Towards the Future Internet
G. Tselentis et al. (Eds.)
IOS Press, 2010

doi:10.3233/978-1-60750-539-6-217

Accelerating Media Business Developments with the MPEG Extensible Middleware

Christian TIMMERER[a,1], Filippo CHIARIGLIONE[b], Marius PREDA[c], and Victor Rodriguez DONCEL[d]
[a] *Klagenfurt University, Austria* / [b] *SmartRM, Italy*
[c] *Institut TELECOM, France* / [d] *Universitat Politècnica de Catalunya, Spain*

Abstract. This document provides an overview of the MPEG Extensible Middleware (MXM), one of ISO/IEC MPEG's latest achievements, defining an architecture and corresponding application programming interfaces (APIs) which enable accelerated media business developments. The paper describes the vision behind MXM, its architecture, and a high level overview of the API. Additionally, example MXM applications are given.

Keywords. Middleware, API, MPEG, Media Applications.

Introduction

The development of media business-related applications is currently becoming a very challenging task due to short deployment cycles and the huge amount of applications flooding the market (e.g., 'app stores'). This calls for standardized and platform-independent application programming interfaces (APIs) for well known media-related functions (e.g., coding, packaging, storing, delivering) so that they do not have to be re-implemented each time a new kind of end user device – possible running on a new platform – emerges on the market. ISO/IEC MPEG working group has recognized this fact and started the development of an MPEG Extensible Middleware (MXM) [1] which exactly addresses this issue.

The paper is organized as follows. Section 1 describes the MXM vision on how media business developments can be accelerated and overviews of the MXM architecture and its APIs are presented in Section 2 and 3 respectively. A selection of MXM applications [2][3][4] that have been developed during the course of the MXM standardization is presented in Section 4. Section 5 concludes the paper.

1. The MXM Vision

With the establishment of the MPEG-21 Multimedia Framework it has been stated that "every human is potentially an element of a network involving billions of content

[1] Corresponding Author. Email: christian.timmerer@itec.uni-klu.ac.at. Phone: +43/463/2700 3621.

providers, value adders, packagers, service providers, resellers, consumers ...” [5]. In particular, MPEG-21 enables the transaction of Digital Items among Users. The former is defined as a structured digital object with a standard representation, identification and metadata whereas the latter may be any kind of creator, end user or intermediary that makes use of Digital Items in the MPEG-21 framework or interacts with other Users. Thus, MPEG-21 is a framework covering the complete range of MPEG technologies defined so far, i.e., from systems (e.g., file formats, delivery formats) to audio/video codecs (e.g., MPEG-2, MPEG-4, AVC) and description formats (e.g., MPEG-7).

However, a framework is almost nothing without a platform on which it can operate and the Digital Media Project (DMP) [6] specifies one such platform. Therefore, DMP specifies an interoperable DRM platform (IDP) by adopting most MPEG-21 technologies and adding a few that were missing [7]. Furthermore, DMP provides an open source software implementation called Chillout® [8] for the functions and protocols defined in or referenced by IDP.

One issue when defining a platform is the portability to other platforms which calls for a middleware to be used in a platform-independent way. That is, the next step is from platform to middleware which is referred to as *MPEG Extensible Middleware* (MXM) [1][9] or MPEG-M, a standard designed to promote the extended use of digital media content through increased interoperability and accelerated development of components, solutions and applications. For this regard, MXM introduces the notion of MXM devices, MXM applications and MXM engines. The MXM standard is organized in the following parts:

- Part 1: Architecture and technologies [10] provides the definition of the architecture and technologies (by reference) used within MXM. This will ensure that MXM devices will be able to run (or play) MXM applications.
- Part 2: MXM API [11] includes normative APIs to so-called MXM engines on top of which MXM applications can be developed independent of the actual engine implementations to be used.
- Part 3: Conformance and reference software [12] as open source software.
- Part 4: MXM protocols [13] enabling means for interoperable communication of MXM devices and, hence, MXM applications.

