Faculdade de Engenharia da Universidade do Porto

FEUP

Digital Item Processing

Rui Filipe Santos Rocha

Dissertação realizada no âmbito do
Mestrado Integrado em Engenharia Electrotécnica e de Computadores
Major Telecomunicações

Orientadora: Prof.ª Dr.ª Maria Teresa Andrade
Orientador na instituição (INESC): Eng. Pedro Carvalho

Junho de 2008

Abstract

MPEG-21 is a standard currently in its final phase of specification, which aims to establish a platform capable of distributing a large variety of multimedia content enabling access and consumption in a flexible and interoperable way, while protecting owner's rights.

The objectives proposed for this dissertation included the conduction of a thorough analysis of benefits and challenges of using the Digital Item Processing (DIP) framework, specified in the 10th part of the MPEG-21 standard for the manipulation of complex multimedia objects. The ultimate goal was to specify and achieve a distributed implementation of this framework, which would be able to satisfy the constraints of "thin" clients while providing a standardized form of interacting with the digital objects. These multimedia objects are referred to as "Digital Items" (DIs) in the MPEG world. The DIP framework specifies a set of standardized methods, designated of Digital Item Methods (DIM), that can be applied to the Digital Item when the DI is being presented to the user. To achieve these goals, the initial phase of this work has included a review of the state of the art, whereby a detailed analysis was made of applications that currently use MPEG-21 DIP. Moreover, an application offering a simple GUI was developed to allow testing the functonalities and operations provided by the reference standard implementation of the DIP framework. The results of this initial phase have indicated that, up to the date, there aren't any applications that use the DIP framework in a distributed way. They have also shown that the current specification of DIP would likely impose some difficulties towards the successful and complete implementation of a distributed solution. Accordingly, (and converging with the initial expectations of the proposers of this work), the second phase of the work has included a formal specification of a distributed solution, clearly identifying the set of methods that could be implemented on the server and those that would need to be implemented on the client. This formal specification was then followed by the implementation of a corresponding solution and its integration in an existing application to browse and present MPEG-21 Digital Items to end-users.

Resumo

O MPEG-21 é uma norma que se encontra na fase final de especificação, que pretende definir uma plataforma capaz de distribuir uma grande variedade de conteúdos multimédia de forma a poderem ser acedidos e consumidos de forma flexível e interoperável protegendo os direitos de autor.

Os objectivos propostos para esta dissertação incluem uma análise os benefícios e dos problemas da uso da *Digital Item Processing (DIP) Framework* especificada na parte 10 da norma MPEG-21, quanto a manipulação de objectos multimédia complexos. O objectivo final era especificar e conseguir implementar Digital esta *framework* de uma forma capaz de ultrapassar as limitações de aplicações clientes que operam em pequenos dispositivos, enquanto se fornece uma forma *standard* de interagir com objectos digitais. Estes objectos multimédia são identificados no mundo da MPEG-21 como *Digital Items* (DIs). A plataforma DIP especifica um conjunto de métodos *standard* designados por *Digital Item Methods* (DIMs), que podem ser aplicados a um item digital (DI) quando este está a ser apresentado ao utilizador. Para atingir estes objectivos, a fase inicial do trabalho incluiu uma revisão do estado da arte, que consistiu em analisar uma série de aplicações que actualmente usam o DIP da MPEG-21. Posteriormente foi desenvolvida uma aplicação que permitiu testar as funcionalidades e as operações disponibilizadas pelo software de referência da MPEG-21 para o DIP. O resultado deste estudo inicial indica que até a data não existem aplicações que usam o DIP de forma distribuída. Também indicaram que a actual especificação do DIP causaria muito provavelmente algumas dificuldades no caminho para uma completa e bem sucedida implementação de uma possível solução distribuída. Posteriormente, (e indo ao encontro das expectativas iniciais de quem propôs este trabalho) a segunda fase do trabalho incluiu uma especificação de uma possível solução para o funcionamento distribuído, identificando claramente o conjunto de métodos que podiam ser implementados no servidor e daqueles que teriam de ser implementados no cliente. Esta especificação foi seguida da implementação desta solução e respectiva integração numa aplicação já existente que permite aos utilizadores finais visualizar de forma distribuída *Digital Items*

Acknowledgements

I would like here to express my gratitude to the people whose help made possible to make this dissertation.

First I would like to say thank you to my supervisors, Professor Maria Teresa Andrade and Engineer Pedro Carvalho for their constructive opinions, suggestions and critics that made me achieve this results.

Also I'd like to say thank you to Giorgiana Ciobanu for her precious help in understanding the MPEG-21 DIP Reference Software and for helping me in the implementation and integration aspects.

Table of Content

List of Figures

List of Tables

List of Acronyms

CAT	Context Aggregation Tool
DI	Digital Item
DIA	Digital Item Adaptation
DIBO	Digital Item Based Operation
DID	Digital Item Declaration
DIDL	Digital Item Declaration Language
DII	Digital Item Identification
DIM	Digital Item Method
DIML	Digital Item Method Language
DIP	Digital Item Processing
DIXO	Digital Item eXtension Operations
DRM	Digital Rights Management
GUI	Graphical User Interface
HTML	Hyper Text Markup Language
IMS	Integrated Management Supervisor
IPMP	Intellectual Property Management and Protection Components
PC	Personal Computer
PDA	Personal Digital Assistant
QoS	Quality of Service

REL	Rights Expression Language
SI	Streaming Interface
SMIL	Synchronized Multimedia Integration Language
SOAP	Simple Object Access Protocol
UMA	Universal Multimedia Access
URL	Uniform Resource Locator
WAP	Wireless Application Protocol
WML	Wirelless Markup Language
XML	eXtensible Markup Language

Chapter 1

1 Introduction

Nowadays, most of the population in the world is a consumer of digital content being information, videos, audio or combination of these or even other kinds of contents. The number of individuals producing digital content not just for professional use but also for personal is increasing very fast. To give a response to this, the number of access devices capable of presenting digital contents is also growing day by day, allowing the access to these contents to be made anywhere and at anytime. This causes some difficulties because content owners want their rights safeguarded. Hence, the platforms developed to consume digital content must be able to manage and protect these rights.

In recent years there has been an evolution in the multimedia content, accompanying the growth of consumption in television services, mobile services, etc. There are numerous research activities in the area of multimedia applications and services to cover all the aspects related to multimedia such as the quality of service, the viewing of remote resources, etc.

This work was developed based on previous developments made by Giorgiana Ciobanu [1] during her master thesis, which are related to the European project ENTHRONE ("*End-to-End QoS through Integrated Management of Content, Networks and Terminals*"). The main innovative contribution of this work, is to combine the distributed approach adopted by Giorgiana Ciobanu with the use of the MPEG-21 Digital Item Processing (DIP) framework. This would allow combining the benefits of moving to the server side the most power-consuming operations required for the presentation of complex multimedia objects, thus suiting the constraints of the so-called thin devices, whilst providing an interoperable solution based on open standards.

DIP framework provides the standard operations that combined can be used by the User to interact with the DI, ensuring interoperability in the processing level [2]. DIP is constituted by several elements which are Digital Item Methods (DIMs) that allow to specify a set of operations, Digital Item Method Language (DIML) which is the language to express these methods making use of Digital Item Basic Operations (DIBOs) and Digital Item eXtension

Operations (DIXOs), this language is ECMAScript, which is a lightweight language "in terms of memory, footprint and processing power". The use of this language made unnecessary to create a new DIML [3].

The main objectives of this work are to study the MPEG-21 standard trying to understand if it is possible to have a distributed implementation of the MPEG-21 DIP, thus solving interoperability problems whilst suiting constraints of thin end-user devices, and then to conceive and implement a solution to demonstrate the concept. Such a solution would allow having standard "tools" to process different kind of resources and present them in different client applications built for different types of terminals. It would work as a bridge between the client applications and the MPEG-21 DIP Reference Software, which is the application that implements all the operations of the MPEG-21 DIP standard that can be used to process a resource (e.g. *play* operation to play a video resource).

This report is structured in six chapters. It is presented the functional specification analysis conducted during this thesis, whereby a clear identification of the requirements of the DDIBrowser (Distributed Digital Item Browser) to enable the remote execution of Digital Item Methods (DIMs) was made, as well as the corresponding functional architecture to support those requirements. Chapter 5 presents the developments made, describing the implementation of the different modules of the system and the results obtained. Chapter 6 presents the conclusion of the thesis.

Chapter 2

2 MPEG-21

The MPEG-21 Standard (multimedia framework) is being developed since June 2000, known as IEC/ISO 21000 and is based on 2 concepts: "*the definition of a fundamental unit of distribution and transaction (the Digital Item) and the concept of Users interacting with Digital Items*" [4]. The goal is to define the necessary technology so that *Users* may access, exchange, manipulate and consume *Digital Items* efficiently, transparently and in an interoperable way through a wide range of networks devices, user preferences and communities [5]. Many applications based on this standard have been or are being developed contributing to MPEG-21 standardization improving the way "*tools*" (*or elements existing for creating, delivering and consumption*) relate to each other creating equal opportunities for content creators, producers, content and service providers in the multimedia market.

The MPEG-21 Standard is built upon the concept of a **Digital Item (DI)** which is a "structured digital object with a standard representation, identification and metadata within the MPEG-21 framework" [4]. A Digital Item is composed of a set of resources (videos files, audio files, etc), metadata (information about the Digital Item or a resource within it) and structural information describing the relationships between resources and metadata. "Any entity that interacts in the MPEG-21 environment or makes use of Digital Items" [6] is called a **User** and it is the other fundamental concept that the MPEG-21 standard defines. A User may be a creator, a resource provider or even a consumer, but to MPEG-21 no distinction is made, they're all Users. The MPEG-21 standard is divided into 18 parts that deal with several aspects such as declaration, identification, digital rights management, adaptation and processing [5] among others. The 18 parts of the MPEG-21 Standard are (list retrieved from [1] and [6]):

- Part 1: Vision, Technology and Strategy
- Part 2: Digital Item Declaration
- Part 3: Digital Item Identification
- Part 4: Intellectual Property Management and Protection Components
- Part 5: Rights Expression Language

- Part 6: Rights Data Dictionary
- Part 7: Digital Item Adaptation and Session Mobility
- Part 8: Reference Software
- Part 9: File Format
- Part 10: Digital Item Processing
- Part 11: Evaluation Tools for Persistent Association
- Part 12: Test Bed for MPEG-21 Resource Delivery
- Part 13: Scalable Video Coding (SVC)
- Part 14: Conformance Testing
- Part 15: Event Reporting
- Part 16: Binary Format
- Part 17: Fragment Identification for MPEG Resources
- Part 18: Digital Item Streaming

This work will mainly focus on parts 2 and 10. Hence this will be covered with more detail. For more information about the remaining parts, the reader is referred to the provided references.

2.1 - MPEG-21 Part 2: Digital Item Declaration

Digital Item Declaration (DID) [7], as the name suggests, this part has the purpose to *declare and describe* the Digital Item. More specifically the purpose is to specify a uniform and flexible abstraction and interoperable schema (by allowing the connection to other parts of MPEG-21) to declare the structure, makeup and organization of a Digital Item. The DID specification is described in three normative sections: Model, Representation and Schema.

Model: The Digital Item Declaration Model describes the tools ("a set of abstract terms and concepts" [7]) to form the model that will allow the definition and description of Digital Item's (DI's).

Representation: The representation specifies the Digital Item Declaration Language (DIDL) which is based upon the terms and concepts defined in the above model. It contains the normative description of the syntax and semantics of each of the DIDL elements, as represented in XML, and intends "to be as flexible and general as possible" [7]. This section also contains some short non-normative examples for illustrative purposes.

Schema: "Normative XML schema" contains all the grammar of the Digital Item Declaration representation, used to validate the declaration and description of the Digital Item (DI), expressed in XML.

DIGITAL ITEM DECLARATION MODEL

Following, each one of the entities of the model will be described. These are the entities that together will allow the description and definition of the DI, involving the specification of resources, metadata and their interrelationships.

Container – like the name suggests it's an entity that contains entities that can be items and/or containers forming a group that can be seen as logical packages (for transport or exchange) or logical shelves (for organization) [7], can be formed using this groups. They also can contain information in the form of labels.

Item – similarly to the containers, an item is also a group formed by sub-items and/or components, and may contain D*escriptors*, *Conditions*, *Choices* and *Annotations*. This way, with these entities, when the item is defined and described it will be possible to configure and customize the *Item* [7]. Plus it will contain information about its sub-parts.

Component – binds resources and D*escriptors*. These D*escriptors* won't have information about the contents of the resource, but may have structural and/or control information about the resource.

Anchor – binds descriptors to specific locations or areas within resources (fragments).

Descriptor – basically its information associated to the enclosing entity.

Condition – the inclusion of the enclosing entity becomes optional and it will be defined by a selection.

Choice – allows choosing one, all or no *Selection*, which will affect the configuration of enclosing *Item*.

Selection – the effect of this entity, it's that it will define if the enclosing entity is included or not.

Annotation – where information about other identified entities is retained without making changes to that element. "The information can take the form of *assertions*, *descriptors*, and *anchors*" [4].

Assertion – indicates the configuration of a choice's state (through the predicates) by asserting true, false or undecided.

Resource – through an unambiguous address, it identifies an Asset such as a video, an audio clip, an image or a textual Asset.

Fragment – designates a location on a resource, being ether a specific location or an area within it.

Statement – it's user readable text giving information about the enclosing entity.

Predicate – it's a declaration that can be true, false, or undecided.

DIGITAL ITEM DECLARATION LANGUAGE, DIDL

Consists in a DIDL root element of a DIDL instance document [7]. This root element can have a single Item child or Container child and also a section where Declarations can be made. The DIDL syntax is based on an abstract structure defined in the Digital Item Declaration Model (using XML Schema), trying to make it as flexible and general as possible, enabling higher level functionality. In Figure 1 there is an example of the hierarchical structure of the Digital Item Declaration Model, with some of the most important entities.

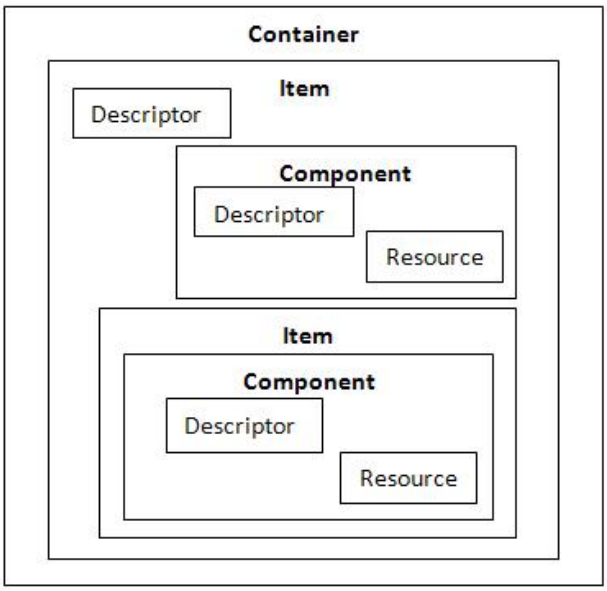


Figure 1 – Example of the hierarchical structure of the Digital Item Declaration Model (from [7])

2.2 - MPEG-21 Part 10: Digital Item Processing

DID and DIDL provide the tools to describe a DI and the way its resources and metadata are related, but do not specify a way for the User to interact with the DI. Digital Item Processing (DIP), the 10th part of MPEG-21 (ISO/IEC 21000-10:2005) [8], is a mean to specify the tools not provided by DID that allow Users (the creators of the DID) to provide their suggestion of interaction with Digital Item, so that the DID may be consumed in a dynamic manner. Some examples of Digital Item Processing are: Digital Item downloading, IPMP handling and rights management, media resources downloading, printing or playing, presentation of the Digital Item, etc [1], but a User can also create new interactions with the Digital Item. Because there are numerous types of Digital Items, and Users might want to consume it in different ways, there is a tool that they can use to specify suggestions of

interactions with the Digital Item when validating, processing and consuming it, which is the Digital Item Method (DIM). This tool is used at the level of the Digital Item Declaration and consists in programmatic scripts [1] that use other tools to create the suggestion of interaction, and these other tools are:

- **Digital Item Base Operations (DIBOs)** – are the more basic operations with a normative interface and semantics. This operations will allow to specify basic functionalities supplied to a User and may be implemented in different ways;
- **Digital Item Method Language (DIML)** – the language used for defining DIM's (which is ECMAScript [3]), from which DIBO's can be called;
- **Digital Item Method linkage with DID** – specifies mechanisms that allow the inclusion of DIM's in DID's;
- **Digital Item Method execution** – specifies the execution environment of a DIM;
- **Digital Item eXtension Operations (DIXO)** – allows the creation of new operations and it can also use DIBOs in the construction of a DIXO.

Despite the existence of DIBO's, Users might need to increase the range of operations that can be used in the DID without compromising efficiency. For that they can use DIXOs in DIMs with the difference of not being normatively defined and that can be created in a programming language chosen by the implementer (e.g. JAVA, C++, etc). This allows the creation of specific methods to a certain working area or even to optimize the implementation of a certain DIM. A DIXO is called by a specified DIBO (runMyDIXO – e.g. runJDIXO in the case of java implementation). Currently it is only possible to create DIXOs using JAVA [8].

DIM is authored using the Digital Item Method Language (DIML). A Digital Item Method definition can be embedded (included in the Digital Item Method declaration) or referenced (located separately). As it was mentioned, DIBOs are basic operations. Some of those standard operations in DIBOs are related to other parts of MPEG-21 with the following operations [8]:

1. DIA – adapt;
2. DID – areConditionsSatisfied, ConfigureChoice, setSelection;
3. DII – getElementsbyIdentifier, getElementsbyRelatedIdentifier, getElementsbyType;
4. DIP – alert, execute, getExternalData, getObjectMap, getObjects, getValues, play, print, release, runDIM, wait, calling DIXOs;
5. REL – getLicense, queryLicenseAuthorization;

Digital Item Method Language includes a binding to Digital Items Based Operations. Despite the fact that DIP specifies the syntax and semantics of the DIBOs, it doesn't specify how it should be implemented therefore the DIBO implementer is free to choose how to implement the semantics.

Object Map

An Object, which is any element in the DID, can be associated to an ObjectType in the DID using an ObjectType descriptor. This ObjectType descriptor is represented by a DIP ObjectType element contained in a DIDL DESCRIPTOR-STATEMENT. An ObjectType may be associated to any DIDL element that may contain a DESCRIPTOR. Object Map provides the link between the arguments of a DIM and a given ObjectType, and the actual Objects in the DID that are associated with the same ObjectType. It's a map consisting of two parts, where the first one is a list of DID Objects (i.e., DIDL elements in a DIDL document that have a DESCRIPTOR-STATEMENT construct containing a DIP ObjectType element), the second one is a list of DIMs (that have zero or more DIM Arguments) [8]. In Figure 2 there is an example showing that each DID Object are of a type, stated by the value of ObjectType, and the DIP Argument element if the type of the object, so we can se here that DIM 1 can be associated to DID Object 1 and DID Object 2, and DIM 2 is associated to DID Object 3.

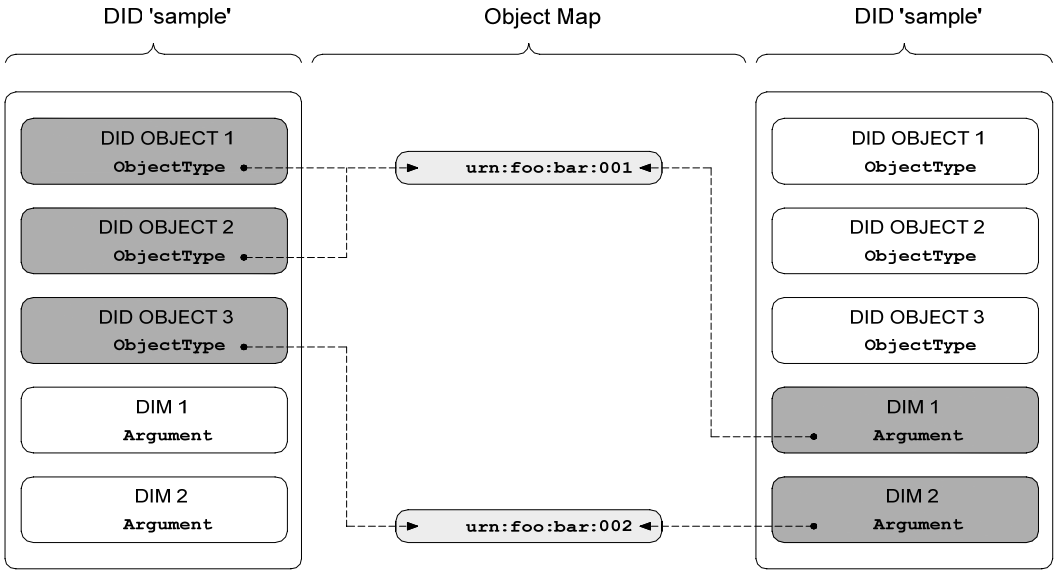


Figure 2- Example of an Object Map relating DID Objects to DIM Arguments (from [8])

To better understand Figure 2, it is shown two pieces of a DID. First a DID object (here represented by an *Item*) with the declaration of an Object Type is presented and then the declaration of a DIM with an argument is presented. Because the Object Type and the argument are equal a relation between them can be created be means of the Object Map.

```
<!--DID Object, with declaration of Object type (urn:foo:PrintableResource)-->
        <Item id="MPEG_LOGO">
                <Descriptor>
                        <Statement mimeType="text/plain">Mpeg Logo</Statement>
                </Descriptor>
                <Component>
                        <Descriptor>
                                <Statement mimeType="text/xml">
                                        <dip:ObjectType>
                                                urn:foo:PrintableResource
                                        </dip:ObjectType>
                                </Statement>
                        </Descriptor>
                        <Descriptor>
                                <Statement mimeType="text/plain">
                                        MPEG Logo
                                </Statement>
                        </Descriptor>
                        <Resource mimeType="image/gif" ref="C:\stuffs\mpeglogo.gif"/>
                </Component>
        </Item>
```

```
<!--DIM declaration with argument (urn:foo:PrintableResource) -->
        <Component>
                <Descriptor>
                        <Statement mimeType="text/xml">
                                <dip:MethodInfo>
                                        <dip:Argument>
                                                urn:foo:PrintableResource
                                        </dip:Argument>
                                </dip:MethodInfo>
                        </Statement>
                </Descriptor>
                <Descriptor>
                        <Statement mimeType="text/xml">
                                <dip:Label>
                                        urn:mpeg:mpeg21:2005:01-DIP-NS:DIM
                                </dip:Label>
                        </Statement>
                </Descriptor>
                <Resource mimeType="application/mp21-method"><![CDATA[
                        function PRINT_PRINTABLE_RESOURCE(arg1)
                        {
                                return DIP.print(arg1);
                        }]]>
                </Resource>
        </Component>
```

Chapter 3

3 State of the art

In this chapter a state of the art survey of applications that use or that are based in MPEG-21 relevant to this dissertation are presented. The presentation of the applications is made with a description of their architecture and when possible, some Figures exemplifying the use of them are presented.

