ftp = new FtpConnection();
Ftp.setLogin(UserName,PassWord);
Ftp.upload(FTPDestinationUrl+"/"+FileName,PIFPath);

Spawn.NotifyUpLoadDone();
Spawn.Shutdown();
```

5.2 Content retrieval

The example shows the content retrieval. The WebServices functions are described by the wsdl (SeahorseWdlUrl).

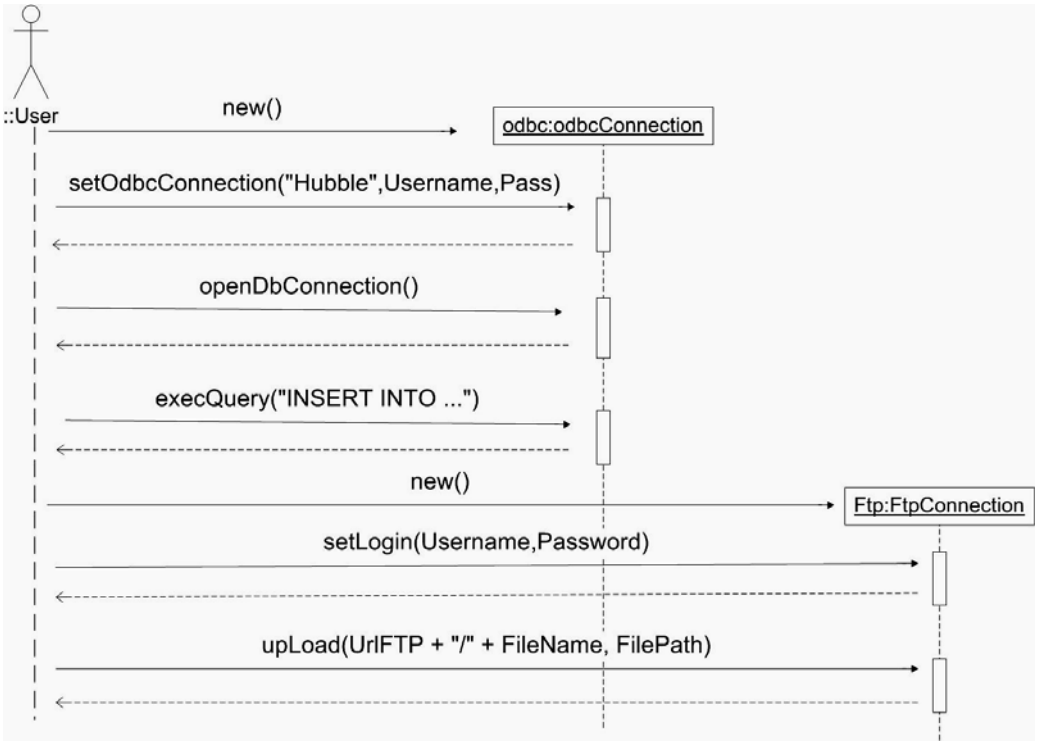
```
var Spawn = new WebServiceConnection(SeahorseWsdlUrl);

var SessionID = Spawn.Login(DOMAIN,USER,PASS);

var GetDomainInfoResponse = Spawn.GetDomainInfo(SessionID);

var Parser = new XMLparser();
Parser.Set(GetDomainInfoResponse);
var DomainUrl = Parser.GetValue("domainurl");
var CourseBasePath = Parser.GetValue("coursebasepath");
var FolderName = Spawn.FindPackage(SessionID,Query);

var Url = DomainUrl + "/" + CourseBasePath + "/" + FolderName;

var Http = new HttpConnection();
Http.getToString(Url+"/manifest.xml");

var Content = Http.getContent();

Spawn.Shutdown();
```

6 ODBC Approach for content management

In this section, examples of scripts for content management based on ODBC are reported. The OdbcConnection class in the AXMEDIS Script Language allows implementing several accesses to own CMS based on ODBC. In this way, content owners can manage the migration of content from their CMS to AXMEDIS.

6.1 Content Publication

The javascript code implements the following Sequence diagram:



The script opens a connection via ODBC, *odbc* object is the instance of *OdbcConnection* class that allows opening, closing and making query to the Database. The publication of the content is made by means the FTP protocol. This is possible by means the *FtpConnection* class.

```
var odbc = new OdbcConnection();
odbc.setOdbcConnection("Hubble",UserName,PassWord);
odbc.openDbConnection();

odbc.execQuery("INSERT INTO DocumentsResources VALUES
('unibo199Bltr','Xeron199B','Tiziano
Casella','http://hubble.astrolabs.it/xeron199B.pdf')");

odbc.closeDbConnection();

var ftp = new FtpConnection();
ftp.setLogin(UserName,PassWord);
ftp.upload("ftp://hubble.astrolabs.it",DestinationPath);
```

6.2 Content retrieval

The script opens a connection via ODBC, *odbc* object is the instance of *OdbcConnection* class that allows opening, closing and making query to the Database. The content retrieval is made by means the HTTP protocol. This is possible by means the *HttpConnection* class.

```
var odbc = new OdbcConnection();
odbc.setOdbcConnection("Hubble",UserName,PassWord);
odbc.openDbConnection();
```

```
odbc.execQuery("SELECT Url FROM DocumentsResources WHERE
DocID='unipi9D12Cdar'");

var Url = odbc.getRow();

odbc.closeDbConnection();

var http = new HttpConnection();
http.getInString(URL);
http.getInFile(Url,FilePath);
```

7 Real cases and prototypes

In this section real cases of automatic content production and formatting into CMS are reported.

7.1 The ILABS learn eXact CMS

As a first real case of use of the content processing area with a legacy CMS the ILABS learn eXact CMS has been used.

7.1.1 Description of the CMS

learn eXact® (LEX) is a proprietary environment developed in GIUNTI Labs during the past ten years and devoted to cover the whole value chain of e-learning publishing and fruition in full compliancy with international standards. Basically it comprises the following elements:

- *Packager*© - the authoring and packaging facility of the *learn eXact*® platform;
- *Lobster*© - the data manager that handles all published content of the *learn eXact*® platform;
- *Siter*© - the e-learning management module of the *learn eXact*® platform;
- *Glove*© - the e-learning rendering and fruition module of the *learn eXact*® platform;
- D-Repository - any repository compliant to the *Digital Repository* standard (WebCT, BlackBoard…)
- Back-end - any existing backend compliant with the set of API exposed by *learn eXact*® platform.

The most relevant aspect in the D-Repository connection is the possibility to use their content either as source or destination of the packaging process as all that is needed is the compatibility with IMS packaging standard. This feature was developed originally to grant users the possibility to continue exploiting the already in place content sources or distribution facilities and infrastructure even in the occasion of a shift in either production or delivery framework. The same principle brought to the development of a set of API for interconnecting with external back-ends; this is actually the way that will be followed to exploit AXMEDIS empowered production and distribution value chain.

7.1.2 Description of prototype and interfacing with AXCP Area

In this first prototype have been developed and tested some generic scripts allowing to perform the basic operation on the CMS: insert, update, query, remove of packages stored as PIF files (Package Interchange Format)

7.1.3 AXCP Scripts and their description

The script defines the class *LobsterConnection* allowing to access to the CMS via the Lobster WebService. The class provides the following services:

new LobsterConnection(wsdl, domain, user, passw)
creates a LobsterConnection object, it connects with the lobster WS, it performs the login and retrieves domain information
Example:

```
var l=new LobsterConnection("http://host.org/lobster.wsdl", "domain", “user", "passw")
```

method void insertPackage(packid, objectType, pifLocalFilepath)
allows to insert the package into the CMS giving the id of the package, the type of object to be uploaded ("Course" or 0, "LearningObject" or 1) and the local filename for the package
Example:

```
l.insertPackage("test","Course","C:/test.zip")
```

method void updatePackage(packid, pifLocalFilepath)
allows to update a package into the CMS giving the id of the package, and the local filename for the updated package.
Example:

```
l.updatePackage("test","C:/test-update.zip")
```

method result findPackage(XPATHQuery, start, size)
allows to perform an XPATH query on the CMS, the start and size values indicate the results to be returned the return value is an array with the xml parsed results
Example:

```
metad=l.findPackage("manifest[@identifier=\"test\"]/metadata", 0, 100)
var title=metad[0].childElement("lom").childElement["general"].childElement("title").getText()
```

method void getPackageFile(packid, remoteFile, localFile)
allows to download files of a package from the server, the file is stored in the localfile
Example:

```
l.getPackageFile("test","imsmanifest.xml","C:/packs/imsmanifest.xml")
l.getPackageFile("test","res90800.htm","C:/packs/res90800.htm")
```

method void deletePackage(packid)
allows to delete a package
Example:

```
l.deletePackage("test")
```

methods logout(), shutdown()
allows to logout from the server, note that no multiple logins of the same user are allowed.
Example:

```
l.logout()
```

Note: The script depends on the jsxml

The following is the script used to test the interaction with the CMS, it tests:

- the query support searching for packages with “Paul” in the title and for each result obtained the imsmanifest.xml is downloaded from the server
- the insertion of a new package into the CMS (from c:\test.zip pif file)
- the update of the same package (from c:\test-update.zip)
- the removal of the package

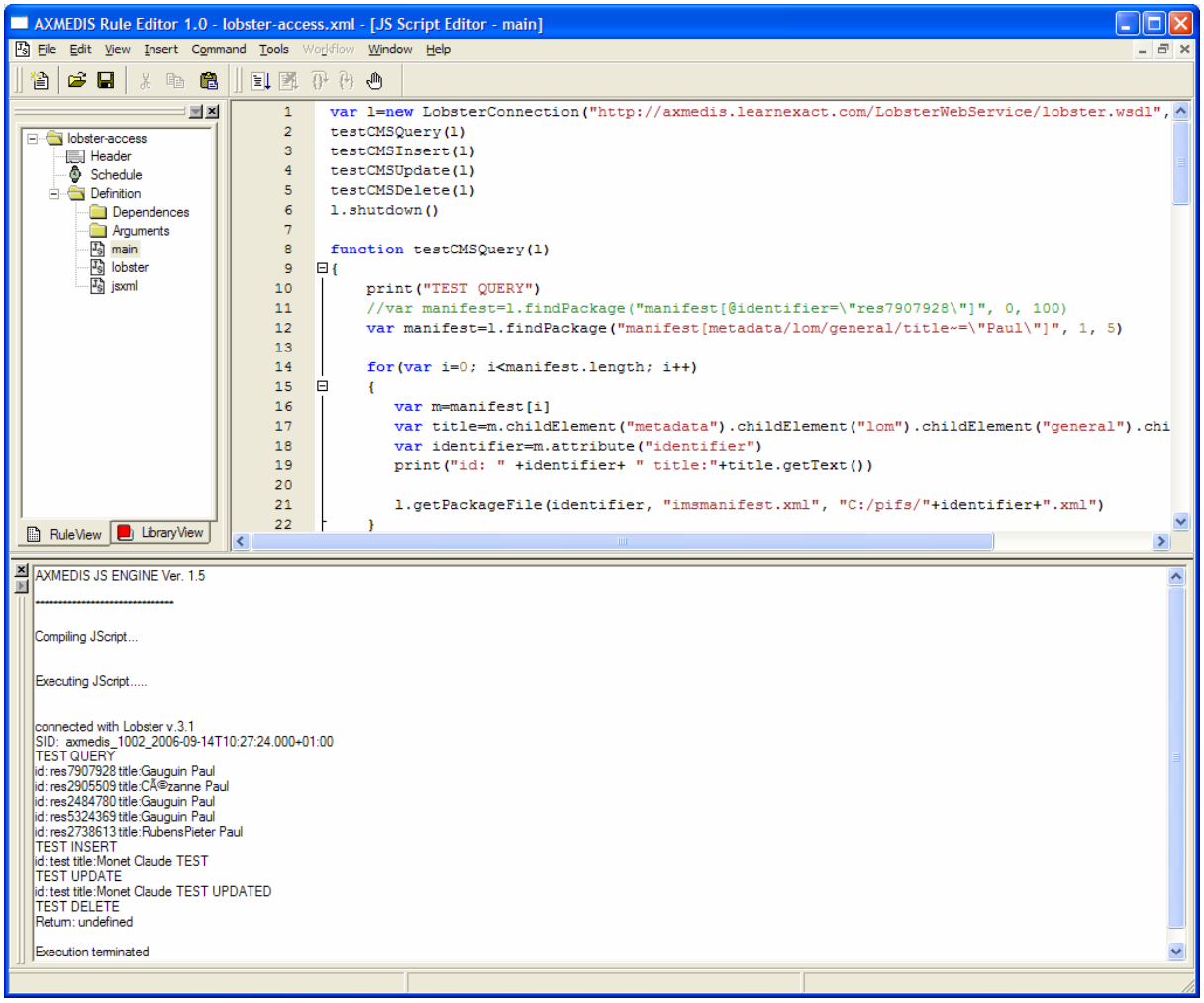
```
var l=new
LobsterConnection("http://axmedis.learnexact.com/LobsterWebService/lobster.wsdl"
, "axmedis", "user", "*****", "http")

try
{
    testCMSQuery(l)
    testCMSInsert(l)
    testCMSUpdate(l)
    testCMSDelete(l)
```

```
}
catch(e)
{
    print(e)
}
print("Closing connection...")
l.shutdown()

function testCMSQuery(l)
{
    print("TEST QUERY")
    var lomns = new Namespace("http://www.imsglobal.org/xsd/imsmd_rootv1p2p1")

    //search for packages containing Paul in the title
    var manifest=l.findPackage("manifest[metadata/lom/general/title~=\"Paul\"]",
1, 5)

    if(!existsDir("C:\\lobster-test"))
        makeDir("C:\\lobster-test")
    if(!existsDir("C:\\lobster-test\pifs"))
        makeDir("C:\\lobster-test\pifs")

    for(var i=0; i<manifest.length(); i++)
    {
       var m=manifest[i]
       var title=m..lomns::title.lomns::langstring.toString()
       var identifier=m.@identifier.toString()
       print("id: " +identifier+ " title:"+title)

       l.getPackageFile(identifier, "imsmanifest.xml", "C:/lobster-
test/pifs/"+identifier+".xml")
    }

    print("the imsmanifest.xml files are in C:\\lobster-test\\pifs")
}

function testCMSInsert(l)
{
    print("TEST INSERT")

    if(!existsFile("C:/lobster-test/test.zip"))
    {
        print("C:/lobster-test/test.zip file is missing")
        return;
    }
    //check if the package is already present
    var lomns = new Namespace("http://www.imsglobal.org/xsd/imsmd_rootv1p2p1")
    var manifest=l.findPackage("manifest[@identifier=\"test\"]", 0, 10)
    if(manifest.length()>0)
    {
        print("Package \"test\" already present, cannot insert it")
        return
    }

    var result=l.insertPackage("test","Course","C:/lobster-test/test.zip")
    print("upload result:" + result)

    var manifest=l.findPackage("manifest[@identifier=\"test\"]", 0, 10)
    // find returns an array with the manifests
```

```
    if(manifest.length()==0)
    {
        print("INSERT FAILED")
        return
    }
    for(var i=0; i<manifest.length(); i++)
    {
       var m=manifest[i]
       var title=m..lomns::title.lomns::langstring.toString()
       var identifier=m.@identifier.toString()
       print("id: " +identifier+ " title:"+title.getText())
       if(title!="TEST")
          print("INSERT FAILED")
    }
}

function testCMSUpdate(l)
{
    print("TEST UPDATE")
    if(!existsFile("C:/lobster-test/test-update.zip"))
    {
        print("C:/lobster-test/test-update.zip file is missing")
        return;
    }

    var lomns = new Namespace("http://www.imsglobal.org/xsd/imsmd_rootv1p2p1")

    var manifest=l.findPackage("manifest[@identifier=\"test\"]", 0, 10)
    if(manifest.length()==0)
    {
        print("Package \"test\" not present, cannot update it")
        return
    }

    var result = l.updatePackage("test","C:/lobster-test/test-update.zip")
    print("upload result:" + result)

    var manifest=l.findPackage("manifest[@identifier=\"test\"]", 0, 10)
    for(var i=0; i<manifest.length(); i++)
    {
       var m=manifest[i]
       var title=m..lomns::title.lomns::langstring.toString()
       var identifier=m.@identifier.toString()
       print("id: " +identifier+ " title:"+title)
       if(title.getText()!="TEST - updated")
          print("UPDATE FAILED")
    }
}

function testCMSDelete(l)
{
    print("TEST DELETE")

    var manifest=l.findPackage("manifest[@identifier=\"test\"]", 0, 10)
    if(manifest.length()==0)
    {
        print("Package \"test\" not present, cannot delete it")
        return
    }
```

```
    l.deletePackage("test")

    var manifest=l.findPackage("manifest[@identifier=\"test\"]", 0, 10)
    if(manifest.length()>0)
       print("DELETE FAILED")
}
```

The output of the script execution in the rule editor is the following:



The javascript code implementing the LobsterConnection class is the following:

```
// class LobsterConnection allows to interact with the LOBSTER WS v 4.0
// constructor LobsterConnection(wsdl,domain,user,passw,uploadtype)
//    allows to connect with the lobster WS, it performs the
//    login and retrieves domain information, the uploadtype can be
"ftp"(default) or http",
//    it indicates how to upload files on the server.
//    Example:
//       var l=new LobsterConnection("http://host.org/lobster.wsdl", "domain",
"user", "passw")
// methods
//    insertPackage(packid, objectType, pifLocalFilepath)
```

```
//       allows to insert the package into the CMS giving the
//       id of the package, the type of object to be uploaded
//       ("Course" or 0, "LearningObject" or 1) and the local
//       filename for the package
//       Example:
//           l.insertPackage("test","Course","C:/test.zip")
//    updatePackage(packid, pifLocalFilepath)
//       allows to update a package into the CMS giving the
//       id of the package, and the local filename for the
//       updated package.
//       Example:
//           l.updatePackage("test","C:/test-update.zip")
//    result findPackage(XPATHQuery, start, size)
//       allows to perform an XPATH query on the CMS, the start
//       and size values indicate the results to be returned
//       the return value is an array with the xml parsed results
//       Example:
//          metad=l.findPackage("manifest[@identifier=\"test\"]/metadata", 0,
100)
//          var
title=metad[0].childElement("lom").childElement["general"].childElement("title")
.getText()
//    getPackageFile(packid, remoteFile, localFile)
//       allows to download files of a package from the server,
//       the file is stored in the localfile
//       Example:
//
l.getPackageFile("test","imsmanifest.xml","C:/packs/imsmanifest.xml")
//          l.getPackageFile("test","res90800.htm","C:/packs/res90800.htm")
//    deletePackage(packid)
//       allows to delete a package
//       Example:
//          l.deletePackage("test")
//    logout(), shutdown()
//       allows to logout from the server, note that no multiple logins
//       of the same user are allowed.
//       Example:
//          l.logout()

function LobsterConnection(wsdl,domain,user,passw,uploadType)
{
    if(uploadType==null)
        uploadType="ftp"
    this.wsdl=wsdl
    this.domain=domain
    this.user=user
    this.passw=passw
    this.uploadType=uploadType
    this.insertPackage=LOBSTER_insert
    this.updatePackage=LOBSTER_update
    this.findPackage=LOBSTER_find
    this.getPackageFile=LOBSTER_getFile
    this.uploadFile=LOBSTER_uploadFile
    this.deletePackage=LOBSTER_delete
    this.logout=LOBSTER_logout
    this.shutdown=LOBSTER_logout

    //create the connection with the wsdl
    this.ws=new WebServiceConnection(wsdl)
```

```
    //request the Lobster version
    this.version=this.ws.GetVersion()[0]
    print("connected with Lobster v."+this.version)
    if(uploadType=="http" && this.version<4.0)
        uploadType="ftp"

    //perform the login and obtain the session id
    this.sid=this.ws.Login(domain, user, passw)[0]
    print("SID:  "+this.sid)