The advantages of having a normative API to MPEG technologies are the following. First, there is no need to have in-depth knowledge of specific MPEG technologies in order to use them. The API provides simple methods to call complex functionalities inside MXM engines leading to “thin” applications because the complexity is hidden in the MXM engines. Second, it offers the possibility of replacing the individual “blocks” (i.e., the MXM Engines) with optimized ones with zero cost for integration thanks to the normative API. Third, it enables the development of innovative applications based on multiple MPEG technologies combined in specific ways. Finally, an infinite number of innovative business models based on MPEG technologies could be developed at much reduced costs. For example, new applications can be developed (and deployed) as soon as reference software for a new video codec becomes available following the normative MXM API. As soon as an optimized version of this video codec becomes available it can be exchanged with the reference software without affecting the application running on top of it.

2. The MXM Architecture and Technologies

The "MXM architecture and technologies" [10] specifies the MXM architecture and references the technologies that are part of an MXM implementation. The architecture is depicted in Figure 1 and comprises a set of MXM engines for which APIs are defined on top of which applications can be developed. The current list of MXM engines includes functionalities for content creation/search, adaptation, streaming/delivery, domain management, IPMP, rights expression, licensing, metadata, event reporting, security, etc. A special role takes the *Orchestrator Engine* which provides access to higher-level functionalities by implementing common scenarios utilizing various MXM Engines in a pre-defined way (e.g., adaptation of content according to the usage context).

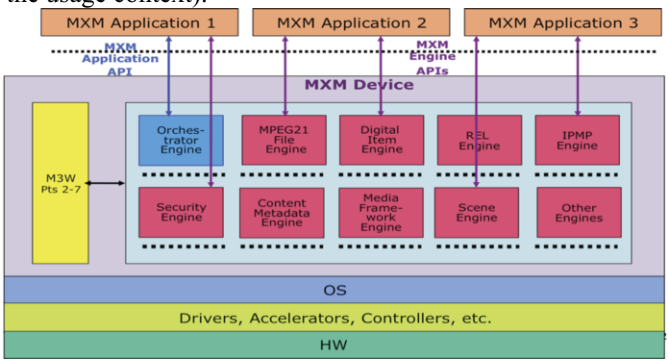

Figure 1. The MXM Architecture.

In order to enable interoperable communication between MXM devices the standard defines MXM protocols as shown in Figure 2. An instantiation example of various MXM devices communicating with each other is illustrated in Figure 3. Let us note that only the payload format, which is XML-based, is specified and it may be actually delivered using any suitable transport protocol (e.g., HTML, Web Services).

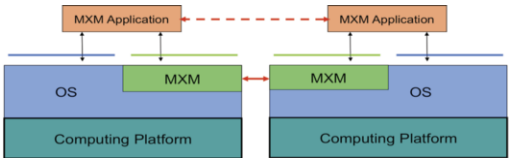

Figure 2. MXM Protocols.

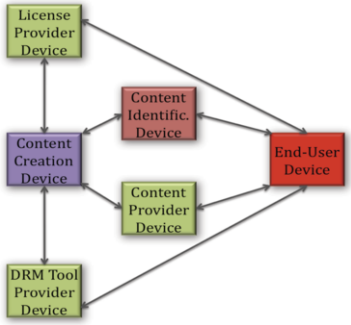

Figure 3. MXM Protocol Instantiation Example.

3. The MXM Application Programming Interface (API)

The MXM API [11] of each engine have been divided with respect to the targeted functionality into *creation* (e.g., encode a raw audio track, create an MPEG-7 metadata

description), *access* (e.g., get data from a Digital Item, decode a video), *editing* (e.g., add an elementary stream to a multiplexed content), and *engine-specific* APIs (e.g., authorize a license as part of the REL Engine).

The next sections introduce two selected APIs – Media Framework Engine and Metadata Engine. For the other APIs the interested reader is referred to [1].

