Automating Production of Cross Media Content for Multi-channel Distribution

www.AXMEDIS.org

DE9.2.3
First Prototype of automating content production and formatting into CMSs of integrators

Version: 1.1
Date: 14/09/2006
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: DE9.2.3 Contractual Date of Delivery: M24 Work-Package contributing to the Deliverable: WP 9.2 Nature of the Deliverable: Prototype and this short report Author(s): DSI

Abstract:
This deliverable is related to the mock up of WP9.2 that has been improved in this period by producing a first real prototype.
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 AND CONTENT PRODUCTION 6
5 WEBSERVICE APPROACH FOR CONTENT MANAGEMENT 7
5.1 CONTENT PUBLICATION 8
5.2 CONTENT RETRIEVAL 10
6 ODBC APPROACH FOR CONTENT MANAGEMENT 10
6.1 CONTENT PUBLICATION 10
6.2 CONTENT RETRIEVAL 11
7 A REAL CASE – THE ILABS LEARN EXACT CMS 12
8 BIBLIOGRAPHY 18

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:

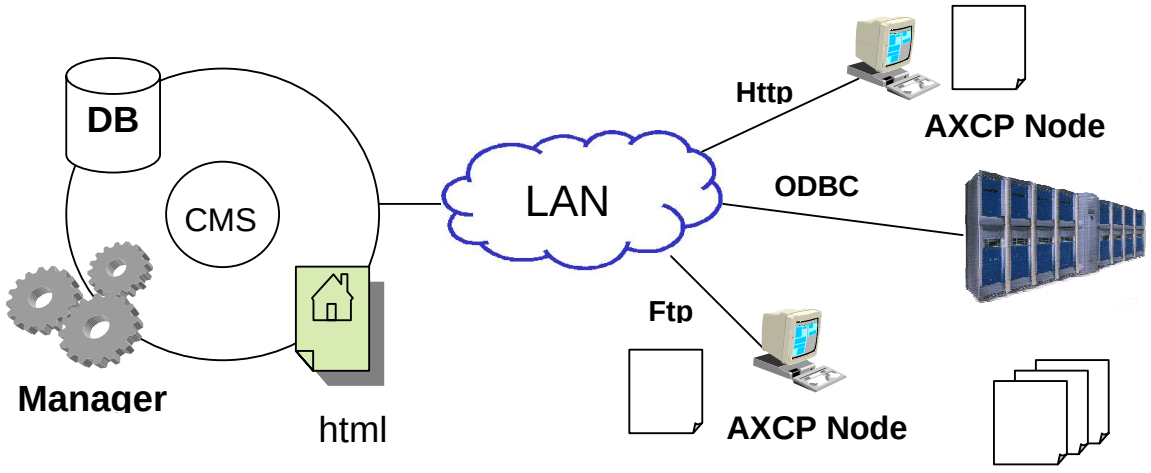
	Content Query		Content Access			Content Upload				
						upload control		file transfer		
CMS	**WS**	**DB**	**http**	**ftp**	**net fs**	**WS**	**http**	**ftp**	**http**	**net fs**
ILABS	X		X			X		X		
TISCALI	(X)	X	X		X	X				X
HP		X				X	X	X	X	
ANSC	X		X			X		?	?	
AFI		X	X	X			X	X		