    //retrieve the domain information and store them in the object
    var domainInfo=this.ws.GetDomainInfo(this.sid)[0]
    var dinfo = domainInfo.substring(domainInfo.indexOf("?>")+2)
    var doc=new XML(dinfo)
    this.domainurl=doc.domainurl.toString()
    this.coursedomainpath=doc.coursedomainpath.toString()
    this.ftpuser=doc.ftpuser.toString()
    this.ftppassw=doc.ftppassword.toString()
    this.ftpport=doc.ftpport.toString()
}
function LOBSTER_insert(packid, objType, pifFilePath)
{
    if(this.sid=="") return "FAIL"
    var oType=0

    if(typeof(objType)=="string")
    {
        if(objType.toLowerCase()=="course" )
            oType=0 //course
        else
            oType=1 //learning object
    }
    else if(typeof(objType)=="numeric")
        oType=objType

    //make request of update for the package
    var filename=this.ws.InsertPackageRequest(this.sid,packid,oType)[0]
    //upload the file
    var result = this.uploadFile(filename, pifFilePath)

    try
    {
        //send the notification of upload
        this.ws.NotifyUploadDone(this.sid)
    }
    catch(e)
    {}

    return result
}
function LOBSTER_update(packid, pifFilePath)
{
    if(this.sid=="") return "FAIL"
    //make request of update for the package
    var filename=this.ws.UpdatePackageRequest(this.sid,packid)[0]

    //upload the file with the name indicated
    var result = this.uploadFile(filename, pifFilePath)
```

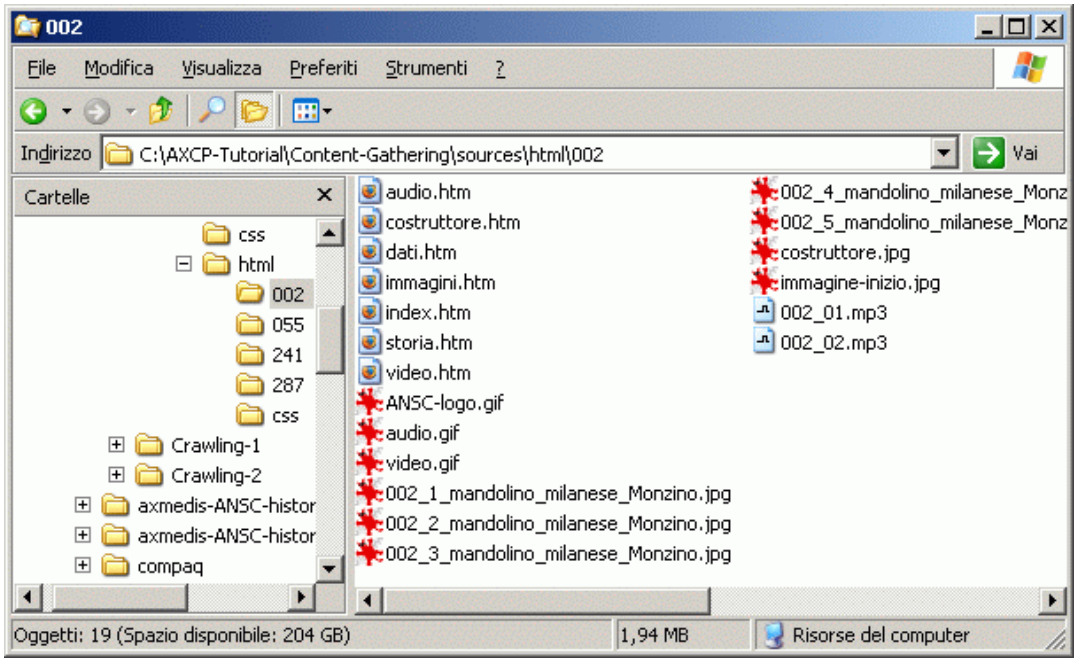
```
    //send the notification of upload
    try
    {
        this.ws.NotifyUploadDone(this.sid)
    }
    catch(e)
    {}

    return result
}
function LOBSTER_find(XPATHQuery, start, size)
{
    if(this.sid=="") return "FAIL"

    //make a XPATH query
    var result=this.ws.FindPackage(this.sid,XPATHQuery, start, size)
    if(result.length>0)
    {
        var res = result[0].substring(result[0].indexOf("?>")+2) //removes the
<?xml...?>

        //parse the xml result removing the tamino elements
        // fill an array with the results
        var resDom=new XML(res)
        var imsns = new
Namespace("http://www.imsproject.org/xsd/imscp_rootv1p1p2")
        return resDom..imsns::manifest
    }
    return []
}
function LOBSTER_getFile(packid, remoteFile, localFilePath)
{
    //request the folder where the files of the package are
    //var foldername=this.ws.GetPackageFolder(this.sid,packid)

    //build the url of the file to get
    //var url=this.domainurl+this.coursedomainpath+foldername+"/"+remoteFile
    var url=this.domainurl+this.coursedomainpath+packid+"/last/"+remoteFile
    //download the file from the server
    var http=new HttpConnection()
    http.getToFile(url,localFilePath);
    return http.getResultMsg()
}
function LOBSTER_uploadFile(remoteFilename,pifFilePath)
{
    var result = ""
    if(this.uploadType=="ftp")
    {
        //obtain the name of the host from the domain url
        var ftpurl=this.domainurl
        if(ftpurl.substr(0,7)=="http://")
            ftpurl=ftpurl.substr(7)
        ftpurl=ftpurl.substring(0,ftpurl.indexOf(":"))
        //upload the file with the name indicated
        var ftp=new FtpConnection()
        ftp.setLogin(this.ftpuser, this.ftppassw)
        ftp.upload(ftpurl+"/"+remoteFilename,pifFilePath)
        result=ftp.getResultMsg()
    }
```

```
    else if(this.uploadType=="http")
    {
        var url = this.domainurl + "/X-Siter/packLight/wsUploadFile.asp"
        var http = new HttpConnection
        var header = [ "LEXUPLOADSID: " + this.sid, "Pragma: no-cache",
"Expect:", "Accept:" ]
        http.postFromFile(url, pifFilePath, header)
        result = http.getResultMsg()
    }
    return result;
}
function LOBSTER_delete(packid)
{
    if(this.sid=="") return "FAIL"

    //perform the delete request
    try
    {
        this.ws.DeletePackage(this.sid,packid)
    }
    catch(e)
    {}
}
function LOBSTER_logout()
{
    //perform the shutdown request
    try
    {
        this.ws.Shutdown(this.sid)
    }
    catch(e)
    {}
}
```

7.2 Content gathering from file system prototype (ANSC)

7.2.1 Description of the CMS and scenario

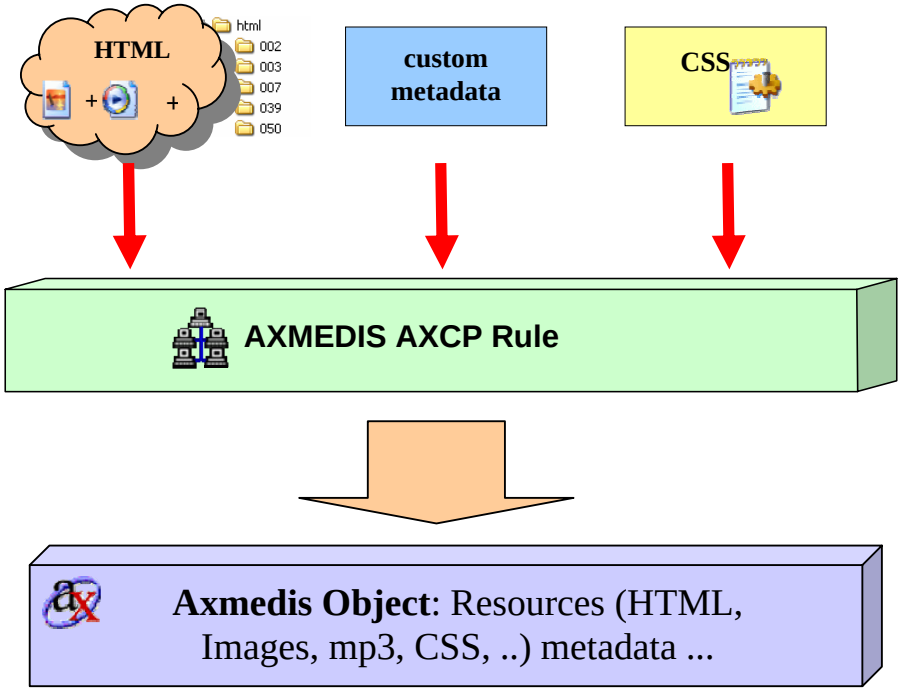
In this case we are not dealing with a real CMS. In fact the media are stored on a file-system. Such file-system could of course be a network one as well. This scenario applies to many realities in the cultural context, where for various reasons, it is not possible to create or mantain an organized CMS and digitization occurs simply storing on hard-disks. Furthermore it applies to those cases in which digitization had started before setting up some form of CMS so that media has been “left behind” on it. In many cases the user may want to gather the media: this is possible creating AXMEDIS objects.In this prototype diverse content is gathered from a file system and aggregated into an AXMEDIS Object. The following example shows how it is possible to gather HTML-based content which is organised in subfolders on a file system with a “master” CSS style sheet.



The HTML presents content about musical instruments, with text, audio, images and video linked by various pages. Following is a browser screenshot of one of the index pages:

Additionally through the AXCP it will be possible to add custom metadata from file[2] created with the AXMEDIS Metadata Editor and automatically extracted: in particular in this prototype a Dublin Core "Title" are automatically created extracting the HTML <title> tag from the index file.

A representation of the general functioning of the process is depicted by the following diagram:



The Rule is pointed to the starting paths or locations of the these elements and automatically creates objects with the right media and metadata. In this prototype the content is logically organized in subfolders, each of which containing an instrument section.

7.2.2 AXCP Scripts and their description

The AXCP script language already offers a set useful functions to deal with the file system. For this prototype firstly a function that will add all the files in a directory (with some extra features) has been created:

```
/***********************
**** Dir2Obj ***********
***********************
Function to add all files in a directory with a
certain mask to an axmedis object.
Optionally an indexfile match can be specified (e.g. "index.htm")
if matched the file will be added as the first resource in the object
Only htm or smil (SMIL) files are allowed for indexfile match.
Dublin Core can be added.
It will be taken from the file DC_file.
This file must have been created from the Axmedis editor DC editor and viewer
If an indexfile is searched and get_title is true
```

[2] Reference to Axmedis Editor tutorial

```
the function will try title extraction on the indexfile and add it as DCtitle to
the object
A css file can be added as resource.
In this case the contentID will be set to "css/style.css" by default
(but this can be customised to any value)
If the output_path doesn't exist it will be created
The function returns true upon success and saves the object to disk. false
otherwise
*/

function Dir2Obj
(
dir,            // input dir (e.g. "c:\\dir") Use "\\" as path separator
input_object,   // for adding from an object. Complete with path and extension.
Use "" for new
mask,                 // files to add (e.g. "*.*")
indexfile,      // a file (including extension) that if found  will be put at
the start of the object. Use "" for none use_extensions
DC_file,        // specify the DC XML file to use to add DC. "" for no Dublin
Core
get_title,      // if true will add DC title trying to extract it from the
<title> tag of indexfile
use_extensions, // add extensions to contentID
css_dir,        // dir containing the css file. "" to skip
css_name,       // css filename leave "" for default ("style.css")
output_path,    // where the object will be saved
output_object   // filename of the object to output INCLUDING extension eg.
"out_obj.axm"
)
{
var error =""; // here all the messages will be pushed
output_path = output_path +"\\";
// initial checks:
// check that the input dir exists
if (!existsDir(dir))
{
      print ("ERROR: The directory " + dir + " doesn't exist");
      return (false)
}
// check that an input_object specified really exists
if (input_object != "" && !existsFile(input_object))
   {
          print ("ERROR: Object file " + input_object + " doen't exist");
          return (false);
   }
// prevent the output object overwriting the input one
if (input_object == (output_path + output_object))
{
    print ("ERROR: I cannot overwrite the same object. Change the output object
or select to create new");
    return (false);
}

// check that indexfile to search (if specified) contains "htm" // TODO this
should be better
if (indexfile != "" && !indexfile.match ("htm") && !indexfile.match ("smil"))
   {
          print ("ERROR: Sorry, I can only take \"htm\" or \"smil\" files as
index");
```

```
          return (false);
   }
// check that if an indexfile has been specified also the mask includes htm,
smil or *.*
if (mask != "*.*" && (!mask.match ("htm")  || !mask.match ("smil")) && indexfile
!= "")
   {
          print ("ERROR: Sorry, Index file must be htm or smil and included in
mask.\n You are using \""+mask+"\" as mask and \""+indexfile+"\" as index
search.\nchange mask or remove index search");
          return (false);
   }

// check that output_path and output_object have been specified
if (output_object == "" || output_path == "\\")
{
          print ("ERROR: Please specify output path (e.g. \"c:\") and output
filename (e.g. \"object001.axm\")");
          return (false);
}
    // warn about missing extension for output object
    if (!output_object.match (".axm"))
    {
        error = error + "\nWARNING: no \".axm\" extension specified for output
object!";
    }
    // If the output_path doesn't exist create it and warn.
    if (!existsDir(output_path))
    {
        if (makeDir (output_path))
        {
        error = error + ("\nWARNING: The output directory \"" + output_path +
"\" doesn't exist: creating it");
        }
        else
        {
        // just in case
        print ("ERROR: Unable to create \"" + output_path + "\"");
        return (false);
        }
    }

    // get the files in an array
    files = getfilelist (dir, mask);
    if (files.length <=0) //check that the folder isn't empty an that there are
files
    {
        print ("ERROR: The directory "+ dir + " is empty or no files of type
\""+mask+"\" found in it");
        return (false);
    }
    error = error + "\nINFO: Taking files of the type "+mask;

    //start:
    print ("-----------creating object from \""+dir+"\""+"-----------------\n");
    var howmany = files.length;         // number of files found
    var resources = new Array(howmany); // array containing the resources that
will be added
    if (indexfile != "")  //if an index file to match was specified
```

```
    {
        theindexfile =""; // this will be used later... if no match is found it
will stay ""
        for (i in files)
        {

            if (filefrompath(files[i]).match (indexfile) &&
filefrompath(files[i]).length ==indexfile.length)
            //if the exact match for string in indexfile is found
            {
                error = error +"\nINFO: I found a match for \""+indexfile+"\":
Adding " + files[i] + " as first resource in the object";
                theindexfile = files[i]; // save the filename to theindexfile
                var res = new AxResource;
                res.load (files[i]);    // load the resource
                resources.push (res);   //push it as firt element of the
resources array
            }
        }
    if (theindexfile == "") // warn if no match was found
    {
    error = error +"\nWARNING: Didn't find any exact file match for
\""+indexfile+"\" in "+dir;
    }
        for (i in files) //add all the other files
        {
            if (filefrompath(files[i]).match (indexfile) &&
filefrompath(files[i]).length ==indexfile.length)
            // but not the index file!
                {
                }
            else
                {
                var res = new AxResource;
                res.load (files[i]);
                resources.push (res);
                }
        }
    }
    else // no index file to match was specified
    {
        error = error + "\nINFO: Adding resources: not using index file";
        for (i in files) //add the files
        {
            var res = new AxResource;
            res.load (files[i]);
            resources.push (res);
        }
    }
    if (input_object == "") //create a new empty axmedis object
    {
        var axObj = new AxmedisObject();
        error = error + "\nINFO: Creating new object";
    }
    else
    {
        var axObj = new AxmedisObject(input_object);  //load an existing object
        error = error + "\nINFO: Using existing object "+ input_object;
    }
```

```
    if (DC_file != "") // if Dublin Core must be added
    {
    // put Dublin Core in the object
        if (indexfile != "" && theindexfile != "" && get_title)
        // if indexfile match was found and <title> tag must be used...
        {
            // try to extract the title from <title></title> in the indexfile
found
            var DC_title = get_HTML_Title (theindexfile);
            var dc = axObj.getDublinCore ();
            dc.addDCElement("title", DC_title);
            error = error +"\nINFO: Explicitly adding DC title
\""+DC_title+"\"\n-- taking it from " +dir + "\\" + theindexfile+" <title> tag";
            if (addDC (axObj, DC_file) != false)
            {
                error = error +"\nINFO: Adding DC from file \""+DC_file+"\"";
            }
            else
            {
                error = error +"\nWARINING: Problem trying to get DC from file
\""+DC_file+"\" please refer to add_DC message";
            }
        }
        else  // just add the DC from the file without extracting any title
        {
            if (addDC (axObj, DC_file) != false)
            {
                error = error +"\nINFO: Adding DC from file \""+DC_file+"\"";
            }
            else
            {
                error = error +"\nWARINING: Problem trying to get DC from file
\""+DC_file+"\" please refer to add_DC message";
            }
        }
    }
    else // if no DC must be added at all
    {
        error = error + "\nINFO: No Dublin Core being added"
    }
    for (i in resources)           //add all the resources to the object
    {
        if (use_extensions)    // if file extensions must be added to the
contentID
        // add file extension to contentID (thanks to Ivan for this)
        /*{
          resources[i].contentID = resources[i].contentID +
mimeTypeToExt(resources[i].mimeType);
        }*/
        axObj.addContent (resources[i]);
    }
  if (use_extensions) // tell what happened about extensions
  {
          error = error +"\nINFO: Adding file extensions to resource ContentID";
  }
  else
  {
          error = error +"\nINFO: Not adding file extensions to resource
ContentID";
```

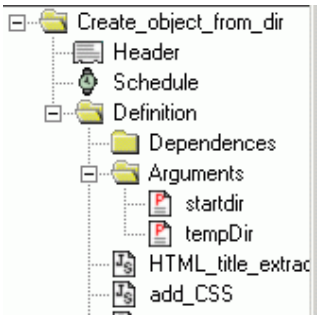
```
  }
  var cssres = add_css (css_dir,css_name);
  //try to get the css if needed
  if (css_dir == "" || !cssres)
  //if the css_dir is empty (skip) or if adding the css failed
  {
      error = error + "\nWARNING: No css file being added.";
  }
  else  // the css must be added and it was found
  {
      cssres.contentID = "css/style.css";       // this is specific to the ANSC
instruments objects
      axObj.addContent (cssres);                //add the css to the object
      error = error + "\nINFO: Adding css file "+css_dir + "\\" + css_name +"
with contentID \""+ cssres.contentID+ "\"";
  }
    // Save the axmedis object
    axObj.save (output_path + output_object); // output_object is in the
Arguments
    // print a report about the object and all the process (error contains all
the messages)
    print (error+ "\n\nSUCCESS: Saved " + output_path + output_object+"\n");

    return (true);
}
```

This function can easily be called recursively on a set of sub-directories from a starting path. The following example shows how to call the Dir2Obj function:

```
function test ()
{
    // get an array of the subdirs of startdir
    // startdir is a parameter in the Arguments.
    // Change it to change the starting dir.
    var directory = subdirs (startdir);
    // DC_xml is a file containin Dublin Core as saved from the AX Metadata
Editor
    var DC_xml = tempDir+"\\dublin-core\\DC-ANSC-instruments.xml";
    // tempdir is specified in the Arguments
    if (directory == null)
    {
        print ("ERROR: no subdirs in "+startdir);
        return (false);
    }
    for (m in directory)
    {
        var success = Dir2Obj
        (
            directory[m],       // input dir (e.g. "c:\\dir") Use "\\" as path
separator
            "",                 // for adding from an object. Complete with path
and extension. Use "" for new
            "*.*",                   // files to add (e.g. "*.*")
            "index.htm",          // a file (including extension) that if found
will be put at the start of the object. Use
            DC_xml,             // specify the DC XML file to use to add DC. ""
for no Dublin Core
            true,               // if true will add DC title trying to extract
it from the <title> tag of indexfile
```