3.1 - Muffins

Muffins - Multimedia Framework for Interoperability in Secure (MPEG-21) Environments was a European Community funded project, which started July 2002 and ended December 2003 [9]. The project dealt with the problem of transparent improved delivery and use of reach-media resources across a wide range of networks and devices, by different categories of users in multiple application domains. The main objective was to contribute to the development of the MPEG-21 standard and to focus on: identifying; defining; developing and implementing components able to solve the problem of description; delivery and protection of reach-media content; building an example application; demonstrating the validity of their solution for example scenarios of usage of that content. The integrated framework of multimedia retrieval, transport and consumption based on open standards, also included the definition and search for content, as well as the delivery and the related rights management handling [9]. The terminal application built by the MUFFINS consortium was meant for consumers and consisted of a web-browser application with the ability to search, upload, download content and even to buy and sell licenses. It has stored in the terminal a media player; and "IPMP Components with features specified in the 4th part of MPEG-21" [1].

3.2 - DANAE

DANAE - Dynamic and distributed Adaptation of Scalable multimedia content in a context-aware environment, is a IST European co-funded project, that addresses "the dynamic and distributed adaptation of scalable multimedia content in a context-aware environment" [10].

To specify, develop, integrate and validate a complete framework able to provide end-to-end quality of (multimedia) service at a minimal cost to the end-user is their main objective [10]. Several features based in other MPEG standards, namely MPEG-4 and MPEG-7 [1], and Digital Rights Management (DRM) support will be integrated. They were mainly concentrated in DIA and DIP contributing to the MPEG-21 standardization effort [1].

The application developed consists of three main modules, with several smaller components:

- *The Server module* where the DIs are stored, and where the DI is adapted to the terminal capabilities;
    - **Server DIP Engine -** in collaboration with the Client DIP Engine it manages communications, sessions, it create and transfer de DI's required by the Client DIP Engine;
    - **Adaptation Engine -** is enabled by the Streaming Interface (SI) when requested by the Multimedia Player, and is responsible for adapting the DI, to the context and/or capabilities of the terminal. This context information is provided by the Context Aggregation tool;
    - **Context Aggregation tool (CAT) -** all the context information coming from the terminals is stored and aggregated here, this information comes through the Client DIP Engine as an input for the server adaptation engine;
    - **Streaming Interface (SI) -** retrieves the information from the Adaptation Engine, sending it to the respective terminals;
- *The Terminal module* is the module, through which the User can consume or even create DIs;
    - **Client DIP Engine -** is the responsible for driving the other modules in the terminal, including the Multimedia Player, as a direct connection to the Server DIP Engine, sends the terminal context information to the CAT, and collects and processes DIDs;
    - **Context Collection Tool -** collects and processes various data from sensors inside each terminal, after which these data is sent to the server to "guide the media adaptation";
    - **Multimedia Player -** the responsible to play the DIs content, and requesting it, and is connected to the SI;

- **DRM Tool -** manages the "decryption keys", when dealing with protected content.

- *The adaptation node module* is a module where manual adaptation can be made, controlled by the CAT;

  - **Adaptation Engine** - that has the same features above;

  - **Streaming Interface (SI)** - that has the same features above, plus if the adaptation node is used, it will stream the Metadata "to enable generic adaptation in the network" [10].

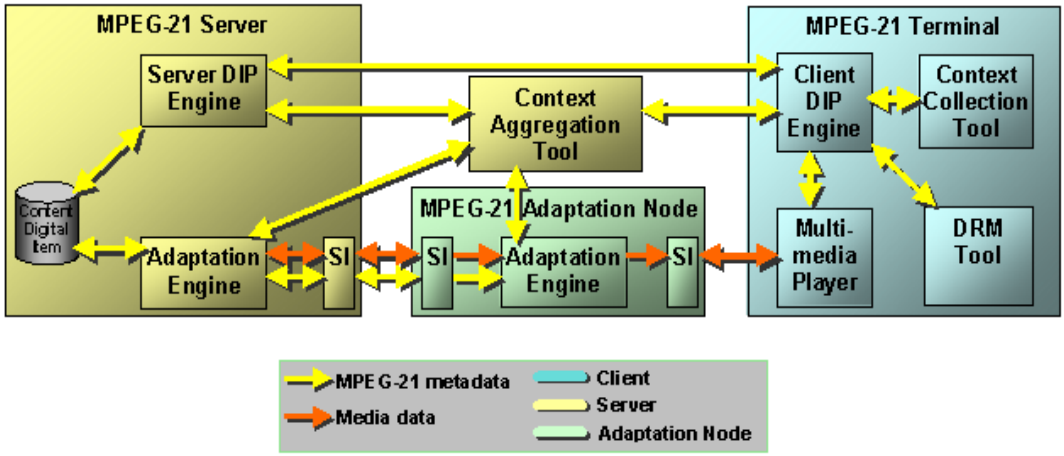


Figure 3 – DANAE's Architecture (from [11])

3.3 - Ghent's MPEG-21 Applications

The University of Ghent, through the research group Multimedia Lab, funded in 2001 is one of the contributors in the effort of standardization of MPEG-21. They have developed several demo applications based in MPEG-21 such as:

- *MPEG-21 Session Mobility* - in this demo application it's demonstrated transparent transfer of multimedia sessions between different devices (e.g., two pc's or a pc and a PDA) [12];

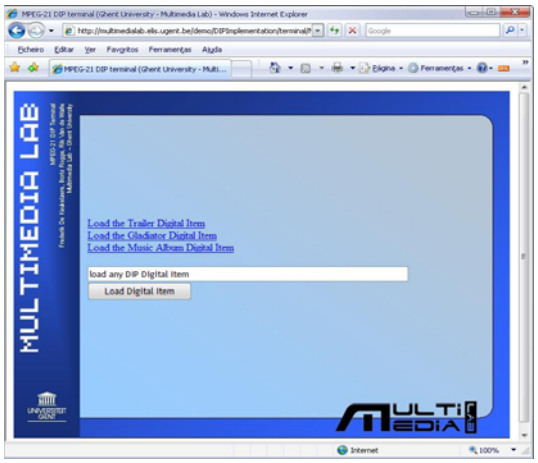
- *MPEG-21 Scalable Video-on-Demand* - this Video-on-Demand demo application uses MPEG-21 and Time-Dependent Metadata (which allow to modify parameters that influence the quality of the stream) to deliver MPEG-4 videos to a client [12]. After choosing the streaming server a list of movies is received, from which the User will be able to choose the video. After choosing the movie a stream is send to the computer;

- *Temporal synchronization of media within MPEG-21 Digital Items Declarations* - this example includes demos showing "how HTML+TIME and SMIL technology can be used combination with MPEG-21 technology to create synchronized multimedia presentations" [12];

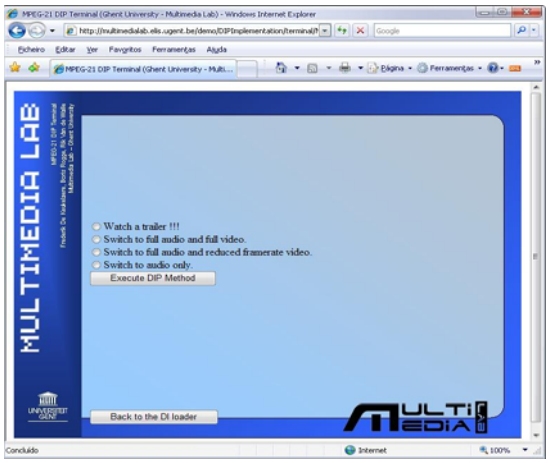
- *MPEG-21 wireless application* - this example combines WAP/WML technology and MPEG-21 technology. The objective is to build a device specific interface to WAP applications;

- *MPEG-21 Digital Item Processing using DIP and DIM* - his example is a terminal application able to use DIP and DIM to implement functionalities of MPEG-21 tenth part. To do so, this generic terminal application has the capability of processing DI and with the application is provided three MPEG-21 Digital Items to demonstrate its use. This MPEG-21 DIP Terminal uses web environment (the application runs in the User terminal and has to be downloaded). The first step when entering the terminal is to choose one of three available Digital Items ("Load the Trailer Digital Item", "Load the Gladiator Digital Item" and "Load the Music Album Digital Item" [12]). If the Trailer Digital Item is chosen a new menu with several messages will appear, here we are able to run a play method or even choose one of three methods used to dynamically configure the streaming video (this may also be done when the trailer is playing, but it will only take place after the buffer of the player is processed), sending SOAP messages to the streaming server. If the choice is o load the Gladiator trailer, then a method able to start the HTML+TIME (used in the demo of the "Temporal synchronization of media within MPEG-21 Digital Items Declarations") will be used. For last if the choice is to load a music album, we are able to choose on of the music tracks available in that music album and run the method that will play the music track.

In Figure 4 there is an example of the usage of the MPEG-21 DIP Terminal showing several menus and possible methods that can be executed in a certain DI.

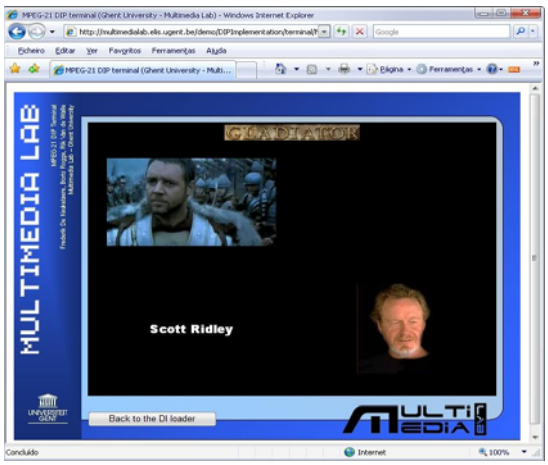


a) Main menu, where the Digital Item is chosen

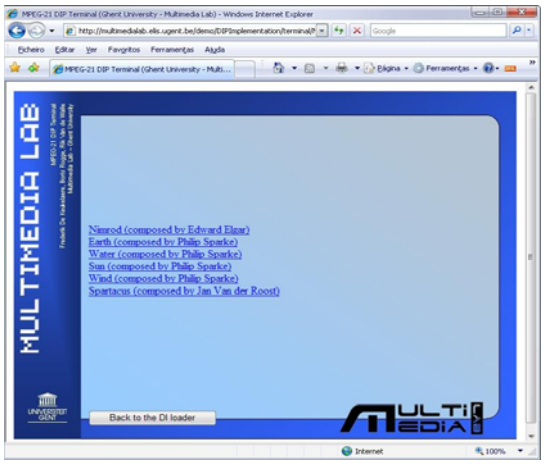


b) Run a trailer method, and the methods to configure the stream

c) The trailers available to the run a trailer method



d) Gladiator Digital Item



e) Music album Digital Item (available tracks)

Figure 4 - MPEG-21 DIP Terminal demo (menus and methods)

3.4 - AXMEDIS

AXMEDIS is a project under the Information Society Technologies programme of the 6th Framework Programme, partially funded/supported by European Community that had an effective start in September 2004, and had the contribution of several partners including User Group Experts and Affiliated Members.

AXMEDIS Multichannel Digital Rights Management is an application able of viewing licenses purchased by the User related to a DI. Plus AXMEDIS developed two other applications which are a DI Player and a DI Editor. Next these two applications are presented.

AXMEDIS Player

This player developed by AXMEDIS supports six different types of media, it has incorporated a video player, an audio player, a SMIL player, an MPEG4 player, an image viewer and a document viewer. Hence, DIs with different kinds of resources (e.g. AXMEDIS has an example of a DI with html documents, audio resources and image resources) can be viewed. In Figure 5 there is an example of a DI being consumed in AXMEDIS Player showing to the user several different resources (image, text).

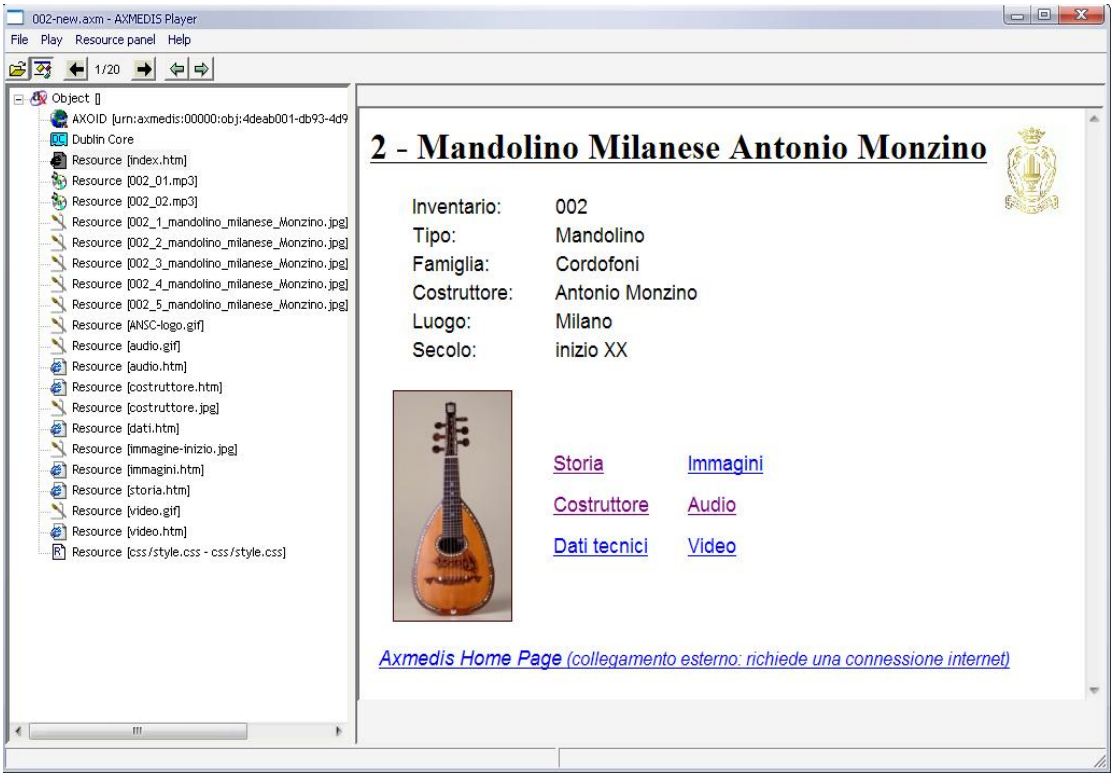


Figure 5 - AXMEDIS Player (example with different resources)

As it can be seen, this player has 2 different areas; in the right side is the area where the User can interact with the resource (in this case this interaction is identical to a web page);

in the left side there is a tree view with the hierarchical structure of the MPEG-21 DID, where the User can navigate and see all the resources within the DI.

AXMEDIS Editor

In this editor, it can be found the same functionalities that the player exposed above, plus several other functionalities, such as DRM editing, Metadata editing, etc. It extends the MPEG-21 standard enabling the User to: protect any content formats and types, control the exploitation of rights and collect and report information about consumption of rights [13]. All the User's actions made upon de content are controlled and validated by the authoring tool.

In Figure 6 we can see the same areas of the player and a tree view, but in this case the tree view has 2 different views, one called "AXMEDIS View" where we can see the resources and act upon them, by editing the resources properties, moving it up/down in the DID hierarchy, etc; and a view called "MPEG-21 View" where it can be seen the hierarchy of the child elements Descriptors and Components are the Item child's, the Statement elements and the Resource elements are respectively the Descriptors and Components child's. Additionally there is also another area (positioned in the right side) that contains the several options available to edit the DI (e.g. add objects, add resources, etc). An example of this is presented in Figure 6.

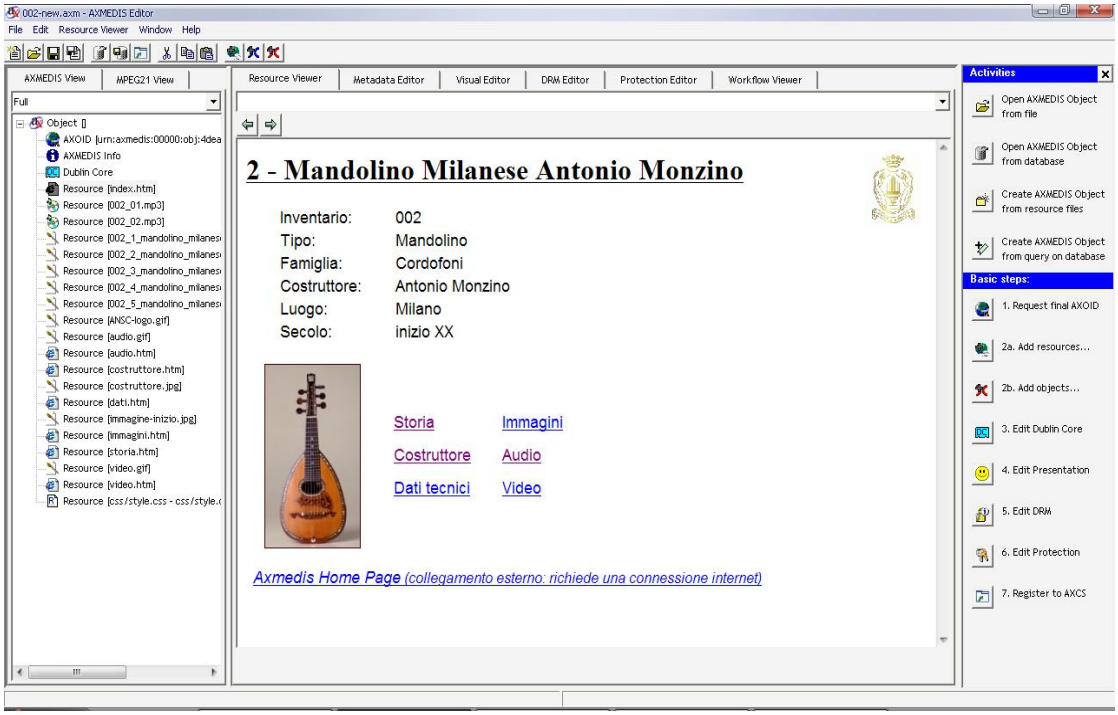


Figure 6 – AXMEDIS Editor (tree view in AXMEDIS View)

For example if want to add a resource, we can go to the right side of the application and click on the second option (add resource) that will open a dialogue box where we choose the resource to add to the Digital Item. The following Figure (Figure 7) shows the dialogue box where resource options can be altered.

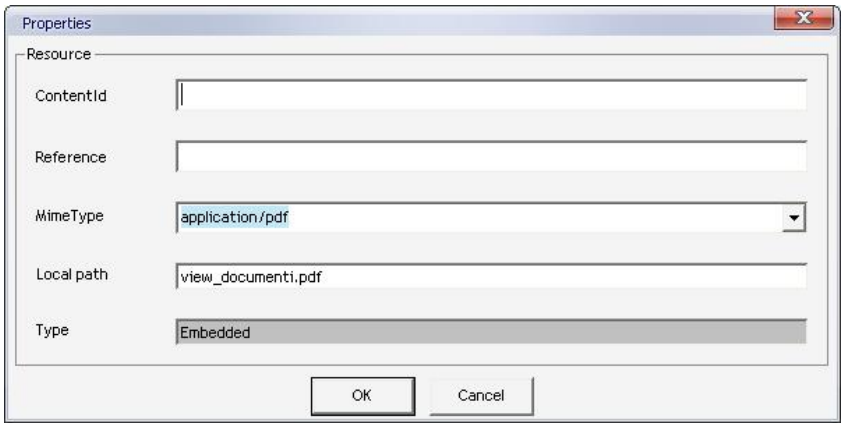


Figure 7 – AXMEDIS Resource properties editing dialogue box

The metadata inside a Digital Item can be altered through the AXMEDIS View by choosing the metafile to be altered and then editing its elements that will show in the middle area of the application.

3.5 - ENIKOS

Enikos is a multimedia company based in Sydney, Australia, that develops and supplies software tools for next-generation multimedia applications based on the MPEG-21 standard [14]. This company developed one of the first's MPEG-21 based applications, called "DICreator" that enabled the creation, editing and browsing Digital Items in a fully validated way conformant to the 2nd Part of MPEG-21 standard [15].

