A LANGUAGE AND ARCHITECTURE FOR AUTOMATING MULTIMEDIA CONTENT PRODUCTION ON GRID

P. Bellini, I. Bruno, P. Nesi
DISIT LAB, Department of Systems and Informatics, University of Florence
Via S. Marta, 3 -- 50139 -- Florence, Italy
Tel:+390554796567, +390554796523, fax: +390554796363
pbellini@dsi.unifi.it, ivanb@dsi.unifi.it, nesi@dsi.unifi.it

ABSTRACT

Possible solutions to the management of multichannel delivering, production on demand, and containment of sale prices in the digital multimedia content production could be the automating, accelerating and restructuring the production process. The proposed solution provides innovative methods and tools to speed up and optimize content production and distribution based on the GRID technology supported by a specific programming language that allow defining the automatic procedure for content processing, production, adaptation, protection, DRM managing, distribution, etc. This paper describes the GRID architecture of the AXMEDIS Content Processing Area and the language adopted to define algorithms executed into the GRID environment.

Keywords: multimedia production, Grid computing, content processing

1. INTRODUCTION

GRID computing is a paradigm of distributed computing that involves coordinating and sharing computing, application, data, storage, and network resources across dynamic and geographically dispersed organizations ([1], [2]). GRID computing is an evolving area of computing, where standards and technology are still being developed to enable this new paradigm. GRID is already being successfully used in many scientific applications where huge amounts of data have to be processed and/or stored. One of the possible new application fields for the GRID Computing is the production and management of multimedia content. It is envisaged that a large number of multimedia services (music, video, radio, television, etc.) will become suitable very soon for real-time (on-demand) public access. Crucial technical issues of providing access to such services are user-friendliness, universal access to services, as well as an efficient service GRID middleware that enables the dynamic service discovery and composition, distributed resource management and adaptive media delivery ([3]). To cope with these goals, a specific solution has been designed in AXMEDIS project as a distributed environment based on the GRID computation.

The AXMEDIS (Automating Production of Cross Media Content for Multi-channel Distribution) IST European Commission R&D Project involves leading European digital content producers, integrators, distributors and researchers and wants to create the AXMEDIS framework to provide innovative methods and tools to speed up and optimize content production and distribution, for production-on-demand, for leisure, entertainment and digital content valorization and exploitation in general ([4], [5]).

The AXMEDIS Content Processing Area is the subsystem of AXMEDIS framework based on GRID computing and aims at realizing an efficient, scalable and flexible solutions for massive content retrieval, adaptation and transcoding, production and distribution on demand, packaging and formatting, protection, licenses production, recognition and tracking of content for broadcast audio visual monitoring. The GRID solution in multimedia content processing and delivering allows meeting the challenges of market demand and providing benefits: (i) reducing costs for content production, indexing, retrieval and management by applying techniques for content composition, representation (format), metadata generation and manipulation, and workflow; (ii) reducing distribution and aggregation costs; (iii) integrating methods and tools for Digital Rights Management (DRM), including the exploitation of emerging standard as MPEG-21.

This paper is partially focused on the architecture of the AXMEDIS Content Processing Area and mainly describes the language adopted to define activities and algorithms for digital content manipulation that are executed into the GRID environment. The paper is organized as follows. In Section 2, the AXMEDIS Content Processing Area inside the AXMEDIS Framework is described. Section 3 describes the concept of Processing Rule that governs all the content processing activities. Section 4 reports the language used in the distributed AXMEDIS Content Processing Rule Engine and describes an example of its use. Finally, conclusions and future work are reported in Section 5.

2. AXMEDIS CONTENT PROCESSING AREA

The architecture of the AXMEDIS Content Processing Area, also called in short AXCP Area (see Figure 1) is based on a GRID infrastructure constituted of an AXCP Rule Scheduler and several AXCP Rule Executors for executing AXCP Rules. AXCP Rules are formalized in AXMEDIS Content Processing Rule Language. The Rules are procedure to be executed according to some constraints which are the conditions for the execution of the AXCP Rule. They are used to script/program and plan the activities to be performed for producing, processing and protecting digital contents in automatic manner and according to possible DRM associated with the content usage.

