Anthony Vetro
Mitsubishi Electric Research Labs

Accelerating the Media Business with MPEG Extensible Middleware

Christian Timmerer
Klagenfurt University, Austria

Filippo Chiariglione
SmartRM

Marius Preda
Institut TELECOM

Victor Rodriguez Doncel
Universitat Politècnica de Catalunya, Spain

The development of successful multimedia applications is becoming a challenging task due to short deployment cycles and the huge amount of applications flooding the market. One major problem that the multimedia industry is facing in this area is the heterogeneity of the content-delivery chain. The classic model in which all the components, such as the authoring, distribution, and playback, were known in advance, has now evolved into an interconnected network of programmable processing units that are no longer dedicated and built for one purpose. While there are many advantages to this classic model, there are new challenges with the introduction of a heterogeneous content chain, including the proliferation of data formats for different kinds of media delivered over the Internet.

This situation calls for standardized and platform-independent APIs for well-known, media-related functions (for example, coding, packaging, storing, and delivering) so that they don't have to be redone each time a new type of consumer device, possibly running on a new platform, emerges on the market. ISO/IEC MPEG has recognized this fact with the development of an MPEG Extensible Middleware (MXM), which specifically addresses this issue. This article describes this new programming framework by reviewing the MXM vision, architecture, and specified APIs. The article also describes an MXM application for content sharing to illustrate the merits of this framework.

Editor's Note

Media applications are becoming increasingly complex. They handle many data formats, run across multiple platforms, and support a wide range of functions. This article describes a standardized set of protocols and APIs that provides efficient access to individual system components, enables rapid deployment of new applications, and improves portability.

—Anthony Vetro

The MXM vision

Proponents of open multimedia frameworks such as MPEG-21 argue that "every human is potentially an element of a network involving billions of content providers, value adders, packagers, service providers, consumers, and resellers."[1] These frameworks enable the production, delivery, and consumption of multimedia content, and use the complete range of media technologies, including system-layer technologies (for example, file and delivery), audio and video codecs (for example, MPEG-2, MPEG-4, and VC-1), and description formats (for example, MPEG-7).

However, a framework is almost nothing without a platform on which it can operate. One such reference platform, specified by the Digital Media Project (DMP, see http://www.dmpf.org/), features an interoperable digital-rights-management (DRM) system comprised of various MPEG-21 technologies and other missing components. DMP provides an open-source implementation called Chillout (see http://chillout.dmpf.org/) for the functions and protocols defined in or referenced by the interoperable DRM platform.

A key issue when defining a platform is the portability to other platforms, a concept that calls for middleware to be used in a platform-independent way. One example of such middleware is MXM 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. MXM introduces the notion of MXM devices, applications, and engines. The MXM standard is organized into the following parts:

- Part 1: defines the architecture and technologies (by reference) used within MXM, ensuring that MXM devices will be able to run (or play) MXM applications.

1070-986X/10/$26.00 © 2010 IEEE

Published by the IEEE Computer Society

- Part 2: MXM API includes normative APIs to so-called MXM engines on top of which MXM applications can be developed independently of the actual engine implementations to be used.
- Part 3: reference software provides an implementation of the API mostly using MPEG technologies and is open-source software.
- Part 4: MXM protocols enable the means for interoperable communication among MXM devices and, hence, MXM applications.

There are several advantages of having a normative API. First, there is no need to have an in-depth knowledge of specific media technologies (for example, MPEG coding or systems standards) 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 (that is, the MXM engines) with optimized ones, with minimal cost for integration thanks to the normative API. Third, it enables the development of standards-compliant applications—for example, applications based on multiple MPEG technologies combined in specific ways.

Finally, an infinite number of innovative business models based on media technologies, including those based on MPEG, could be developed at reduced costs by using a normative API. For example, new applications could be developed and deployed as soon as reference software for a new video codec is available. And as soon as an optimized version of this video codec becomes available, it could be exchanged with the reference software without affecting the application running on top of it.

MXM architecture and API

Part 1 of MXM specifies its architecture and references the technologies that are part of MXM. Figure 1 depicts the architecture, which comprises a set of MXM engines for which APIs are defined and on top of which applications can be developed. The current list of MXM engines includes functionalities for content creation, search, adaptation, streaming and delivery, domain management, intellectual property management and protection (IPMP), rights expression, licensing, metadata, event reporting, security, and so on. The orchestrator engine has a special role in providing access to higher-level functionalities by implementing common scenarios using various MXM engines in a predefined way (for example, adaptation of content according to the usage context). An MXM application can make use of as many or as few engines as required to support the intended functionality.