DICreator

DICreator application is constituted by full GUI based DI Editor and a customizable DI Browser.

This DI Editor has the following characteristics:

- Simple hierarchical view of the Digital Items;
- Intelligent menus to add Digital Items elements in the hierarchical position desired;
- Display and rendering of media resources;
- Quick and easy addition of resources by direct reference or adding it directly to the digital item (making it embedded);
- Intelligent tip screens for easy insertion of common element combinations and resources;
- Uses pull-down menus to assist the entry of attribute values;
- When a Digital item is demanded or loaded or even saved they may be validated;

- Full copy and paste of hierarchy across and within DIs.

This DI Browser characteristic:

- Split panel windows for navigational views, content views and an area for messages;
- Displays Description about the selected item;
- Intelligent and appealing presentation of Choices;
- Full playback of the most common media types, using external players to play the others;
- Simple interface with a history of the browsed Digital Items.

With these 2 parts, the DICreator becomes a very simple and easy application, and both of them have "internal multimedia capabilities the ability to launch external viewers for specialist file types" [15].

MPEG-21 DIP Desktop Peer

This is a most recent application released in 2006; it's a Java application which is a demonstrator of the DIP functionality. When the application is started, the first thing that appears is a dialog where DID's can be locally or remotely opened, after opening a DID it appears a list of DIM's able to run, therefore the User only needs to choose the DIM he wants to run. At any point the User can see the DID source and the DIM source in View menu. Enikos has in their web page a few examples of DID that can be used in the DIP Desktop Peer. An example of opening a Music Album DID, choosing a method and executing it is shown in Figure 8 and 9.

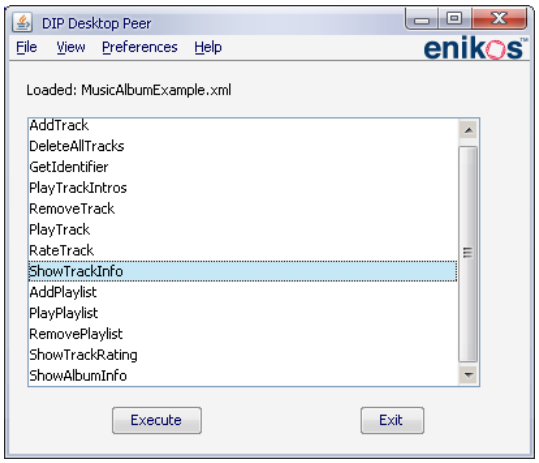


Figure 8 - Enikos dialogue box after opening a Music Album Example DID and selecting ShowTrackInfo DIM

Figure 9 - Enikos dialogue box executing ShowTrackInfo DIM, after choosing that DIM and selecting the track (track 1 in this case)

3.6 - Klagenfurt University Demos

The Klagenfurt University has, like other institutions and groups referred above, contributed to the effort of standardization of MPEG-21. To do that they developed several applications, most of them within the 7th part (Digital Item Adaptation - DIA) of MPEG-21 (DIA gBSDtoBin Web Demo, DIA Web Demo and DIA BSDLink Webeditor). Two other applications developed by the Department of Information technology (ITEC) were the DIBuilder and the DIConsumer, both will be described next.

DIBuilder

DIBuilder is an online Java Applet application that allows a User to create simple Digital Items according to MPEG-21. These Digital Items are constituted by one or more video resources and it may have Descriptors (to describe the item as a whole), Choices (that allow selecting other resources), Licenses ("to grant rights to principals" [16]) and Components (allowing to reference, for example other video resources and other descriptors).

To create the Digital Item several steps have to be made. When the applet begins the User is presented with a description of the applet and information about how to use the applet. The next step consists in choosing the video resources for the Digital Item (shown in Figure 10), the next steps consist in creating an MPEG-7 Description, a DIA Description,

License creation and the creation of Choices (this 4 steps are all optional), in the end the User just needs to create the Digital Item by adding (if he chooses so) a global description, configuration that allows further more to export the DI or the resources and it has to define the location where the Digital Item will be saved (shown in Figure 11).

Figure 10 – Klagenfurt DIBuilder, Selecting resources dialogue

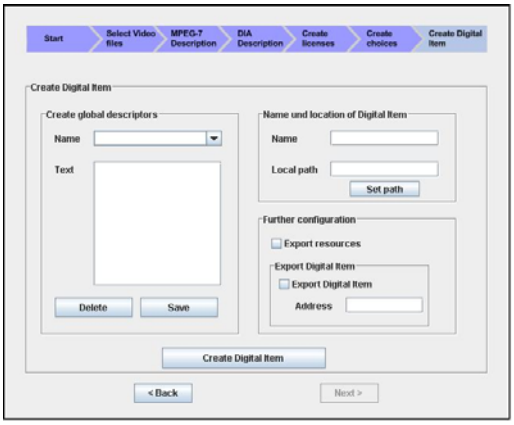


Figure 11 - Klagenfurt DIBuilder, Creating DI dialogue

DIConsumer

Just like the DIBuilder, this is a Java applet application very simple to use. The first thing presented to the User is a description of the applet and information on how to use it, after that the User has 3 steps to consume the Digital Item.

Thee first step is to choose the DI that is to be consumed (shown in Figure 12), the second step if to choose the resources to be consumed according to the choices that the item contains, the final step is consuming the Digital Item by selecting the resources available according to the choices made.

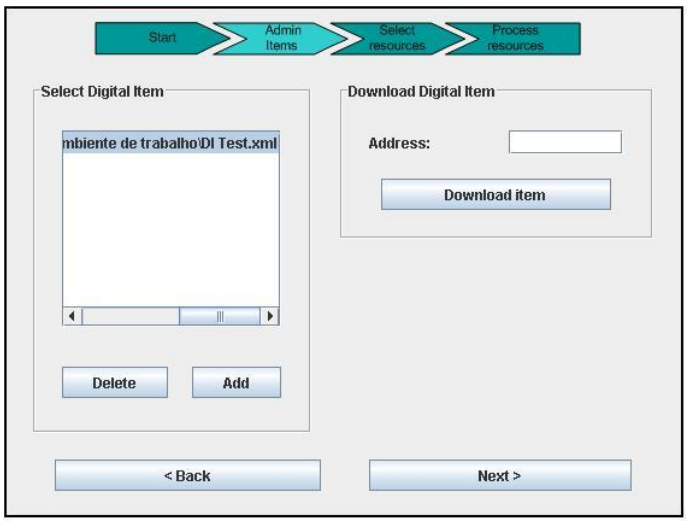


Figure 12 – Klagenfurt DIConsumer, selecting a DI to be consumed

These two applications have the advantage of being very simple and intuitive, which allows any User to create Digital Item with video resources. The disadvantage is that it is limited to MPEG-4 video resources, restricting its applicability to the MPEG-21 framework [1].

3.7 - ADACTUS Products

This company has several products related to the MPEG-21 standard. Their goal is to offer solutions for delivery and presentation of multimedia contents adapted to mobile terminals giving their best in maintaining the Quality of Service and Quality of Experience [17].

mobilize

This is the simplest product in ADACTUS portfolio [17] and other products like mobilizeOnDemand and mobilizePublisher were built based in this product. It's a "highly scalable cross-platform content delivery system that enables content providers and broadcasters to deliver their content to all the new multimedia-capable terminals"; the content will be adapted to fit the terminals capabilities, safeguarding the owner's rights (DRM mechanisms) and having the ability to report about the contents usage and consumption [17]. The function of this application is to take care of all the processes of ingesting, adapting and publishing the multimedia contents. The contents can be presented by general browsers or by downloadable applications prepared to present such content. For instance, it allows commercials to be presented as metadata to the video streams or during the Users navigation through the content [17].

Mobilize allows the owner of content, to adapt it to:

- Mobile Phones
- Internet
- iPod

- PSPs
- Generic multimedia players compliant to the standards

In the consumer's perspective, there are several consumption scenarios provided by Mobilize:

- Live streaming of video
- Streaming of on demand- accessible video- clips
- Download of videos
- Subscription to a podcast/ blog
- Download of video content for standards- compliant multimedia devices

The Mobilize platform contains a JAVA ME-Client that enables, for instance the broadcaster or other content creators, to build an application to present a specified content. From a broadcaster point of view this client can create applications with an advantage over browser environments that is to communicate with the User and/or delivering to him in a much more advanced way (more dynamically) graphics, metadata and content. Also it was designed to be integrated into different content workflows, through a flexible XML interface that prepares and publishes some of that workflows content in an easy way. The contents publication is made through an XML file posted towards the Mobilize server's URL, containing a description of the several resources used in the content, which will be validated by the server notifying registered users for the existence of this new MPEG-21s Digital Item [17].

mobilizeOnDemand

This platform built to drive highly interactive applications, is based on the mobilize platform, as said above. The content is distributed (by content providers) in custom designed applications (applications built to "present specified content, to handle user interactions, to send breaking news or alerts to the user, to render data locally on the client terminal (commercials, tickers, graphics), to report content back to the newsroom (text, images, audio or video) or to create user communities" [17]) in order to achieve optimal presentation and usage. Broadcaster's brands can be distributed to new devices through mobilizeOnDemand.

mobilizePublisher

This system was created to enable publication of contents in browser platforms or terminals able to play multimedia files. It can be used by the content provider either to create the presentation or to just handle the multimedia content. The contents that this platform enables to create are, "on demand or live mobile TV, on demand or live content to a PC or a set top box, encoding of video and creation of a podcast, encoding a video for PSP playback, or integration with a broadcast mobile TV platform" [17] and it can distribute videos as SMS or as a WAP Push message for mobile terminals. The platform also enables the

content provider to protect the content (if he chooses so, if not its freely available) through an authentication scheme

mobilizeLive

Created to enable live video encoding in mobilizeOnDemand and mobilizePublisher platforms, by capturing the video live in the platform and then create several outputs from the same captured video, to be send to the several terminals for playback [17].

mobilizeAds

It's a plug-in for mobilizOnDemand enabling commercials to be distributed. The distribution can be made, by sending the commercials as metadata at the same time as the content or as "part of the video stream, as part of the menu structure presenting the content or even as a combination of these" [17].

3.8 - ENTHRONE II

ENTHRONE is an acronym for *End-to-End QoS through Integrated Management of Content, Networks and Terminals*, which started its 2nd phase in September 2006 with the duration of 24 months. Their main objective is to propose an efficient, distributed and open management solution supporting end-to-end Quality of Service (QoS) that ensures the distribution across heterogeneous networks and reception in various user terminals of audio-visual contents (or services) not forgetting to handle the protected content having has main costumers the services providers (e.g. Broadcasters, Telecommunications Operators, among oters) [18]. Using the MPEG-21 Standard the project aims to achieve an interoperable transparent access to multimedia resources contributting to "the realisation of *Universal Multimedia Access* (UMA)" [1].

The content to be delivered to the User terminal is limited to its capabilities and requested quality. Therefore it has to be generated, protected and distributed in a scalable manner by managing the whole distribution chain by taking in consideration 2 aspects: interoperability that can be achieved using common protocols; and adaptation achieved in this case by using the open standard MPEG-21. The proposal of the ENTHRONE project is to have protocols defining the messages to be exchanged by different distributed components and using the open standard MPEG-21 to transparently manage, access, generate, transfer, adapt and consume multimedia content in order to provide QoS guarantees.

To acomplish this objectives, the ENTHRONE project has developed an *Integrated Management Supervisor* (IMS) which is a "distributed context-aware content management system to enable the customised access to multimedia content from diverse client devices" [19]. This system supports the transaction of DIs and makes the necessary adaptations trying to maximize the quality according to the capabilities of the terminal, the network conditions

and user preferences but also according to the MPEG-21 standard specifications. In Figure 13 its represented the functional architecture of the IMS.

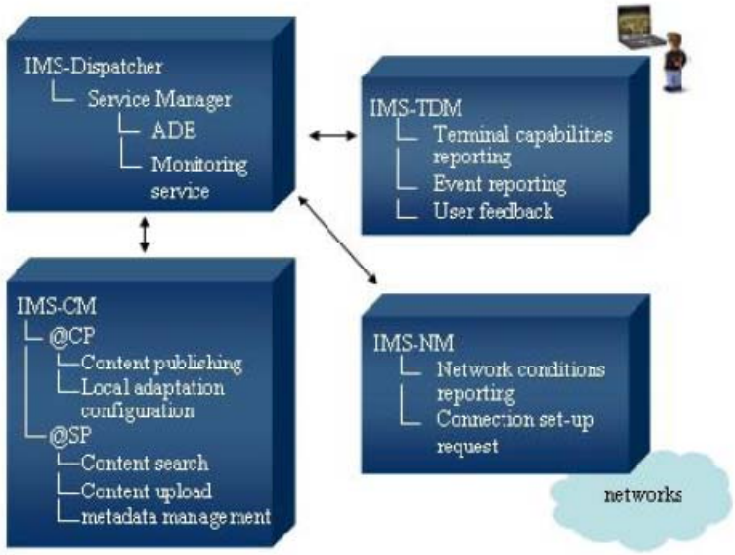


Figure 13 - The IMS functional architecture (from [19])

As seen in Figure 13 the ENTHRONE IMS system is constituted by 4 modules:

- **IMS-Dispatcher** – makes the overall coordination of the IMS system, manages user requests using the values delivered by the other sub-modules and monitors the agreed services [19];
- **IMS-TDM –** *IMS-terminal* Device Manager gathers and passes to the *IMS-Dispatcher* information about the user terminal, the surrounding environment and delivers notifications about events through a platform-independent interface;
- **IMS-CM –** *IMS-Content Manager* is responsible for storing, accessing and handling multimedia metadata and resources; it also publishes the DI through several specific modules running at the Content Provider side (IMS-CM@CP) and there are modules at the Service Provider side (IMS-CM@SP) to manage the access to the contents using a data base;
- **IMS-NM –** *IMS-Network Manager* is responsible for managing the network services;

The *IMS-Dispatcher* has a module called *Service Manager* which is the responsible for support the end-to-end QoS through the implementation of all the functionalities needed at application layer. This module is constituted by other sub-modules (represented in Figure 14):

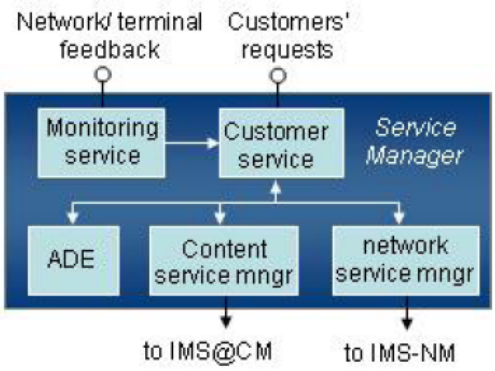


Figure 14 - The IMS-Dispatcher Service Manager elements (from [19])

The sub-modules of the Service Manager are:

- **Costumer Service** module receives the user requests and coordinates all the operations necessary for the adaptation decision;
- **Adaptation Decision Engine (ADE)** is the module that in fact makes the adaptation decision;
- **Content Service Manager** module establishes connections to other adaptation engines giving them the required parameters;
- **Service monitoring** module it is an "essential element of the service monitoring functionality provided by the *Service Manager* " because it's the interface between the *Service Manager* and the external quality measurements devices spread along the delivery path;

Chapter 4

4 MPEG-21 Reference Software

This chapter has the objective of presenting the MPEG-21 DIP Reference Software which is a sub part of the 8th part of the MPEG-21 standard. It corresponds to the study made to verify if the Reference Software can be used in a distributed architecture. It is presented the general architecture of the DIP Reference Software and afterwards their main blocks are described. In the end it is presented the results of tests made to the DIBOs implemented in the Reference Software to see which can be executed in the server side and which ones have to be executed in the client side.

4.1 - DIP Reference Software

Before implementing the functions needed to execute DIP in a distributed way, that allows to free the User terminal from most of the processing required for DIP requiring only the User terminal have to present to the User the resources and necessary metadata, the MPEG-21 DIP Reference Software was analyzed to understand how the software was built, which steps are made to execute a DIM and which DIBOs have to be implemented in the client. Next, Figure 15 shows the architecture of the MPEG-21 DIP Reference Software.

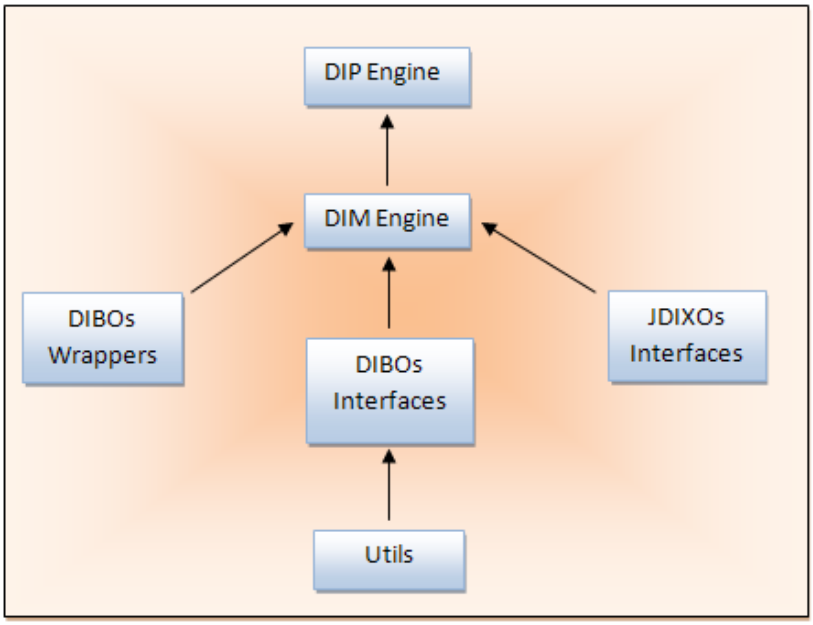


Figure 15 - MPEG-21 DIP Reference Software top architecture

The blocks shown in Figure 15 have specific purposes which are:

- **DIP Engine** – is a small implementation to execute DIMs locally, contains functions to verify if there are DIMs in the DID, to select arguments for the DIMs and to execute DIMs;
- **DIM Engine** – is responsible for the interpretation of the Java Script code used to write DIMs within the DID;
- **DIBOs Wrappers** – wraps the DIBOs into Java Script functions so they can be executed by the DIM Engine;
- **DIBOs Interfaces** – provides the interfaces for the DIBOs implementation to be used by de Wrappers;
- **JDIXOs Interface** – provides the interface for the JDIXOs to be executed in the DIMs;
- **Utils** – provides the implementation for all the DIBOs specified in MPEG-21 Part 10.

When trying to understand how the MPEG-21 DIP Reference Software was built, some aspects that could cause some difficulties were found. First, the behavior of the MPEG-21 DIP Reference Software is similar to a "black box". It is presented to the User a box to select the DIM to execute (in JavaScript code), then if the DIM has any arguments it is presented a box

to select the arguments and the necessary elements to process and then the software executes the DIM only displaying the result and not returning that result or any other information. Since the client is expected to present the result of the execution of the DIM to the User, changes in the MPEG-21 DIP Reference Software had to be made to ensure that the result of the execution is returned in order to be seen in the client. These changes concerned the module *DIM Engine* of the MPEG-21 DIP Reference Software.

Another aspect to consider is the implementation in the MPEG-21 DIP Reference Software of some DIBOs (e.g. *alert*, *play*, *configureChoice*, etc) that include graphic interfaces. For example, the *alert* DIBO receives a message and a message type, and then presents to the User a message box. The problem consists in how to remotely present this message box or other graphical interfaces to the User if the Web Services don't have the ability to execute Visual Classes of functions with visual elements?

In order to have a point of comparison that allows knowing if the remote execution of the DIBO is being made correctly it is necessary to make some tests to the DIBOs. These tests would be made locally and remotely. To be able to perform these tests it was necessary to create a few DIDs that would implement the DIBOs in DIMs. These DIDs were created based in example DIDs provided with the MPEG-21 DIP Reference Software and in the DID created when the first version of the DDIBrowser was developed.

Plus to make the tests for the DIBOs and see if they needed a new implementation, to see if the Web Services are working properly and see if they do the required operations, a test application was created in JAVA. This application uses the Web Services to remotely test the DIBOs and calls directly the java methods created to work as Web Services to test DIBOs locally. It allows the access to a repository on the server containing DIDs. This way the User would be able to change the DID without closing the application. The application has the same functionalities that the WDI Browser has referring to DIP. Plus it allows a faster interaction between the User and the services, because it is not necessary for the User to browse an *Item*, he only needs to choose an Object Type and then the list of DIMs that use that object type as an argument will be presented allowing then the User to execute the selected DIM. Hence a small description of the DID is presented to the User. This description is available has a metadata inserted in the DID.

Next is presented an example of this metadata inserted in the DID, which is *Title* and *docDescriptor*:

```
<Item>
        <Descriptor>
                <Statement mimeType="text/plain">
                        <Title>DIP-test 1</Title>
                </Statement>
        </Descriptor>
        <Descriptor>
                <Statement mimeType="text/plain">
                        <docDescriptor>
                                This example DID allows to test: PRINT, alert, runDIM...
                        </docDescriptor>
                </Statement>
        </Descriptor>
</Item>
```

The JAVA application created to test the DIBOs has the GUI presented in Figure 16.