3.1. Media Framework Engine API

The Media Framework Engine API defines a set of APIs related to different media modalities such as image, audio, video, and graphics. At the time of writing of this paper it provides means for creating (i.e., encoding) and accessing (i.e., decoding) of audio, graphics 3D, image and video resources.

The API is organized in a hierarchical fashion where higher levels provide more generic functions applicable to all types of media (e.g., play, pause, seek) and lower levels provide specific functions only applicable to certain media types (e.g., `setVolume` for audio or `getDecodedImage` for image).

3.2. Metadata Engine API

This API can be divided into two parts, one being generic to potentially all kinds of metadata standards (including those developed outside of MPEG) and one being specific to MPEG-7 metadata. As for the Media Framework Engine, this API provides means for creating and accessing metadata. For example, the former could be used by authoring software to create metadata associated to actual content whereas the latter could be used for parsing the metadata at the consumption stage.

In particular, the generic metadata creator/parser provides an interface defining the methods to create/parse generic metadata structures in possibly any format depending on the specific implementation of the metadata engine of choice. Similarly, the MPEG-7 creator and parser define methods for creating and parsing metadata structures compliant to the MPEG-7 standard respectively.

4. Examples of MXM Applications

4.1. Fully Interoperable Streaming of Media Resources in Heterogeneous Environments

This section describes an MXM-based architecture for the fully interoperable streaming of media resources (i.e., scalable and non-scalable) in heterogeneous usage environments (i.e., with different and varying terminal capabilities, network conditions, and user preferences) [14].

The architecture for this framework is depicted in Figure 4 and in the following we will give a brief walkthrough.

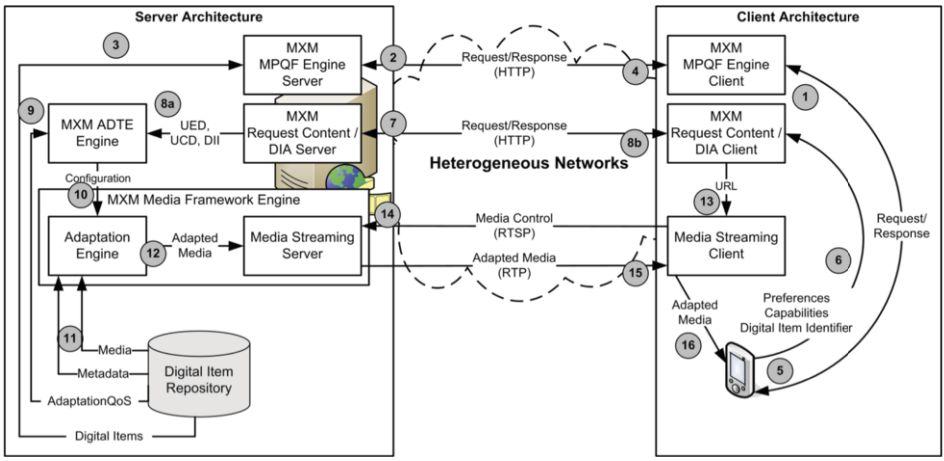

Figure 4. MXM-based Architecture for a Fully Interoperable Streaming Framework of Media Resources in Heterogeneous Environments.

- **Query for available Digital Items (Steps 1-5)**: The terminal requests its local *MPEG Query Format (MPQF)* engine to issue a query for the available Digital Items to its counterpart on the server side (1-2). The MPQF engine at the server examines its repository and responds to the client with a list of available Digital Items which are presented to the user on the terminal's display (3-5). In this case the MPQF is transmitted via HTTP.

- **Select Digital Item (Steps 6-13)**:

  The user is now able to select the desired Digital Item she/he wants to consume. Therefore, the corresponding identifier is provided to the *Request Content / Digital Item Adaptation (DIA) client* which also assembles the user preferences, terminal capabilities, and network conditions into a Usage Environment Description (UED) and Universal Constraint Description (UCD). This information is included within the MXM Request Content protocol which is transmitted via HTTP to the server (6-7). The server responds with an personalized RTSP URL (8b) which is then used by the *Media Streaming Client* in the subsequent stage to initialize the actual streaming session (13).