The AXCP Rule Scheduler performs the rule firing/activation, discovering of Rule Executors and management and dispatching of Rules to be executed. The scheduler may receive commands (to invoke a specific rule with some parameters) and provide reporting information (e.g. notifications, exceptions, logs, etc…) to external workflow and tools by means of a WEB service. Rule Executor receives the Rules to be executed from the Scheduler, and performs the initialization and the launch of the script program execution of the Rule. During the run, the Executor could send notifications, errors and output messages to the Scheduler. Moreover, the Executor could invoke the execution of other Rules sending a specific request to the Scheduler so as to divide a complex Rule/procedure into sub rules/procedure running in parallel and rationally use the computational resources accessible in the content factory, on the GRID. This solution maintains advantages of a unified solution and allows enhancing the capabilities and the scalability of the AXMEDIS Content Processing Area ([6], [7]).

The processing tools in the AXCP Area are supported by the AXMEDIS Plugin technology that allows each AXCP Rule Executor to dynamically link any content processing tool and algorithm (e.g. audio, video and image adaptation, transcoding, encryption) and coping with possible customized algorithms and tools.

3. AXCP RULE GENERAL FORMAT

An XML formalization of AXCP Rules is reported in Figure 2 and it is comprised of three main sections. The *Header* contains general metadata such as: rule name, AXRID (Rule ID), rule version, rule type, software name, version of software, date of production, time of production, author, affiliation, URL, comment, last modification date and time. *Schedule* section contains temporal constraints describing the rule status ("*active*" or "*inactive*") and conditions for firing it such as: start date, start time, periodicity (monthly, daily, weekly, etc.), expiration date and expiration time. Such information is used by the AXCP Rule Scheduler in the GRID environment for planning the activity and associating active rules with available computational resources. *Definition* is the section that contains the rule

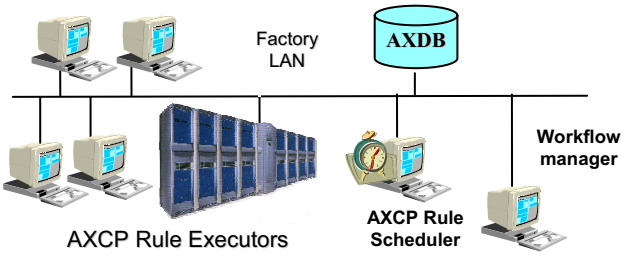

Figure 1 – The AXMEDIS Content Processing Area

signature in terms of list of arguments (parameters and selections), list of dependences (required AXMEDIS plug-ins in terms of plugin name and version) and the rule body (the script code to run). Dependences define constrains on plugins that a Rule Executor has to have installed to run the script (e.g., plugins for fingerprint estimation, audio/video transcoding, descriptors extraction).

Formally, the rule signature of the AXCP Rule is:

$$R = f(S_1, S_2, .., S_n, P_1, ..., P_m)$$

Where:

- S_i is a *Selection* (sequence of queries), to be sent to the AXMEDIS Database to retrieve digital object (content) IDs or a set of object IDs to AXMEDIS objects or a mix of them; such parameter is exploded in terms of list of objects IDs during the execution of $f(.)$.
- P_i is a parameter (basic type as integer, common string, XML string, Boolean, etc.), it could represent, for example, the scale factor or the MIME type of the output format, the number of objects collections to be created, name of the author, etc.
- f is the identifier of rule (e.g., the ID of rule);
- R is the consumptive result of the rule application. It could be a status, a new AXMEDIS object, or a metadata manipulation result, the license of an AXMEDIS object, a message to be returned to the AXMEDIS Content Processing Area, etc.

Other results are resulting objects processed and/or produced during the execution of the Rule and they can be directly posted into the file system or database.