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

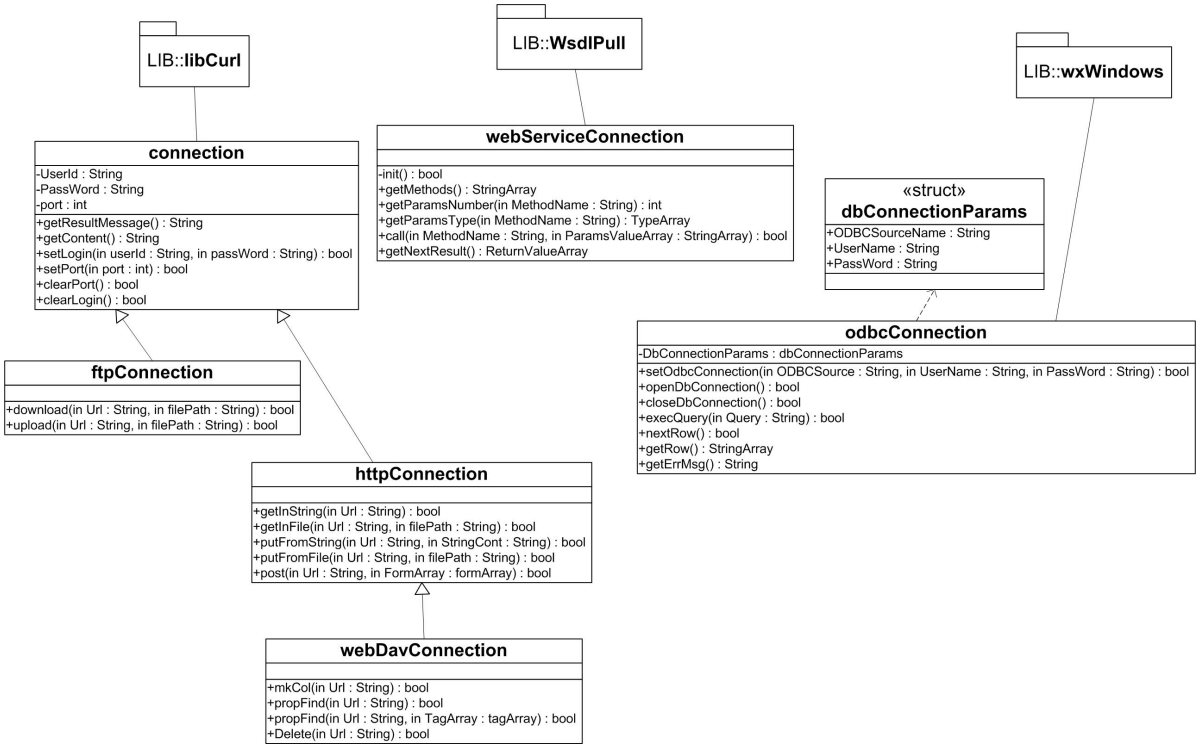
3 Prototype Description

The added technologies in the prototype are:

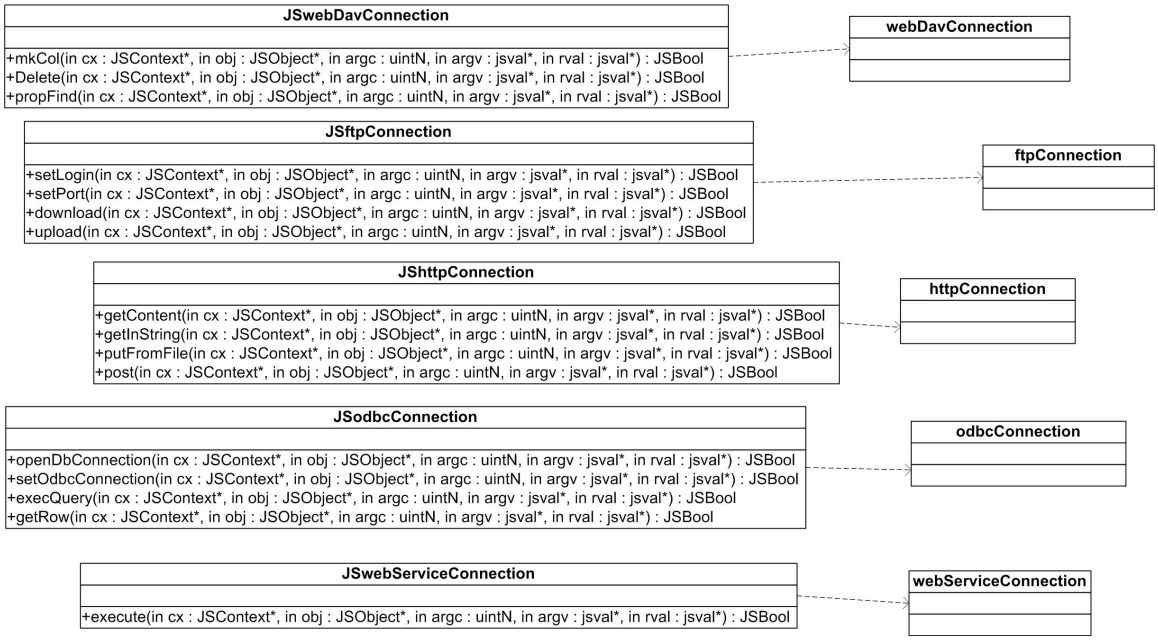
- Possibility of using into the AXMEDIS script language FTP commands to read and write files;
- Possibility of using into the AXMEDIS script language WSDL actions to access to Web Services in general. It can be used for accessing to database or for other means;
- Possibility of using into the AXMEDIS 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.