  At the server, the DIA information (i.e., UED and UCD) together with the Digital Item Identifier (DII) is passed to the *Adaptation Decision-Taking Engine (ADTE)* which is used to configure the *Adaptation Engine* based on metadata (i.e., AdaptationQoS) associated with the DII (8a-10). The Adaptation Engine adapts the media resource according to the parameters provided by the ADTE and forwards the adapted media resource bitstream to the *Media Streaming Server* (11-12).

- **Streaming of adapted the media resource (Steps 14-16)**: The *Media Streaming Client* requests the adapted media resource bitstream from the *Media Streaming Server* via RTSP and the server provides it via RTP (14-15). Finally, the bitstream is presented to the user at the terminal's display (16). Optionally it is possible to update the UED information during streaming (e.g., change in available network bandwidth) and the corresponding bitstream is adapted during the streaming session. That is, everything starting from step 6 is repeated in a loop except that a

new RTSP URL is issued (steps 8b, 13, and 14). The updated version of the adapted media resource bitstreams becomes visible as soon as the client receives this information as the adaptation is performed in a timely manner.

For further details the interested reader is referred to [14].

4.2. Including MPEG-4 3D Graphics in Third-Party Application

Including audio, image, and video resources (e.g., MP3, AAC, JPEG, MP4) in third-party applications is nowadays a beginner job. The complexity of such codecs is hidden behind a very simple communication interface once the content is decoded, i.e., matrix of pixels for images and wave samples for audio. Transposing the same principle in the computer graphics world is a challenge due to the variety of representation forms and also the complexity and heterogeneity of data to be transferred, i.e., vertex position, normals and tangents, color and texture as well as their variation in time.

The application introduced in this section shows how using the MXM `3DGraphicsEngine` and its set of APIs to simplify the complex integration work. Specifically, with only some lines of code, Ogre3D – a very well known 3D graphics rendering engine – is transformed into an MPEG-4 3D graphics player.

Ogre3D is a simple and easy to use oriented object interface designed to minimize the effort required to render 3D scenes, and to be independent of 3D implementation, i.e., Direct3D/OpenGL. It provides advanced features for object definition and rendering (mesh, appearance and animation) as well as scene management.

Built on top of VRML97, MPEG-4 contained, already in its first specifications from ten years ago [15], tools for the compression and streaming of 3D graphics assets, enabling to describe compactly the geometry and appearance of generic, but static objects, and also the animation of human-like characters. Since then, MPEG has kept working on improving its 3D graphics compression toolset and published two editions of MPEG-4 Part 16, AFX (Animation Framework eXtension) [16], which addresses the requirements above within a unified and generic framework and provides many more tools to compress more efficiently generic, textured, animated 3D objects. While the 3D content is a complex data structure involving the coexistence of heterogeneous data types (geometry, that can be specified as surface or volume, appearance that exposes material properties and texture and finally animation that can be obtained from a variety of rigid motion or deformation models), it is relatively easy to specify a formalism for representing such data. For this reason, there are currently several tons of formats for describing 3D content. MPEG-4 AFX is not yet another format for representing 3D data since its key contribution is related to compression more that to the representation itself. Recent standardization activity formalized in MPEG-4 Part 25 shows how the MPEG-4 tools for compressing 3D assets can be applied to other formalisms such as COLLADA and X3D. However, in order to make the compression layer transparent to the application developer, the MXM API transport the information between the media framework engine and the developed application in a flat format, very similar to the ones used by Direct3D or OpenGL. Thus after loading an MPEG-4 3D object the engine exposes to the application the corresponding vertex, normal, texture coordinates buffer and the associated index buffers in a data structure ready for rendering. Figure 5 shows several MPEG-4 3D objects loaded in the Ogre engine by using the MXM engine for data decoding.