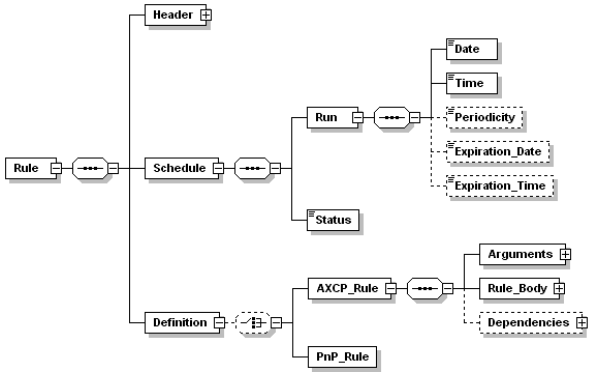

Figure 2 – XML schema of the AXCP rule

4. THE AXMEDIS SCRIPT LANGUAGE

The script language used in the body of the AXCP Rule is based on the Javascript. The Rule Executor uses the SpiderMonkey Javascript Engine v.1.5 released by Mozilla ([8], [9], [10]) to execute scripts. The JavaScript built-in objects were extended by wrapping partially or totally AXMEDIS data types (AXMEDIS JS Classes) and adding functions (JS Functions) deriving from the AXMEDIS Framework. Both could provide direct services or serve as interfaces to external services. For example, an AXMEDIS JS Class for direct services might be one that handles the network access, while a JS Class interfacing an external service might be an intermediary broker of database services. In this terms, the script language was enriched to provide several data types, operators and accessible algorithms to manipulate any digital resources in a large number of formats for: images, audio files, video, documents, multimedia (including MPEG-4, HTML, LOM, etc.), plus MPEG-21 aspects, MPEG-21 REL, MPEG-21 IPMP, etc. Therefore, the script language allows writing simple procedures and combining them to: (i) import and retrieve from other sources (e.g., existing CMSs) content and metadata using different channels/interfaces (e.g. ODBC, Http, ftp, web services); (ii) bind the structure of content and content collection to presentation and formatting styles by means of the SMIL languages and features; (iii) find/produce alternatives/adaptations for components that present potential distribution problems (too big files); (iv) format/adapt content and metadata for publication on different distribution channels; (v) protect digital content according to MPEG-21 models and related AXMEDIS tools, including license production and adaptation; (vi) process and produce metadata and managing the estimation of fingerprint and descriptors; (vii) interact with commercial tools and custom dynamic libraries for exploiting their functionalities ([6]). For the sake of completeness, a brief description of the most relevant AXMEDIS JS Classes is reported ([5], [6], [7]):

- The *AxmedisObject* class wraps the AXMEDIS Object Model. It is responsible of AXMEDIS object management in terms of: creation, embedding digital resources and metadata, storing/retrieving into/from database, etc.
- The *AxInfo* class maps and allows managing the B2B set of metadata for AXMEDIS objects.
- The *AxDublinCore* class maps the metadata related to the Dublin Core in the script.
- The *AxResource* class allows managing different raw digital content in the script.
- The *AxCPPlugin* class is a meta JS class that allows dynamic access to AXMEDIS Plugins providing several functions and algorithms for: Fingerprinting, Digital Resource Adaptation, Metadata Adaptation, Transcoding, Watermark, Descriptors extraction, etc…
- The *AxSelection* class allows using Selection objects to manage the access and making queries to the AXMEDIS database.
- The *AxSearchBox* class interfaces with the Searchbox Crawling application which is used to ingest content from the a large set of factory CMSs. It allows querying and retrieving content and metadata for creating AXMEDIS objects.
- The *License* class allows creating and managing licenses associated with AXMEDIS objects. It wraps the MPEG-21 REL license model and provides access to Principal, Grants and Resource items.
- The *IPMPInfo* class wraps the IPMP information necessary for protecting AXMEDIS object. It is used to indicate the kind of protection algorithm and the related parameters to be used.
- The *SMIL* classes allows selecting on the basis of the user profile and device capabilities, the best SMIL template and the style to be applied on the content of a specific AXMEDIS object.
- The *NetConnection* classes model different types of connection/protocol (e.g. http and ftp protocols, ODBC, WebService, WebDav) and provide primitive methods for accessing and retrieving metadata and digital resources.