```
            true,               // add extensions to contentID
            cssDir,             // dir containing the css file. "" to skip
            "",                 // css filename leave "" for default
("style.css")
            outDir,             // where the object will be saved
            filefrompath(directory[m])+".axm" // filename of the object to
output INCLUDING extension eg. "out_obj.axm"
        )
    }
    return (success);
}
```

In this function two variables are used – “startdir” and “tempDir” – which are specified by the user in the Arguments list of the rule tree view in the AXCP Rule Editor, as can be seen in the following screenshot:



The subdirs function simply gets a list of sub-directories given a starting path.

For adding the metadata, two different functions are used, one for extracting the <title> tag from an html file an returning it as a string upon succes:

```
/* tries to extract the title from an HTML file
the <title> tag must be within a <head></head> section
and the closing </title> tag must exist
The funcion will trim the contents from leading and trailing spaces, tabs
newlines etc.
the function returns the title found or an exmpty string on unsuccess
*/

function get_HTML_Title (file)
{
    if (! existsFile(file))     // check that the file exists
    {
      print ("html title ERROR: File " + file + "doen't exist");
      return ("");
    }
    var all = readFromFile(file);       // read from the file into a string
    var headpos = all.indexOf("<head>");//search for the <head> tag
    if (headpos < 0) // if the <head> tag is not find: error
    {
      print ("html title ERROR: File " + file + " contains no <head> tag");
      return ("");
    }
    var titlepos = all.indexOf ("<title>",headpos); //search for the <title> tag
    if (titlepos < 0) // if not found: error
    {
      print ("html title ERROR: File " + file + "contains no <title> tag");
```

```
      return ("");
    }
    //search for the </title> closing tag
    var endtitlepos = all.indexOf ("</title>",titlepos);
    if (endtitlepos < 0) // if not sound: error
    {
      print ("html title ERROR: File " + file + "contains no </title> end tag");
      return ("");
    }
    var output = all.slice (titlepos+7,endtitlepos); // get the substring in
<title> ... </title> substring
    var trim = 0;
    for (i in output) // check for leading spaces, tabs and newlines...
    {
        if (output.charCodeAt(i) ==  32 || output.charCodeAt(i) ==  9 ||
output.charCodeAt(i) ==  10)
        {
            trim = trim + 1; //leading index for slice
        }
        else break   // ...up to the first nonspace char found
    }
    var trimlength = output.length;
    for (j = output.length-1; j >= 0; j--) // check for trailing spaces, tabs
and newlines from ednd of string...
    {
        if (output.charCodeAt(j) ==  32 || output.charCodeAt(j) ==  9 ||
output.charCodeAt(j) ==  10)
        {
            trimlength = j ; //trailing index slice
        }
        else break   // ... down to the first nonspace char found
    }

    output = output.slice(trim,trimlength); // slice the string
    print ("html title SUCCESS: extracted \""+ output + "\" from " + file);
    return (output);
}
```

The other function relies on the JSXML XML Tools[3] to parse a – specified – XML file containing the metadata and then adding it to a – specified – AXMEDIS Object. In fact the AXCP provides a set of functions to handle metadata (and thus Dublin Core) within AXMEDIS objects.[4]

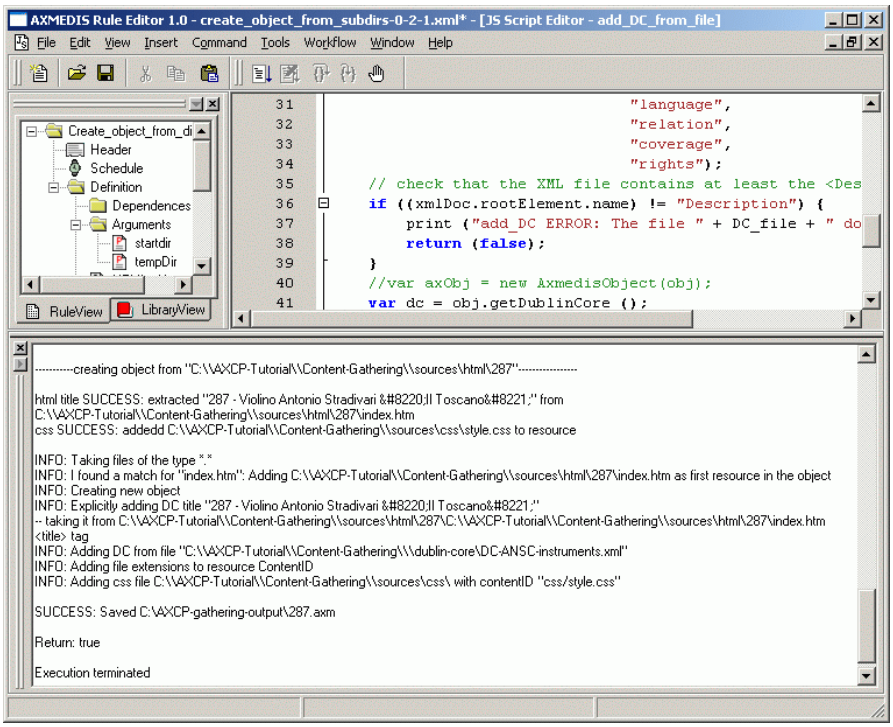
```
/* add Dublin Core metadata to the object obj
taking it from an XML file as produced by the save DC
in the Axmedis Editor.
If DC is already present, it will be added after that.
the updated object is returned
false is returned in case of error.
CAUTION: no check for duplicates is made!!
*/
function addDC(obj, DC_file)
{
    // check if the file exists otherwise print an error and return false
    if (!existsFile (DC_file)) {
        print ("add_DC ERROR: "+ DC_file + " doesn't exist");
```

[3] Reference also to modified version in AXCP...

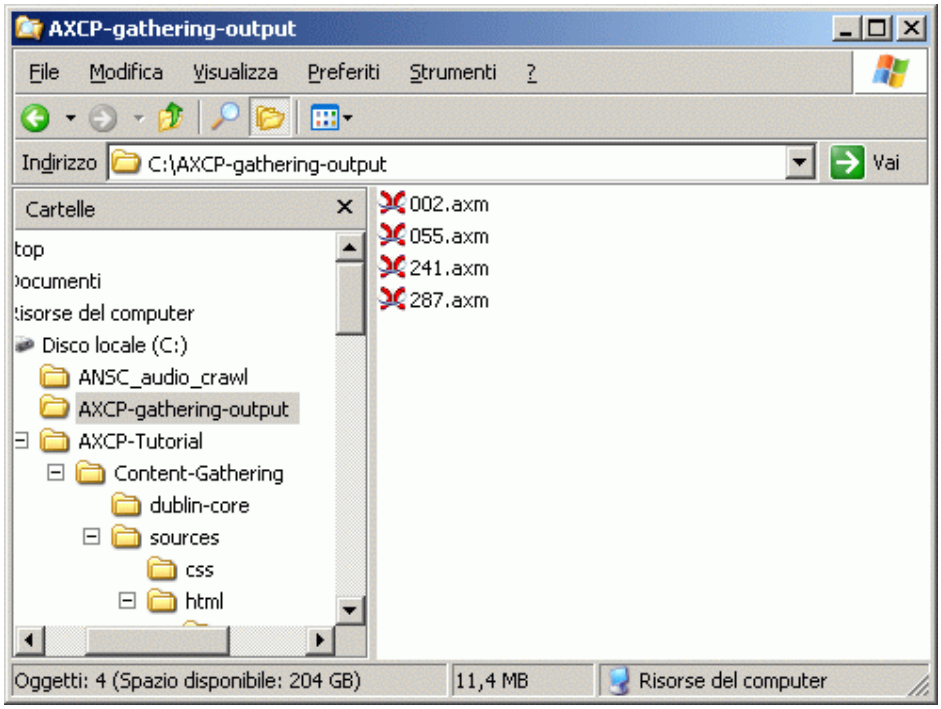
[4] Se the AXMEDIS JavaScript Reference Manual for further details.

```
        return (false)
    }
    strXML = readFromFile (DC_file)
    var xmlDoc = new REXML(strXML);

    // these are the 15 possible DC elements
    var DC_elements = new Array ("title",
                                 "creator",
                                 "subject",
                                 "description",
                                 "publisher",
                                 "contributor",
                                 "date",
                                 "type",
                                 "format",
                                 "identifier",
                                 "source",
                                 "language",
                                 "relation",
                                 "coverage",
                                 "rights");
    // check that the XML file contains at least the <Description> element
    if ((xmlDoc.rootElement.name) != "Description") {
        print ("add_DC ERROR: The file " + DC_file + " doesen't seem a valid DC
XML file - Could not find \"Description\" root element");
        return (false);
    }
    //var axObj = new AxmedisObject(obj);
    var dc = obj.getDublinCore ();
    for (var xmlIterator=new JSXMLIterator(xmlDoc.rootElement);
xmlIterator.getNextNode();) {
        for (j = 0; j < DC_elements.length; j++) {
            // just in case: check that the element is a valid DC one and that
it is not empty
            if ((xmlIterator.xmleElem.name == DC_elements[j]) &&
(xmlIterator.xmleElem.text.length >0))
            {
                dc.addDCElement (xmlIterator.xmleElem.name,
xmlIterator.xmleElem.text, xmlIterator.xmleElem.attribute ("lang"));
                //print ("Adding DC element *" + xmlIterator.xmleElem.name
+"*\twith value: \"" + xmlIterator.xmleElem.text+ "\" and lang attribute:
\""+xmlIterator.xmleElem.attribute ("lang")+"\"");
            }
        }
    }
    return (obj);
}
```

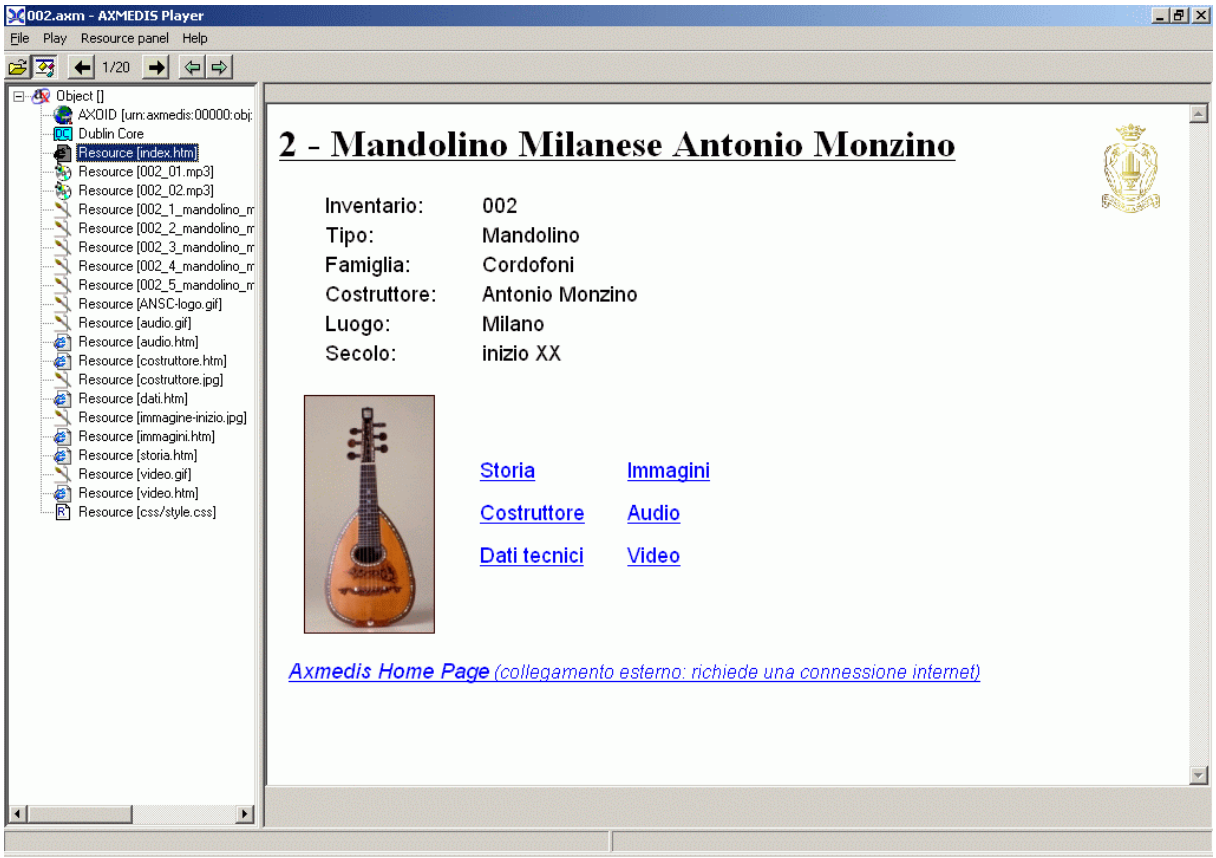
For the user's convenience a report of the operation and possible errors and warnings is printed to the output (in the AXMEDIS Rule Editor the Output window) as can be seen in the following screenshot:



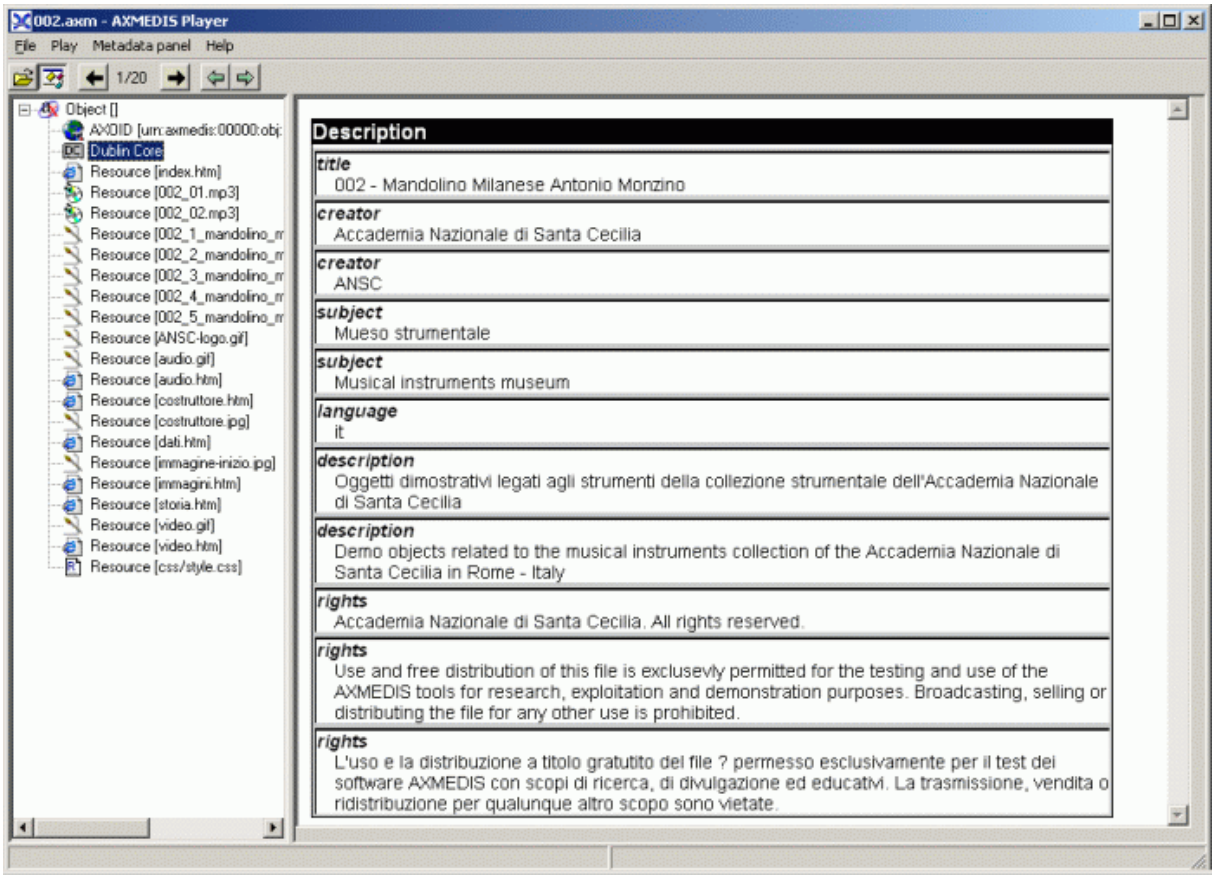
As a final result AXMEDIS Objects containing media from the subfolders, the css and metadata are created. In this case they are again saved to the file system, but they could of course be uploaded to the database, a server and also further processed (for example adding all of them into a unique object).
Here are the final objects saved in a directory after running the script:



The objects can be opened in the AXMEDIS Editor or Player (as shown here):



The following screenshot shows the DC metadata that has been added automatically:



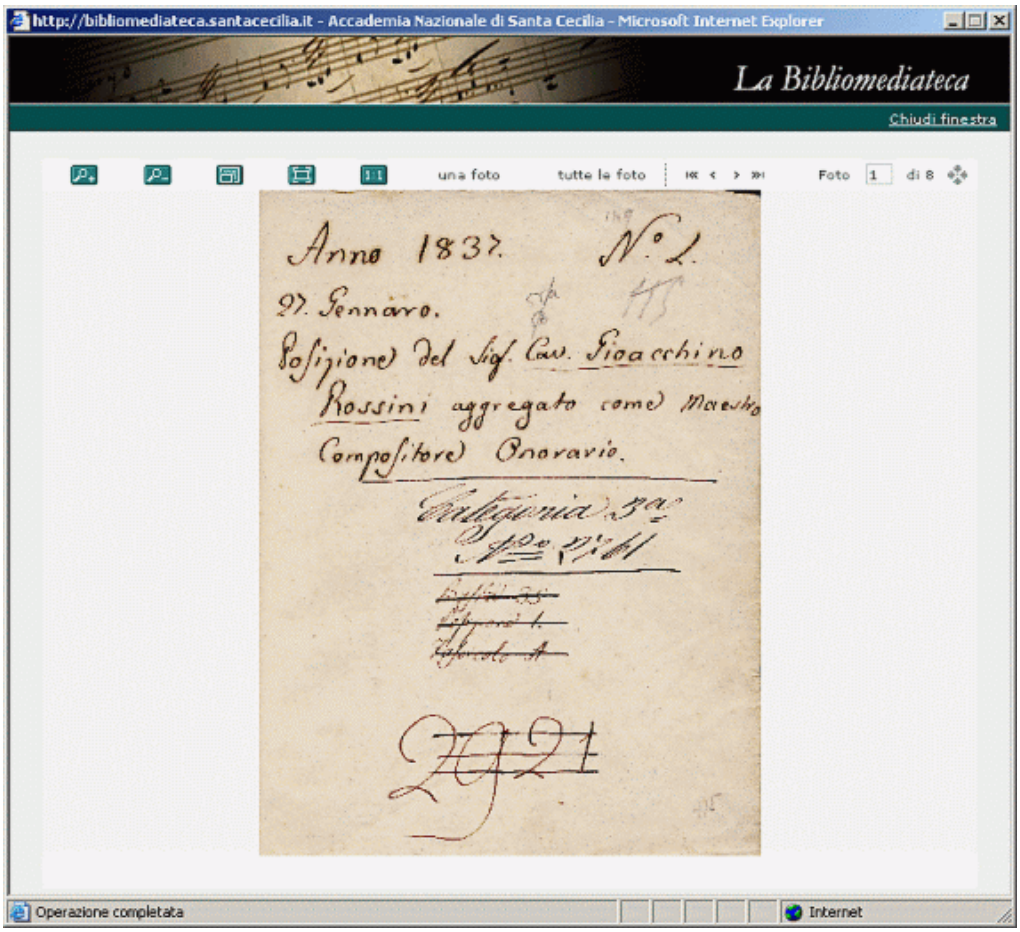
7.3 Crawling from CMS using SearchBox Tool (ANSC)

The following example shows the possibility to crawl CMS extracting media and metadata and adding them to AXMEDIS objects.

7.3.1 Description of the CMS and scenario

For a first prototype the digitized Historical Archive from the Accademia Nazionale di Santa Cecilia has been used. The Historical Archive holds documents relating to the Accademia since its foundation. The documents are currently being digitized and consequently catalogued in a CMS.[5] As the documents (scanned images) are catalogued specific metadata is added and applied; the documents are then accessible both to the public[6]through a specific OPAC and internally for back-office purposes. The images are stored on a server and an XML database (conforming to specific EAD)[7] drives the web applications, like for example a document viewing interface (shown here):

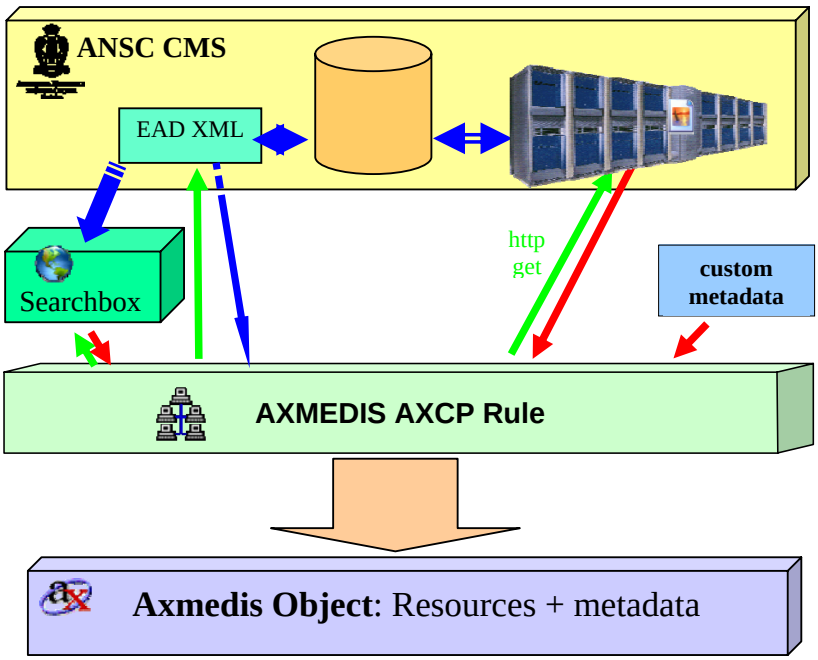
[5] Here only an overview for our purposes is given. For further details see http://bibliomediateca.santacecilia.it/bibliomediateca/guida/guida_gen.jsp.
[6] See http://bibliomediateca.santacecilia.it/bibliomediateca/patrimonio/0/patrimonioArchivioSingolo.jsp?idA=0&hl=0.
[7] See http://www.loc.gov/ead/



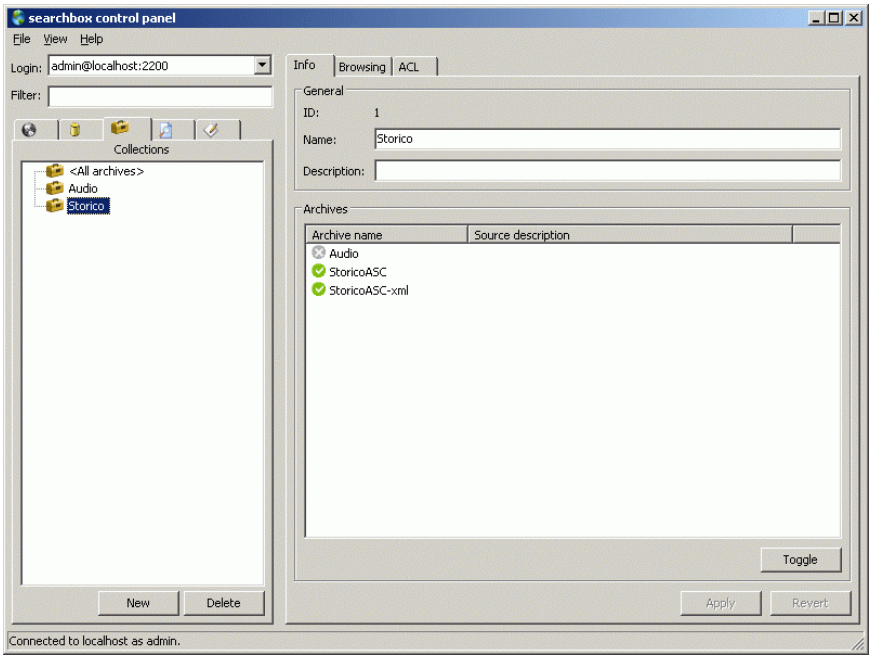
7.3.2 Content gathering prototype description

The aim of this prototype is to crawl the CMS searching for the title[8] and extract media and metadata adding them to and AXMEDIS Object as shown in the following general diagram:
Since in this case the capability of searching within the database is required a Search Engine is also needed. This can be done by using Focuseek's Searchbox opportunely integrated into the AXMEDIS AXCP script language[9]. Within Searchbox XML can be indexed through a specific XML parsing plugin.[10]

[8] Which in this case is the title of the archival unit. Of course further search keys can be defined and used.
[9] See http://www.focuseek.com/
[10]



Searchbox – once the indexing has been performed - can be queried directly via AXCP scripts as it will show.[11] In the following picture, the Searchbox control panel interface is showing two collections. One of the collections relates to the Historical Archive:



7.3.3 AXCP Scripts and their description

After launching a query, the Searchbox Engine (if the search was successful) returns a link(s) to the XML document(s) containing the search key. The document(s) will have to be appropriately parsed so as to

[11] For further details see the AXMEDIS JavaScript Reference Manual

extract the relevant EAD elements for our purpose. To do this a specific JavaScript class is created in the Rule to hold the extracted elements:

```
// StoricoASC
function incomplete_component (
    c_id,
    container_type,
    container_text,
    did_odd_p,
    unittitle_type,
    unittitle,
    physdesc_extent,
    physdesc_genreform,
    normal_date,
    daoloc_hrefs
    )
{
    this.c_id = c_id;
    this.container_type = container_type;
    this.container_text = container_text;
    this.did_odd_p = did_odd_p;
    this.unittitle_type = unittitle_type;
    this.unittitle = unittitle;
    this.physdesc_extent = physdesc_extent;
    this.physdesc_genreform = physdesc_genreform;
    this.normal_date = normal_date;
    this.daoloc_hrefs = daoloc_hrefs;
}
```

The class is not very big in this prototype, but it could be further enlarged if needed.
The crawling process is controlled by the function crawlStorico. This function is intended specifically for the Historical archive, but could be easily customized for another source.

```
function crawlStorico
    (title_string,      // string to search in the titles
    temp_dir,           // temp dir (full path) - will be created if it doen't
exist
    output_dir,         // output dir where ALL the objects (and log) will be
saved - will be created if it doen't exist
    DC_file,            // DC xml file (full path) to add to each object. "" for
none
    searchbox_host,     // searchbox host (e.g. "localhost")
    searchbox_port,     // searchbox port (e.g. "2200")
    searchbox_username, // searchbox username
    searchbox_password, // searchbox password
    http_prefix,        // prefix for getting images via http for ALL objects
    create_object)      // if true all the possibly created
                        //objects will be added to a unique one (only if 2 or
more)
{
    var foundURLs =  searchbox_getURL (title_string,
                            searchbox_host,
                            searchbox_port,
                            searchbox_username,
                            searchbox_password);
    var all_records = new Array();
    print
("************************************************************");
```

```
    var global_error = ""; // hold all messages
    var total_objects = 0;
    var all_objects = new Array();
    appendToFile (output_dir+"\\_log.txt","\n------------ Report searching for
\""+title_string+"\"----------\n\n");
    for (ix=0; ix < foundURLs.length; ix++)
    {
        foundURLs[ix] = eliminate_file_prefix (foundURLs[ix]);
        var error = "";
        print ("Searching for images in " +foundURLs[ix] +" for
\""+title_string+"\". please wait for results...");
        if (!existsFile (foundURLs[ix]))
        {
            break;
        }
        URL_contents = readFromFile (foundURLs[ix])
        record = get_matching_components (URL_contents, title_string)
        // try to create an object
        for (kx = 0; kx < record.length; kx++)
        {
            var this_record = record[kx];
            error= error + "Found ID:"+this_record.c_id + " -> title: "
+this_record.unittitle+" in  "+ foundURLs[ix]+"\n";
            error= error + "Trying to create object...\n";
            // only if daoloc_hrefs (i.e. images) are found create an object...
            if (this_record.daoloc_hrefs == null)
            {
            error = error + "no images found for "+this_record.c_id + " ->
title: " +this_record.unittitle+"\n";
            break;
            }
            var axObj = new AxmedisObject(); // new object
            res = new AxResource;
            // now add the attachments
            error = error+ "Trying to get resources via http...\n"
            var foundfiles = 0;
            var clean_title_string = epurate_pathname(title_string);
            var savedir = output_dir+"\\"+ clean_title_string
+"\\"+this_record.c_id;
            images_to_add = this_record.daoloc_hrefs.length;
            if (images_to_add > 100) {
                print ("WARNING: Found " +images_to_add  +" images. it may take
long - please be patient!");
            }
            for (iy=0; iy < this_record.daoloc_hrefs.length; iy++)
            {
                var noadd = false;
                if (Math.floor((iy+1) % 50) == 0) {
                    print ("saving...");
                    if (!existsDir (temp_dir)) {
                        createDir (temp_dir);
                    }
                    axObj.save  (temp_dir+"\\~"+this_record.c_id +iy+".axm")
                    axObj = new AxmedisObject (temp_dir+"\\~"+this_record.c_id
+iy+".axm");
                }

                if (this_record.daoloc_hrefs[iy].match (".ASC.")) {
                    // this is archive and server specific.
```

```
                    var the_file = gethttp
(http_prefix+"low"+this_record.daoloc_hrefs[iy],
temp_dir+"\\"+this_record.c_id);
                    if (the_file != false)
                    {
                        var res = new AxResource ();
                        res.load (the_file);
                    }
                    else
                    {
                        noadd  = true;
                    }
                }
                else {
                    this_record.daoloc_hrefs[iy] =
this_record.daoloc_hrefs[iy].replace ("\\","/");

                    var the_file = gethttp
(http_prefix+this_record.daoloc_hrefs[iy], temp_dir+"\\"+this_record.c_id);
                    if (the_file != false)
                    {
                        var res = new AxResource ();
                        res.load (the_file);
                    }
                    else
                    {
                        noadd  = true;
                    }
                }
                if (noadd == false)
                {
                  axObj.addContent (res);
                  foundfiles = foundfiles +1;
                  print ("- Adding "+this_record.daoloc_hrefs[iy]+" to object -
"+(iy+1)+" of "+images_to_add+" ("+(Math.round(iy*100/images_to_add))+"%)");
                }
                else
                {
                // uncomment the line below to have a more detailed error
                //error= error + "HTTP ERROR trying to add
"+http_prefix+this_record.daoloc_hrefs[iy]+"\n";
                }
            }

            if (foundfiles >0) // i.e. if at least a resource was successfully
added
            {
                // add DC related to this object
                dc = axObj.getDublinCore ();
                dc.addDCElement ("title",this_record.unittitle,"");
                dc.addDCElement ("identifier",this_record.c_id,"");
                if (this_record.unittitle_type != null &&
this_record.unittitle_type != "")
                {
                    dc.addDCElement ("type",this_record.unittitle_type ,"it");
                }
                if (this_record.container_type != null)
                {
                    if (this_record.container_text == null) {
```

```
                        dc.addDCElement
("description",this_record.container_type,"it");
                    }
                    else {
                        dc.addDCElement ("source","Archivio Storico -
"+this_record.container_type+" "+this_record.container_text,"it");
                    }
                }

                if (DC_file != "" && (addDC (axObj, DC_file))) // add custom DC
from file
                {
                    error = error+ "Adding DC from "+DC_file+"\n"
                }
                else
                {
                    error = error+ "ERROR: Impossible to add DC from "+DC_file+"
Does it exist?\n"
                }
                createDir (savedir);

                axObj.save (savedir+"\\"+this_record.c_id +".axm");
                error = error + "Managed to find and add "+foundfiles+"
resource(s)\n"
                error = error + "SUCCESS: Saved object:
\""+savedir+"\\"+this_record.c_id +".axm\"\n";
                total_objects = total_objects+1;
                all_objects.push (savedir+"\\"+this_record.c_id +".axm");
            }
            else
            {
            error = error + "ERROR: Did not save any object because there were
no resources to add: maybe http error? (see above)\n";
            }
        print (error);
        appendToFile (output_dir+"\\_log.txt", error);
        }
    }
    print ("--- Saved "+total_objects+ " objects in total: see
"+output_dir+"\\_log.txt for details");
    appendToFile (output_dir+"\\_log.txt","\n--- Saved "+total_objects+ "
objects in total ---\n");

    // if a unique object containing all the others must be created
    if (create_object && total_objects >1)
    // check that at least 2 objects have been created
    {
    all_axObj = new AxmedisObject ();
    this_object = new AxmedisObject ();
    var save_name = output_dir+"\\"+epurate_pathname (title_string)+"-all.axm";
    print ("\n--- Adding all found objects to "+save_name);
        for (iobj =0; iobj < all_objects.length; iobj++)
        {
            if (Math.floor((iobj+1) % 50) == 0) {
                    print ("saving...");
                    if (!existsDir (temp_dir)) {
                        createDir (temp_dir);
                    }
```

```
                    all_axObj.save  (temp_dir+"\\~"+epurate_pathname
(title_string)+"-all.axm");
                    all_axObj = new AxmedisObject
(temp_dir+"\\~"+epurate_pathname (title_string)+"-all.axm");
                }

            print ("- Adding "+all_objects[iobj]);
            this_object = new AxmedisObject(all_objects[iobj]);
            var this_DC = this_object.getDublinCore ();
            var the_title = this_DC.getDCElementValue ("title");
            this_object.contentID = the_title;
            all_axObj.addContent (this_object);
        }
    // add DC related to this object
    all_dc = all_axObj.getDublinCore ();
    all_dc.addDCElement ("title","Search results for \""+title_string+"\"","");
    all_axObj.contentID = ("Object for \""+title_string+"\"");
    all_axObj.save (save_name);
    appendToFile (output_dir+"\\_log.txt","\n--- Added all objects to
"+save_name+"---\n");
    }
    else // <= 1 objects were created so no global object will be created.
    {
        print ("**WARNING: Did not create global object because only
"+total_objects+ " objects produced");
        appendToFile (output_dir+"\\_log.txt","WARNING: Did not create global
object because only "+total_objects+ " objects produced\n");
    }
    appendToFile (output_dir+"\\_log.txt","\n -------- end report searching for
\""+title_string+"\"-------\n");
}
```

The function queries Searchbox for the given title, parsees the matching XML documents and extracts the relevant components. Among the records only those containing digitized images are filtered: then via the AXCP http functionalities[12] the images are retrieved from the CMS server[13] which holds all the images. Metadata such as the title (archival), unique id, type etc. are extracted from the XML records and added to the Dublin Core;[14] also custom (fixed) metadata is taken from a file previously created in the AXMEDIS Metadata Editor (with metadata such as the creator, rights etc.) and added to the object.
The images are added as AXMEDIS Resources to the object, which is finally saved. Optionally multiple objects resulting from the search can be all added to a global object. For convenience a subdirectory with the name of the search string is created where the objects are saved. Again the object could also uploaded, sent via ftp etc.
The Searchbox query is carried out by a specific function that will return an array of URLs containing the searched keyword:

```
// returns an array with all the urls of files containing the searchstring
// as returned from the searchbox query
function searchbox_getURL (searchstring,
                           host,
                           port,
                           username,
                           password)
```

[12] See AXMEDIS Javascript Reference Manual
[13] This server is only accessible from a machine within the Accademia's intranet. For demonstration purposes a snipplet of the server was recreated on a local server, and the script appropriately modified.
[14] Different metadata could also be added.

```
{
    var the_host = host;
    var the_port = port
    var address = "http://"+the_host+":"+the_port+"/";
    print ("Connecting to searchbox at "+address);
    searchstring
    // this will use http connection just to check if the server exists

    connection = new HttpConnection();
    connection.getToString(address);
    //str = connection.getContent ();
    msg = connection.getResultMsg();
    if (msg != "Exit Success") {
        print ("ERROR: trying to connect to: " +address+" - "+ msg);
        return (false);
    }
    // we are assuming that it is a valid searchbox server.
    var URL = new Array();
    var sb = new AXSearchbox();
    sb.host = the_host;
    sb.port = the_port;
    sb.username = username;
    sb.password = password;
    var qs = new QuerySpec();
    qs.collection = 1; // Storico Collection has ID 1
    qs.queryString = searchstring;
    var qr = new Array();
    var maxres = sb.query(qs, qr);
    var i, j; for(i = 0; i < qr.length; ++i)
    {
        URL[i] = sb.getDocumentURL(qr[i].id);
    }
    return (URL);
}
```

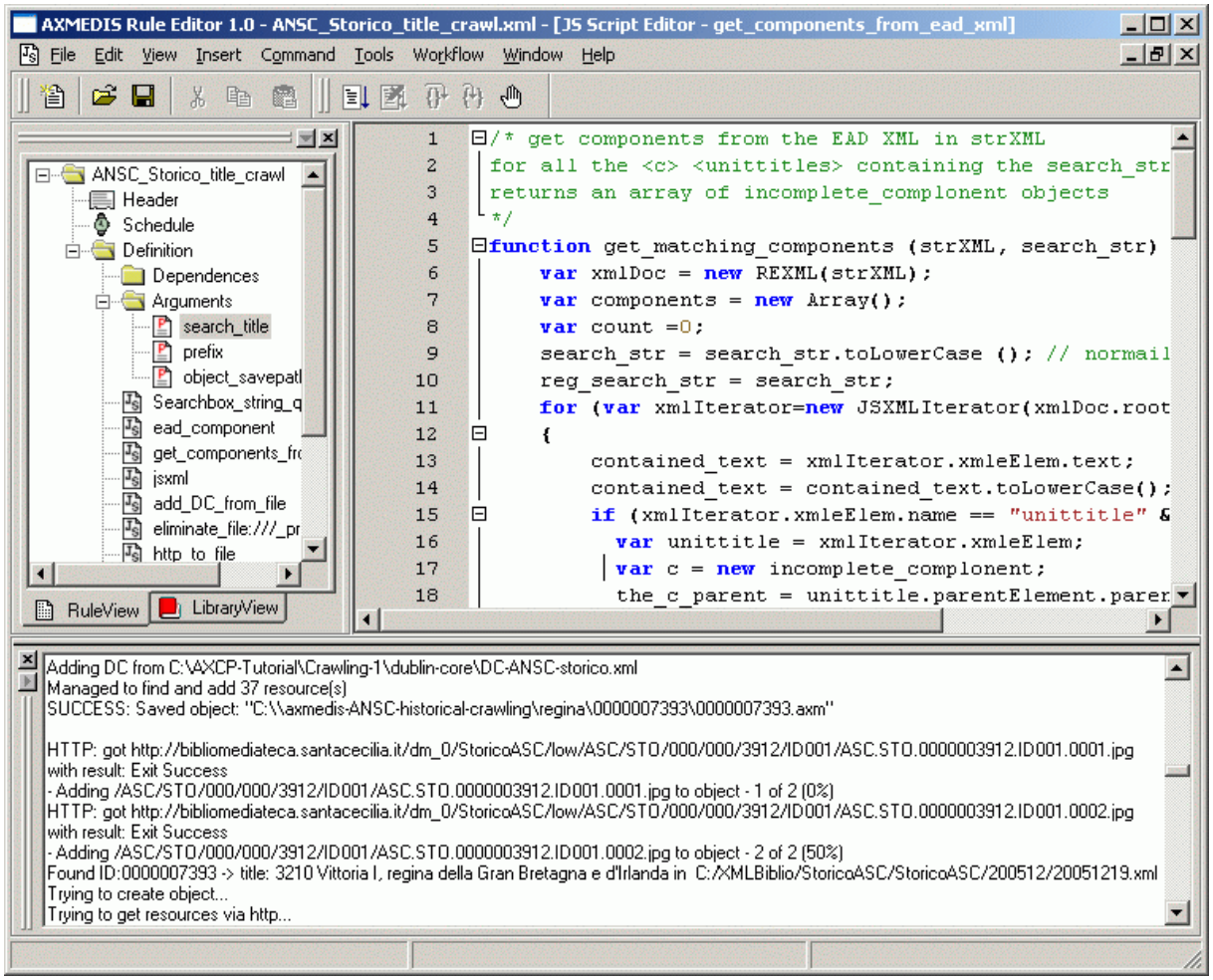
The URLs obtained are passed to a function which extracts (parsing the XML) the relevant components returning an array of components of the class “incomplete_complonent” (see above):