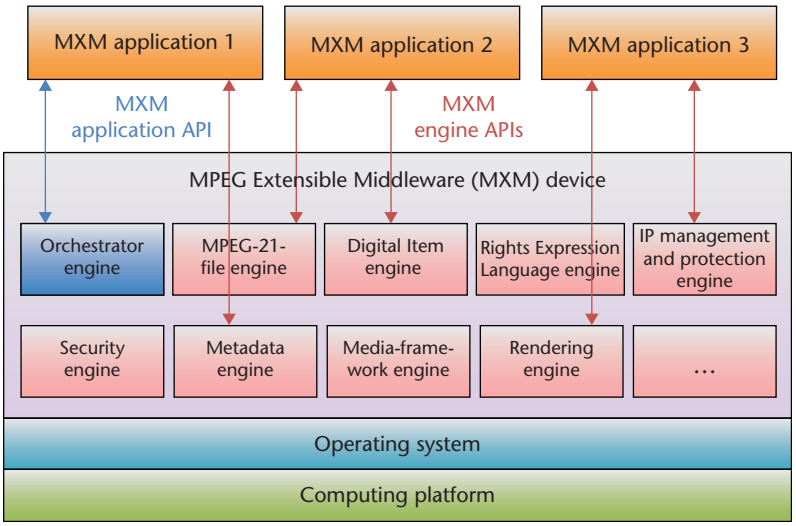

Figure 1. The MPEG Extensible Middleware architecture.

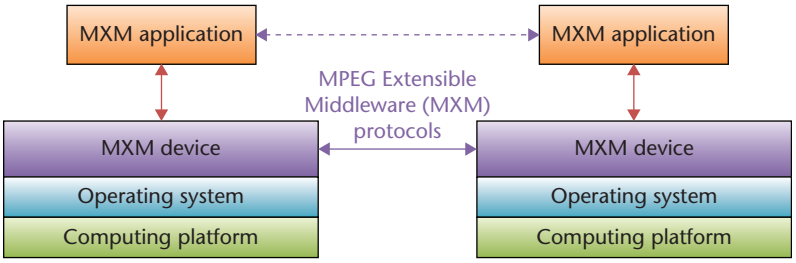

Figure 2. MPEG Extensible Middleware protocols.

To enable interoperable communication between MXM devices, the standard defines MXM protocols as shown in Figure 2. Only the payload format, which is XML-based, is specified here. This format can actually be delivered using any suitable transport protocol, such as HTML or SOAP.

Figure 3 (next page) illustrates the MXM protocol system architecture, with the devices and protocols defined as follows:

- Content-creation device creates content items, possibly including audiovisual resources, metadata, rights information, and so on.

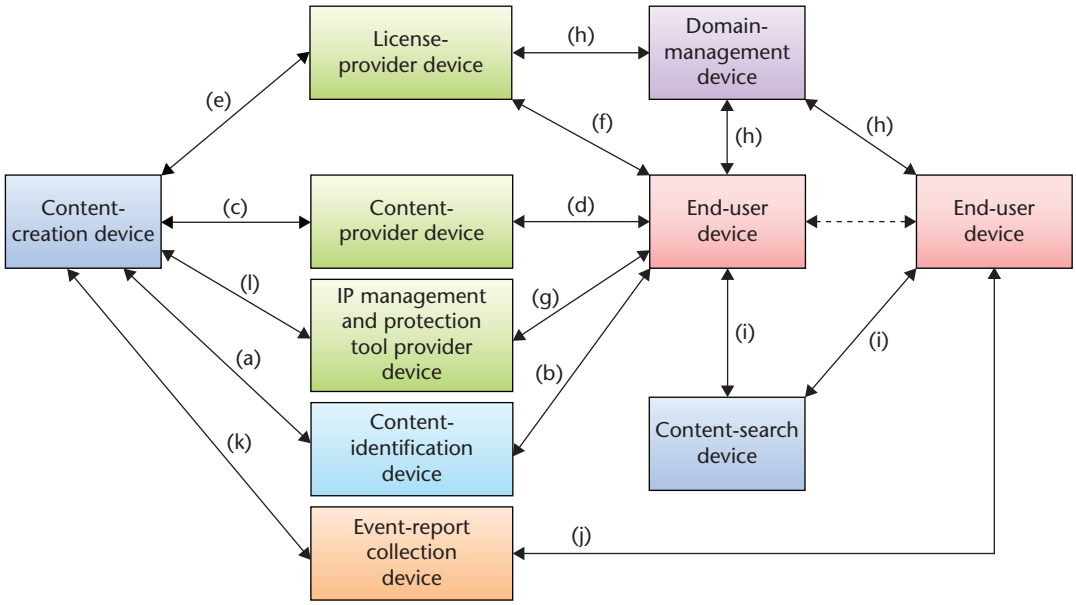

Figure 3. The MPEG Extensible Middleware protocol system architecture.