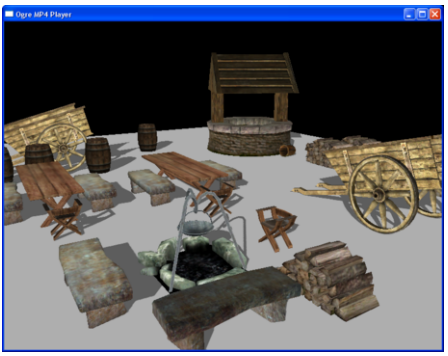

Figure 5. MPEG-4 3D objects loaded in the Ogre engine.

4.3. Sharing Content in a Smart Way

MXM can be used in a variety of environments and can serve for multiple purposes. This section describes how MXM is employed by SmartRM (http://www.smartrm.com), an innovative system for sharing content while retaining control of it.

SmartRM is a viral service based on social networks enabling everyone to share confidential content with friends or colleagues in a protected way. The SmartRM software allows its users to convert confidential files (e.g., PDF documents, videos or audio tracks) into encrypted MPEG-21 files that can be shared with others: only the contacts that have been enabled can read, view, listen, print, etc. the protected content, at the conditions that have been specified. The SmartRM service makes it possible to know when and how many times a protected file has been accessed by someone, as well as grant or remove permissions dynamically.

The SmartRM system architecture is based on a client-server model. The client is a Mozilla Firefox plug-in based on MXM [17]. Most of the Web browsers available today, in fact, are capable of initializing, creating, destroying, and positioning plug-ins inside the browser window when certain events occur, often through standard APIs such as NPAPI [18]. Figure 6 shows an example of a high-level architecture of the SmartRM client software.

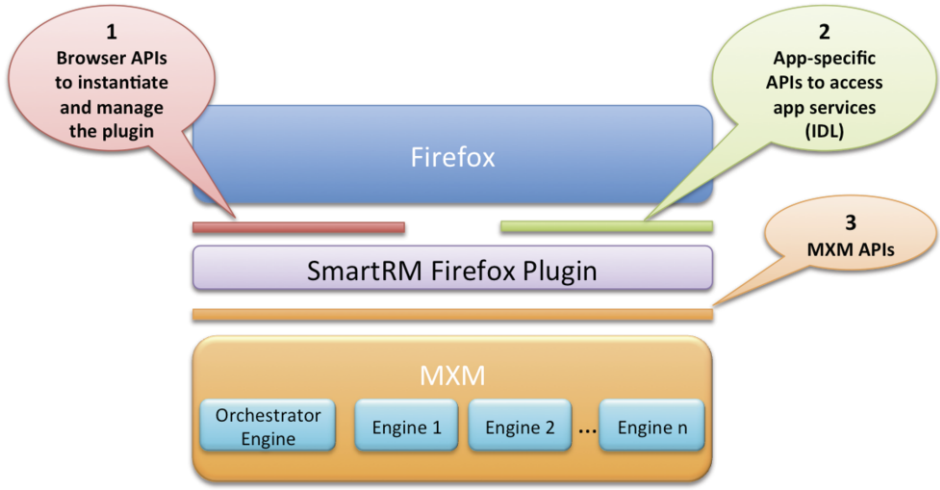

Figure 6. High-level architecture of the SmartRM MXM-based Firefox plug-in.

The SmartRM functionalities can be accessed either by code running on an HTML page (e.g., Javascript) or by other Firefox plug-ins, extensions or components through an API described in IDL. By relying on MXM, the SmartRM Firefox plug-in delegates media access and presentation, as well as security-related functionalities to C++ MXM engines that encapsulate the complexity of those functionalities inside easily-manageable modules that can be replaced at ease because access to them is made through the standard MXM APIs.