```
/* get components from the EAD XML in strXML
for all the <c> <unittitles> containing the search_str string.
returns an array of incomplete_component objects
*/
function get_matching_components (strXML, search_str) {
    var xmlDoc = new REXML(strXML);
    var components = new Array();
    var count =0;
    search_str = search_str.toLowerCase (); // normailze the search string to
lower case
    reg_search_str = search_str;
    for (var xmlIterator=new JSXMLIterator(xmlDoc.rootElement);
xmlIterator.getNextNode();)
    {
        contained_text = xmlIterator.xmleElem.text;
        contained_text = contained_text.toLowerCase(); // normalize the eleement
text to lower case
        if (xmlIterator.xmleElem.name == "unittitle" &&
(contained_text.match(reg_search_str))) {
          var unittitle = xmlIterator.xmleElem;
          var c = new incomplete_component;
```

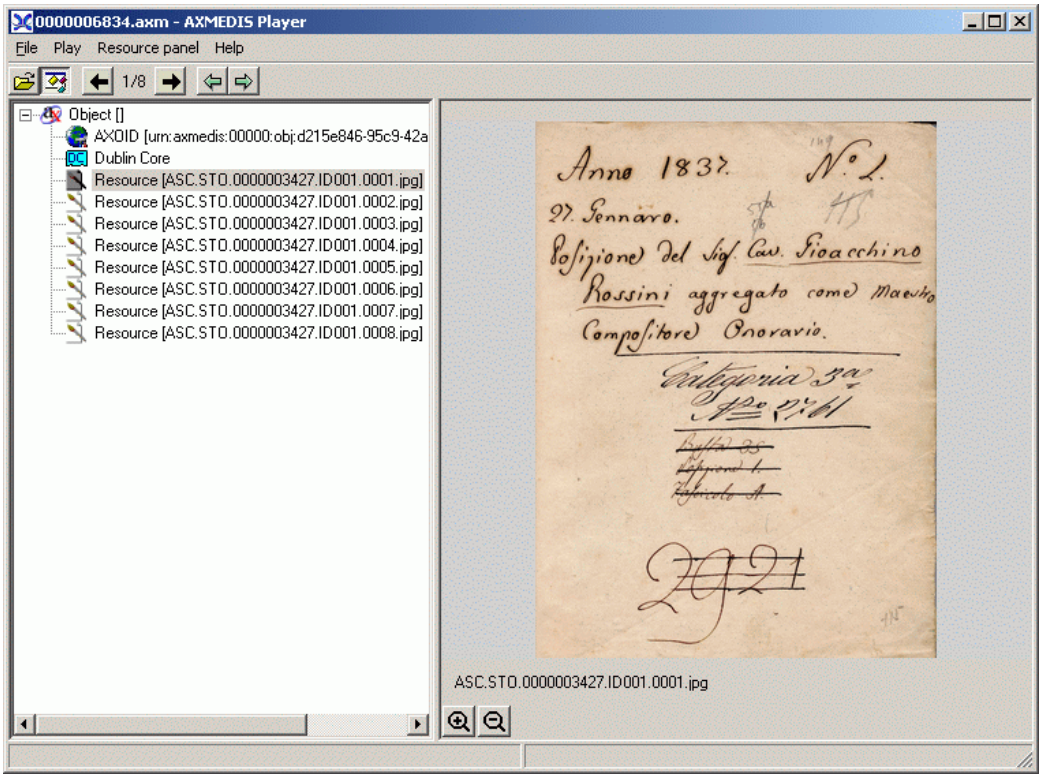
```
          the_c_parent = unittitle.parentElement.parentElement;
          c.c_id = the_c_parent.attribute ("id");

          daogrp = the_c_parent.childElement ("daogrp");
          if (daogrp != null)
          {
              var daogrp_childeren = daogrp.childElements;
              c.daoloc_hrefs = new Array();
              if (daogrp_childeren != null) {
                  for(i=0;i<daogrp_childeren.length;i++)
                  {
                    if (daogrp_childeren[i].attribute ("href") != "") {
                        c.daoloc_hrefs.push (daogrp_childeren[i].attribute
("href"));
                    }
                  }
               }
          }
          else {break}; // this way only records with daolocs (i.e. images) will
be taken
          count = count+1;

          did = the_c_parent.childElement ("did");
          if (did != null)
          {
          container = did.childElement ("container");
          }

          if (container != null)
          {
          c.container_type = container.attribute ("type");
          c.container_text = container.text;
          }

          odd = did.childElement ("odd");
          if (odd != null) {
              p = odd.childElement ("p");
              if (p != null) {
                  c.did_odd_p = p.text;
              }
          }
          else c.did_odd_p = "";

          physdesc = did.childElement ("physdesc");
          if (physdesc != null) {
              physdesc_extent = physdesc.childElement ("extent");
              if (physdesc_extent != null) {
                  c.physdesc_extent = physdesc_extent.text;
              }
              physdesc_genreform = physdesc.childElement ("genreform");
              if (physdesc_genreform != null) {
                  c.physdesc_genreform = physdesc_genreform.text;
              }
          }

          if (unittitle.attribute ("type") != null) {
              c.unittitle_type = unittitle.attribute ("type");
          }
```

```
            c.unittitle = unittitle.text;

            unitdate = unittitle.childElement ("unitdate");
            if (unitdate != null) {
                c.normal_date = unitdate.attribute ("normal");
            }

          components.push (c); // array of found components
          }
    }
return (components);
}
```

These components are used by the main function to assemble the objects appropriately. Since the whole script is quite complex and may take some time to execute, not only appropriate feedback is given to the user but also a log.txt file is created in the main output directory containing information about the process. Here is some output given to the user (in the low part of the screen) while executing the script:



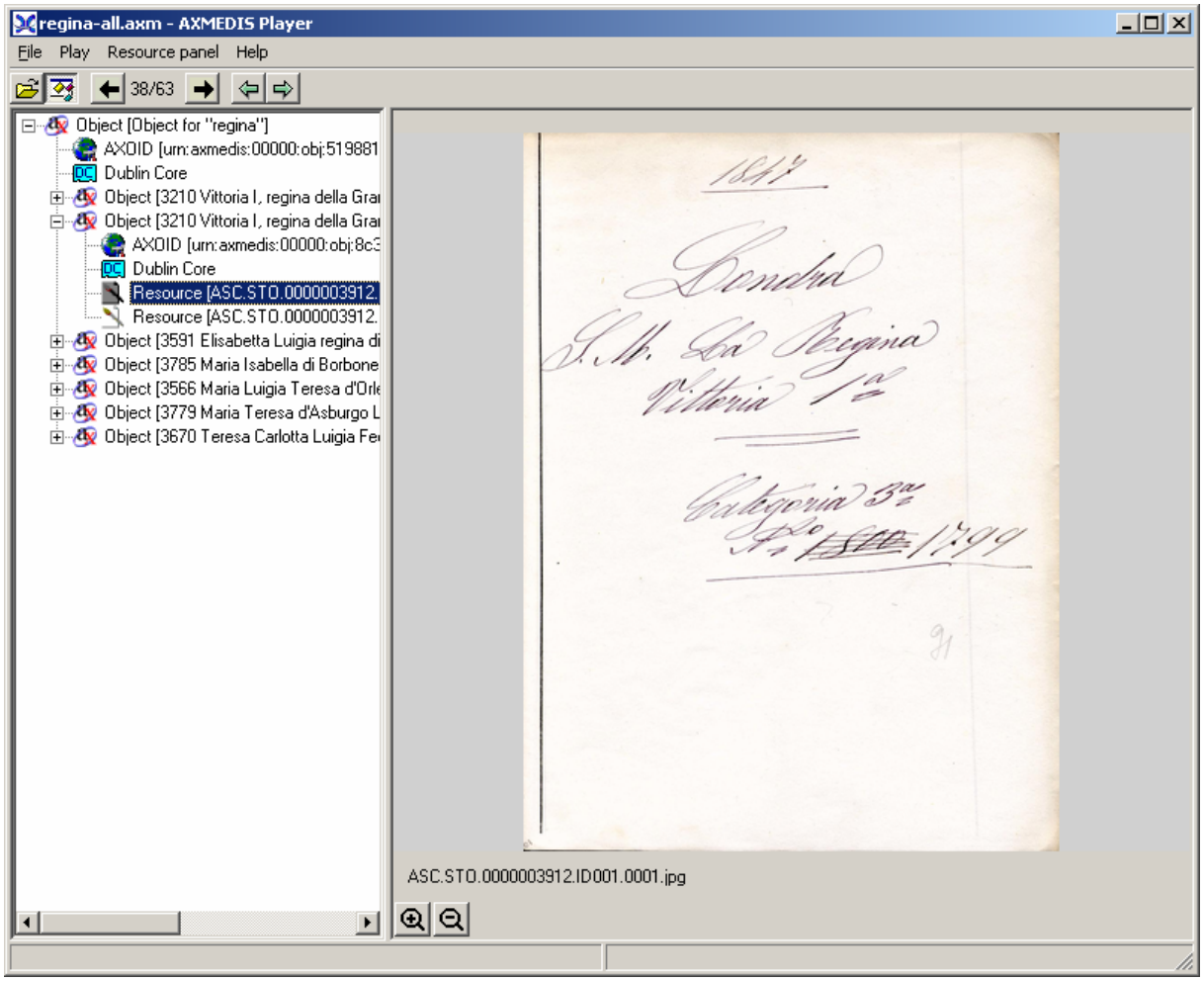
Following is an object obtained by searching for “rossini":



The images added as resources can be seen in the hierarchical view of the object on the left.
In the following the metadata (as seen in the Axmedis Player) is shown:

Title, identifier, source, and type were automatically extracted, but further metadata could be added.

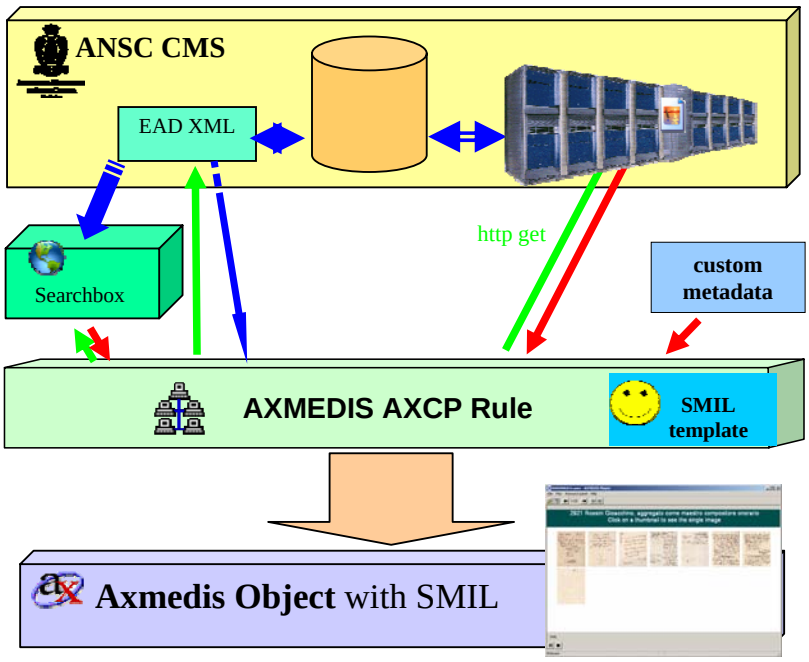
Here is an object containing all the objects resulting for the search “regina”. Each object has its own resources and metadata.



7.3.4 Content Formatting using SMIL (ANSC)

As a further development and exploitation of the AXCP tools an improved version of the crawling script has been developed that will create a SMIL-based user interface presenting clickable image thumbnails. SMIL is the Synchronized Multimedia Integration Language, a W3C Recommendation;[15] it is a mark-up language for rich media presentation with interactive functionalities. For this prototype a SMIL template was created; the template will be automatically filled in by the script. Here is the diagram with the general concept:

[15] See http://www.w3.org/TR/SMIL2/



This is just like the crawling rule described above, but here a specific create_smil function has been added which will be called by the main crawlStorico function. This function could in fact also be used independently:

```
/*
This function creates an 800x600 smil containing
clickable thumbnails of images given in the
imagenames array
the title will be added to a text-file called
dctitle.txt in the same dir.
Also a file called instructions.txt with the text
for going back must be in same dir.
For the SMIL to work properly
all the images must be in the same directory.
*/
function create_smil (outdir, outfile, imagenames, smil_title)
{
    var titlebcgcolor="#004F4F"; // the color that will appear in the title
background
    var titlefontcolor="#FFEFFF";  // the color for the title text
    // SMIL header template
    var top_smil = "<?xml version=\"1.0\" encoding=\"ISO-8859-1\"?>\n"
        +"<smil xmlns=\"http://www.w3.org/2005/SMIL21/\">\n<head>\n"
        +"<layout type=\"text/smil-basic-layout\">\n"
        +"<root-layout width=\"800\" height=\"600\"
backgroundColor=\"white\"/>\n"
        +"  <region id=\"title\" width=\"800\" height=\"60\" top=\"0\"
left=\"0\" backgroundColor=\""+titlebcgcolor+"\" z-index=\"1\"/>\n
      <region id=\"thumbs\" width=\"790\" height=\"auto\" top=\"65\" left=\"5\"
backgroundColor=\"transparent\" fit=\"scroll\" z-index=\"1\"/>\n\n  </layout>\n
      \n  <transition id=\"fade\" type=\"fade\" />\n  <transition id=\"wipeLR\"
type=\"barWipe\" subtype=\"leftToRight\" dur=\"2.5s\"/>\n  <paramGroup id
=\"titletext\">\n       <param name=\"font-size\" value=\"20\"/>\n
```

```
      <param name=\"font-family\" value=\"arial\"/>\n      <param name=\"color\"
value=\""+titlefontcolor+"\" />\n        <param name=\"transIn\" value=\"fade\"
/>\n  </paramGroup>\n</head>\n<body>\n\n<par begin=\"0s\" dur=\"indefinite\">\n
<text region=\"title\" src=\"dctitle.txt\" paramGroup=\"titletext\"/>\n\n";
    var thumbs_smil ="";
    // add thumbnails to SMIL
    for (im=0;im < imagenames.length; im++)
    {
        counter = im+1;
        var num="";
        if (counter < 10) num = "000"+counter;
        else if (counter >= 10 && counter < 100) num = "00"+counter;
        else if (counter >= 100 && counter < 1000) num = "0"+counter;
        else if (counter >= 1000 && counter< 1000) num = counter;
        var top = (Math.floor(im/7) * 125)+10;
        var left= ((im % 7)*105)+30;
        var height="120";
        var width="95";
        thumbs_smil = thumbs_smil +"  <img id=\""+num+"_thumb\"
region=\"thumbs\" src=\""+imagenames[im]+"\" top=\""+top+"\" left=\""+left+"\"
height=\""+height+"\" width=\""+width+"\" fit=\"fill\">\n      <area
title=\"Click here to see the image\" href=\""+num+".smil\" top=\""+top+"\"
left=\""+left+"\"   height=\""+height+"\"  width=\""+width+"\" />\n</img>\n";
        var imageview_top ="<?xml version=\"1.0\" encoding=\"ISO-8859-
1\"?>\n<smil xmlns=\"http://www.w3.org/2005/SMIL21/\">\n<head>\n  <layout
type=\"text/smil-basic-layout\">\n    <root-layout width=\"800\" height=\"600\"
backgroundColor=\"transparent\"/>\n      <region id=\"title\" width=\"800\"
height=\"30\" top=\"0\" left=\"0\" backgroundColor=\""+titlebcgcolor+"\" z-
index=\"1\"/>\n        <region id=\"thumbs\" top=\"30\" heigth=\"570\"
regAlign=\"center\" regPoint=\"center\" backgroundColor=\"transparent\"
fit=\"meet\" z-index=\"1\"/>\n\n  </layout>\n  \n  <transition id=\"fade\"
type=\"fade\" />\n  <transition id=\"wipeLR\" type=\"barWipe\"
subtype=\"leftToRight\" dur=\"2.5s\"/>\n <paramGroup id =\"titletext\">\n
      <param name=\"font-size\" value=\"18\"/>\n           <param name=\"font-
family\" value=\"arial\"/>\n       <param name=\"color\"
value=\""+titlefontcolor+"\" />\n        <param name=\"transIn\" value=\"fade\"
/>\n        <param name=\"transOut\" value=\"wipeLR\" />\n </paramGroup>\n
      \n</head>\n<body>\n  <par begin=\"0s\" dur=\"indefinite\">\n     <text
region=\"title\" src=\"instructions.txt\" paramGroup=\"titletext\">\n
                  <area title=\"torna\" href=\""+outfile+"\" />\n
      </text>\n";
        var image_section = "<img id=\""+num+"_show\" region=\"thumbs\"
src=\""+imagenames[im]+"\" regAlign=\"center\" regPoint=\"center\"
fit=\"meet\">\n \n           </img>\n";
        var imageview_bottom ="    </par>\n</body>\n</smil>";
        var imageview_smil = imageview_top+image_section+imageview_bottom;
        writeToFile (outdir+"\\"+num+".smil",imageview_smil);
    }
    // create dctitle.txt and add to SMIL
    writeToFile (outdir+"\\"+"dctitle.txt",smil_title);
    // create instructions.txt and add to SMIL
    writeToFile (outdir+"\\"+"instructions.txt","Click here to go back");
    var bottom_smil = "\n </par>\n</body>\n</smil>";
    var smil_index =top_smil+thumbs_smil+bottom_smil;
    writeToFile (outdir+"\\"+outfile,smil_index);
}
```

Following is the crawlStorico function where the SMIL files are created and then added as resources to the object:

```
function crawlStorico
    (title_string,      // string to search in the titles
    temp_dir,           // temp dir (full path) - will be created if it doen't
exist
    output_dir,         // uotput dir where ALL the objects will be saved - will
be created if it doen't exist
    DC_file,            // DC xml file (full path) to add to each object. "" for
none
    searchbox_host,     // searchbox host (e.g. "localhost")
    searchbox_port,     // searchbox port (e.g. "2200")
    searchbox_username, // searchbox username
    searchbox_password, // searchbox password
    http_prefix,        // prefix for getting images via http for ALL objects
    create_object)      // if true all the possibly created
                        //objects will be added to a unique one only if 2 or
more
{
    var foundURLs =  searchbox_getURL (title_string,
                            searchbox_host,
                            searchbox_port,
                            searchbox_username,
                            searchbox_password);
    var all_records = new Array();
    print
("****************************************************************");
    var global_error = "";
    var total_objects = 0;
    var all_objects = new Array();
    appendToFile (output_dir+"\\_log.txt","\n------------ Report searching for
\""+title_string+"\"----------\n\n");
    for (ix=0; ix < foundURLs.length; ix++)
    {
        foundURLs[ix] = eliminate_file_prefix (foundURLs[ix]);
        var error = "";
        print ("Searching for images in " +foundURLs[ix] +" for
\""+title_string+"\". please wait for results...");
        if (!existsFile (foundURLs[ix]))
        {
            break;
        }
        URL_contents = readFromFile (foundURLs[ix])
        record = get_matching_components (URL_contents, title_string)
        // try to create an object
        for (kx = 0; kx < record.length; kx++)
        {
            var this_record = record[kx];
            error= error + "Found ID:"+this_record.c_id + " -> title: "
+this_record.unittitle+" in  "+ foundURLs[ix]+"\n";
            error= error + "Trying to create object...\n";
            // only if daoloc_hrefs (i.e. images) are found create an object...
            if (this_record.daoloc_hrefs == null)
            {
            error = error + "no images found for "+this_record.c_id + " ->
title: " +this_record.unittitle+"\n";
            break;
            }
            var axObj = new AxmedisObject(); // new object
            res = new AxResource;
```

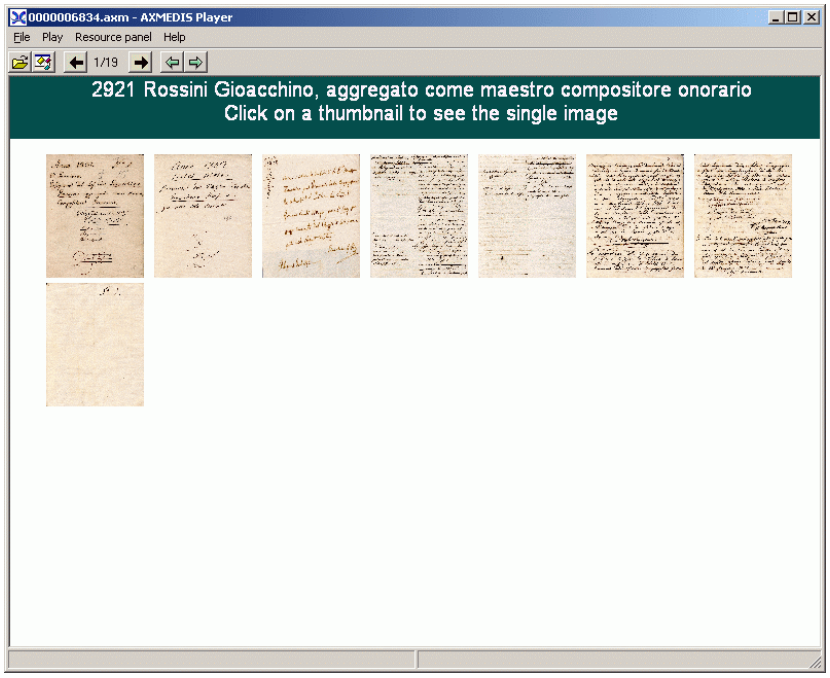
```
            // now add the attachments
            error = error+ "Trying to get resources via http...\n"
            var foundfiles = 0;
            var clean_title_string = epurate_pathname(title_string);
            var savedir = output_dir+"\\"+ clean_title_string
+"\\"+this_record.c_id;
            images_to_add = this_record.daoloc_hrefs.length;
            if (images_to_add > 100) {
                print ("WARNING: Found " +images_to_add  +" images. it may take
long - please be patient!");
            }
            var images4smil = new Array();
            for (iy=0; iy < this_record.daoloc_hrefs.length; iy++)
            {
                var noadd = false;
                if (Math.floor((iy+1) % 50) == 0) {
                    print ("saving...");
                    if (!existsDir (temp_dir)) {
                        createDir (temp_dir);
                    }
                    axObj.save  (temp_dir+"\\~"+this_record.c_id +iy+".axm")
                    axObj = new AxmedisObject (temp_dir+"\\~"+this_record.c_id
+iy+".axm");
                }

                if (this_record.daoloc_hrefs[iy].match (".ASC.")) {
                    var the_file = gethttp
(http_prefix+"low"+this_record.daoloc_hrefs[iy],
temp_dir+"\\"+this_record.c_id);
                    if (the_file != false)
                    {
                        var res = new AxResource ();
                        res.load (the_file);
                        // uncomment if using a version of the rule editor
earlier than 1.0
                        //var newID = res.contentID +
mimeTypeToExt(res.mimeType);
                        //res.contentID = newID;
                    }
                    else
                    {
                        noadd  = true;
                    }
                }
                else {
                    this_record.daoloc_hrefs[iy] =
this_record.daoloc_hrefs[iy].replace ("\\","/");

                    var the_file = gethttp
(http_prefix+this_record.daoloc_hrefs[iy], temp_dir+"\\"+this_record.c_id);
                    if (the_file != false)
                    {
                        var res = new AxResource ();
                        res.load (the_file);
                        // uncomment if using a version of the rule editor
earlier than 1.0
                        //var newID = res.contentID +
mimeTypeToExt(res.mimeType);
                        //res.contentID = newID;
```

```
                    }
                    else
                    {
                        noadd  = true;
                    }
                }
                if (noadd == false)
                {
                 axObj.addContent (res);
                  images4smil.push (res.contentID);
                  foundfiles = foundfiles +1;
                  print ("- Adding "+this_record.daoloc_hrefs[iy]+" to object -
"+(iy+1)+" of "+images_to_add+" ("+(Math.round(iy*100/images_to_add))+"%)");
                }
                else
                {
                  // uncomment the line below to get extensive error
                  //error= error + "HTTP ERROR trying to add
"+http_prefix+this_record.daoloc_hrefs[iy]+"\n";
                }
            }

            if (foundfiles >0)
            // i.e. if at least a resource was successfully added
            {
                // add DC related to this object
                dc = axObj.getDublinCore ();
                dc.addDCElement ("title",this_record.unittitle,"");
                dc.addDCElement ("identifier",this_record.c_id,"");
                if (this_record.unittitle_type != null &&
this_record.unittitle_type != "")
                {
                    dc.addDCElement ("type",this_record.unittitle_type ,"it");
                }
                if (this_record.container_type != null)
                {
                    if (this_record.container_text == null) {
                        dc.addDCElement
("description",this_record.container_type,"it");
                    }
                    else {
                        dc.addDCElement ("source","Archivio Storico -
"+this_record.container_type+" "+this_record.container_text,"it");
                    }
                }

                if (DC_file != "" && (addDC (axObj, DC_file))) // add custom DC
from file
                {
                    error = error+ "Adding DC from "+DC_file+"\n"
                }
                else
                {
                    error = error+ "ERROR: Impossible to add DC from "+DC_file+"
Does it exist?\n"
                }
                // create the SMIL
                print ("Creating SMIL for "+this_record.unittitle);
                error = error + "Creating SMIL for "+this_record.unittitle+"\n";
```

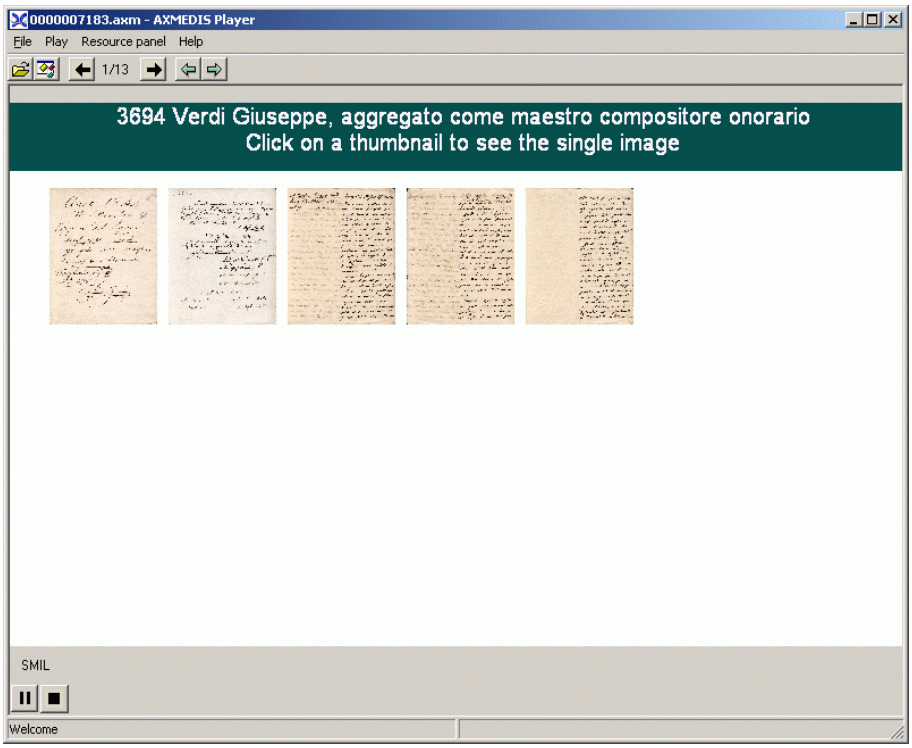
```
                var smiltempdir =temp_dir+"\\"+this_record.c_id+"\\SMIL";
                createDir (smiltempdir);
                create_smil (smiltempdir, "index.smil", images4smil,
this_record.unittitle+"\nClick on a thumbnail to see the single image");
                createDir (savedir);
                var smile_files = getfilelist (smiltempdir,"*.*");
                for (ismil=0;ismil<smile_files.length;ismil++)
                {
                    var smil_res = new AxResource();
                    smil_res.load (smile_files[ismil]);
                    // UNCOMMENT if using rukle editor version < 1.0
                    //if (smil_res.mimeType == "application/smil") smil_resID =
smil_res.contentID + ".smil";
                    //else smil_resID = smil_res.contentID +
mimeTypeToExt(smil_res.mimeType);
                    //smil_res.contentID = smil_resID;
                    // END UNCOMMENT
                    if (smil_res.contentID == "index.smil")
                    {
                        var all_res = axObj.getContent ();
                        axObj.insertContent (smil_res,all_res[0],true);
                    }
                    else
                    axObj.addContent (smil_res);
                }
                axObj.save (savedir+"\\"+this_record.c_id +".axm");
                error = error + "Managed to find and add "+foundfiles+"
resource(s)\n"
                error = error + "SUCCESS: Saved object:
\""+savedir+"\\"+this_record.c_id +".axm\"\n";
                total_objects = total_objects+1;
                all_objects.push (savedir+"\\"+this_record.c_id +".axm");
            }
            else
            {
            error = error + "ERROR: Did not save any object because there were
no resources to add: maybe http error? (see above)\n";
            }
        print (error);
        appendToFile (output_dir+"\\_log.txt", error);
        }
    }
    print ("--- Saved "+total_objects+ " objects in total: see
"+output_dir+"\\_log.txt for details");
    // save the log
    appendToFile (output_dir+"\\_log.txt","\n--- Saved "+total_objects+ "
objects in total ---\n");
    // if a unique object must be created
    if (create_object && total_objects >1)
    {
    all_axObj = new AxmedisObject ();
    this_object = new AxmedisObject ();
    var save_name = output_dir+"\\"+epurate_pathname (title_string)+"-all.axm";
    print ("\n--- Adding all found objects to "+save_name);
        for (iobj =0; iobj < all_objects.length; iobj++)
        {
            if (Math.floor((iobj+1) % 50) == 0) {
                    print ("saving...");
                    if (!existsDir (temp_dir)) {
```

```
                        createDir (temp_dir);
                    }
                    all_axObj.save  (temp_dir+"\\~"+epurate_pathname
(title_string)+"-all.axm");
                    all_axObj = new AxmedisObject
(temp_dir+"\\~"+epurate_pathname (title_string)+"-all.axm");
                }

            print ("- Adding "+all_objects[iobj]);
            this_object = new AxmedisObject(all_objects[iobj]);
            var this_DC = this_object.getDublinCore ();
            var the_title = this_DC.getDCElementValue ("title");
            this_object.contentID = the_title;
            all_axObj.addContent (this_object);
        }
    // add DC related to this object
    all_dc = all_axObj.getDublinCore ();
    all_dc.addDCElement ("title","Search results for \""+title_string+"\"","");
    all_axObj.contentID = ("Object for \""+title_string+"\"");
    all_axObj.save (save_name);
    appendToFile (output_dir+"\\_log.txt","\n--- Added all objects to
"+save_name+"---\n");
    }
    else
    {
        print ("**WARNING: Did not create global object because only
"+total_objects+ " objects produced");
        appendToFile (output_dir+"\\_log.txt","WARNING: Did not create global
object because only "+total_objects+ " objects produced\n");
    }
    appendToFile (output_dir+"\\_log.txt","\n -------- end report searching for
\""+title_string+"\"-------\n");
}
```

Here is the object created after searching for “rossini”:



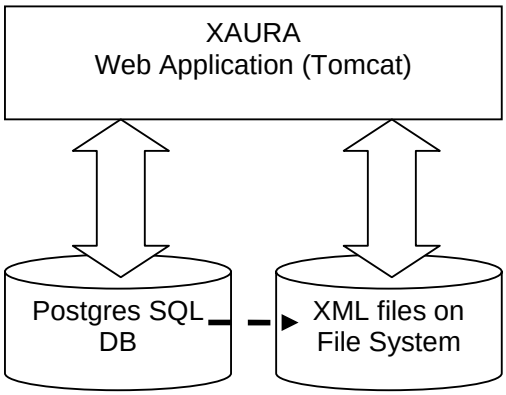
Notice that the title is extracted from the Dublin Core Title. The metadata are still present and some resources for the SMIL have been added. Here is an object created by searching for “verdi”:



7.4 XAURA - TISCALI

7.4.1 Description of the CMS

The structure of the CMS is shown in figure:



The XAURA web application uses the Postgres SQL DB to contain minimal metadata and associated with each record is present in the file system an XML file with the metadata related with the content..

7.4.2 Description of prototype and interfacing with AXCP Area

The searchbox tool is used to import metadata from the XAURA CMS, afterwards using an AXCP rule some of the movie information get from CMS can be transformed into axmedis objects.

The searchbox is configured in this way:

- a source is configured to get from the postgres database all the movies using an ODBC plugin and an XML plugin to parse the associated information
- a source is configured to get from postgres database information on all the persons (actors, directors) using the ODBC plugin and an XML plugin to parse the associated information

The odbc connection to get the movies from the database is configured as:

```
select id,name, 'file:///D:/Programmi/Apache Software
Foundation/Tomcat
5.5/webapps/cms/projects/mediaclub/content/film/'||id||'.xml' from
contents where content_type_id='film' --!!!PKA!!!—
```

And the ODBC plugin is configured as:

```
FFF
DM        title id   1
T         title      2
L         centralNorm          3
```

The third field is a link to an xml file (containg the real metadata) that is parsed using the XML plugin.
The XML plugin is configured with:

```
XPATH
DM   NT   title !Title     @string(/document/title)
DM   NT   title !DCMI.title     @string(/document/title)
T         title      @string(/document/title)
DM   NT   title !Description    @string(/document/abstract)
DM   NT   title !DCMI.description    @string(/document/abstract)
T    NT   centralNorm          @string(/document/abstract)
DM   NT   centralNorm     !Type @string(/document/genre)
DM   NT   centralNorm     !DCMI.type @string(/document/genre)
DM   NT   centralNorm     !directorid
     @string(/document/director/@id)
DM   NT   centralNorm     !actorid   @/document/actor/@id
DM        centralNorm     !filmid    @substring-
before(string(/document/image/@id),'.image')
DM   NT   centralNorm     !genre     @string(/document/genre)
DM   NT   centralNorm     !credits   @string(/document/credits)
DM   NT   centralNorm     !Distributor
     @string(/document/distributor)
DM   NT   centralNorm     !plot @string(/document/plot)
DM   NT   centralNorm     !DCMI.date @string(/document/year)
DM   NT   centralNorm     !date @string(/document/year)
DM        centralNorm     !imagePath @concat("D:/Programmi/Apache
Software Foundation/Tomcat
5.5/webapps/cms/projects/mediaclub/media/image/jpeg/",string(/docum
ent/image/@id),".jpg")
DM        centralNorm     !posterPath
     @concat("D:/Programmi/Apache Software Foundation/Tomcat
5.5/webapps/cms/projects/mediaclub/media/image/jpeg/",string(/docum
ent/poster/@id),".jpg")
DM        centralNorm     !downloadPreviewUrl
     @string(/document/prevdown/url)
```

```
DM          centralNorm     !streamPreview512Url
      @string(/document/prevstream[bitrate=512]/url)
DM          centralNorm     !streamPreview256Url
      @string(/document/prevstream[bitrate=256]/url)
DM          centralNorm     !downloadVideoUrl
      @string(/document/videodown/url)
DM          centralNorm     !streamVideo512Url
      @string(/document/videostream[bitrate=512]/url)
DM          centralNorm     !streamVideo256Url
      @string(/document/videostream[bitrate=256]/url)
L           centralLink
      @concat("file:///D:/Programmi/Apache Software
Foundation/Tomcat
5.5/webapps/cms/projects/mediaclub/media/image/jpeg/",string(/docum
ent/poster/@id),".jpg")
```

The configuration to get the information related to person is:

```
select id,name, 'file:///D:/Programmi/Apache Software
Foundation/Tomcat
5.5/webapps/cms/projects/mediaclub/content/person/'||id||'.xml'
from contents where content_type_id='person' --!!!PKA!!!--
```

The configuration of the ODBC plugin is:

```
FFF
DM          title id    1
T           title       2
L           centralNorm           3
```

The third field is a link to an XML file that is parsed using an XML plugin with the configuration:

```
XPATH
T     NT    title       @string(/document/name)
DM    NT    title !name @string(/document/name)
DM          centralNorm     !personid  @substring-
before(string(/document/picture/@id),'.picture')
DM    NT    centralNorm     !bio  @string(/document/bio)
DM    NT    centralNorm     !filmography
      @string(/document/filmography)
DM          centralNorm     !picturePath
      @concat("D:/Programmi/Apache Software Foundation/Tomcat
5.5/webapps/cms/projects/mediaclub/media/image/jpeg/",string(/docum
ent/picture/@id),".jpg")
DM    NT    centralNorm     !films     @string(/document/films)
```

The basic metadata of the movie are mapped to Dublin core and specific metadata is stored in the searchbox as shown in the next two tables:

Searchbox metadata name	**XPATH**
Title	/document/title
DCMI.title	/document/title
Description	/document/abstract
DCMI.description	/document/abstract
Type	/document/genre
DCMI.type	/document/genre
Directorid	/document/director/@id
Actorid	/document/actor/@id
Filmid	substring-before(string(/document/image/@id),'.image') genre /document/genre

Credits	/document/credits
Distributor	/document/distributor
Plot	/document/plot
DCMI.date	/document/year
Date	/document/year
imagePath	concat("D:/Programmi/Apache Software Foundation/Tomcat 5.5/webapps/cms/projects/mediaclub/media/image/jpeg/", string(/document/image/@id),".jpg")
posterPath	concat("D:/Programmi/Apache Software Foundation/Tomcat 5.5/webapps/cms/projects/mediaclub/media/image/jpeg/", string(/document/poster/@id),".jpg")
downloadPreviewUrl	/document/prevdown/url
streamPreview512Url	/document/prevstream[bitrate=512]/url
streamPreview256Url	/document/prevstream[bitrate=256]/url
downloadVideoUrl	/document/videodown/url
streamVideo512Url	/document/videostream[bitrate=512]/url
streamVideo256Url	/document/videostream[bitrate=256]/url

Searchbox metadata name	**XPATH**
title	/document/name
name	/document/name
personid	Substring-before(string(/document/picture/@id),'.picture')
bio	/document/bio
filmography	/document/filmography
picturePath	concat("D:/Programmi/Apache Software Foundation/Tomcat 5.5/webapps/cms/projects/mediaclub/media/image/jpeg/", string(/document/picture/@id),".jpg")
films	/document/films

7.4.3 AXCP Scripts and their description

```
sb.host="localhost"
sb.port="2200"
sb.username="user"
sb.password="****"

var r=new Array
cmsQuery("film",importQuery,r)
print("Starting to import " + r.length + " films...");
for(i=0; i<r.length; i++)
{
    var docid=r[i].id
    print("building movie "+(i+1))
    buildMovie(docid)
}

function buildMovie(docid)
{
    var axobj=buildFilmObject(docid);

    if(uploadOnDB)
```

```
    {
        axobj.uploadToDB()
        storeAXOID(docid,axobj.AXOID)
    }
    else
        axobj.save(savePath+"/films/"+axobj.contentID+".axm")
    axobj=null
}

function buildFilmObject(docid)
{
    newObject=true
    var m=sb.getDocumentMetadata(docid)
    var links=sb.getDocumentOutlinks(docid);