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 and content production

The AXCP area allows processing both raw digital resource and AXMEDIS object. 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<ciclo; 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.

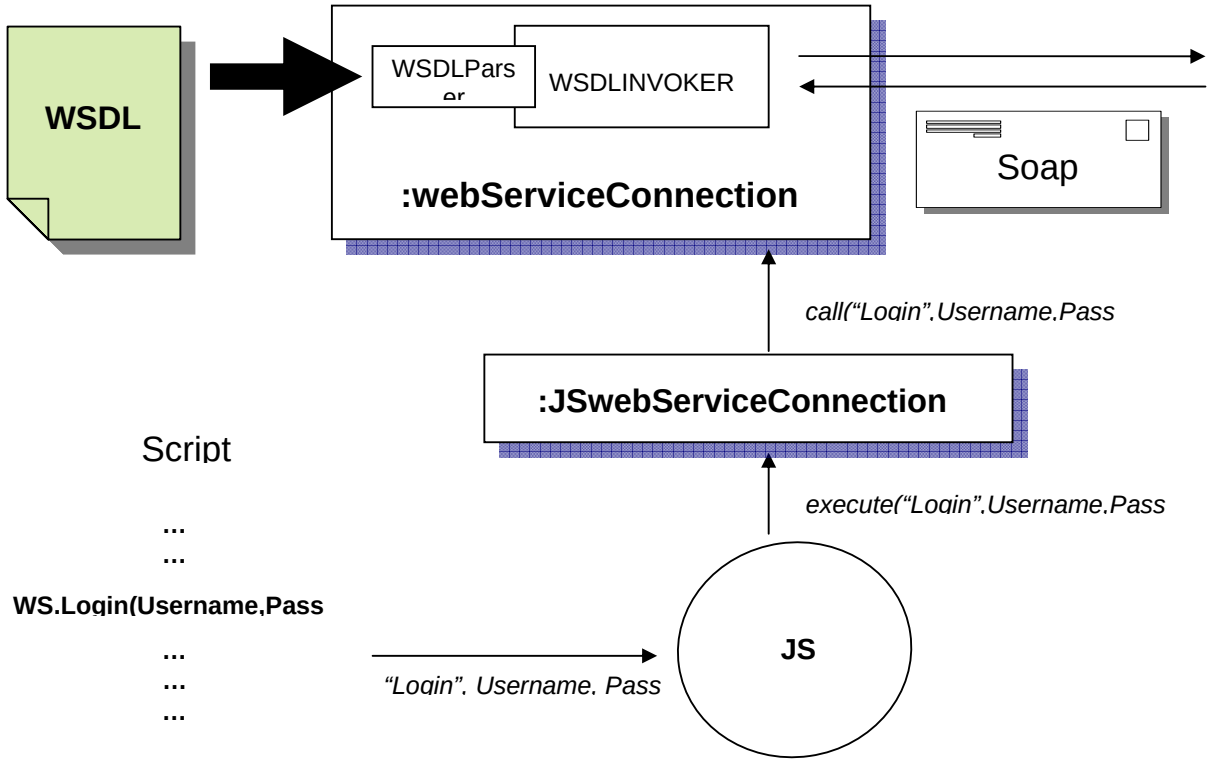
```
function getTitle(axObj)  {
      var dc = axObj.getDC();
      return dc.getElement("dc.title");