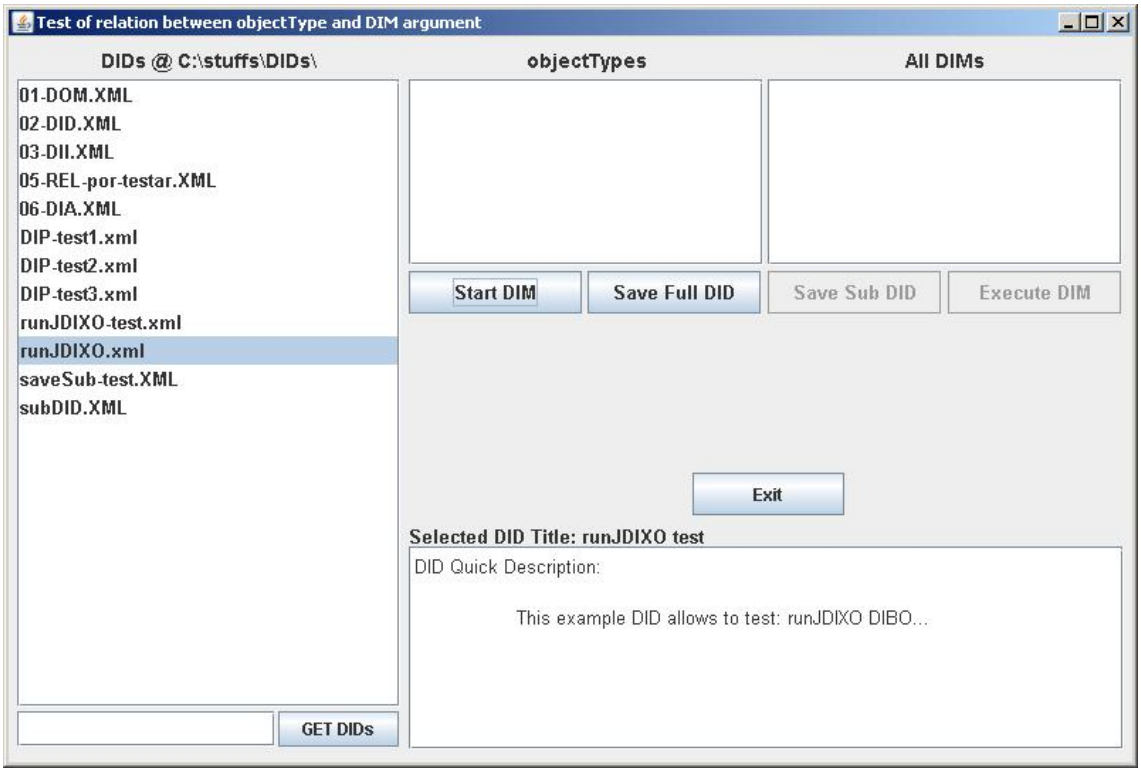


Figure 16 - JAVA Test Application to test DIBOs

Next, in Figure 17, it can be seen that there is an area where the list of DIDs is presented, an area for the Object Types, an area for the DIMs and an area for the description included in the new metadata. To begin executing DIP it's necessary to press the "Start DIM" button,

after selecting the DIM. When the "Start DIM" button is pressed the list of object types and the list of DIMs existing in the DID are presented in its respective area.

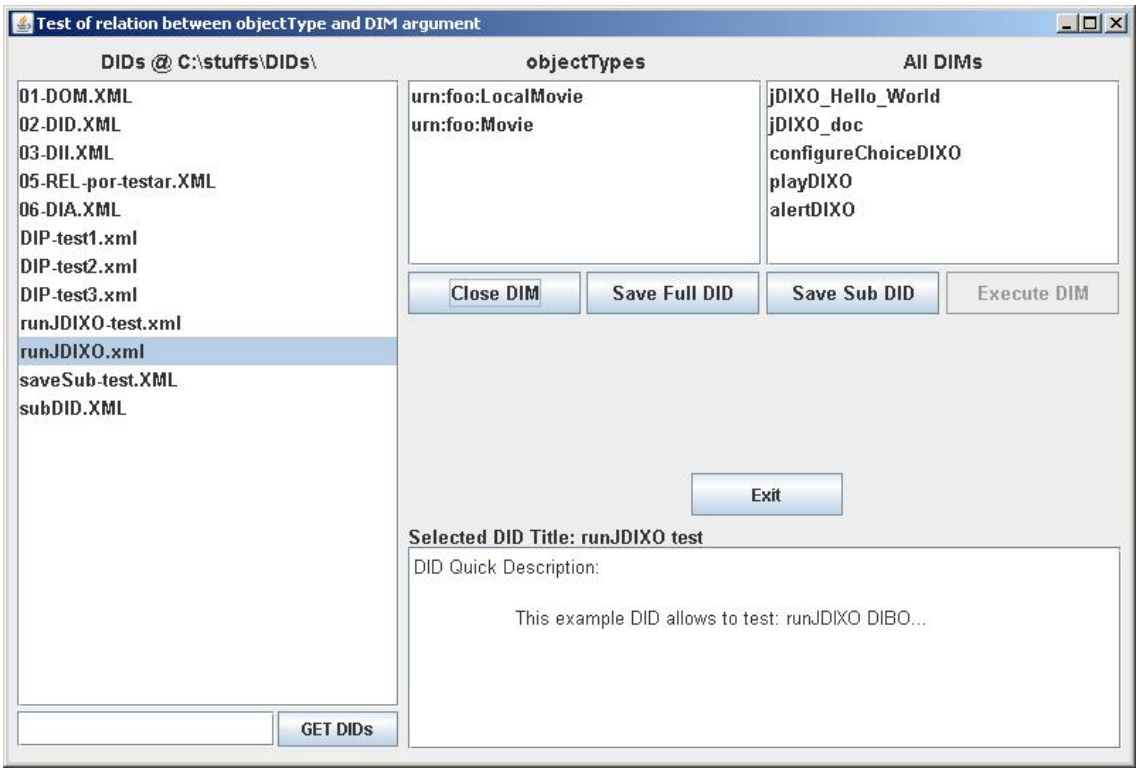


Figure 17 - JAVA Application GUI, presenting the list of Object Types and DIMs

To filter the list of DIMs it is only necessary to select an Object Type and then a new list of DIMs is presented. It is also possible to execute a DIM without filtering the DIMs.

Figure 18 presents the GUI with the DIMs list filtered by an Object Type.

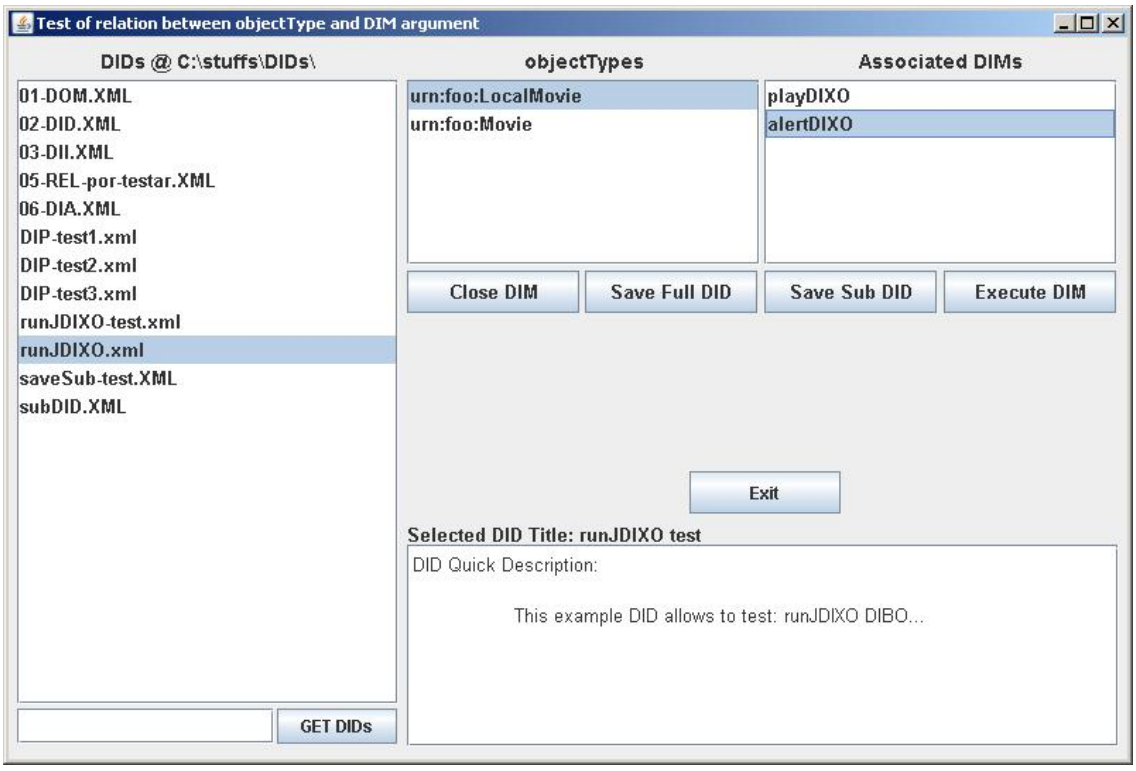


Figure 18 - JAVA Application GUI with new list of DIMs

Has it can be seen the list of DIMs of Figure 17 has been changed from a list of five DIMs to a list of two DIMs (Figure 18). Selecting one DIM and then pressing "Execute DIM" button allows executing the DIM. The button "Save Full DID" allows to save in the terminal where the application is running all the DID that is being executed. The "Save Sub DID" allows to save to the terminal where the application is running, an *Item* that is identified by an ID (e.g. *<Item id="item_13">)*. This GUI is exactly the same when testing DIBOs locally and remotely.

4.2 - Testing the DIBOs

In order to test the DIBOs it was decided that first, local tests using only the MPEG-21 Reference Software should be made in order to register the results of the execution of the DIBOs. Afterwards a remote test using the test application would be made. This way there would be a point of comparison to verify if the results of the remote execution were correct. For a better comparison of the results and to ensure that the results wont change each time the DIM is executed, each DIM will have the minimum possible number of DIBOs. Next two examples of DIM are presented to better understand this point:

```
//Example 1:
function get_object_type_Name()  //the name of the DIM
{
        var objectMap=DIP.getObjectMap(didDocument);
        return objectMap.getObjectTypeName(0);
}
```

Notice that in the DIM of Example 1, two DIBOs are used, *getObjectMap* and *getObjectTypeName*, this is because the objective of the DIM is to obtain the name of the first object type existing in the DID and to achieve that it is necessary first to obtain the objectMap.

```
//Example 2:
function execute(arg1)  //the name of the DIM
{
        DIP.execute(arg1);
}
```

In example 2 only the *execute* DIBO is used, this is because this DIBO doesn't require other DIBOs to be executed before it and it only needs to receive the argument, which is the *Item* where the resource to be executed is declared.

In table 1 is presented the results of the local tests and the remote tests made to the DIBOs, before making a new implementation to some DIBOs. Notice the result Undefined, that simply means that the DIBO implementation doesn't return any value; the comparison results indicate that if the local and remote execution returned the same result or not (Equal if the result is the same, Different if the DIBOs can't be executed remotely); and the result of the remote execution indicated with "No result" indicates that these are the DIBOs that can't be remotely executed and is needed a new implementation.

DIBO	**Result**		
	local execution	**remote execution**	**Comparison**
adapt	XML String	XML String	Equal
areConditionsSatisfied	True/False	True/False	Equal
configureChoice	True	**No result**	Different
setSelection	Undefined	Undefined	Equal
getElementsByIdentifier	Lorg.w3c.dom.Element	Lorg.w3c.dom.Element	Equal
getElementsByRelatedIdentifier	Lorg.w3c.dom.Element	Lorg.w3c.dom.Element	Equal
getElementsByType	Lorg.w3c.dom.Element	Lorg.w3c.dom.Element	Equal
alert	Undefined, but presents a message box	**No result**	Different

execute	True	**No result**	Different
getExternalData	Ljava.lang.String	**No result**	Different
getObjectMap	org.iso.mpeg.util.mpeg21.dip.dibo.ObjectMapImp	org.iso.mpeg.util.mpeg21.dip.dibo.ObjectMapImp	Equal
getObjects	Lorg.w3c.dom.Element	**No result**	Different
getValues	Returns the inserted values	**No result**	Different
play	org.iso.mpeg.util.mpeg21.dip.dibo.PlayStatusImp	**No result**	Different
print	True and prints the file	True but prints the file in server instead of client	Different
release	Undefined	Undefined, but only works if this is executed in the same machine where the resource is being played	Equal
runDIM	Undefined for the execution of the runDIM DIBO plus shows the result of the other DIM executed	Undefined for the runDIM execution plus the result of the called DIM	Equal, if the DIBOs in the DIM execute OK remotely; Different if not
runJDIXO	It is the result of the return defined in the JDIXO	It is the result of the return defined in the JDIXO	Equal/Different depends on the JDIXO implementation
wait	Undefined	Undefined	Equal
getLicense	Lorg.w3c.dom.Element	Lorg.w3c.dom.Element	Equal
queryLicenseAuthorization	False/True	False/True	Equal
getDIPErrorCode	Returns the code of the last thrown error	Returns the code of the last thrown error	Equal
getArgumentList	Ljava.lang.String	Ljava.lang.String	Equal
getArgumentListCount	2	2	Equal
getMethodCount	16	16	Equal
getMethodWithArgs	[Component: null]	[Component: null]	Equal
getMethodsWithArgs	1	1	Equal
getObjectOfType	[Descriptor: null]	[Descriptor: null]	Equal
getObjectsOfType	Lorg.w3c.dom.Element	Lorg.w3c.dom.Element	Equal

getObjectsOfTypeCount	2	2	Equal
getObjectTypeCount	3	3	Equal
getObjectTypeName	urn:foo:PrintableResource	urn:foo:PrintableResource	Equal
getStatus	Returns an integer depending on the playStatus	Returns an integer depending on the playStatus	Equal

Table 1 - Result of executing DIBOs locally and remotely (before new implementations)

The DIBOs implemented in the Reference Software that require a new implementation and why the new implementation is required, are presented in Table 2. The rest of the DIBOs do not need a new implementation.

DIBO	Requires new implementation?
configureChoice	Yes – because the implementation made in the Reference Software implements a visual elements and these elements can´t be executed in the client; therefore an implementation to maintain this idea is required;
alert	Yes – this implementation presents to the User a message box and therefore it is needed a new implementation capable of presenting this message box to the User in the client side;
execute	Yes – this operation executes external files like ".exe" files, etc. To ensure the proper execution, this implementation has to be made in the client where the server will ensure the security related issues;
getExternalData	Yes – this operation requests the User to select resources external to the current DID [8], so another implementation is needed, to ask to the User the URLs for the external resources;
getObjects	Yes – this operation requests to the User to select one or more DOM Elements of a given object type and this selection (in this software) is made through a GUI, that should also be implemented in the client;
getValues	Yes – similar to other operations, it requests to the User to enter a Boolean, a string or a number value by means of a GUI, therefore it also needs another implementation;
play	Yes – because the implementation plays the resource locally, another solution should be implemented, to allow the User to view the resource in its terminal;
print	Yes – despite the fact of this operations requires another implementation, it is not because of visual elements, it is because it allows to print remotely, but the printing is made in the server, therefore the new implementation should in some way print the resource in the clients terminal;
runDIM	No/Yes – it depends on the DIM that this operation calls;
runJDIXO	No/Yes – it depends on the implementation made on the JDIXO; the JDIXO will be running in the server and if the implementation of the JDIXO

	contains any kind of GUI then it can cause the same problems of the DIBOs, if not, its execution is made without incidents;

Table 2 - List of DIBOs implemented in MPEG-21 DIP Reference Software that require new implementation

Chapter 5

5 DDIBrowser

In this chapter it is presented the DDIBrowser (Distributed Digital Item Browser) which is an application based on the work developed by Giorgiana Ciobanu in her master thesis [1]. First (in section 5.1) it will be presented the DDIBrowser on its first version developed by Giorgiana Ciobanu (called DIBrowser) and it is an application with a distributed architecture for viewing DIs. Then (in the 3 remaining sections) it will be presented the functionalities implemented and integrated in the DDIBrowser to execute DIP in a distributed way. DIBOs and DIXOs were intended to be executed. Following the web philosophy used in the development of the DIBrowser [1] and using its engine to browse Digital Items "step-by-step" presenting at each step the DIMs existing for the resources of the item that is being presented to the user. This will be achieved through the Object Type of the resource that is described in the item.

Next it will be presented the architecture and the result obtained by the User when using this first version of the DDIBrowser for viewing DIs.

5.1 - DDIBrowser first version

The DDIBrowser is the application that served as a source of requirements to this dissertation and that was used as a basis for integration of the work developed. The work developed by Giorgiana Ciobanu in [1] aimed to develop an application based on INESC DIDBrowser (a desktop application for consuming Digital Items) or changing it to create a WEB application within a client-server WEB architecture. The reasons that initially lead to the development of this application was that the INESC DIDBrowser had some limitations in the portability field where the Java that was used to develop the application isn't "compatible with all types of terminal devices" [1], and the portable devices have limitations in complex processing of Digital Items making impracticable the use of that browser in these devices.

This application provides a web environment enabling *Users* to consume DI's, the processing of the DI is made in a step-by-step manner instead of processing the entire DI at

once. This allows reducing the presentation time because resources not requested by the *User* are not processed. To create this application the author had to focus in three main aspects: the way the contents would be presented to *Users* (Web Philosophy), the way to process DI (Processing DI) and the modality of representing the parts of MPEG-21 (Object Model). These topics are discussed next.

Web Philosophy

Due to the great amount of information that a DID can have, the creation of a complex structure, can make the visualization of that information to be slower or limited. Therefore the author has chosen to present the information progressively step by step so that it can be accessible by any type of devices. This philosophy is based in a tree structure which allows to identify some similarities between the DID representation and a website. This similarity is shown in Figure 19.

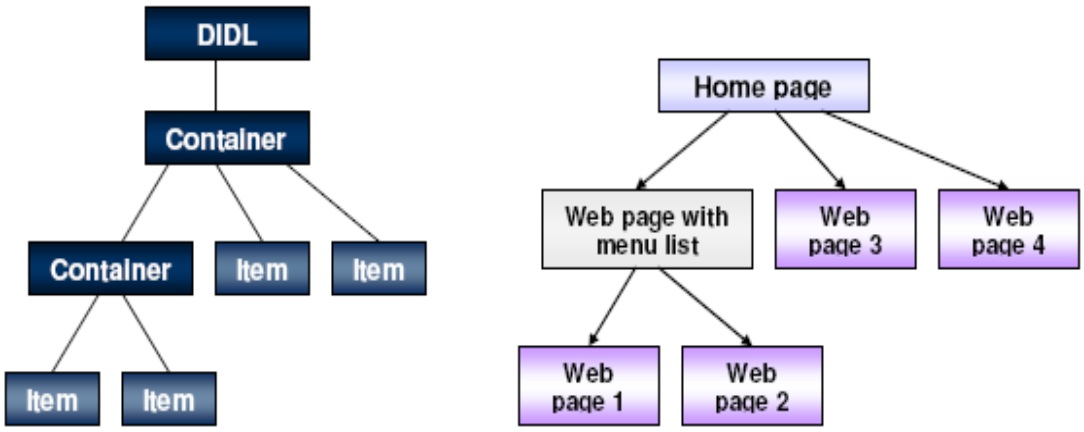


Figure 19 - Similarity between a DID representation and a Website (from [1])

For each element that the *User* is visualizing is presented a webpage with the element information and hyperlinks for the next elements in the hierarchical structure and for possible sub-elements. When elements contain selections an additional webpage is presented so that the *User* may choose its preferences after which they will be presented to the *User*.

Processing DI

Processing consists of preparing the chosen DI for browsing. To do so, the system simplifies the contents discarding the information that is not necessary (for example: low-level metadata intended for system processing or even elements that do not meet some conditions). The *Container* and *Item* are considered here the base elements for browsing. An example of the simplification of a DID by processing the base elements is shown in Figure 20.

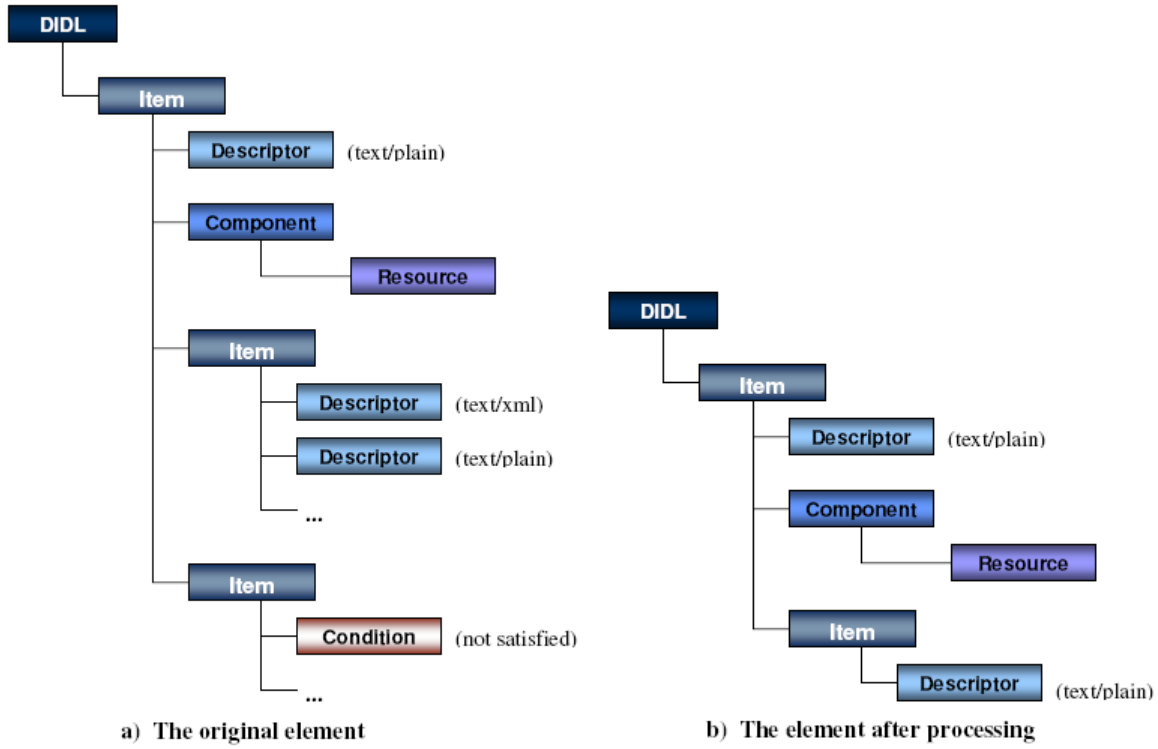


Figure 20 - Example of processing the base elements (from [1])

Object Model

Consists in creating an object representation for mapping the XML elements of the MPEG-21 parts to ensure a fast and easy access to the elements contributing for an improvement on overall performance, not forgetting the rules and constrains defined in the schema files that are defined in the standard.