    //printCmsMetadata(m)

    title=m.getValue("Title")
    axoid=m.getValue("axoid")
    print("building film "+title+"...")

    if(forceNewObject)
        axobj=new AxmedisObject
    else if(!uploadOnDB || axoid=="")
    {
        if(!uploadOnDB && existsFile(savePath+"/films/"+toID(title)+".axm"))
        {
            print("  loading old object from file...")
            axobj=new AxmedisObject(savePath+"/films/"+toID(title)+".axm")
        }
        else
            axobj=new AxmedisObject
    }
    else
    {
        print("  retrieving old object from DB...")
        axobj=new AxmedisObject("axdb://"+axoid)
        newobject=false
    }

    axFilmDC=axobj.getDublinCore()
    updateAxInfo(axobj.getAxInfo())
    axobj.getAxInfo().objectRevision=-1

    updateDCMetadata(axFilmDC, "title", title, "ita")

    type=m.getValue("Type")
    updateDCMetadata(axFilmDC, "type", type, "ita")

    date=m.getValue("date")
    updateDCMetadata(axFilmDC, "date", date)

    description=m.getValue("Description")
    updateDCMetadata(axFilmDC, "description", description, "ita")

    posterPath=m.getValue("posterPath")
    if(posterPath!="")
    {
        axres=addResourceById(axobj,"poster.jpg")
```

```
        axres.load(posterPath)
        axlogo=new AxResource
        axlogo.load("C:/Documents and
Settings/Piero/Documenti/Immagini/axlogo.png")
        ImageProcessing.Paste(axres,axlogo,0,0,1,axres)
    }

    plot=m.getValue("plot")
    if(plot!="")
    {
        writeToFile("c:/tmp.htm",
        "<html><head><meta content='text/html;charset=utf-8' http-
equiv='content-type'></head>" +
        "<body><h1>"+title+"</h1>"+
        "<a href='poster.jpg'><img src='poster.jpg' height='200'/></a><br><br>"+
        plot+"<br><br>"+
        "<img src='image.jpg'/></body></html>")
        axres=addResourceById(axobj,"plot.htm")
        axres.load("C:/tmp.htm")
        removeFile("C:/tmp.htm")
    }

    imagePath=m.getValue("imagePath")
    if(imagePath!="")
    {
        axres=addResourceById(axobj,"image.jpg")
        axres.load(imagePath)
    }

    downloadPreviewUrl=m.getValue("downloadPreviewUrl")
    if(downloadPreviewUrl!="")
    {
        axres=addResourceById(axobj,"preview-download")
        axres.ref=downloadPreviewUrl
        axres.mimeType="video/x-ms-wmv"
    }
    streamPreview512Url=m.getValue("streamPreview512Url")
    if(streamPreview512Url!="")
    {
        axres=addResourceById(axobj,"preview-stream-512")
        axres.ref=streamPreview512Url
        axres.mimeType="video/x-ms-wmv"
    }

    directorid=m.getValue("directorid")
    if(directorid!="")
    {
        rs=buildPersonObject(directorid, "directors")
        if(rs!=null)
        {
            updateDCMetadata(axFilmDC, "creator", rs[0])
            if(buildCompositeObject=="embed")
            {
                removeContentById(axobj,rs[1].contentID)
                axobj.addContent(rs[1])
            }
            if(uploadOnDB)
            {
                rs[1].uploadToDB()
```

```
                storeAXOID(rs[2], rs[1].AXOID)
            }
            else
                rs[1].save(savePath+"/directors/"+toID(rs[0])+".axm")
        }
        else
            ; //print("    director: "+directorid+ " not found")
    }

    actorids=m.getValues("actorid")
    for(actor in actorids)
    {
        rs=buildPersonObject(actorids[actor], "actors")
        if(rs!=null)
        {
            updateDCMetadata(axFilmDC, "contributor", rs[0])
            if(buildCompositeObject=="embed")
            {
                removeContentById(axobj,rs[1].contentID)
                axobj.addContent(rs[1])
            }
            if(uploadOnDB)
            {
                rs[1].uploadToDB()
                storeAXOID(rs[2], rs[1].AXOID)
            }
            else
                rs[1].save(savePath+"/actors/"+toID(rs[0])+".axm")
        }
        else
            ; //print("    actor: "+actorids[actor]+ " not found")
    }

    axobj.contentID=toID(title)
    return axobj
}

function buildPersonObject(personid, type)
{
    persons=new Array
    if(cmsQuery("person","personid:"+personid, persons)>0)
    {
        pm=sb.getDocumentMetadata(persons[0].id)

        name=pm.getValue("name")
        axoid=pm.getValue("axoid")
        contentId=toID(name)

        newobject=true
        print("    building person "+name+"...")
        if(forceNewObject)
            axPersonObj=new AxmedisObject
        else if(!uploadOnDB || axoid=="")
        {
           if(!uploadOnDB &&
existsFile(savePath+"/"+type+"/"+toID(name)+".axm"))
            {
                print("      loading old object from file...")
```

```
                axPersonObj=new
AxmedisObject(savePath+"/"+type+"/"+toID(name)+".axm")
            }
            else
                axPersonObj=new AxmedisObject
        }
        else
        {
            print("      retrieving old object...")
            axPersonObj=new AxmedisObject("axdb://"+axoid)
            newobject=false
        }

        ret=new Array
        axPersonDC=axPersonObj.getDublinCore()
        updateAxInfo(axPersonObj.getAxInfo())

        if(name!="")
        {
            axPersonObj.contentID=contentId
            updateDCMetadata(axPersonDC,"title", name)
        }

        bio=pm.getValue("bio")
        if(bio!="")
        {
            writeToFile("c:/tmp.htm","<html><head><meta
content='text/html;charset=utf-8' http-equiv='content-type'></head>"+
              "<body><h1>"+name+" - Biografia</h1>"+
              "<img src='picture.jpg'/><br><br>"+bio+"<br><br>"+
              "<a href='filmography.htm'>Filmografia</a></body></html>")
            axres=addResourceById(axPersonObj,contentId+"/biography.htm")
            axres.load("C:/tmp.htm")
            removeFile("C:/tmp.htm")
        }
        picturePath=pm.getValue("picturePath")
        if(picturePath!="")
        {
            axres=addResourceById(axPersonObj,contentId+"/picture.jpg")
            axres.load(picturePath)
        }
        filmography=pm.getValue("filmography")
        if(filmography!="")
        {
            writeToFile("c:/tmp.htm","<html><head><meta
content='text/html;charset=utf-8' http-equiv='content-type'></head>"+
              "<body><h1>"+name+" -
Filmografia</h1>"+filmography+"</body></html>")
            axres=addResourceById(axPersonObj,contentId+"/filmography.htm")
            axres.load("C:/tmp.htm")
            removeFile("C:/tmp.htm")
        }
        ret[0]=name
        ret[1]=axPersonObj
        ret[2]=persons[0].id
        return ret
    }
    return null
}
```

```
function updateAxInfo(axinfo)
{
    if(axinfo.objectCreatorCount==1 && axinfo.getObjectCreatorAXCID(0)=="")
    {
        axinfo.setObjectCreatorAXCID(0,"CRE_a45168bf-d02d-4ba6-b4ad-
c4e94b270369")
        axinfo.setObjectCreatorName(0,"automatic importer")
        axinfo.setObjectCreatorNationality(0,"ITA")
    }
    axinfo.distributorName="TISCALI"
    axinfo.objectVersion=1
    axinfo.objectRevision=0
}

function putLogo(resource, logo)
{
    imageprocessingplugin.Paste(resource,logo,0,0,1,resource)
}
function updateDCMetadata(axDC, dcName, dcValue, dcLang)
{
    if(axDC.getDCElementCount(dcName)==0)
    {
        if(dcValue!="")
        {
            if(dcLang==null)
                axDC.addDCElement(dcName, dcValue)
            else
                axDC.addDCElement(dcName, dcValue, dcLang)
        }
    }
    else
    {
        axDC.setDCElementValue(dcName,0,dcValue)
        if(dcLang!=null)
            axDC.setDCElementLanguage(dcName,0,dcLang)
    }
}
function addResourceById(axobj, id)
{
    content=axobj.getContent()
    if(content!=null)
    {
        for(xx=0; xx<content.length; xx++)
            if(content[xx].contentID==id)
                return content[xx]
    }
    axres1=new AxResource;
    axres2=axobj.addContent(axres1);
    axres2.contentID=id
    return axres2
}
function removeContentById(axobj, id)
{
    content=axobj.getContent()
    if(content!=null)
    {
        for(xx=0; xx<content.length; xx++)
            if(content[xx].contentID==id)
```

```
            {
                axobj.removeContent(content[xx])
                return
            }
    }
    return
}
function toID(str)
{
    while(str.indexOf(" ")>=0)
        str=str.replace(" ","-")
    return str
}
function xmlEncode(str)
{
    while(str.indexOf("<")>=0)
        str=str.replace("<","&lt;")
    while(str.indexOf(">")>=0)
        str=str.replace(">","&gt;")
    while(str.indexOf("<")>=0)
        str=str.replace("<","&lt;")
    return str
}