}

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.

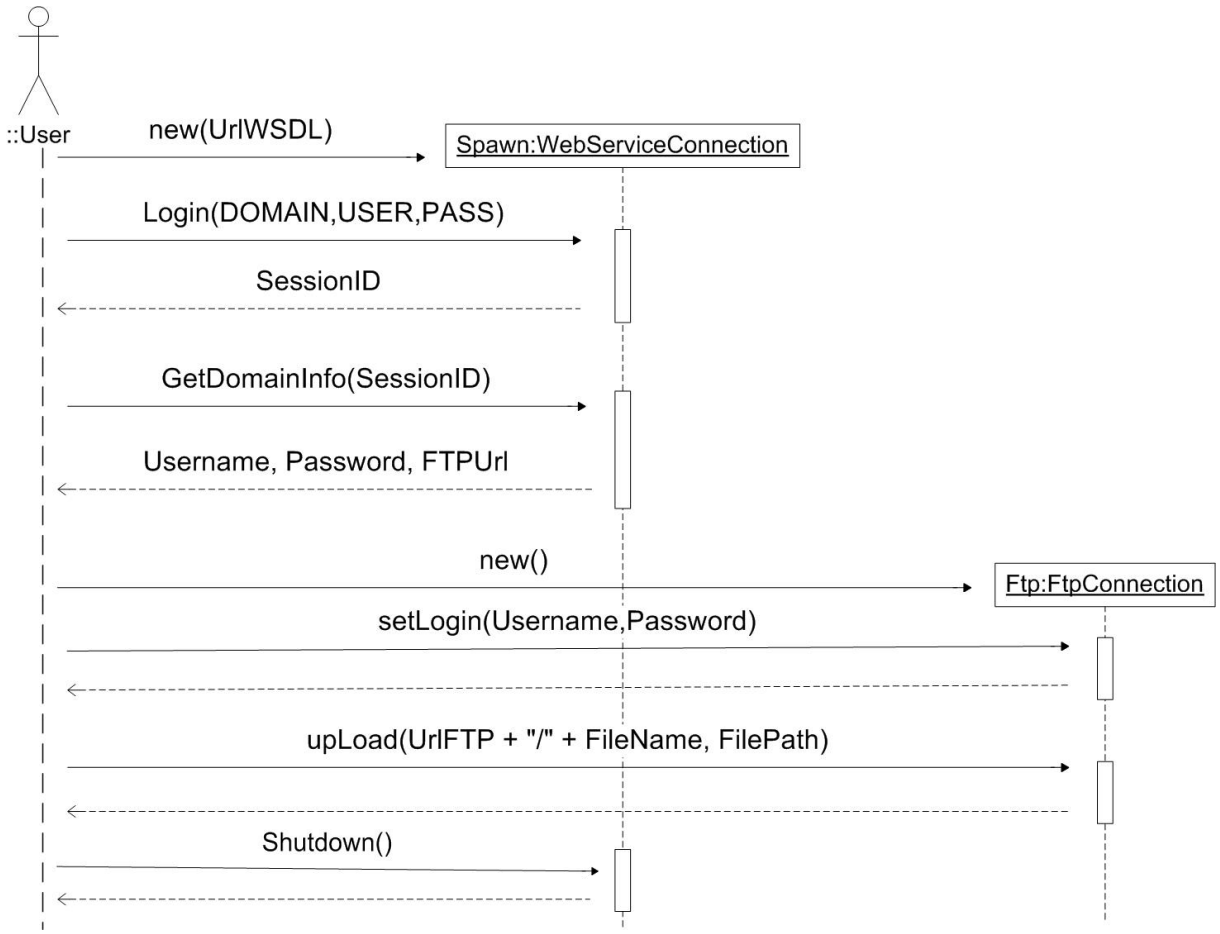


In the following some scripts have been reported. When possible, the sequence diagram is associated with the corresponding implementation in Javascript.

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).

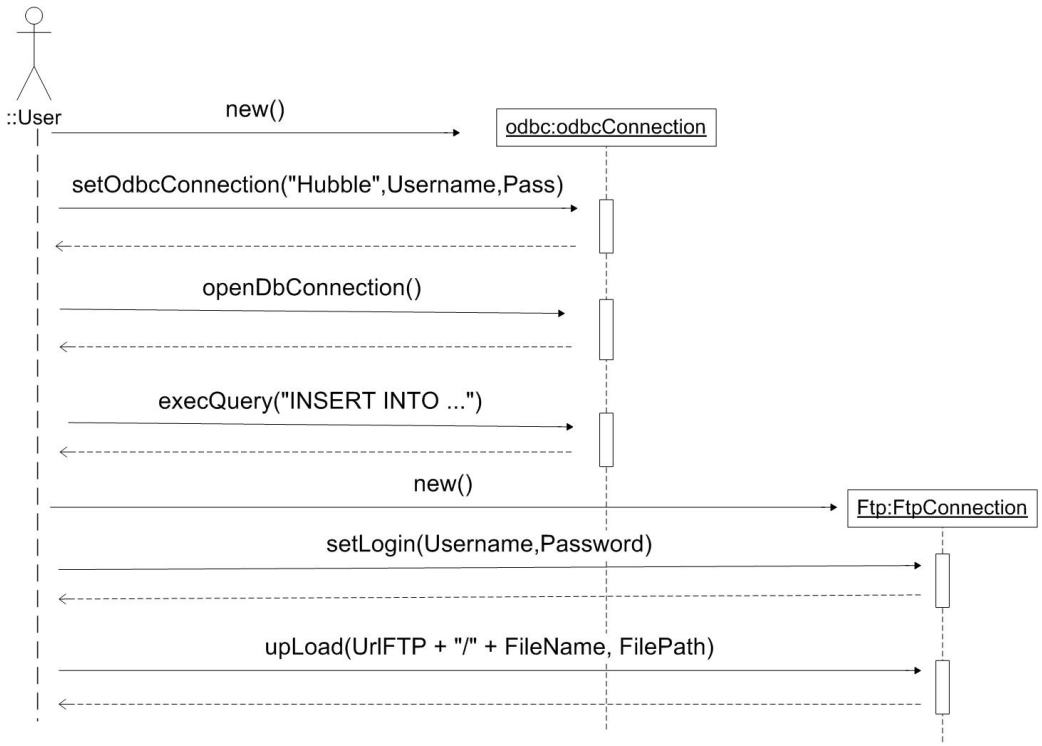
```
var Spawn = new WebServiceConnection(SeahorseWsdlUrl);