5.1.1 Architecture

The architecture is composed of 3 main parts (represented in Figure 21), the User terminal that makes the content presentation and deal's with the *Users* interactions, the Server constituted by 2 components, the WDIBrowser (Web Digital Item Browser) which is the web services client that generates all the web pages to be visualized in the User Terminal and the web services server called IDIBrowser (INESC Digital Item Browser) which is the system that processes the DI (e.g. for checking conditions) and the repository where the DI are stored (that can be in the same machine than the server or in different machines).

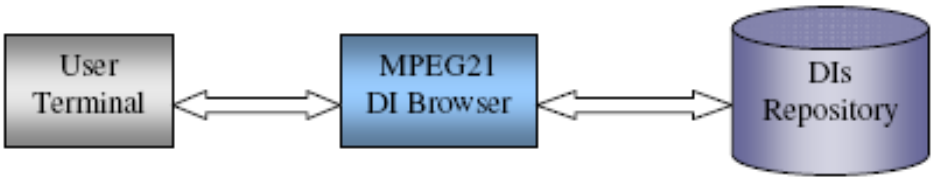


Figure 21 - MPEG-21 DDIBrowser (first version) components (from [1])

On the server side it was implemented a Web Service architecture to ensure interoperability and transparent data transfer. So, the API in the IDIBrowser supplies as Web Services the methods that make the bridge between the Client applications and the internal processing part. The communications between the Client and the Web Services running in the server are made through XML messages where the Client sends a message indicating the method to be executed and optionally some parameters. After executing the operation the server sends back to the Client the results of the executed operation also in XML messages. In Figure 22 it's shown the types of exchanged messages between the client and server applications with the respective *User* interactions.

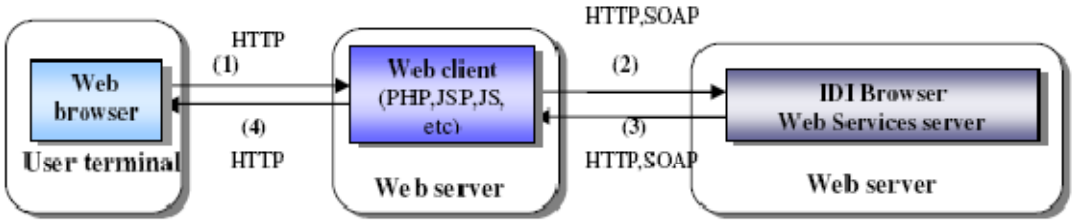


Figure 22 - The communication between the User terminal and MPEG-21 DDIBrowser (first version) components (from [1])

The messages exchanged by the User Terminal and the Web Server are the following:

(1) – using a Web Browser (e.g. Internet Explorer) the User accesses the Web client to request DI or to select an operation;

(2) – the client application requests the execution of the respective operation;

(3) – the Web Service sends the result after executing the operation;

(4) – the User visualizes the results in the web browser.

Figure 23 shows the general architecture with all the DDIBrowser components. These components are described next.

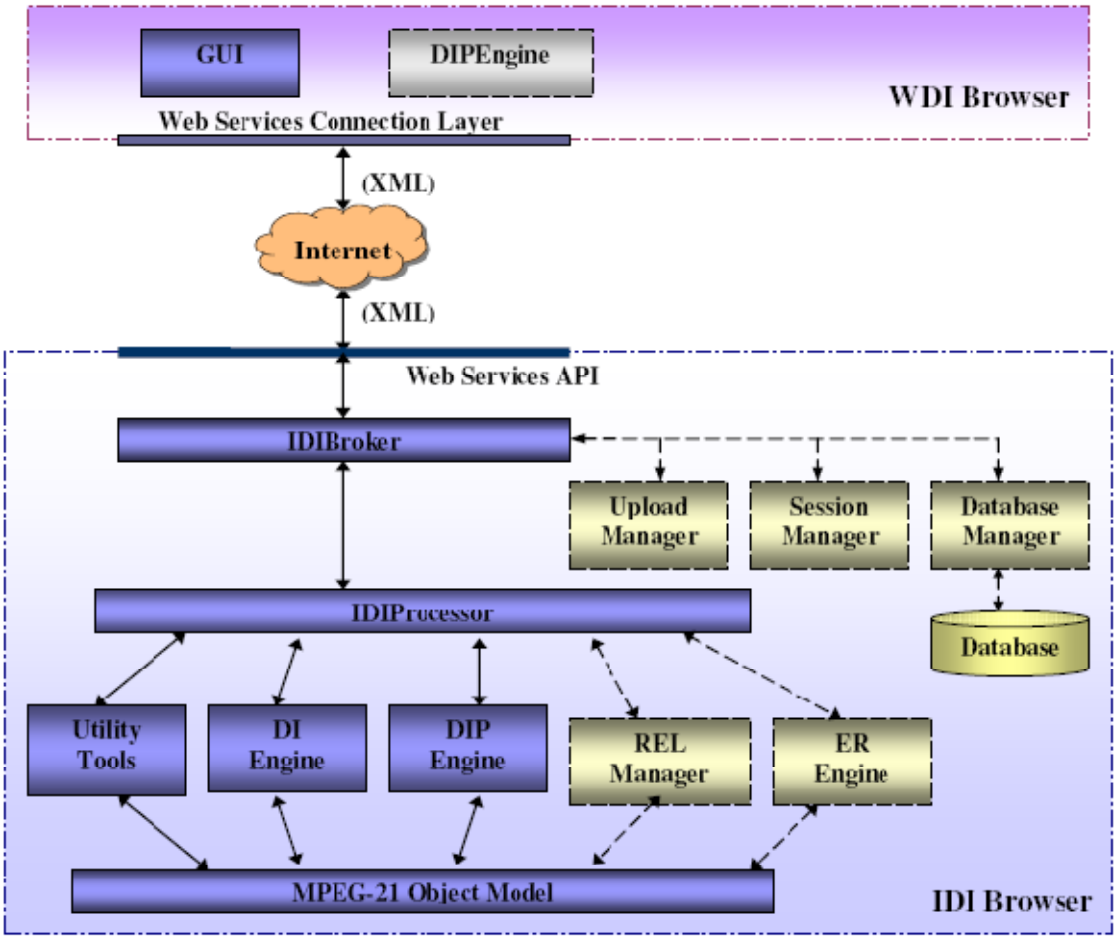


Figure 23 - General architecture of MPEG-21 DDIBrowser (first version) (from [1])

WDI Browser

Runs in the server terminal (but it is the Web Services client) to generate the graphical Web interface that allows the User to interact with the MPEG-21 DDIBrowser system through a web browser working in the User terminal; its sub-modules are:

- **GUI –** it's the graphical interface generator (generates web pages) made visible in the User's Web browser;
- **Web Services Connection Layer -** makes the connection to the server and requests the necessary operations for the Digital item presentation;

IDI Browser

It's the server, the main engine of the MPEG-21 DDIBrowser (first version) system and it is responsible for processing the DIs and providing the operations of the Web Service; its sub-modules are:

- **Web services API –** interface (or bridge) between the User actions and the internal processing; it defines the web services operation available to the WDI Browser [1];
- **IDIBroker – "**intermediary layer that ensures the communication between the various modules of the system" [1];
- **IDIProcessor –** implements the general processing of MPEG-21 DIBrowser;
- **DIEngine –** Dedicated to process DI at DID level simplifying it for a faster and easier browsing;
- **DIPEngine – "**essential for realizing the dynamic interaction of the User with the DI"; defines "methods (DIMs) available for a Digital Item" through the implementation of Digital Item Based operations (DIBOs); these DIMs are executed by the DIPEngine;
- **Utility tools –** its all the tools necessary for the MPEG-21 DI processing and other specific operations;
- **MPEG-21 Object Model –** maps the MPEG-21 elements into objects;
- **REL Manager –** makes the verification of the licenses included in the DID, sending a notification to the User to purchase a license (if needed) giving authorization to the consumption of the DI;
- **ER Engine –** implements part 15 – Event Reporting of MPEG-21 Standard [1];
- **Upload Manager Module –** useful for uploading DI for the repository from the Client and to download a DI to the User terminal;
- **Session Manager –** to control User sessions and their identification;
- **Database –** contains User sessions information, User licenses information and contains descriptions of *Items* in DI [1];
- **Database Manager –** "a transparent and flexible interface for accessing the database" [1];

These last 6 components, represented in light-grey in Figure 19, are the components designed to achieve the objectives of the extended requirements explained in [1].

5.1.2 Using WDI Browser

Here it will be shown the use of the WDI Browser, how to open a DI, browser *Items*, etc. The first the User needs to do, is to request a DI, and this is shown in Figure 24.

Figure 24 - Requesting a DI by entering its location (from [1])

After opening the DI the User will be able to navigate by seeing the content of the top element of the DI.

If the User wants to open an element/sub-element it needs to choose a Description shown in the Content Overview menu placed in the right side of the web page (Figure 25):

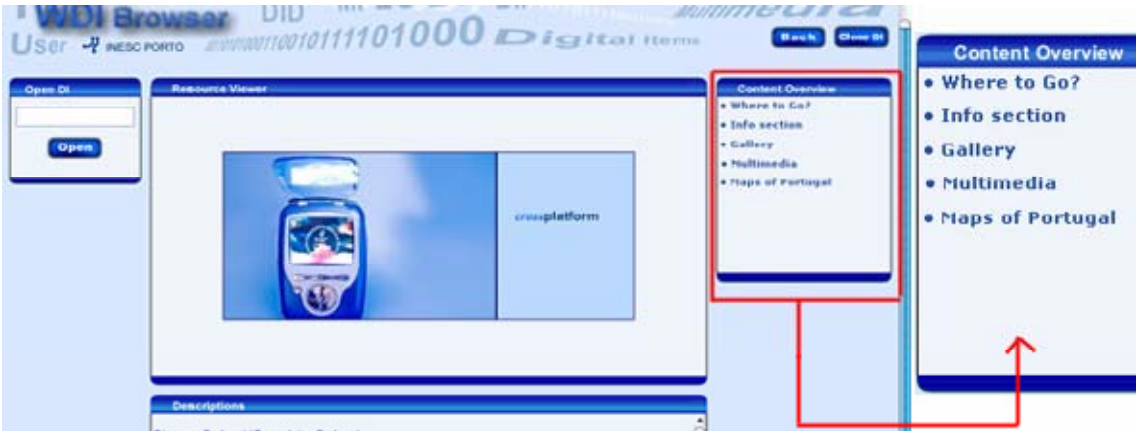


Figure 25 - Content Overview menu with the available sub-elements (from [1])

In the interface presented to the User, he will be able to see several types of resources, such as videos, audio, images, etc. Figure 26 shows these resources that are be presented in the Resource Viewer in the middle of the interface, long descriptions will be shown in a short version but giving the User the option of seeing the entire description, video and audio resources are played in an available media player (e.g. Quick Time Player).

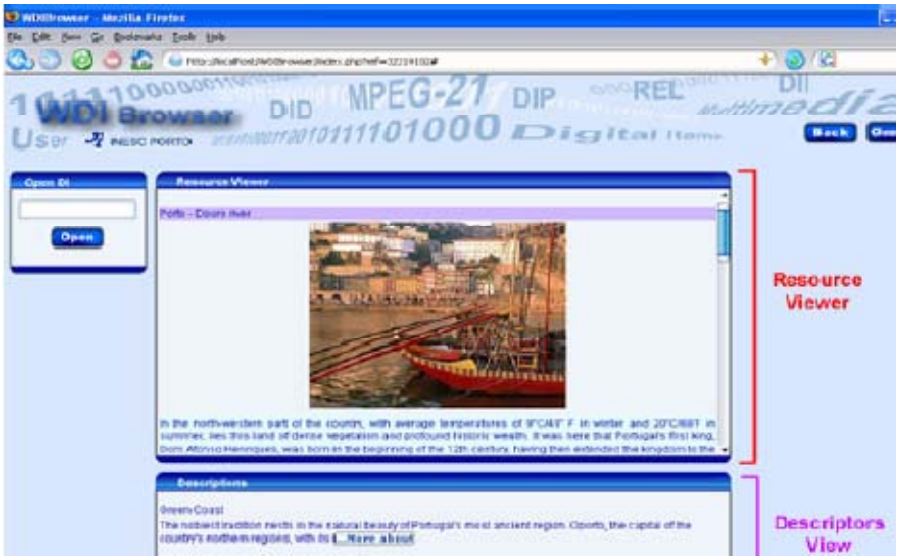


Figure 26 - Examples of Resource and Descriptors view existing in an element (from [1])

If an element contains choices then another window will be presented showing the possible choices for the User from which he will choose 1 and the elements related to it will be shown. A choice window is presented in Figure 27.

Figure 27 - Example of a Choice Window (from [1])

When an element contains DIP DIM methods they can be executed and the list is presented in the DIMs menu in the left side of the interface, after pressing execute the User is presented with a list of arguments that can be used in that method. This approach was based in the inclusion of the DIP functionality by means of an applet Web application implemented in the client side. The problem of this approach is that for the applet to work properly it is necessary to install in the User terminal a Java Virtual Machine and also its necessary to download the applet. These two aspects require a considerable processing capacity which can be prohibitive for slower machines. Therefore a new solution is required. The interface of the applet is presented in Figure 28.

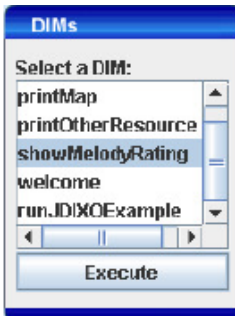


Figure 28 - Example of DIMs menu with some methods (from [1])

In addiction to these menus there is also a *Close DI* button placed in the right side of the interface and pressing it will clear the Contents View, the *Descriptors* view and the resources clearing also the DI data in the server side. After this only the *Open DI* and resource Viewer will stay visible.

Next to the *Close DI* button is the *Back* button that when pressed the previous element will be displayed to the User. It is recommended that the User uses this Back button instead of the Web Browser to ensure a correct browsing through the DI elements.

5.2 - System Requirements

It was necessary that the system could ensure the execution of DIMs. For this, the User only needed to select the DIM from a list presented to him, where this list can also be a specific list with only the DIMs that use a certain object type as an argument. Because these functionalities had to be integrated in the DDIBrowser developed in [1], the execution is required to be made in a distributed way, where the client will do the minimal processing needed.

The requirements identified for this system are:

1. It was required for the server, to return a list of all the DIMs existing in the DID that can be executed, to be shown to the User and enabling the User to execute a DIM if he chooses so;

2. The server had to execute DIMs returning the result;

3. The implementation should allow thin devices to execute DIP. It is because of the limited capabilities of these devices that the client has to be free of most of the processing;

4. Has to avoid the need of installing any software on the user terminal, making use of the players and other applications that already exist in the terminal for viewing the resources.

5.3 - System Architecture

Next an analysis on the system architecture and a description of the main modules that constitute the system is presented.

Considering, that the required functionalities are to be integrated in the DDIBrowser the functional architecture will be in some way derived from the DDIBrowser (first version) architecture. In fact only another entity had to be added to the system and it is called the *DIPEngineServer*. This entity provides the necessary methods to make possible DIP in a distributed way. The entities and its interaction are shown in Figure 29.

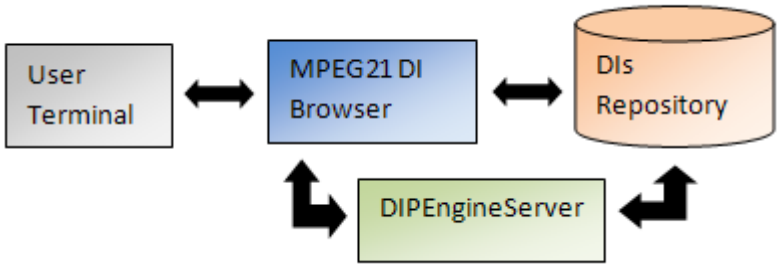


Figure 29 - DDIBrowser entities

The entities of the system are described in [1], but the purpose of the new entity is said next.

- *DIPEngineServer* - is where all the DIP Processing is made;

Keeping the DDIBrowser (first version) architecture and integrating the *DIPEngineServer* requires the communication between the *DIPEngineServer* and the Client (in this case, the WDIBrowser) to be made using Web Services, because of the advantage of using the HTTP protocol for data transport guaranteeing a maximum interoperability and an efficient data transfer [1]. This also increases the flexibility of this module because it enables its use in other clients built in different environments that can access Web Services. The purpose of the *DIPEngineServer* is to enable the processing of DIMs that are contained in the DID by the MPEG-21 DIP Reference Software and to be the interface between the MPEG-21 DIP Reference Software and the client through a series of functions developed as Web Services. These DIMs are constituted by JavaScript code where DIBOs are called. The DIBOs implementation and the processing made to the DIBOs to meet the requirements is made by the MPEG-21 DIP Reference Software that implements the DIBOs as specified in DIP (MPEG-21 Part 10).

The messages that will be exchanged between the several modules of the DDIBrowser are those shown in Figure 30.

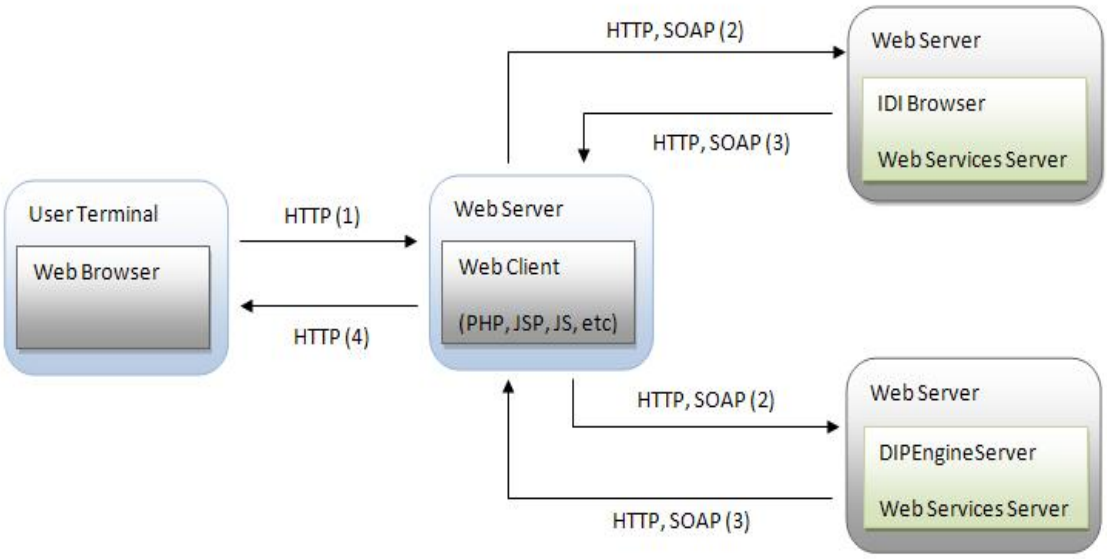


Figure 30 - Communication between the DDIBrowser components

The messages exchanged between the User Terminal and the Web Client and those exchanged between the Web Client and the IDI Browser, are explained in Figure 18. The messages exchanged between the Web Client and the DIPEngineServer have exactly the same objectives of those exchanged between the Web Client and the IDI Browser, and these objectives are:

(2) – the client application requests the execution of the respective operation;

(3) – the Web Service sends the result after executing the operation;

The contents of the messages are simple Strings (e.g. the DIM name: *myDIM*) that identify the DIM to be executed when requesting the execution of an operation and Strings with the result of the execution (e.g. the returned result of the execution of a DIM to obtain an object type name can be *urn.foo.LocalMovie*).

In Figure 31 is presented the general architecture of the *DIPEngineServer* including the module that was required to be implemented in the client (WDIBRowser) in order to present to the User the GUIs required by DIBOs that cant be remotely executed.