- Content-provider device stores content and in turn providing it to other devices (for example, via streaming, downloading, and so on).
- Content-search device provides search results for a given content query, for example, according to the MPEG Query Format (MPQF).
- License-provider device issues licenses to other devices upon request.
- IPMP-tool-provider device interacts with other devices to provide IPMP tools.
- Content-identification device provides identifiers to new content items and parts thereof, and allowing applications to verify the authenticity of the identified objects.
- Domain-management device manages the various functions of a domain.
- End-user device accesses content, licenses, and IPMP tools from other devices; authenticates content; and becomes part of a domain of devices.
- Event-report collection device processes event-report requests and issuing event reports.

Figure 3 illustrates the following protocols (letters correspond to labels in the figure), which are defined as follows:

- (a) identify content protocol to identify content items and elements thereof;
- (b) authenticate content protocol to authenticate content items and elements thereof;
- (c) store content protocol to store content items and elements thereof;
- (d) access content protocol to obtain content items and elements thereof;
- (e) store license protocol to configure a license service to issue licenses;
- (f) access license protocol to obtain licenses granting rights over content items and elements thereof;
- (g) access IPMP tool protocol to obtain IPMP tools necessary to access protected content;
- (h) manage domain protocol to create, join, and administer a group of users and devices;
- (i) content-search protocol to enable the end-user device to search for content;
- (j) store event report to request the event-report collection device to store an event report;
- (k) register event-report request to request the event-report collection device to register a class of event reports triggered by a certain event-report request; and
- (l) access IPMP tool list protocol to request the list of available IPMP tools.

The MXM API of each engine can be roughly clustered with respect to the targeted functionality into the following groups: creation (for example, encode a raw audio track or create an MPEG-7 metadata description), access (for example, get data from a Digital Item or decode a video), and editing (for example, add an elementary stream to multiplexed content). The following generic APIs are available and are not tied to a specific MPEG technology:

- The Digital Item adaptation engine specifies the means to access and create information pertaining to the usage environment context, such as screen resolution and supported coding formats.
- The media framework engine defines a set of APIs related to different media modalities, such as image, audio, video, and graphics.
- The metadata engine provides the means to parse and potentially create all kinds of metadata standards, including those developed inside and outside of MPEG.
- The rendering engine includes a comprehensive set of APIs related to rendering media.
- The security engine defines APIs pertaining to the management of security certificates, keys, devices, and so on.

In addition to the generic APIs that could work with a broad class of engines (including proprietary ones), MXM also specifies several MPEG-specific APIs to enable efficient use of MPEG technologies:

- The content-protocol engine defines APIs for accessing, authenticating, identifying, servicing, and storing content.
- The content search engine provides an API to the MPQF.
- The Digital Item engine has APIs for creating, parsing, and editing of Digital Item declarations and parts thereof.
- The Digital Item streaming engine includes APIs for creating Bitstream Binding Language (BBL) files (that is, metadata that steers the streaming) and the actual streaming based on this file.
- The event-reporting engine defines APIs for event-report requests and the actual event reports.
- The domain engine provides the means for domain management.
- The IPMP and IPMP-tool engines comprise APIs related to IPMP and its protocols.
- The license-protocol engine is used to handle requests pertaining to licenses.
- The MPEG-21-file engine enables creation and parsing of MP21 files.
- The media value-chain ontology engine defines the API for accessing information related to Part 19 of MPEG-21.
- The Rights Expression Language engine is a powerful API for the REL standard.
- The orchestrator engine defines orchestration operations related to adaptation, IPMP, play-out, and REL.

Example MXM application

MXM can be used in several environments and can serve multiple purposes. To illustrate the capabilities of MXM, this section describes how MXM is employed by SmartRM (see http://www.smartrm.com), a system for sharing content while retaining control of it.[2]

SmartRM, a viral service based on social networks, enables people to share confidential content in a protected way. The SmartRM software lets users convert confidential files (such as PDFs, videos, or audio tracks) into encrypted MPEG-21 files. Interaction with the protected content through reading, viewing, listening, or printing is limited to only those contacts that have been granted permission and are subject to the set of specified conditions. The service makes it possible not only to know when and how many times a protected file has been accessed by someone, but also to grant or remove permissions dynamically.