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 A real case – 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. 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)

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:

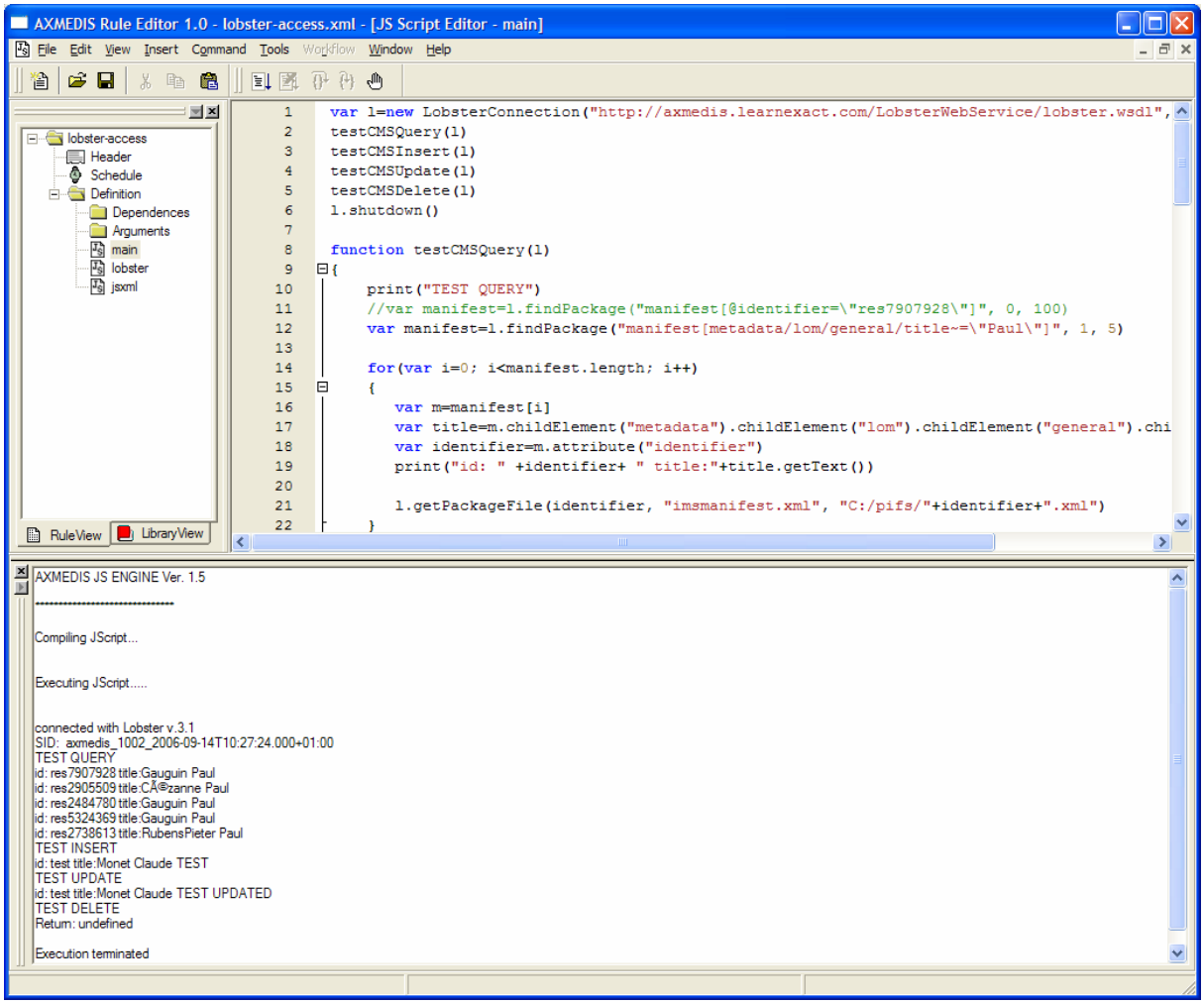
- the query support searching for packages with “Paul” in the title and for each result obtained the imsmanifest.xml is downloaded from the server
- the insertion of a new package into the CMS (from c:\test.zip pif file)
- the update of the same package (from c:\test-update.zip)
- the removal of the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