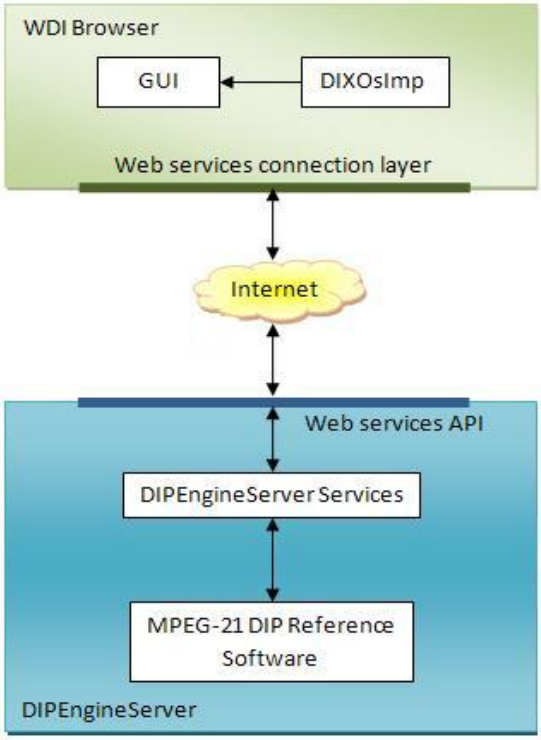


Figure 31 - *DIPEngineServer* **general architecture**

The components shown in Figure 31 are:

- **WDI Browser** – It is the client application (a series of web pages) that the User will access through a common web browser (e.g. Internet Explorer, Mozilla Firefox, etc);
    - **GUI** – is responsible for creating the web pages that the User will see and was implemented in the first version of DDIBrowser;
    - **DIXOsImp** – is responsible for the interpretation of the messages returned from the execution of the DIM, and generates the Visual (or GUI) implementation for those DIBOs that have to be partially or totally implemented in the client (DIBOs with visual elements);
    - **Web services connection layer** – is responsible for the communication with the server for the execution of DIMs;
- *DIPEngineServer* – is responsible for the DIP processing (execution of DIMs)
    - **Web Services API** – represents the interface between the User actions and the internal processing part of the application; it defines the Web Services operations available to the WDI Browser;

- *DIPEngineServer* **Services** – represents the implementation of the provided web services; it processes the requests and uses the MPEG-21 DIP Reference Software functions to obtain data for the execution of DIMs;
- **MPEG-21 DIP Reference Software** – implements the necessary methods to execute DIMs and DIBOs.

5.4 - UML Specification of *DIPEngineServer*

Next, UML diagrams [20] are presented. These diagrams describe the User's interaction with the system functionalities associated to DIP and the interaction that the Client has with the *DIPEngineServer*. First, Use Case diagrams will be presented. In Figure 32, is shown the Use Cases that represent the User interactions with the system, more specifically, represents the interactions between the User and the client application.

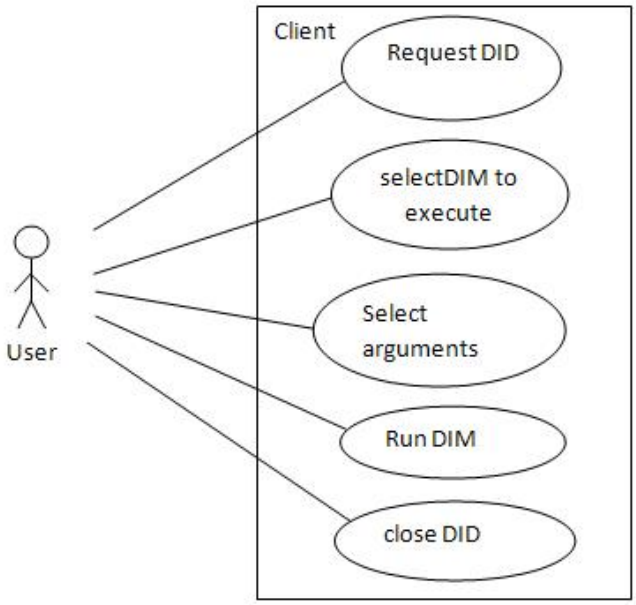


Figure 32 – Use Case Diagram of the interactions that the User has with the system for processing DIP

The description of the use cases in Figure 32 is:

- **Request DID –** this Use Case is common to the *Request DID* specified in [1]; when opening the WDIBrowser in the User terminal, the User will provide to the client (WDIBrowser) the path to the DID stored in a remote repository and as response the User will be able to see the top element of the DID;
- **Select DIM to execute –** when a User browses an *Item* which contains DIMs that use as an argument the object type of this *Item,* he can select a DIM to be executed;

- **Select Arguments –** the User is viewing the DID top element and decides to run a selected DID; he can choose an element of the same type of the argument used by the DIM;

- **Run Dim –** the User has chosen a DIM to execute and decides to execute it and the response will be the result of the execution of the DIM;

- **Close DID –** after browsing and executing DIMs the User decides to close the DID, in this case the DID will be cleared from the system engines.

Considering that the client (WDIBrowser) works as an intermediary between the User terminal and the server, he will be considered as an actor in the next use cases diagram (Figure 33), that present the interactions between the client and the server for the processing of DIMs (showing the lists of available DIMs and executing DIMs).

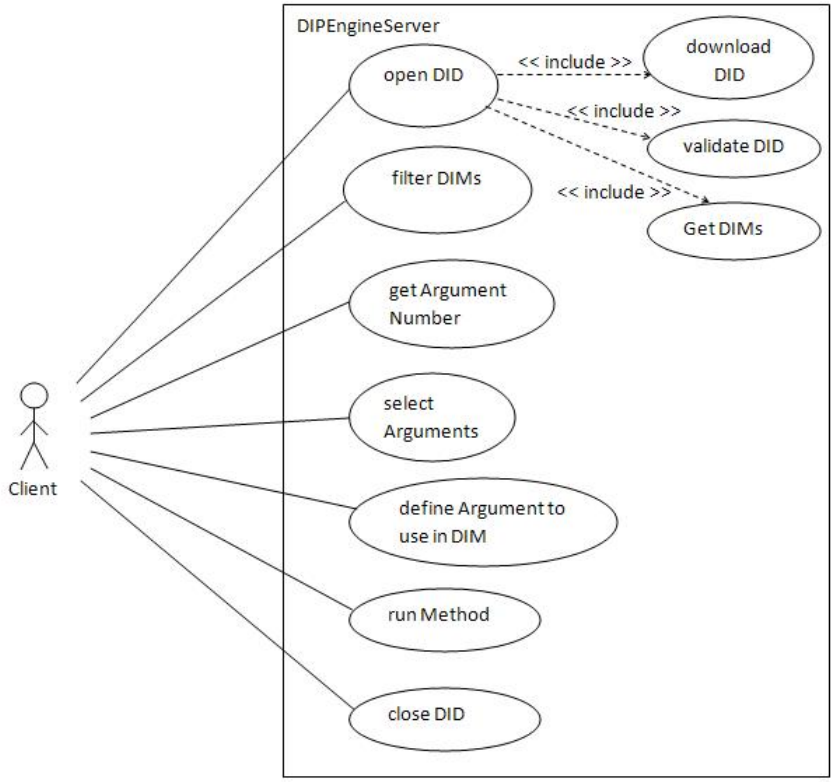


Figure 33 - Use Case Diagram of the interaction of the Client (WDIBrowser) with the *DIPEngineServer* (DIP web services server)

The description of the use cases of Figure 33 is:

- **Open DID –** this Use case was already specified and implemented in [1] but it is used in this work to retrieve the URL of the DID, therefore is referenced here;

- **Filter DIMs –** when browsing a certain *Item* the User will be able to see DIMs that use the object type defined in that *Item*; to achieve this, the client requests to the server to filter the DIMs by this object type; the response list of DIMs that use that object type;

- **Get argument number –** the client requests the number of arguments to the server, so that he determine if the object type of the item is equal to one or all the arguments in the DIM;

- **Select arguments –** the client requests a list of all the *Items* that have the same object type than the *Item* the User is browsing; this way the client can assign the Item the User is browsing to be the one for which the DIM will be executed; this way, the DIM execution will be regarding to the item the User is browsing and not for the full DID;

- **Define argument to use in DIM –** after receiving the list of the *Items* that use that object type, the client will compare the *Items* description with the description of the arguments and will set as the argument for the DIMs execution the corresponding to the *Item* the User is browsing;

- **Run method –** when a User decides to run a DIM, and the client has defined the argument to be used, it calls the service to run the DIM receiving as result, the result of the execution of the DIM;

- **Close DID –** when the User decides to stop browsing the DID, the client will clear the DID from the server, stopping the engines of the MPEG-21 DIP Reference Software.

After defining the use cases, their behavior, the actions that the actor (User or WDIBrowser) can perform must be defined.

In Figure 34 and 35 it is shown the activity diagrams that describe the activities flux for the actions that can be carried out by the actors, which are the User interaction with the client and the client interaction with the server (respectively).

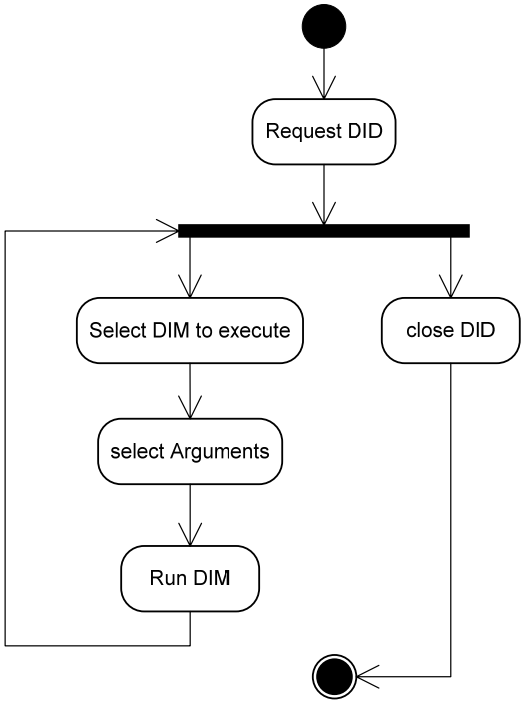


Figure 34 - Activity Diagram of User interaction with the client

This activity diagram shows that the User initially provides to the client the URL for the DIM receiving as response the list of DIMs existing in the DID. Afterward, when browsing it is possible to close the DID or it is possible to select a DIM from a filtered list of DIMs that use the object type of the browsed item as an argument and then execute that DIM. If the DIM to be executed is selected in the top element it is necessary for the User to specify the argument for that DIM to execute.

Next, Figure 35 shows the activity diagram that represents the client interaction with the server. Here the client after receiving the URL, requests the server to open the DID returning to the client the list of DIMs. If browsing an *Item* the client requests the server a list of DIMs that use the object type of that *Item* as an argument in the DIM, and then if the User chooses so, it requests the server to execute the DIM. Plus when the execution is requested, the client automatically sets that *Item* (or element) as the argument for the DIMs execution, after which the server returns to the client the result of the execution.

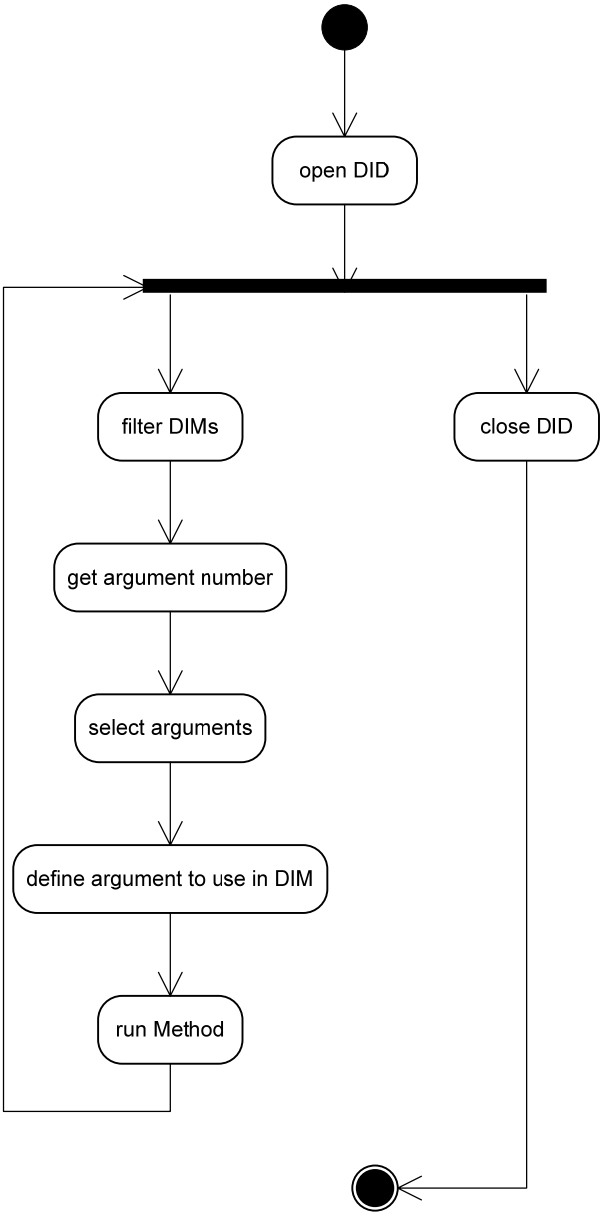


Figure 35 - Activity diagram of the client interaction with the server

Next, the sequence of messages and interactions that the User terminal has with the Client and that the Client has with the server will be presented through a sequence diagram. The Sequence Diagram depicted in Figure 36 shows all the steps made in order to execute a DIM between the User Terminal, the Client and the Web Services Server.

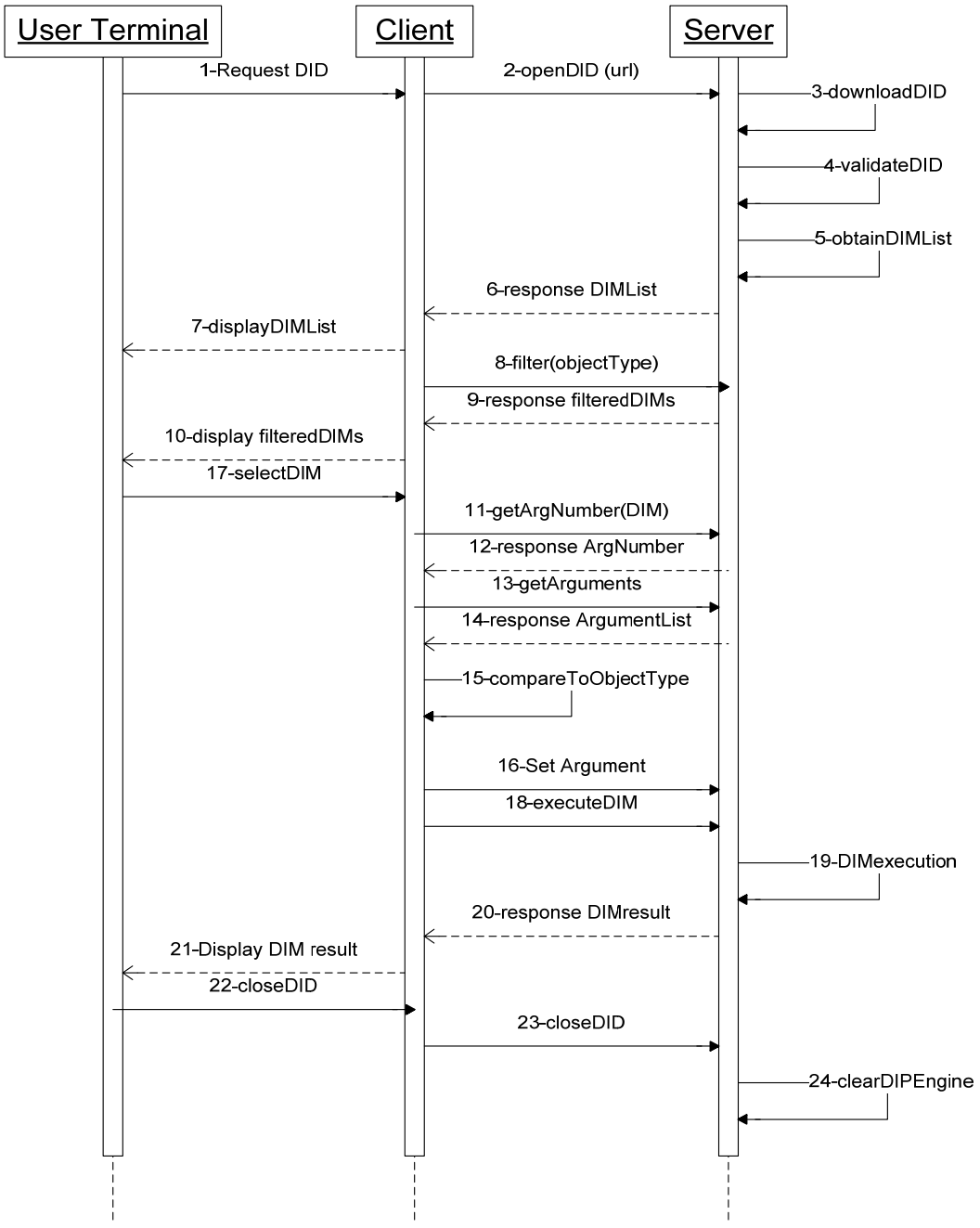


Figure 36 - Sequence diagram with the entities interactions

Chapter 6

6 DDIBrowser Implementation

This chapter presents the implementation of the DDIBrowser. The implemented elements were the *DIPEngineServer*, the necessary modifications made to the MPEG-21 DIP Reference Software, the implementation of two DIBOs (the *alert* and *play*) made through JDIXOs and changes made to the DDIBrowser (first version) to enable the execution of DIMs and JDIXOs in a distributed manner.

6.1 - DIPEngineServer

This is the class where all the services are implemented to allow distributed DIP. This class is constituted by several functions that can be called by the client, but it also has functions for internal processing in the service execution. First, the functions available for the client are presented, followed by the functions for internal processing, and by the integration in the client.

The functions existing in the *DIPEngineServer* available for the client (public functions) are:

- **startDIP** – this function receives as an input the location of the DID and the output will be a list of all the DIMs existing in the DID; to achieve this output the necessary engines existing in the MPEG-21 DIP Reference Software have to be initiated and then the list will be retrieved. The elements initiated in the MPEG-21 DIP Reference Software have the purpose of converting the DID into a DOM document and validating the DID (DID Engine), to execute DIP operations providing an implementation for DIBOs and one possible implementation for the execution of JDIXOs (DIP Engine) and to execute DIMs (DIM Engine). Figure 34 shows the function/elements used and initiated by this function.

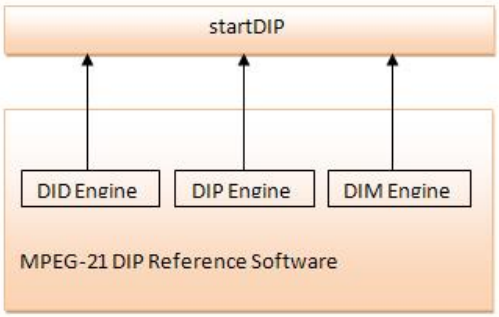


Figure 37 - MPEG-21 DIP Reference Software engines used in *DIPEngineServer* and initiated in startDIP method

Next, in Figure 38, it is presented the flowchart of the startDIP method.

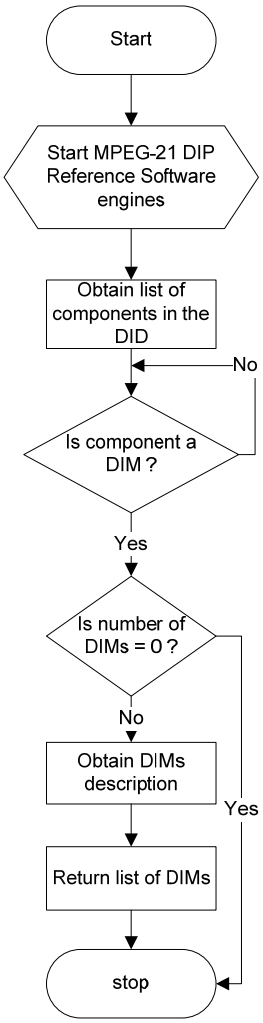


Figure 38 - Flowchart of StartDIP method

- **filter** – receives as an input a string with the object type from which the DIMs will be filtered and gives as output a list of DIMs that use that object type as an argument in its execution; this function will use the list of DIMs existing in the DID and then will use the function *MyArguments* to get the DIMs that contain arguments and then it will compare this argument to the object type received as input and if they are equal the name of that DIM will be return. The flowchart that expresses this method behavior is presented in Figure 39.

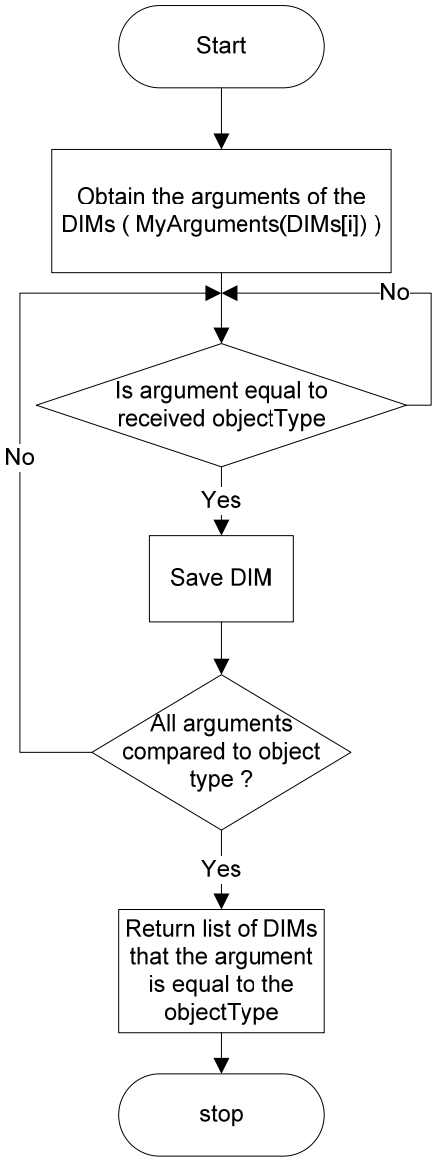


Figure 39 - filter method flowchart

- **getArgNumber** – this function receives as input the name of a DIM and return as output the number of arguments that, that DIM uses; Figure 40 shows this method flowchart.