The SmartRM system architecture is based on a client–server model. The client is a Mozilla Firefox plug-in that relies on MXM. Most Web browsers available today are capable of initializing, creating, destroying, and positioning plug-ins inside the browser window when certain

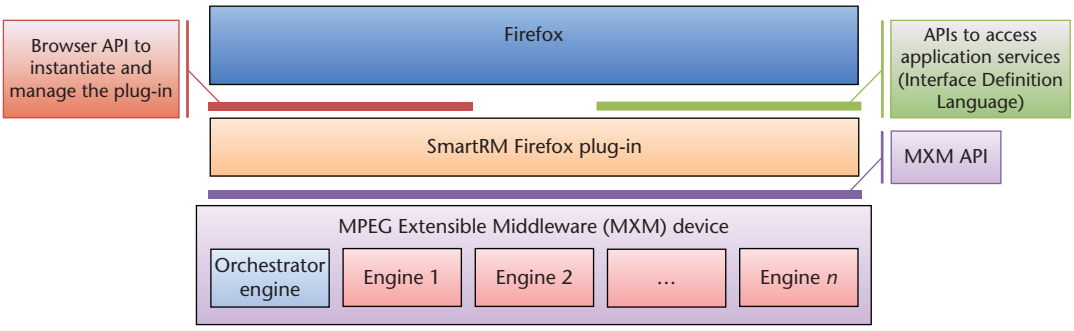

Figure 4. High-level architecture of the SmartRM MPEG Extensible Middleware-based Firefox plug-in.

events occur, often through standard APIs such as the Netscape Plug-in API (see http://en.wikipedia.org/wiki/NPAPI). Figure 4 shows an example of a high-level architecture of the SmartRM client software.

The SmartRM functionalities can be accessed either by code running on an HTML page (such as, JavaScript) or by other Firefox plug-ins, extensions, or components through an API described in Interface Definition Language. 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 manageable modules that can be replaced easily because access to them is made through the standard MXM APIs. Exchanging MXM-protocol messages over SOAP and XMPP does most of the communication between the SmartRM client and the SmartRM server. Both the client and the server are then relieved from the complexity of generating, dispatching, and interpreting such messages, as MXM engines can do these operations.

Open issues

Open issues to examine further include the efficiency of the middleware itself, such as a low overhead that is especially required for mobile devices, and scalability in terms of the number of requests that can be handled by the middleware. Further information on MXM is available at http://mxm.wg11.sc29.org/ as well as in other articles.[3-6] **MM**

Acknowledgments

We thank the following people for their contribution and encouragement to the work presented here: Leonardo Chiariglione (Cedeo.net), Michael Eberhard (Klagenfurt University), Ivica Arsov (Institut TELECOM), Angelo Difino (Cedeo.net), and Wonsuk Lee (ETRI). This work is supported in part by the European Commission in the context of the Alicante project (FP7-ICT-248652; http://www.ict-alicante.eu/).

References

1. I. Burnett et al., ''MPEG-21: Goals and Achievements,'' *IEEE MultiMedia,* vol. 10, no. 4, 2003, pp. 60-70.
2. A. Difino and F. Chiariglione, ''An MXM-Based Application for Sharing Protected Content,'' *1st Int'l MXM Developer's Day,* 2009; http://mxm.wg11.sc29.org/2009/06/an-mxm-based-application-for-sharing-protected-content/.
3. M. Eberhard et al., ''An Interoperable Streaming Framework for Scalable Video Coding Based on MPEG-21,'' *IEEE Wireless Comm.,* vol. 16, no. 5, 2009, pp. 58-63.
4. I. Arsov and M. Preda, ''Including MPEG-4 3D Graphics in Your Application,'' *1st Int'l MXM Developer's Day,* 2009; http://mxm.wg11.sc29.org/2009/06/including-mpeg-4-3d-graphics-in-your-application/.
5. V. Rodriguez-Doncel and J. Delgado, ''A Media Value Chain Ontology for MPEG-21,'' *IEEE MultiMedia,* vol. 16, no. 4, 2009, pp. 44-51.
6. C. Timmerer et al., ''Accelerating Media Business Developments with the MPEG Extensible Middleware,'' *Towards the Future Internet—Emerging Trends from European Research,* G. Tselentis et al., eds., IOPress, 2010.

Contact author Christian Timmerer christian.timmerer@itec.uni-klu.ac.at.

Contact editor Anthony Vetro at avetro@merl.com.