Finally, JS Functions includes a set of auxiliary functions for different purposes: Statistical, Combinatorial, Set Management, Generic (e.g., file system functions), External Calls (e.g., invoking an AXCP Rule at runtime) ([6], [7]).

4.1. Using the AXMEDIS Script Language

In Figure 3, an example of AXMEDIS script for composition and adaptation of AXMEDIS objects is reported. The script shows the use of some classes mentioned above and 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();
}
```

Figure 3 - Example of Script code for the production and adaptation of AXMEDIS objects

5. CONCLUSIONS

In this paper, the aims and the script language of the AXMEDIS Content Processing Area have been described. This Area is a flexible and scalable core subsystem of the AXMEDIS Framework and architecture. Such subsystem is involved in the automatic content production, protection, formatting, metadata adaptation, etc., in AXMEDIS. The adopted solution is based on GRID Computing and on a script language. The AXCP Rule Language extends the Javascript by adding a set of object types wrapping the AXMEDIS data types and MPEG-21 data types provided by the AXMEDIS framework. In addition, a specific javascript object was defined in order to link dynamically different AXMEDIS plugins that allow extending capabilities and functionalities without changing the system. The full documentation can be recovered on AXMEDIS portal http://www.axmedis.org. AXMEDIS is an open platform in the sense that you can join the AXMEDIS community.

6. ACKNOWLEDGMENTS

The authors would like to thanks to all AXMEDIS project partners including the Expert User Group and all affiliated members, for their contributions, funding and collaborations. A specific acknowledgment to EC IST FP6 for partial funding of AXMEDIS project. A warm thanks to all AXMEDIS people that have helped us in starting up the project and sorry if they have not been involved in the paper and have not been mentioned. We trust in their understanding.

7. REFERENCES

[1] D. W. Erwin and D. F. Snelling. “UNICORE: A Grid computing environment”. Lecture Notes in Computer Science, 2150, 2001.

[2] Foster. The anatomy of the Grid: Enabling scalable virtual organizations. Lecture Notes in Computer Science, 2150, 2001.

[3] J. Mathe, K. Kuntner, S. Pota, Z. Juhasz, “The use of Jini technology in distributed and Grid multimedia systems”, in Proc. MIPRO 2003, Hypermedia and Grid Systems, Opatija, Croatia, 19-23. May 2003., pp. 148-151.

[4] P. Bellini and P. Nesi, “An Architecture of Automating Production of Cross Media Content for Multi-channel Distribution”, in Proc. AXMEDIS 2005, Florence, Italy, 30 Nov. - 2 Dec., IEEE Press, pp 123-133

[5] AXMEDIS DE3.1.2A Framework and Tools Specifications http://www.axmedis.org/documenti/view_documenti.php?doc_id=1379

[6] AXMEDIS DE3.1.2C Framework and Tools Specifications http://www.axmedis.org/documenti/view_documenti.php?doc_id=1381

[7] P. Bellini, I. Bruno and P. Nesi, “A Distributed Environment for Automatic Multimedia Content Production based on GRID”, in Proc. AXMEDIS 2005, Florence, Italy, 30 Nov. - 2 Dec., pp 134-142, IEEE Press.

[8] “JavaScript C Engine Embedder's Guide”, http://www.mozilla.org/js/spidermonkey/apidoc/jsguide.html

[9] J. Thiele, “Embedding SpiderMonkey - best practice” http://egachine.berlios.de/embedding-sm-best-practice/embedding-sm-best-practice-index.html

[10] “Scripting C++ with JavaScript using SpiderMonkey”, http://home.tiscali.be/franky.braem17/spidermonkey.htm