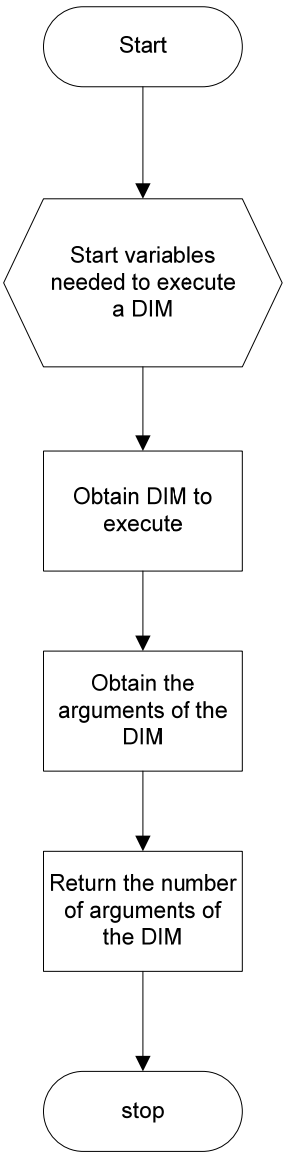


Figure 40 - getArgNumber method flowchart

- **getArguments** – the input of this function is the argument index and the output is the description of available items with that argument as object type; if there's no arguments this function doesn't need to be called, but if there are one or more arguments then this function is called (the number of times that the function *getArgNumber* returns) to retrieve the items for each argument, and the implementation for the item selection is left for the client implementer. The flowchart representing the behavior of the method is presented in Figure 41.

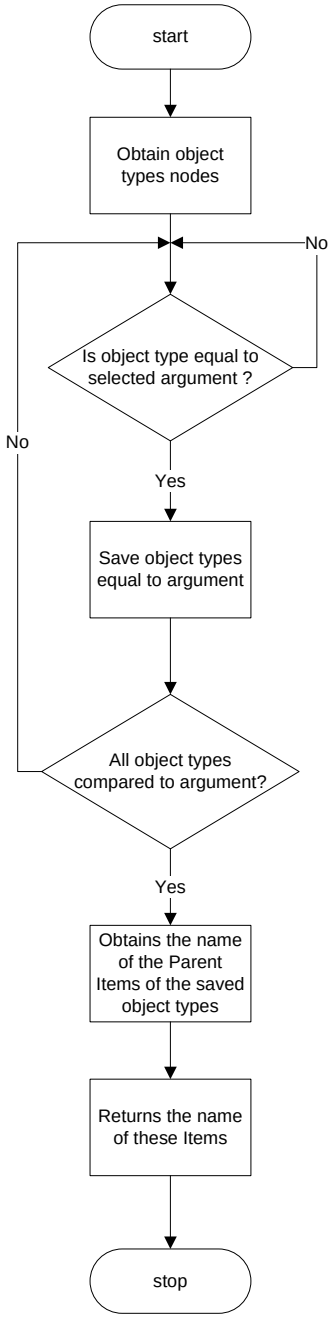


Figure 41 - getArgument method flowchart

- **setArgument** – it has as an input the index of the item in the list returned by the function *getArguments* and has no output; it simply sets the selected item as the item for which the DIM will execute. The flowchart is presented in Figure 42.

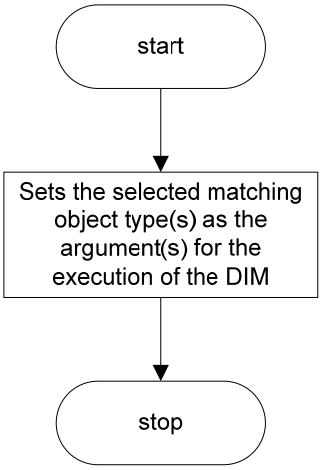


Figure 42 - setArgument method flowchart

- **closeDID** – it has no input or output and it stops the MPEG-21 DIP Reference Software engines (DID Engine, DIP Engine and DIM Engine); this method flowchart is presented in Figure 43.

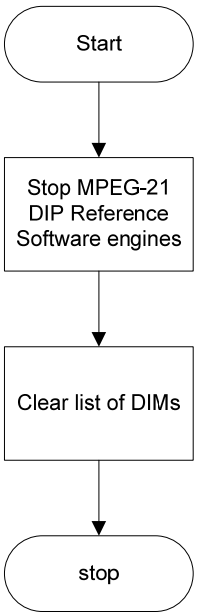


Figure 43 - closeDID method flowchart

- **runMethod** – this function has no input and the output will be the result of the execution of the DIM; this function is responsible for obtaining the necessary items, and it uses the DIM Engine to execute the DIM. The steps made in this method are presented in the flowchart of Figure 44.

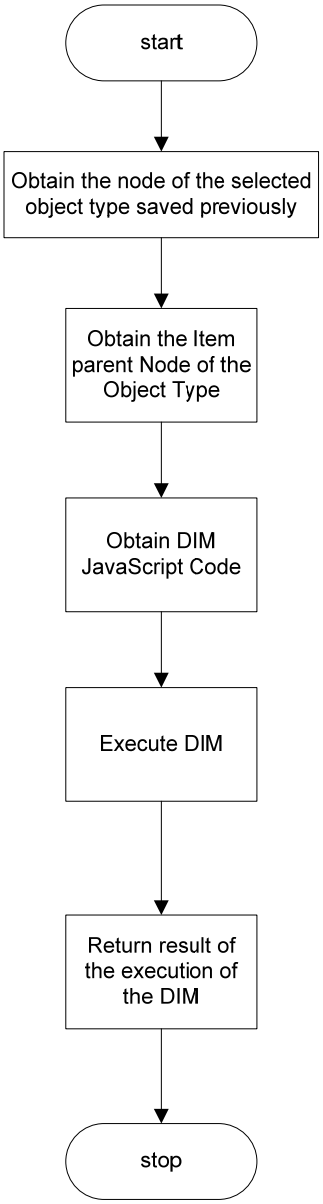


Figure 44 - runMethod method flowchart

- **getDIDs** - this function receives as an input the URL to a repository and returns as output a list of DID; it will obtain a list of the files with the extension ".xml" or ".XML" (the listing is case sensitive) existing in that repository and will obtain from the DIDs the data contained in a new metadata created to allow the User to give a title and a small description for the DID. The flowchart of this method is presented in Figure 45.

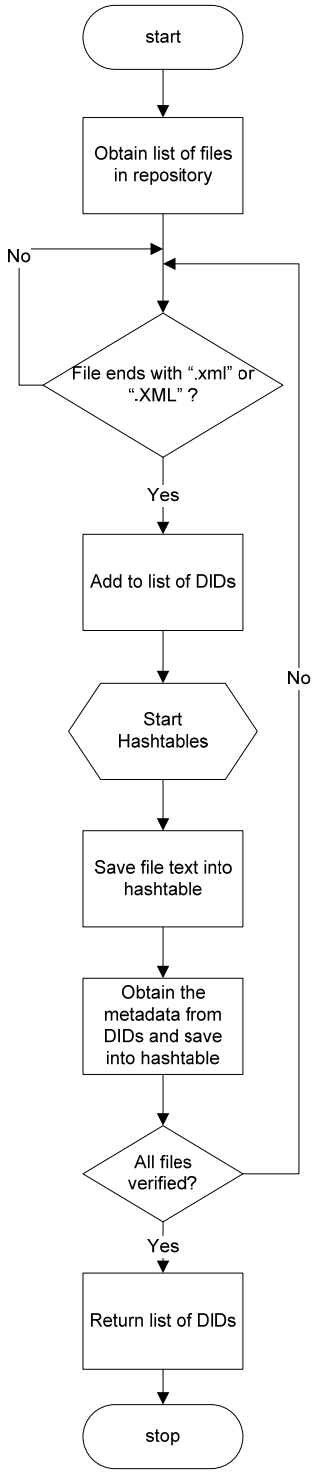


Figure 45 - getDIDs method flowchart

- **getDidTitle** – this function receives as an input the name of the DID as gives as an output the title metadata retrieved in the function *getDIDs*. Figure 46 shows the method flowchart.

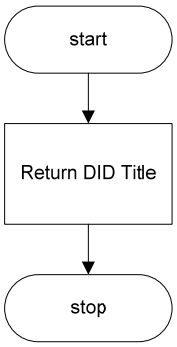


Figure 46 - returnDidTitle method flowchart

- **getDidDescription** - similar to the previous function receives the name of the DID as an input and gives as an output the description metadata, also obtained in the function *getDIDs*. Figure 47 shows the method flowchart.

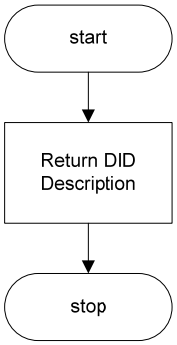


Figure 47 - getDidDescription method flowchart

- **getFullDid** – this function receives as an input the DID name and gives as an output a string with all the content of the file allowing for example to save the file in the client terminal. Figure 48 shows the method flowchart.

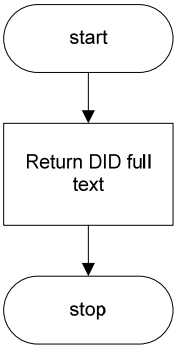


Figure 48 - getFullDid method flowchart

These three last methods presented above were built only to assist in knowing what the example DID created contains and needs the method getDids to be executed first. Next, the functions existing for internal processing (private functions) are:

- **containsMethodInfo** – this function is used in the *MyArguments* function and the *getArgNumber* function and receives as an input a *DOM Node* which is a component in the DID, and gives as an output a Boolean value; the function will determine if the *Component* has an argument. If so it returns true, if not, it returns false. This behavior is shown through a flowchart presented in Figure 49.

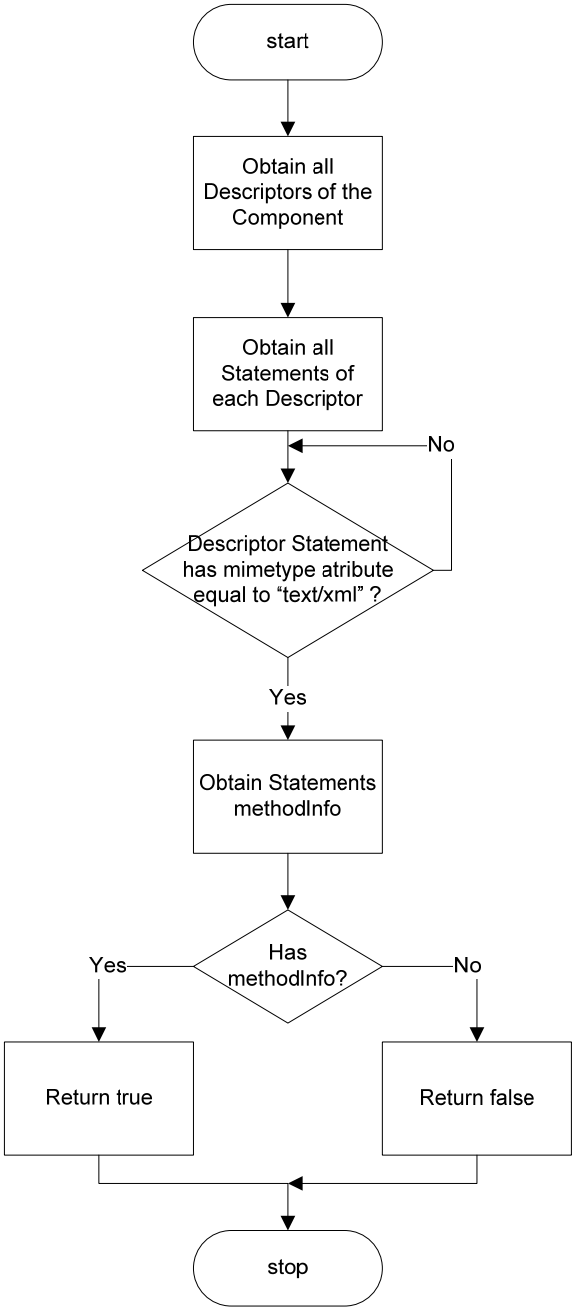


Figure 49 - containsMethodInfo private method flowchart

- **MyArguments** – this function is used in the *filter* function and receives a DOM Node as an input and the output will be the DOM Nodes of the arguments existing in a DIM. Figure 50 shows the flowchart with its behavior.

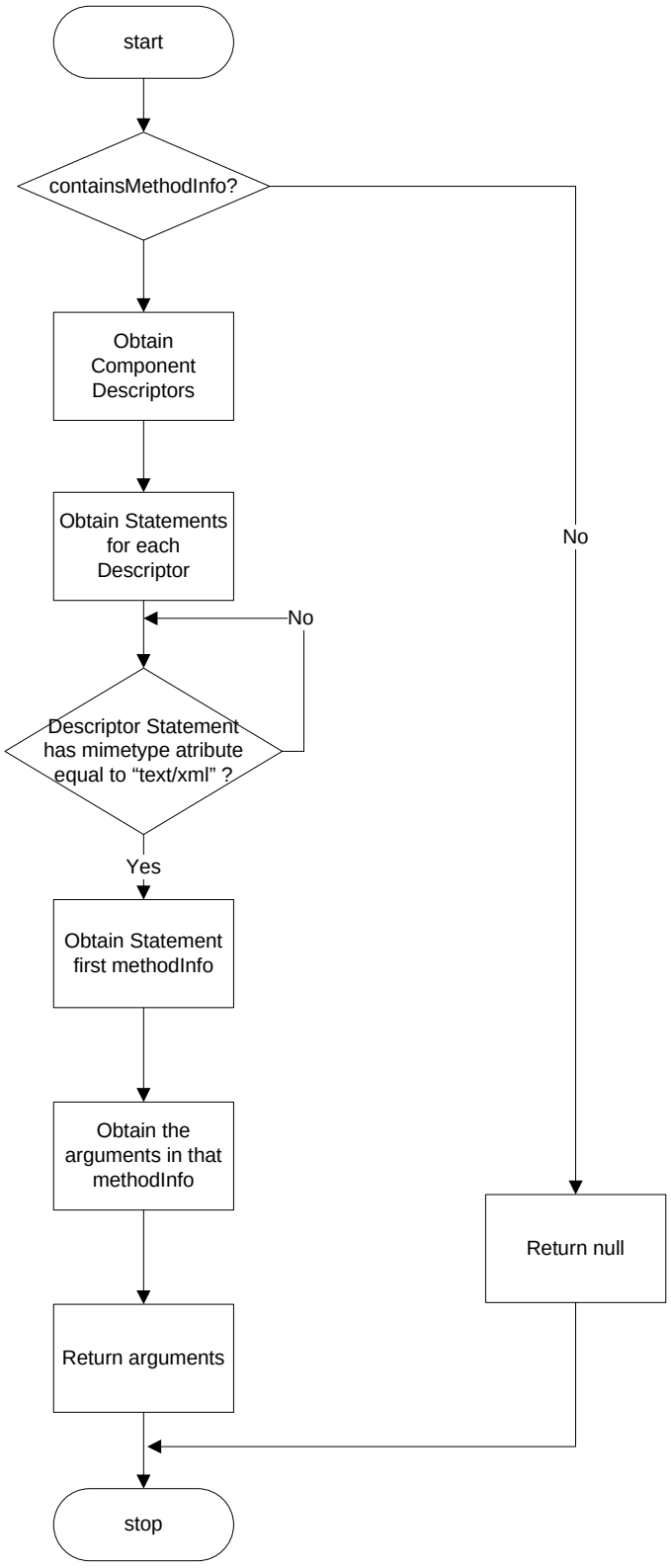


Figure 50 - MyArguments private method flowchart

6.2 - MPEG-21 DIP Reference Software Modifications

In order for the system to work, changes to the behavior of the MPEG-21 DIP Reference Software had to be made. The first one was to make the software able to return the result of the execution of the DIM. To do this the *interpret* function header, which is the function that interprets the JavaScript code that implements the DIM, existing in the *DIMEngine class* of the software had to be changed from:

```
/**
* Interprets a DIM.
*
* @param method
*          The textual representation of the DIM.
* @param args
*          The arguments for the DIM.
*/
          public void interpret(String method, Object[] args) {…}
```

To:

```
/**
* Interprets a DIM.
*
* @param method
*          The textual representation of the DIM.
* @param args
*          The arguments for the DIM.
*/
          public String interpret(String method, Object[] args) {…}
```

Plus the line "*return Context.toString(result);*" was added to the function.

```
public String interpret(String method, Object[] args)
{
          …
          Object result = f.call(cx, scope, scope, args);
          return Context.toString(result);
}
```

Another problem that had to be solved was the fact that the original implementation made to execute JDIXOs was unable to execute JDIXOs when the DIM execution was made remotely. Despite the research made, that included some e-mail exchanges with one of the authors of the MPEG-21 DIP Reference Software trying to understand what the problem was no explanation for this problem was found. The solution was to change the line "*Class dynamicClass = defineClass(null, classData, 0, classData.length, null);*" to "*Class dynamicClass = Class.forName(className);*"

The solution adopted here for these DIBOs, that require a new implementation was to create JDIXOs to avoid changing the DIBOs implementation in the MPEG-21 DIP Reference Software. Therefore two of these DIBOs were implemented as JDIXOs. For example the new implementation of the *alert* DIBO could be made by receiving from the DIM (by calling the *runJDIXO* DIBO) the message and the message type, and then it would return a specific message that the client has to interpret and present to the User. With this method the visual implementation of the DIBO is implemented in the client and according to the type of machine where the client is supposed to run. Therefore two of these DIBOs were implemented as JDIXOs to show that it is possible to execute DIP remotely. The DIBOs implemented were the *alert* DIBO and the *play* DIBO. This solution can also be used in the implementation of the other DIBOs that use visual elements.

The implementation of a JDIXO must follow a defined structure where it is necessary to define the name of the constructor, define the arguments that the JDIXO will receive in the JDIXO method *getArgumentTypes()* and define the return types in the method *getReturnType()*. After this it is necessary to write the implementation of the JDIXO in the method *callJDIXO*.

6.2.1 alert DIXO

This operation is expected to receive a message to present to the User and a type of message (it can be an info message, a warning, an error or a plain message).

- Problem: The problem here is how to present the message box to the User if the web services aren't able to remotely execute visual classes or simple visual elements.

- Solution: For the JDIXO, and because it has to obey to the structure presented above, the argument that it will receive is a String. The return will be XML String. The argument contains the message and the message type separated by a special character because, the interface for the JDIXO is prepared to receive one argument, this way the message and the message type can be sent in the same argument. The JDIXO objective is to receive this argument and construct a XML string that the client will receive and interpret the data in the String.

The XML String message will have the following structure:

```
<DIBO>
        <className>alertDIXO</className>
        <msg>the message specified in the DID</msg>
        <msgType>the message type specified in the DID</msgType>
</DIBO>
```

This XML String presented above will be the String that the JDIXO will return. It will be up to the client implementer to decide what to do with this message. For means of test the implementation will do the same thing that the *alert* DIBO implementation does.

- Limitation: The limitation of this solution is that is needed that the calling of this JDIXO has to be made in the end of the DIM, on other words, the result of calling the JDIXO should be the final result of the DIM execution, nothing more should be implemented in the DIM after calling the JDIXO, except if the User that writes the DIM decides to create the implementation embedded in the DIM. For example if the DIM would be to present to the User the object type count, the DIM could obtain this value through the operation *getObjectTypeCount* and then it would call the JDIXO and pass that count as the message, then the client would present a message box with that object type count.

6.2.2 play DIXO

This operation has the purpose of playing resources for the User. The implementation on the DIBO simply plays the resource returning the play status.

- Problem: The problem is how to play the resource in the client when this implementation only allows playing the resource locally.

- Solution: The most simple solution was to develop a JDIXO that will do almost everything that the DIBO does, but instead of obtaining the resource URL and reproduce the resource in a created implementation based in Quicktime, it will only obtain the URL and then returns a XML String that will contain the URL for the resource, allowing to play the resource according to the players available in the User terminal, not requiring the client implementer to create an implementation .Like in the *alert* this implementation must obey to the same rules, therefore the argument that the JDIXO will receive will be a DOM Element and after obtaining the URL the JDIXO will return a XML String with the following format:

```
<DIBO>
        <className>playDIXO</className>
        <url>the url for the resource</url>
</DIBO>
```

After receiving this XML String it will be up to the client to decide what to do with this URL, the implementation made follows the same idea of the DIBO and the resource is to be played in the User terminal and the playback.

- Limitation: This solution has the same limitation of the *alert* DIXO presented above which is the fact that this JDIXO should be the last thing to be executed in the DIM, in order to be able to return the XML String to the client.

These solutions may have limitations if trying to implement other operations (e.g. *configureChoice*) because after returning to the client and presenting the choices to the User allowing him to change that choices, it would require that the client would call another service, that should be created to set the new values to those choices. All these DIBOs implemented through this method would have the limitation exposed above in the *alert* and *play* DIXO which obligates these JDIXOs to be the last line in the DIM in the return enabling the DIM to return the XML String returned by the JDIXOs.

6.3 - Integration in the WDIBrowser

The integration in the WDIBrowser developed in [1], consists in creating the necessary modules to make use of the *DIPEngineServer* services to execute DIMs. The integration was divided in three phases that were:

- First to make the WDIBrowser to present to the User a list of DIMs provided by the *DIPEngineServer*;
- Secondly it would be to guaranty that when browsing an item, the WDIBrowser presents the DIMs that use that *Item* object type as an argument;
- And, for last, to guarantee that it is possible to execute the DIMs, including the newly created JDIXOs by creating the module that would receive the XML String to then present to the User the result.