At the same time, most of the communication between the SmartRM client and the SmartRM server is done by exchanging ISO/IEC 23006-4 (MXM Protocols) messages over SOAP and XMPP. Both the client and the server are then relieved from the complexity of generating, dispatching and interpreting such messages, as this operations can be done by MXM engines.

4.4. Tracking the Intellectual Property Value Chain

This section describes an MXM-based scenario where intellectual property attribution and trace of the value chain is put in a first place of importance. MXM is about handling multimedia, and the content represented by the Digital Items most of the times representing something that is protected by intellectual property laws. Going beyond the elementary task of representing who is the copyright owner of this "something", MXM also provides engines, devices and protocols to trace the complete intellectual property value chain, being possible to find out who was the original author, the adaptors who made derivative works, the performers, the producers or the distributors or broadcasters who took part in the chain. Moreover, within the MXM framework it is possible to categorize precisely the kinds of objects subject to the intellectual property protection, as well as the different actions taken on the Digital Items which are relevant in regard to the intellectual property.

All of this is accomplished by means of the role verification device, a server application able to receive notice of the important events for the intellectual property and answer queries. This server hosts the Media Value Chain Ontology (MVCO) [19], implementing an API on top of it (the MXM MVCO Engine), and maintains a set of

ontology class instances corresponding to all the relevant users, intellectual property entities and actions taken, being able to respond to the queries as the result of the execution of an ontology reasoner.

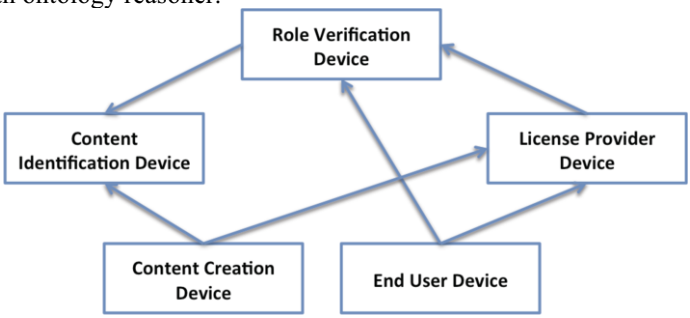

Figure 7. MXM-based architecture for tracking the intellectual property value chain.

To describe the interoperation of the devices in the presented scenario (see Figure 7), an example is given. In this scenario, the user called Alice creates a content. She does not merely create a resource, i.e., a Digital Item, she is a creator in the sense that she makes a new artistic work. Then, by means of the *Content Creation Device*, uploads her content to the *Content Identification Device* (an abstract reference called work and a first manifestation of the work). The *Role Verification Device* is also informed of this novelty, and will keep record of it.

Thus, when another user, called Bob, wants to make an adaptation of that work, he will need to identify the nature and creator of the work, and manage to get a license authorizing him to register an adaptation. The license Alice has to issue for this regard, given by the *License Provider Device*, can be verified to satisfy the intellectual property requirements as a check against the role verification device. Perhaps then a third user called Chang may upload a performance of Bob's adaptation, and a distributor called Diana may offer the resulting record for purchase. Thanks to the role verification device these – and other – actions on the material can be performed assuming the originator has granted the right to do so and it provides means for tracking the whole process. For example, the end user Erik may be interested in knowing the original composer of the work or wants to know which role does Bob play in this. Additionally, the *License Provider Device* may verify the conformance of the licenses issued in the process to the intellectual property model, certifying, for example, that the performance Chang made took place with the necessary consent of Alice, etc.

5. Conclusions

In this paper we have introduced the MPEG Extensible Middleware enabling accelerated media business developments by defining a normative APIs to the vast amount of MPEG technologies currently available (i.e., systems, audio, video, 3D graphics, metadata, etc.). Thanks to these normative APIs, applications can be developed on top of these APIs even though the underlying technologies (e.g., advanced audio/video codec or delivery format) are still, for example, subject to standardization and where only proof-of-concept reference software is available. At a later stage this software can be replaced by optimized one at no cost thanks to the standardized API. Furthermore, the middleware effectively hides the complexity below

the API and, thus, almost eliminates the burden for newcomers to enter the highly competitive media business. That is, complex media applications become very tiny and easy to develop without digging into thousands of pages of various MPEG standards.