function cmsQuery(archive, queryString, results)
{
    var qs=new QuerySpec;
    if(archive=="film")
        a = [17];
    else if(archive=="person")
        a = [15];
    else
        return 0
    qs.archives = a;
    qs.info=QueryInfo.INFO_ALL_METADATA
    //qs.firstDoc = 0;
    qs.lastDoc = 1000;
    qs.queryString=queryString
    return sb.query(qs,results);
}
function storeAXOID(sbDocId, axoid)
{
    templateMetadata= [ new MetadataValue ]
    templateMetadata[0].key="axoid"
    templateMetadata[0].slice=13
    templateMetadata[0].value=axoid
    sb.applyMetadataTemplate(sbDocId,3,templateMetadata)
}
function printCmsMetadata(m)
{
    for(x=0; x<m.length; x++)
        print(m[x].key+" --> "+m[x].value)
}
```

7.5 Digital Media Platform (DMP) - HP

7.5.1 Description of the Digital Media Platform

HP Digital Media Platform is an integration platform for media solutions.

It provides a comprehensive framework that offers:

- A plug and play model for integrating solution components across multiple platforms
- Pre-built set of services that provide infrastructure functionality
- A workflow model that provides supports for flexible and granular workflows
- Extensible metadata model that provides core entities required in media solutions
- Seamless interoperability between services running on the Windows and Unix platform.

The following picture gives an overview of the DMP architecture:

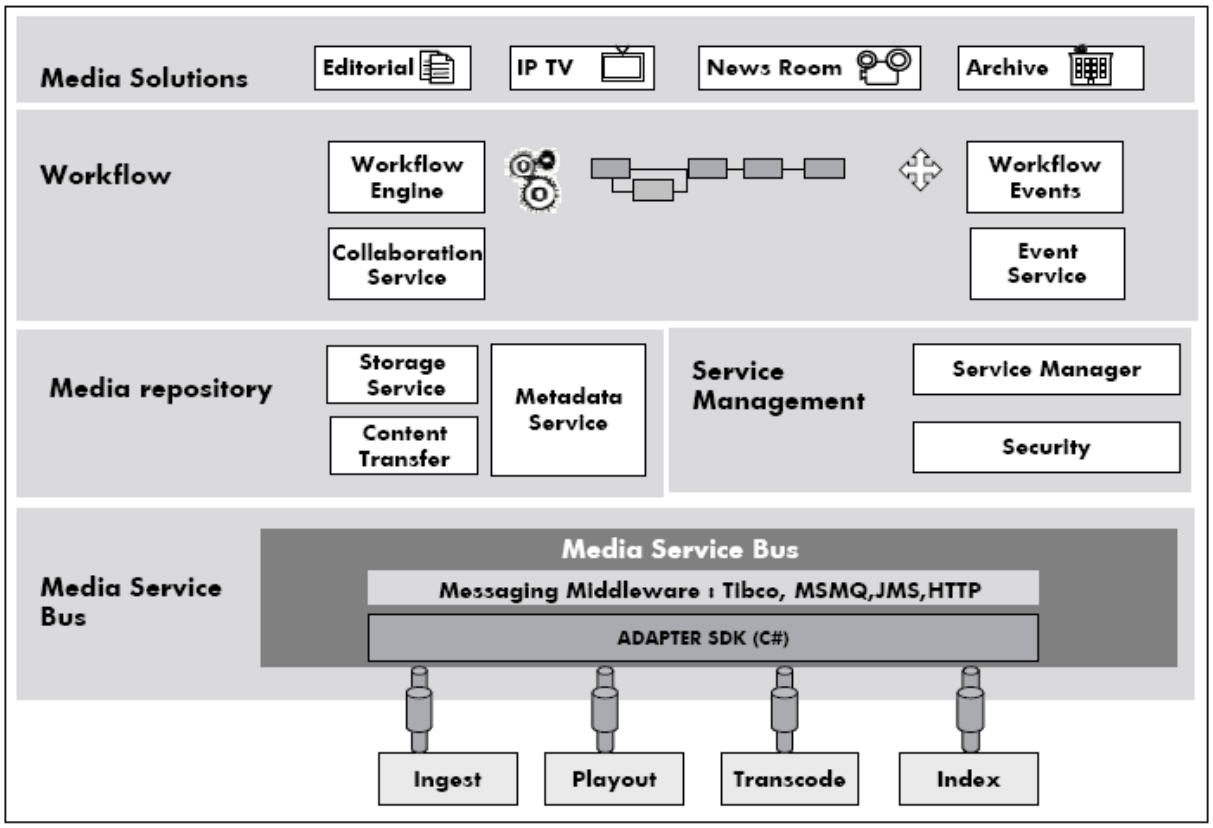


Figure 1: DMP architecture

The Digital Media Platform metadata service provides centralized access to all assets, that DMP manages. The metadata service provides an implementation of the DMP core metadata model over MS SQL Server. However any popular Digital Asset Management (DAM) product can be integrated as the platform's asset store.

The metadata service implements functionality to create, manage and locate assets from the metadata store. The metadata service is installed as an NT service on the DMP server host by the DMP server installer.

The DMP provides access to content to content distributors; content can be streamable content, or any kind of content that can be distributed through various channels. Stramable content could be files stored on a file system (such as MPEG2 files), content received from air transmission (satellite or antenna), which will be made available to encoders, broadcasters, portals, etc etc.

7.5.2 Description of prototype and interfacing with AXCP Area

The DMP is an orchestrator for content distributors. One kind of content it can manage is files stored on filesystem, which can be streamed.

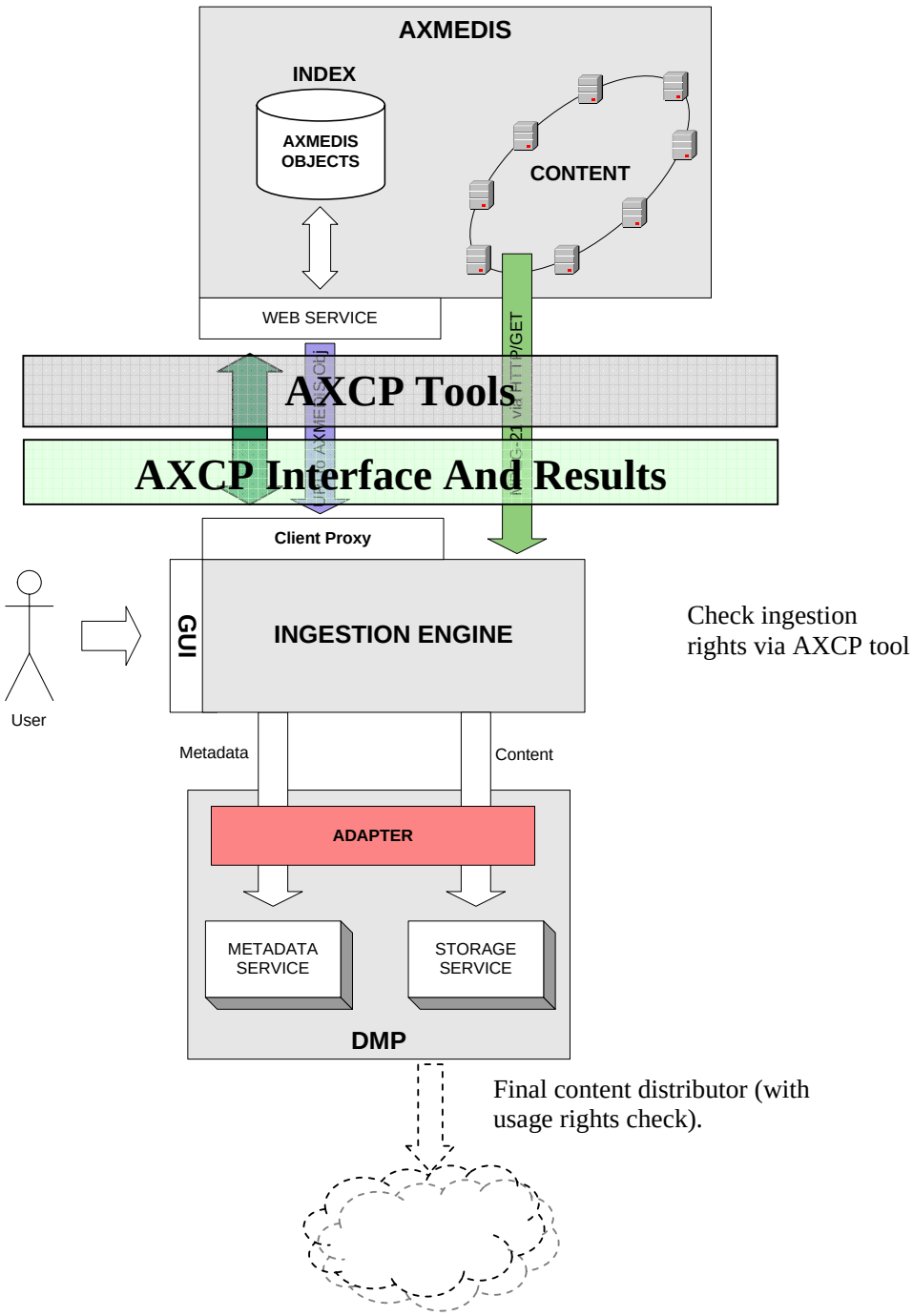
The prototype that was implemented consists in (all actions are made through a completely integrated GUI):

- Integrate AXMEDIS tools for query production, in order to find MPEG21 objects into the AXDB given some conditions on the metadata and, if possible, the PAR (Possible Available Rights).
- Once the query is composed, it can be launched against the AXDB in order to get the corresponding results.
- On the obtained results, the user can choose which content to download. The content could be protected or unprotected. User rights are checked using the dedicated AXMEDIS tools, in particular the AXCP tool.
- Both in case of protected and unprotected the DMP will access the public metadata, extract then and save them in its metadata db.
    - If the downloaded object is unprotected, the DMP will extract the actual resources and save them into its own filesystem in order to make it available for content distributors.
    - If the downloaded object is protected, the DMP will first check if the user (usually a DMP Administrator) has the rights to unpack the object. If the user has those rights, the DMP will unpack the MPEG21 file and save che resources in to its own filesystem. If the user has no rights to unpack the MPEG21, the DMP will save the entire MPEG21 file as is into its filesystem.

Being a tool to support content distributors, the DMP will only check the user unpacking rights, it will not check final user usage rights. Each content will be made available together with its ows usage rights, but the check of those rights is demanded to actual content distributors.

All this was accomplished by writing an ingestion engine (adapter) used to communicate with AXMEDIS AXCP tools.

The following picture explains what was implemented.



The scope of the integration demo is to integrate the **DMP** with **AXMEDIS DB**, importing AXMEDIS objects into the DMP stores.

Referring to the above picture that sketches the structure and the flow (from up downward), several steps are made to obtain this result:

- A DMP user (usually an Administrator) logs in to the DMP interface.
- The administrator builds a query against the AXDB by using the integrated query interface facility.
- He/she chooses which object to download among the query results.

- The MPEG21 object is downloaded into parking area, waiting for the public metadata to be extracted and the unpacking license to be checked.
- The system checks, by using the AXCP tool, the possibility for the user to unpack the MPEG21 object and extract the actual resources and their private metadata. All PARs are inserted in the object metadata, but are not actually considered by the DMP scope.
- At this step, the DMP actually ingests the resources and the related metadata. In event of the user has no rights to unpack, the DMP will ingest the MPEG21 object only, with public metadata.

This flow concludes the scope of the project regarding the DMP.

As depicted in dotted line in the picture above, the ingested content, together with its own usage rights, will be ready to be used by distributors which in turn will transmit it to their end-users. The usage rights enforcement will be then left to final distributors. This step is currently out of scope of Axmedis project.

7.5.2.1 AXCP Interface and Results

By integrating the AXCP tool within the DMP, users are able to crawl the AXMEDIS database, select the content they are going to import and download it. The system will ingest the downloaded content according to the user unpacking license. The DMP will integrate a web interface which will be able to interact with the AXCP tool and fire rules against it, in order to accomplish all the operations a single user can do.

Here below is a description of the use cases each user will go through. Each use case will correspond to a script and a set of rules according to the AXCP script language.

1. **Query the database**

   Each user is allowed to build a query to the AXDB by using a simple interface for building queries. The GUI will be completely integrated with the DMP GUI. The query will then be fired against the actual AXDB by means of the AXCP tool.
   The resulting object list will be presented to the user.

2. **Download the selected objects**

   Among the object list returned by previous query, the user can select which of the content he/she wants to download. The selected MPEG21 objects are downloaded in a specific repository, waiting for ingestion.

3. **Ingest a MPEG21 object**

   The user can now select any MPEG21 object from the downloaded MPEG21 objects repository. The system will check if the user has rights to unpack the object (i.e. access to the embedded resources and metadata). If the MPEG21 object can be unpacked, the DMP will extract the metadata to populate its metadata database, and will save each embedded resource in to its own filesystem. If the MPEG21 object cannot be unpacked (i.e. user has no rights for that), the DMP will only save the public metadata into its metadata database, and the MPEG21 object as it is into its own filesystem.

If necessary, all the process can be completely automated by creating a single rule thea wraps all the 3 use cases.

The Digital Media Platform runs on a Windows 2003 server and uses SQL Server 2000 as DataBase.

The integration is performed by developing a new adapter for the MPEG21 ingestion, using visual C# and ASP.NET as languages. The GUI will be completely integrated with the DMP GUI, while the ingestor will be a Windows NT service running in the background accepting the commands coming from the GUI interface.

7.5.2.2 Demonstrator fact sheet:

- Main purpose:
  - The main purpose of this demonstrator is to show the possibility to integrate the HP Digital Media Platform with the AXMEDIS framework and tools. The demonstrator will show how users query the AXMEDIS Database in order to find the content they need, download the content, separate the metadata from the actual content, if they have rights rights to do so, and store the two in the appropriate repositories.
- Review of the architecture integration with AXMEDIS:
  - Users are given the possibility to search the AXDB through the integration with the AXCP tool, using a GUI which is completely integrated with the DMP GUI. All queries can be made both in synchronous and asynchronous mode. Once the user has chosen the content she/he needs, this can be downloaded on the DMP and then integrated, after the system checked user rights.
  - Production of content, where and how:
    - This integration is only meant for content consumption, not for content production.
  - Content is meant to be first processed upon arrival. Once the MPEG-21 file is downloaded, the DMP ingestor will check if the user has rights to access the inner resources. If he/she has rights, the system will separate the metadata from the actual resources. Metadata are saved into the DMP metadata DataBase, the content is saved on the filesystem. Is the user has no rights to read the resources, the DMP will store only the public metadata into its database and save the MPEG21 file as it is into its filesystem.
    - At present there is no clue on how many content items can be processed per day
    - At present there is no clue on how many content items can be processed simultaneously
  - Integration with the AXMEDIS DRM is implemented through the usage of the AXCP tool.
  - Being a content consumer, the DMP is not meant to produce any kind of license.
  - Users and devices registration depends on the final content distributor.
  - Content distribution is made through streaming or broadcasting encoded content to actual distributors. These can distribute content to different target devices.
  - Being just a step of the content distribution chain, the DMP has no accounting collection or action monitoring facility. Accounting is demanded to the actual distributor.
- Description of the effective installation
  - Servers: Cluster for the DMP, hardware for the adapters, encoders, STORAGE.
  - Portals: The DMP is exposed through an administration portal, which, depending on the accessing user profile, shows different functionalities. The AXMEDIS search and ingestion functionality is completely integrated with this portal. Final users have no access to this portal.
  - Distribution infrastructure needed if any: Portals, encoders, streaming servers or any other communication device (such as SMS/MMS gateways).
  - Streaming/downloads: both.
  - Players needed: PC/ PDA/ STB/ any device that can receive content.
- AXMEDIS tools
  - List of major AXMEDIS tools: AXCP tool, AXDB.
  - AXMEDIS P2P usage, yes or no, where and how: MPEG21 files are mostly downloaded via HTTP calls. They could be also downloaded via P2P by using the AXCP tool.
  - AXCP usage, yes/no, where and how: Used to integrate query production and execution, file downloading and unpacking rights checking.
  - Workflow tools usage, yes/no, where and how: Not used.
  - Programme and publication usage, yes/no, where and how: Not used.

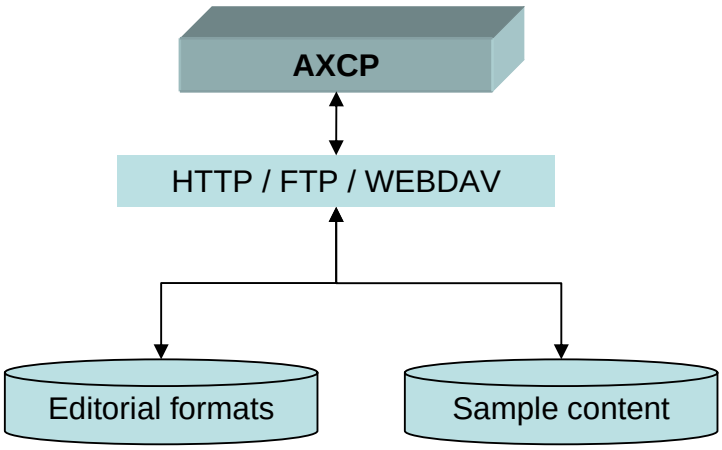
  - PMS/AXCS usage, yes or no, where and how: Not used
  - AXMEDIS database usage, yes or no, where and how: Yes, used for content queries and to retrieve actual object download address.
- Target Market:
  - The target market is the content distributors market. Each content distributor can retrieve AXMEDIS objects, ingest them with their metadata (and rights in the future), work on the content and then distribute (for streaming, download or other) it to the final users.
- Description of the business model
  - Each content distributor can implement its own business model, choosing when and how and to whom distribute the content. The distributor can also decide everything about the rights to impose to the content regarding the final user.
- Description of content:
  - How many content AXMEDIS objects will be distributed: At present there is no clue about how many content objects will be distributed.
  - Who is going to provide digital resources with the needed clearance of rights: digital resources and rights will be provided by the actual distributor.
  - Content description: The DMP can ingest and distribute any kind of content, but it mainly used with streamable content such as audio or video.
  - Kind of resources: The content topics can be of any kind, from films, to tv shows, music videos, sport events, etc. etc. either recorded or live streamed.
  - Typical Content size for each content type: Audio: circa 1MB/min. Video: circa 8MB/min depending on the bitrate encoding.
- Final Users/Clients:
  - How many final users will be reached: this depends on the final distributor.
  - Their description: Users are of various kinds. All people, ranging from kids to grown-ups, male and female, who have the possibility to access the content provided by the DMP (or better by the actual content distributor).
  - Their registration is needed: User registration is controlled by the Client or by the final distributor.
- Partners involved and roles:
  - No partner is involved regarding the development of this solution.
  - All content production partners are indirectly involved, since the ingestion could use all the content saved in the AXDB.

7.6 Content and format gathering from filesystem (XIM)

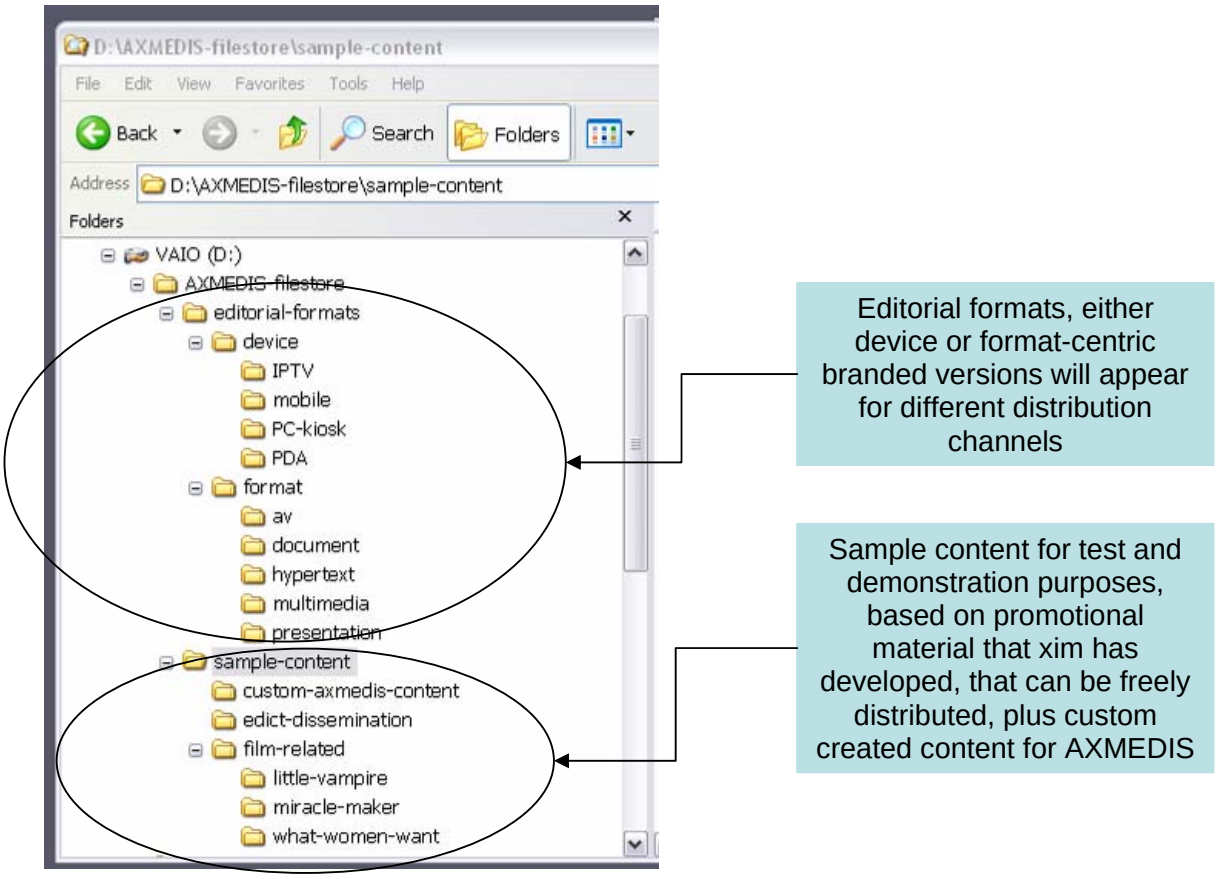
7.6.1 Description of the CMS and interfacing with AXCP

XIM's role within the AXMEDIS demonstrator is as a content integrator, with a particular focus on developing and customising editorial formats. The archive of resources that XIM is sharing within AXMEDIS therefore consists of a combination of editorial formats (in the form of SMIL and HTML+CSS documents) which will be constantly added to and adapted during the demonstration phase, plus a small library (less than 1000 objects) of sample content (as specified in DE8.1.1.1 and DE8.5.1.1) to used for testing and demonstration. As with ANSC, XIM is not using an actual CMS as part of the demonstrator due to the relatively small quantity of structured content. Instead XIM is storing its media and metadata in a file system which can be shared with consortium partners, and accessible by AXCP via HTTP and FTP.

The following diagram illustrates the XIM demonstrator content and format repository for content integration:



Content managed and structure
The XIM demonstrator filesystem is organised into following structure:



Access modes
The filesystem will be accessible via HTTP and FTP. WEBDAV is also possible if required. It will be equally important that XIM is able to access the content of other partners' databases and to save modified

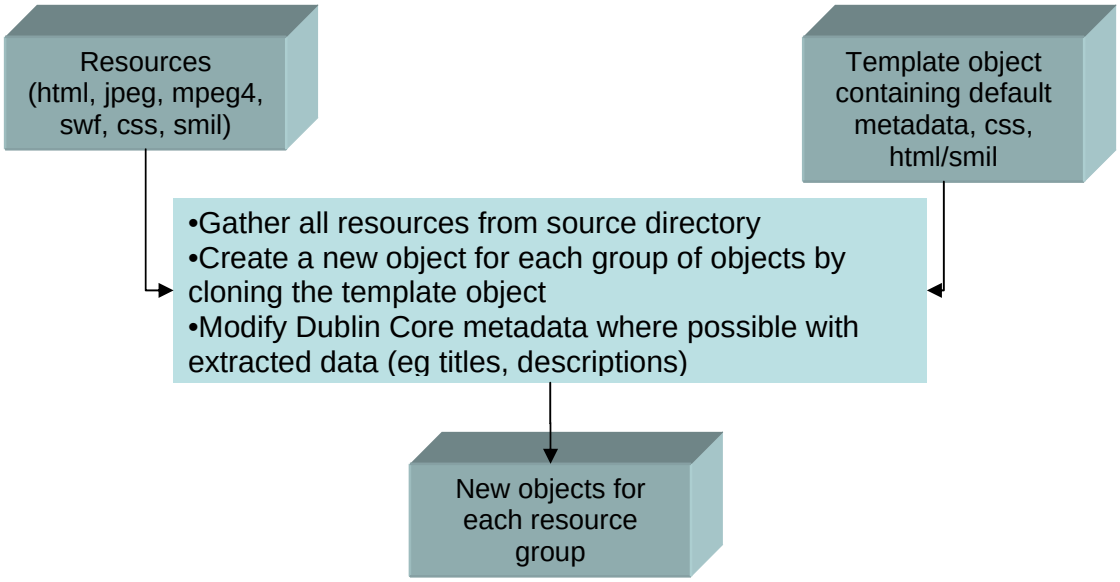
objects to relevant database destinations for distribution. By storing editorial formats as AX objects, these will be able to be easily shared via P2P directly from XIM's filesystem.

Metadata
For most of the sample objects, XIM is manually creating metadata where none previously existed and entering this using the AX Editor. Where possible, certain Dublin Core elements can be extracted from some types of content (eg titles, desciptions, language, author, format from tags within html pages).

7.6.2 AXCP Scripts and their description

All resources from XIM will be shared via HTTP/FTP, both as AXMEDIS objects and in their original format, so no real-time ingestion will be required. XIM is using some simple scripts to automate the local ingestion of resources into AXMEDIS objects and the association of relevant metadata plus gallery/presentation templates to present groups of data.

The Javascript used is very similar to that used by ANSC and is based on the samples developed by DSI. The process is illustrated below:



7.7 (www.lacentraldigital-com) - SDAE

7.7.1 Description of the CMS

Predefined content packages are made available to clients from content catalogued at www.lacentraldigital.com: First the client must obtain a client from La Central Digita, with the client code the client is able to acces their content packages (video and audio) and can search their packeges (audio or video) obtain caontent lists and view Content metadata that include image files, url of low quality content trial fragments and the url of content.

The two content types are audio and video represented by the internal La Central Digital structure as follows:

Video:

Element	Field	Type	Constraints	Req	Description
deliver	action	string	6 chars	x	Insert, Update or Delete
deliver	parent_id	string	5 chars	x	internal id of the comerce. Assigned by LaCentralDigital
deliver	video	video		x	info of the video
video	code	integer		x	Video Id
video	color	boolean		x	1 if is color / 0 if is B&W
video	sound	boolean		x	1 if has sound / 0 if hasn't
video	classic	boolean		x	1 if is a big classic video / 0 if not
video	original_title	string	255 char	x	original title of the movie
video	title	string	255 char	x	title in the languaje of the movie
video	country	string	2 char	x	country code ISO 3166-1
video	language	string	3 char	x	language code ISO 639. Empty for the silent movies
video	duration	string	6 char	x	duration of the film, in hhmmss
video	year	string	4 char	x	year of the video
video	genre	integer		x	genre of the video
video	size	string	255 char	x	size in bytes of the movie
video	comercial_des	string	2000 char	x	comercial description of the video
video	synopsis	string	2000 char	x	Synopsis of the movie
video	short_synopsi	string	255 char	x	Short synopsis
video	keywords	string	2000 char	x	keywords for the searchs
video	parental_advi	string	3 char	x	
video	artists	list		x	
artist	role	integer		x	code of role of the artist. At least one have to be Actor
artist	name	string	255		name of the artist
artist	last_name	string	255		last name of the artist
artist	surname	string	255	x	surname of the artist

	FIELDS
	ATTRIBUTES

Audio

Element	Field	Type	Constraints	Req	Description
deliver	action	string	6 chars	x	Insert, Update or Delete
deliver	parent_id	string	5 chars	x	internal id of the comerce. Assigned by LaCentralDigital
deliver	album	album		x	info of the album
album	code	integer		x	Album Id
album	title	string	255 char	x	title of the album
album	country	string	2 char	x	country code ISO 3166-1
album	upc	string	10-13 char	x	UPC-12 or EAN-13
album	main_artist	artist		x	main artist of the album.
album	year	string	4 char	x	year of the album
album	publisher	string	255 char	x	publisher for the album
album	copyright	string	255 char	x	(P) Line/copyright information from album
album	label	string	255 char	x	label for the album
album	tracks	list	1 - n	x	list of tracks
artist	role	integer			code of role of the artist (only for tracks). At least one have to be Main Artist
artist	name	string	255		name of the artist
artist	last_name	string	255		last name of the artist
artist	surname	string	255	x	surname of the artist
track	number	integer		x	number of the track in the album
track	title	string	255 char	x	title of the track
track	movement	string	255 char		only in classical music
track	duration	integer		x	duration in secs
track	artists	list	1-n	x	artists in the tracks
track	genre	integer		x	code of genre

	FIELDS
	ATTRIBUTES

Access mode

After content has been chosen (see content catalogue) and the client XML format received, the content is packaged and made available either through a Web Service (SOAP) or exported to the client system.,

Content managed
```
Audio: Flac    (also, WAV, MP3, etc…..)
Vídeo: MPEG-2, WMV 1.5 MB, WMV 256 MB (shortly MPEG-4)
```

Relationships between metadata and digital resource:
Internal proprietary metadata used in conjunction with either the clients XML or La Central's preferred XML

8 Bibliography

- DE9.2.1 specification of automating content production and formatting into CMSs of integrators
- AXMEDIS demonstrators: axmedis-major-tools-full-version-author-grid-v2-7-2.exe, AXMEDIS tools full version, March 2007, authoring tool, GRID tools, AXCP Content processing tools and plug ins, documentation, example of MPEg-21/AXMEDIS obejcts with SMIL, MPEG-4, HTML, any kind of resources, DRM licenses, etc.
    - http://www.axmedis.org/documenti/view_documenti.php?doc_id=2885
- AXMEDIS tools user manual
    - http://www.axmedis.org/documenti/view_documenti.php?doc_id=2857
- DE3-1-2-2-6 Specification of AXMEDIS Content Processing
    - http://www.axmedis.org/documenti/view_documenti.php?doc_id=1958
- AXMEDIS Javascript Reference Manual
    - http://www.axmedis.org/documenti/view_documenti.php?doc_id=2855
- http://curl.haxx.se/libcurl/
- http://wsdlpull.sourceforge.net/
- http://www.wxwidgets.org/