In Figure 51 is shown the architecture of the WDIBrowser.

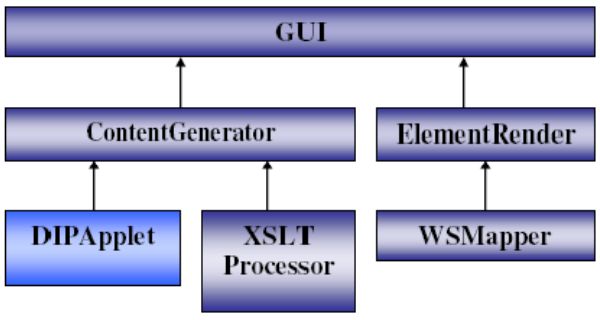


Figure 51 - Architecture of WDIBrowser client (from [1])

The relevant sections of the WDIBrowser and the modifications made on them, are described next:

- **WSMapper** - it is the lower module of the WDI Browser that directly accesses the published WSDL description of the IDI Browser's Web services [1] and here, under the rate of this work a new class to work as the SOAP client for the *DIPEngineServer* Web Services was created;

- **ContentGenerator** – this module is composed by several .PHP files which create the HTML views of the essential parts of the WDI Browser interface when showing the *Descriptors* information, displaying the *Resources*, or containing the HTML forms corresponding to the Back and Close buttons, or the *Choices* window, etc. These files use the method provided by the XSLTProcessor for transforming the XML elements into HTML [1], and here three steps were followed to change this module. First a new PHP file that would obtain the list of DIMs existing in the DID was created. Secondly another PHP file that would obtain the list of DIMs that use the object type of the item being browsed as an argument (the filtered DIMs) and it would present that list to the User. For last, two new PHP files were created, one that would send the request for executing DIMs and receiving its result and another that would implement the necessary methods for successfully viewing the result of the *alert* JDIXO and *play* JDIXO.

- **DIPApplet** – this applet runs the MPEG-21 DIP Reference Software and was used like this to be easily included in the WDIBrowser. In the DDIBrowser this part was removed from the client so that the *DIPEngineServer* could be included in its place.

The description of the rest of the modules that weren't changed can be seen in [1].

6.4 - Results

Next, the results of the implementation will be presented showing how the User can select DIMs and execute them. The results are similar to those presented in section 5.1.2, but now these results are regarding to DIP. It will be shown the obtained list of DIMs, the filtered list of DIMs and the execution of one DIBO that didn't require a new implementation and the execution of a DIBO that was implemented by means of a JDIXO (in this case it will be the *alert* and the *play*).

6.4.1 Opening the DID

This method was implemented in the first version of the DDIBrowser but it was used to obtain the URL of the DID. In Figure 52 shows the "DIMs" view after opening the DID and the list of existing DIMs in the DID.

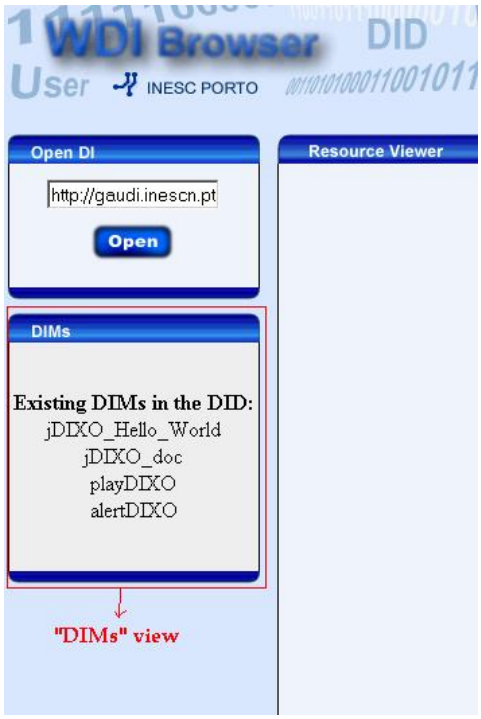


Figure 52 - DIMs view. List of existing DIMs in the DID

What is seen in the DIMs view is the list of existing DIMs in the DID (the DIMs name and not a description), which are: "jDIXO_Hello_World", "jDIXO_doc", "playDIXO" and "alertDIXO".

6.4.2 Filtering DIMs

To filter DIMs it is necessary that the User browses an *Item*, by selecting an *Item* in the "Content Overview" view. Next, Figure 53, shows a list of filtered DIMs when browsing an item. The *Item* is chosen in the "Content Overview" view in the right side of the page, like shown in section 5.1.2. After choosing the *Item*, the client will request for the *DIPEngineServer* to filter the list of DIMs.

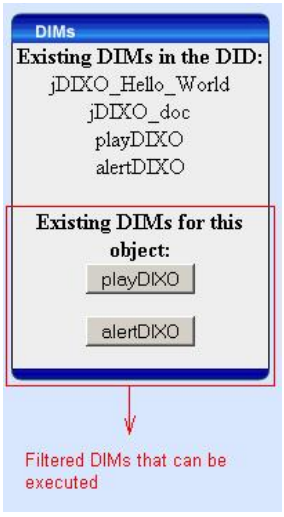


Figure 53 - List of DIMs that use the Item object type as an argument

6.4.3 Executing DIMs

After filtering the DIMs it is possible to execute DIMs that are related to the *Item* being browsed through the object type. To achieve this, the User should choose a DIM to execute and then request its execution, then the client will automatically set the DIM argument as the *Item* being browsed and then will request the execution of the DIM by calling the *runMethod* operation of the *DIPEngineServer*. Figure 54 presents the list of DIMs existing in the DID and those that can be executed for the *Item* being browsed.

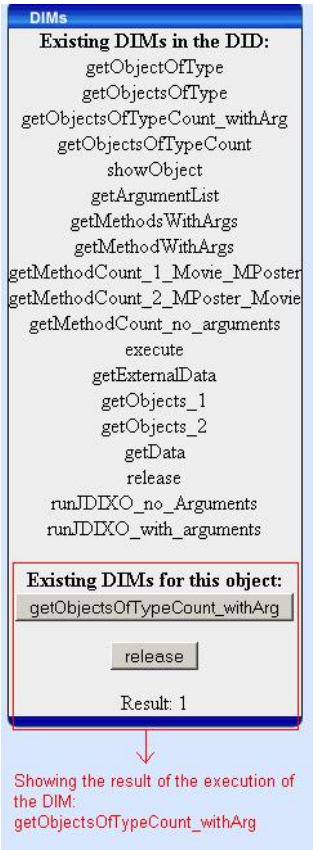


Figure 54 - Shows the result of executing the DIM getObjectsOfTypeCount_withArg

The DIM *getObjectsOfTypeCount_withArg* simply returns the number of *Items* existing in the DID with a certain *ObjectType*.

Considering now the specific alert DIBO and the play DIBO that were implemented in two JDIXOs lets see the result of their execution. Figure 55 shows the result of executing the *alert* JDIXO in a DIM. This DIM is supposed to present to the User an informative message with the first object type existing in the DID.

Figure 55 - Result of the execution of the *alertDIXO*

Notice that another button appeared after the execution, this button is part of the alert implementation, and now the User should press that button to see the message. If he chooses so, the message is presented in the "Resourve View", shown in Figure 56, if not he can continue browsing, or even execute another DIM.

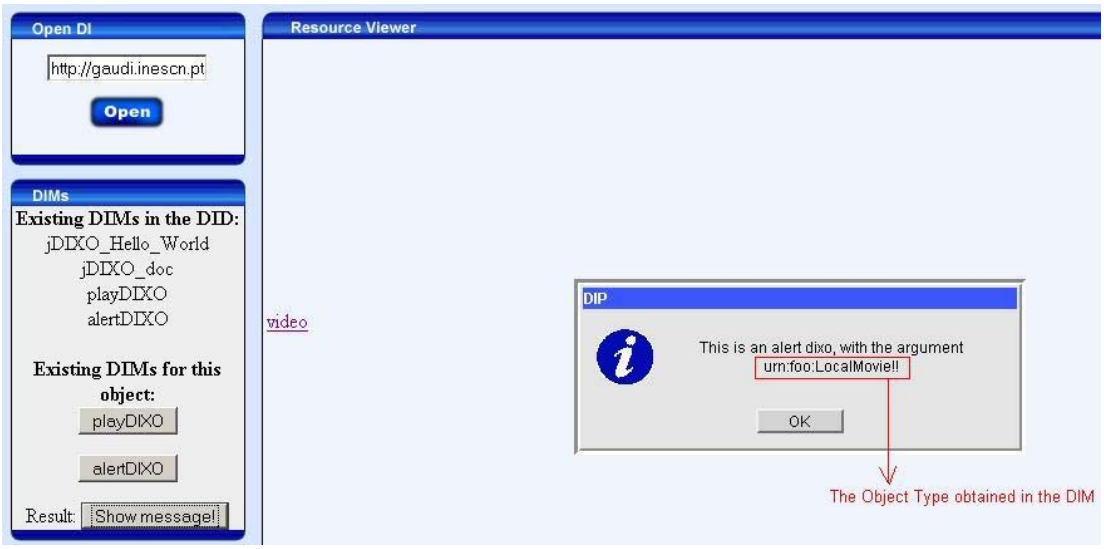


Figure 56 - Message Box presented to the User if he decides to see the message resulting in the DIM execution

In Figure 57 it is shown the result of the execution of the *play* DIXO. After pressing the button to execute the DIM the resource will be presented in the "Resource Viewer" view.

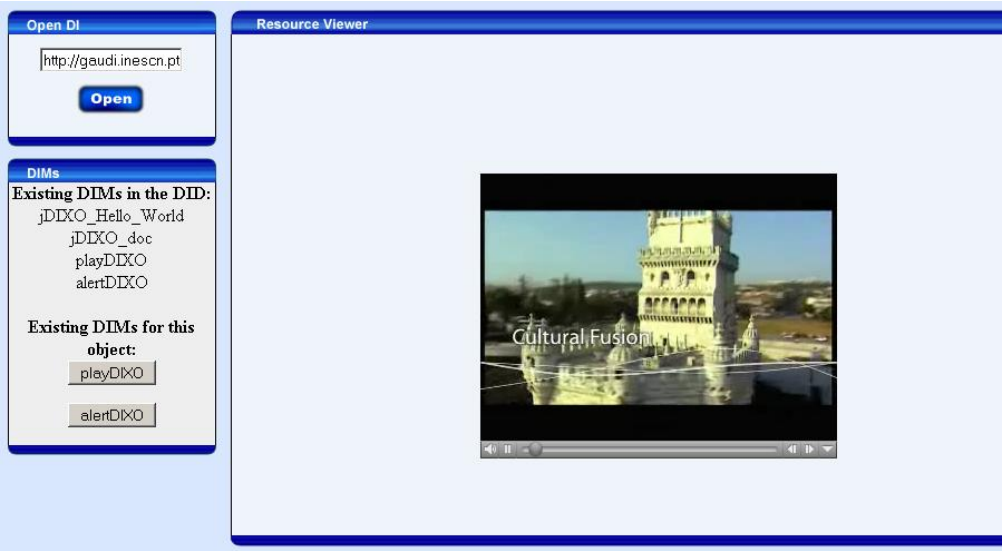


Figure 57 - Video Resource being viewed after pressing *playDIXO* button

Chapter 7

7 Conclusions

This chapter is meant to end this work with a few observations regarding to the possibility of executing DIP on a distributed way (using MPEG-21 DIP Reference Software) and analyzing the implemented solution.

The solution adopted in [1] to use DIP was to use a JAVA Applet, but this solution requires more processing capability from the User terminal, on the contrary the new solution, implemented here requires less capabilities and allows the implementation to be made according to the User terminal capabilities. In a certain way this solution solves the problem of the DIBOs implementation but with limitations that have to be solved. These limitations can be, as said before, the fact that it is necessary that the DIBO is the last code line in the DIM implementation so that it can be possible for the client to receive the result of the DIM execution witch is the XML String of the DIBO to interpret and execute the GUI or even the possibility that the implementation of the DIBOs GUI is not normalized. Having this in consideration it can be said that in fact this solution doesn't solve the problem, a normalized implementation and an implementation that allows the DIBO to be called at any point of the DIM is required.

Another possible solution for this would be to create some kind of DIPEngine (similar to the one existing in the MPEG-21 DIP Reference Software) that could be able to interpret the DIM making use of the implementation made to the rest of the DIBOs in the Reference Software that can be remotely executed. This engine would be able to call these DIBOs, executing himself the DIBOs that must be executed in the client (e.g. *alert*). The possible architecture for this solution is presented in Figure 58.

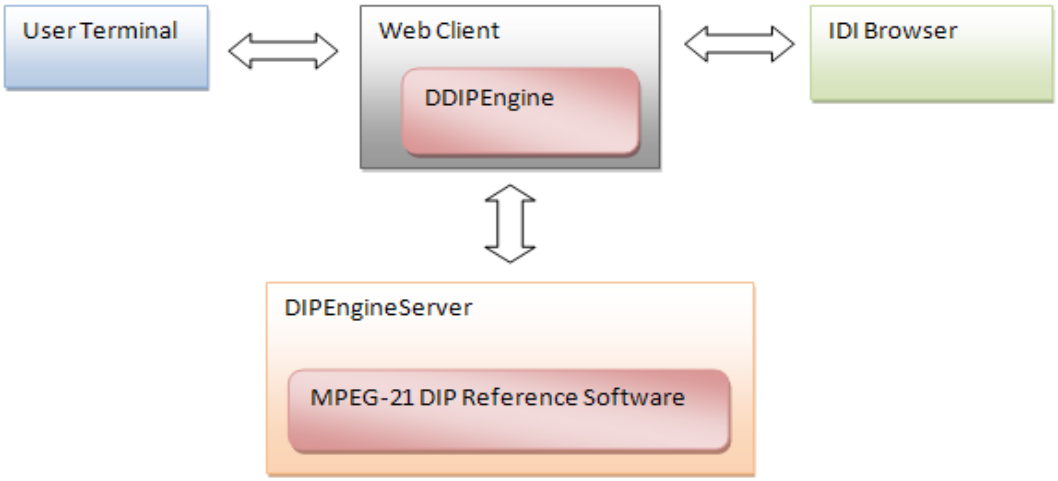


Figure 58 - General architecture of possible solution for Distributed DIP

The DDIPEngine would be the new module to assist in the interpretation of the DIMs and perhaps other modules would be necessary in the DIPEngineServer to make the DDIPEngine to work with the engines of the MPEG-21 DIP Reference Software. An attempt to specify and implement this solution or another possible solution, until the public presentation of this thesis will be made.

Other solution to consider, due to the fact that the DIBOs implementation made in the Reference Software works correctly when the purpose is to execute DIP locally allowing interaction between the User and the declared resources in the DID and that it causes problems when the purpose it to execute in a distributed way, is to change or to create a new implementation of the MPEG-21 DIP Reference Software, where all the implementation of the DIBOs would allow the remote execution of all DIBOs overcoming the limitations exposed in this work with the presented solution.

Considering some future work possible to be made, the main work should be to implement the rest of the DIBOs with the new solution. The implementation to execute JDIXOs should be completed because for now the MPEG-21 DIP Reference Software only allows the execution of JDIXOs by means of the DIBO *runJDIXO*, but the JDIXO should also be executed in a similar way of the DIM, because it is possible to declare it with the parameter *Label* that indicates if a *Component* is a DIM or a DIXO.

Plus, improvements can be made to the *DIPEngineServer* that would allow more functionalities, for example there is the possibility of configuring choices in DIMs and on the DIBrowser is possible to configure choices in order to browse items, but in the *DIPEngineServer* it should be possible to develop a module that would allow the User to configure choices regarding to the use of resources before any execution of DIMs, or even a mechanism to automatically define the states of choices can be created.

Annex A – DIBOs (informative)

A list of all the DIBOs implemented in the MPEG-21 DIP Reference Software is presented next.

DIBO
adapt
areConditionsSatisfied
configureChoice
setSelection
getElementsByIdentifier
getElementsByRelatedIdentifier
getElementsByType
alert
execute
getExternalData
getObjectMap
getObjects
getValues
play
print
release
runDIM
runJDIXO

wait
getLicense
queryLicenseAuthorization
getDIPErrorCode
getArgumentList
getArgumentListCount
getMethodCount
getMethodWithArgs
getMethodsWithArgs
getObjectOfType
getObjectsOfType
getObjectsOfTypeCount
getObjectTypeCount
getObjectTypeName
getStatus

Table 3 - Complete List of DIBOs

Annex B – DIPEngineServer Class

```
/** This is the class that implements the web services
*/
public class DIPEngineServer
{
        /** Initializes all the necessary engines and variables to Digital Item Processing
        * @param local the location of the DID
        * @return a String containing the list of DIMs
        */
        public String startDIP(String local){…}

        /** Stops all engines, and clears the list of DIMs
        */
        public void closeDim(){…}

        /** Filters the list of DIMs
        * @param obT is the object type that will be the condition to filter the DIMs
        * @return a String containing the new list of DIMs
        */
        public String filter(String obT){…}

        /** Obtains the number of arguments that the DIM requires
        * @param dim is the name of the DIM
        * @return an integer that s the number of arguments
        */
        public int getArgNumber(String dim){…}

        /** Obtains the arguments necessary for the DIM
        * @param pos is the position of the DIM in an internal list
        * @return a String object with the argument description
        */
        public String getArgs(String pos){…}

        /** Sets the argument selected by the User for the execution of the DIM
        * @param selected is the position of the argument in a list that contains
        * the arguments of the DIM
        */
        public void setArgument(int selected){…}
```

```
        /** Executes the DIM with the help of the MPEG-21 DIP Reference Software
        */
        public void runMethod(){…}

        /** Obtains a list of DIDs existing in a repository
        * @param local is the location of the repository
        * @return a String with the list of DIDs existing in the repository
        */
        public String getDIDs(String local) {…}

        /** Gets the title metadata inserted in the DID
        * @param did is the name of the DID from which we want the title
        * @return a String with the title of the DID
        */
        public String getDidTitle(String did){…}

        /** Gets the description metadata inserted in the DID
        * @param did is the name of the DID from which we want the description
        * @return a String with the description of the DID
        */
        public String getDidDescription(String did){…}

        /** Gets the DID contents
        * @param did is the name of the DID we want to retrieve
        * @return a String object with the full content of the DID
        */
        public String getFullDid(String did){…}

        /** Private method to verify is a DIM contains arguments
        * @param component is a DOM Node of the DID (the Node of the DIM)
        * @return Boolean object that indicates if the DIM contains or not an argument
        */
        private static boolean containsMethodInfo(Node component) {…}


        /** Obtains the argument Nodes necessary for a DIM
        * @param component is the DIM from which is necessary to obtain the argument
        * Nodes
        * @return an array object of Nodes containing all the arguments of a DIM
        */
        private static Node[] MyArguments(Node component) {…}
}
```

References

[1] Giorgiana Ciobanu, "MPEG21 DI Browser, an MPEG-21 based architecture for the consumption of Digital Items", Master Degree Dissertation, Faculty of Engineering of University of Porto, 2006

[2] Frederik De Keukelaere, Saar De Zutter, Rik Van de Walle, "MPEG-21 Digital Item Processing", IEEE TRANSACTIONS ON MULTIMEDIA, VOL. 7, NO. 3, June 2005

[3] Ian Burnett (UoW), Rik Van de Walle (Ghent University), "Current Vision on MPEG-21 Digital Item Processing", December 2002

[4] http://www.chiariglione.org/mpeg/standards/mpeg-21/mpeg-21.htm#_Toc23297968, October 2002

[5] http://multimediacommunication.blogspot.com/2007/06/mpeg-21-multimedia-framework.html, June 2007

[6] http://mpeg-21.itec.uni-klu.ac.at/cocoon/mpeg21/, January 2005

[7] Information technology – Multimedia framework (MPEG-21) - Part 2: Digital Item Declaration, ISO/IEC FDIS 21000-2:2005(E), April 2005

[8] Information technology – Multimedia Framework (MPEG-21) – Part 10: Digital Item Processing, ISO/IEC FDIS 21000-10:2005(E), July 2005

[9] http://www.ist-world.org/, December 2007

[10] http://danae.rd.francetelecom.com/, March 2006

[11] http://danae.rd.francetelecom.com/project-technology.php, December 2005

[12] http://multimedialab.elis.ugent.be/demo.asp, July 2006

[13] http://www.axmedis.org/, 2006

[14] http://www.enikos.com, 2007

[15] http://www.prweb.com/releases/2003/11/prweb89412.htm, November 2003

[16] http://www-itec.uni-klu.ac.at/~harald/MPEG-21/, August 2004

[17] http://www.adactus.no, 2006

[18] http://www2.inescporto.pt/utm/projecto/em-curso/projecto-enthrone, 2008

[19] Maria Teresa Andrade, P.F. Souto, Pedro Carvalho, H. Castro, L. Ciobanu, B. Feiten, "Dynamic service management in heterogeneous environments using MPEG-21 DIA for multimedia content adaptation", INESC Porto, Deutsche Telekom, 2006

[20] Alberto Manuel Rodrigues da Silva e Carlos Alberto Escaleira Videira, "UML metodologias e ferramentas CASE", Centro Atlântico, Lisboa 2005

[21] G. Ciobanu, M. Andrade, P. Carvalho, E. Carrapatoso, "An MPEG-21 Web Peer for the consumption of Digital Items", *Proc. CISTI 2007*, June 2007

[22] Ian S. Burnett, Fernando Pereira, Rik Van de Walle, Rob Koenen, "The MPEG-21 Book", Wiley, 2006