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

testCMSQuery(l)
testCMSInsert(l)
testCMSUpdate(l)
testCMSDelete(l)

l.shutdown()

function testCMSQuery(l)
{
    print("TEST QUERY")
    var manifest=l.findPackage("manifest[metadata/lom/general/title~=\"Paul\"]",
1, 5)

    for(var i=0; i<manifest.length; i++)
    {
       var m=manifest[i]
       var title= m.childElement("metadata"). childElement("lom").
                       childElement("general"). childElement("title")
       var identifier=m.attribute("identifier")
       print("id: " +identifier+ " title:"+title.getText())

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

function testCMSInsert(l)
{
    print("TEST INSERT")
    //check if the package is already present
    var manifest=l.findPackage("manifest[@identifier=\"test\"]", 0, 10)
    if(manifest.length>0)
    {
        print("Package \"test\" already present, cannot insert it")
        return
    }

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

```
    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.childElement("metadata").childElement("lom").
                  childElement("general"). childElement("title")
       var identifier=m.attribute("identifier")
       print("id: " +identifier+ " title:"+title.getText())
       if(title.getText()!="Monet Claude TEST")
          print("INSERT FAILED")
    }
}

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

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

    var manifest=l.findPackage("manifest[@identifier=\"test\"]", 0, 10)
    for(var i=0; i<manifest.length; i++)
    {
       var m=manifest[i]
       var title=m.childElement("metadata").childElement("lom").
                  childElement("general").childElement("title")
       var identifier=m.attribute("identifier")
       print("id: " +identifier+ " title:"+title.getText())
       if(title.getText()!="Monet Claude 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:

```
// ILABS LOBSTER Connection service
// class LobsterConnection allows to interact with the LOBSTER WS
// constructor LobsterConnection(wsdl,domain,user,passw)
//    allows to connect with the lobster WS, il performs the
//    login and retrieves domain information
//    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)
{
    this.wsdl=wsdl
    this.domain=domain
    this.user=user
    this.passw=passw
    this.insertPackage=LOBSTER_insert
    this.updatePackage=LOBSTER_update
    this.findPackage=LOBSTER_find
    this.getPackageFile=LOBSTER_getFile
    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)

    //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 doc=new REXML(domainInfo)
```

```
    var docinfo=doc.rootElement
    this.domainurl=docinfo.childElement("domainurl").getText()
    this.coursedomainpath=docinfo.childElement("coursedomainpath").getText()
    this.ftpuser=docinfo.childElement("ftpuser").getText()
    this.ftppassw=docinfo.childElement("ftppassword").getText()
    this.ftpport=docinfo.childElement("ftpport").getText()
}
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]

    //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+"/"+filename,pifFilePath)

    //send the notification of upload
    this.ws.NotifyUploadDone(this.sid)

    return "OK"
}
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]

    //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+"/"+filename,pifFilePath)
    //send the notification of upload
    this.ws.NotifyUploadDone(this.sid)
```

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

    //make a XPATH query
    var res=this.ws.FindPackage(this.sid,XPATHQuery, start, size)[0]
    //print(res)

    //parse the xml result removing the tamino elements
    // fill an array with the results
    var resDom=new REXML(res)
    var result=resDom.rootElement.childElement("xq:result")
    var r=new Array()
    if(result!=null)
    {
        for(var i=0; i<result.childElements.length; i++)
            r[i]=result.childElements[i].childElements[0]
    }
    return r
}
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

    //download the file from the server
    var http=new HttpConnection()
    http.getToFile(url,localFilePath);
}
function LOBSTER_delete(packid)
{
    if(this.sid=="") return "FAIL"

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

8 Bibliography

- DE 9.2.1 specification of automating content production and formatting into CMSs of integrators
- DE3-1-2-2-6 Specification of AXMEDIS Content Processing
- http://curl.haxx.se/libcurl/
- http://wsdlpull.sourceforge.net/
- http://www.wxwidgets.org/