Acknowledgments

Although only a few names appear on this paper, this work would not have been possible without the contribution and encouragement of many people, particularly Leonardo Chiariglione (CEDEO.net), Michael Eberhard (Klagenfurt University), Ivica Arsov (Institut TELECOM), Angelo Difino (CEDEO.net), and Wonsuk Lee (ETRI).

References

[1] MPEG Extensible Middleware, http://mxm.wg11.sc29.org/, (last access: January 2010).
[2] M. Eberhard, C. Timmerer, H. Hellwagner, "Fully Interoperable Streaming of Media Resources in Heterogeneous Environments", 1st Int'l MXM Developer's Day, London, UK, June 2009. Available at http://mxm.wg11.sc29.org/ (last access: January 2010).
[3] I. Arsov, M. Preda, "Including MPEG-4 3D graphics in your application", 1st Int'l MXM Developer's Day, London, UK, June 2009. Available at http://mxm.wg11.sc29.org/ (last access: January 2010).
[4] A. Difino, F. Chiariglione, "An MXM-based application for sharing protected content", 1st Int'l MXM Developer's Day, London, UK, June 2009. Available at http://mxm.wg11.sc29.org/ (last access: January 2010).
[5] I. Burnett, F. Pereira, R. Van de Walle, R. Koenen (eds.), *The MPEG-21 Book*, Wiley, 2006.
[6] Digital Media Project, http://www.dmpf.org/, (last access: January 2010).
[7] Digital Media Project, *Interoperable DRM Platform v3.2*, October 2008. http://www.dmpf.org/project/ga20/idp-32.html.
[8] Chillout, http://chillout2.dmpf.org/wordpress/, (last access: January 2010).
[9] L. Chiariglione, "The MXM Vision", 1st Int'l MXM Developer's Day, London, UK, June 2009.
[10] ISO/IEC 23006-1, *Information Technology – MPEG extensible middleware (MXM) – Part 1: MXM architecture and technologies*, Final Committee Draft, London, UK, June 2009.
[11] ISO/IEC 23006-2, *Information Technology – MPEG extensible middleware (MXM) – Part 2: MXM API*, Final Committee Draft, London, UK, June 2009.
[12] ISO/IEC 23006-3, *Information Technology – MPEG extensible middleware (MXM) – Part 3: MXM conformance and reference software*, Final Committee Draft, London, UK, June 2009.
[13] ISO/IEC 23006-4, *Information Technology – MPEG extensible middleware (MXM) – Part 4: MXM protocols*, Final Committee Draft, London, UK, June 2009.
[14] M. Eberhard, C. Timmerer, E. Quacchio, and H. Hellwagner, "An Interoperable Streaming Framework for Scalable Video Coding Based on MPEG-21", *IEEE Wireless Communication*, vol. 16, no. 5, pp. 58-63, October 2009.
[15] ISO/IEC 14496-2, *Information technology – Coding of audio-visual objects – Part 2: Visual*, September 2009.
[16] ISO/IEC 14496-16, *Information technology – Coding of audio-visual objects – Part 16: Animation Framework eXtension (AFX)*, December 2009.
[17] A. Difino, "Use of MXM in a web browser environment", ISO/IEC JTC1/SC29/WG11/M16836, Xian, China, Oct 2009.
[18] Netscape Plugin Application Programming Interface (NPAPI), http://en.wikipedia.org/wiki/NPAPI (last access: January 2010).
[19] V. Rodriguez-Doncel, J. Delgado, "A Media Value Chain Ontology for MPEG-21", *IEEE MultiMedia,* vol. 16, no. 4, pp. 44-51, October 2009.