

The Multimedia Home Platform

- an overview

J.-P. Evain

EBU Technical Department

The Multimedia Home Platform (MHP) encompasses the peripherals and the inter-connection of multimedia equipment via the in-home digital network. The MHP solution covers the whole set of technologies that are necessary to implement digital interactive multimedia in the home – including protocols, common API languages, interfaces and recommendations.

This article offers an introduction to the design and harmonization of MHP receivers, starting with a reference model which has been derived from the DVB and UNITEL reference models.

Introduction

At the beginning of 1996, the *UNITEL – universal set-top box* project was launched by the ISIS Programme of the European Commission. The main aim of this project was to raise awareness of the benefits of developing a common platform for user-transparent access to the widest range of multimedia services. Promising progress has since been achieved towards the harmonization of what is now widely called the *Multimedia Home Platform* (MHP).

The MHP Launching Group was born from the UNITEL initiative in order to open the project to external parties via joint meetings. Key representatives of the High Level Strategy Group took part in this group, and this collaboration eventually led to the transfer of these activities to the DVB Project. Two DVB working groups were subsequently set up:

- A commercially-oriented group, DVB-MHP, to define the user and market requirements for enhanced and interactive broadcasting in the local cluster (including Internet access).
- A technical group, DVB-TAM (Technical issues Associated with MHP), to work on the specification of the DVB Application Programming Interface (API).

DVB-TAM is currently considering several API candidates:

- MHEG-5/Java;
- *Mediahighway +*;
- *JavaTV*;
- HTML/Java.

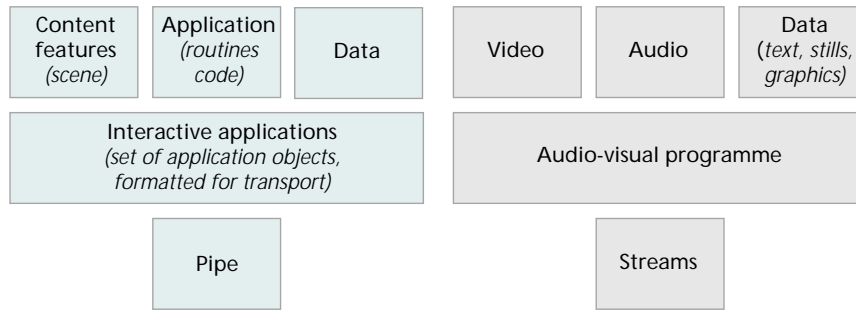
The API chosen by DVB will have to be open, in order to suit the requirements of a horizontal market. It will have to be CA-independent but will support compatibility in a multi-CA environment.

Reference model

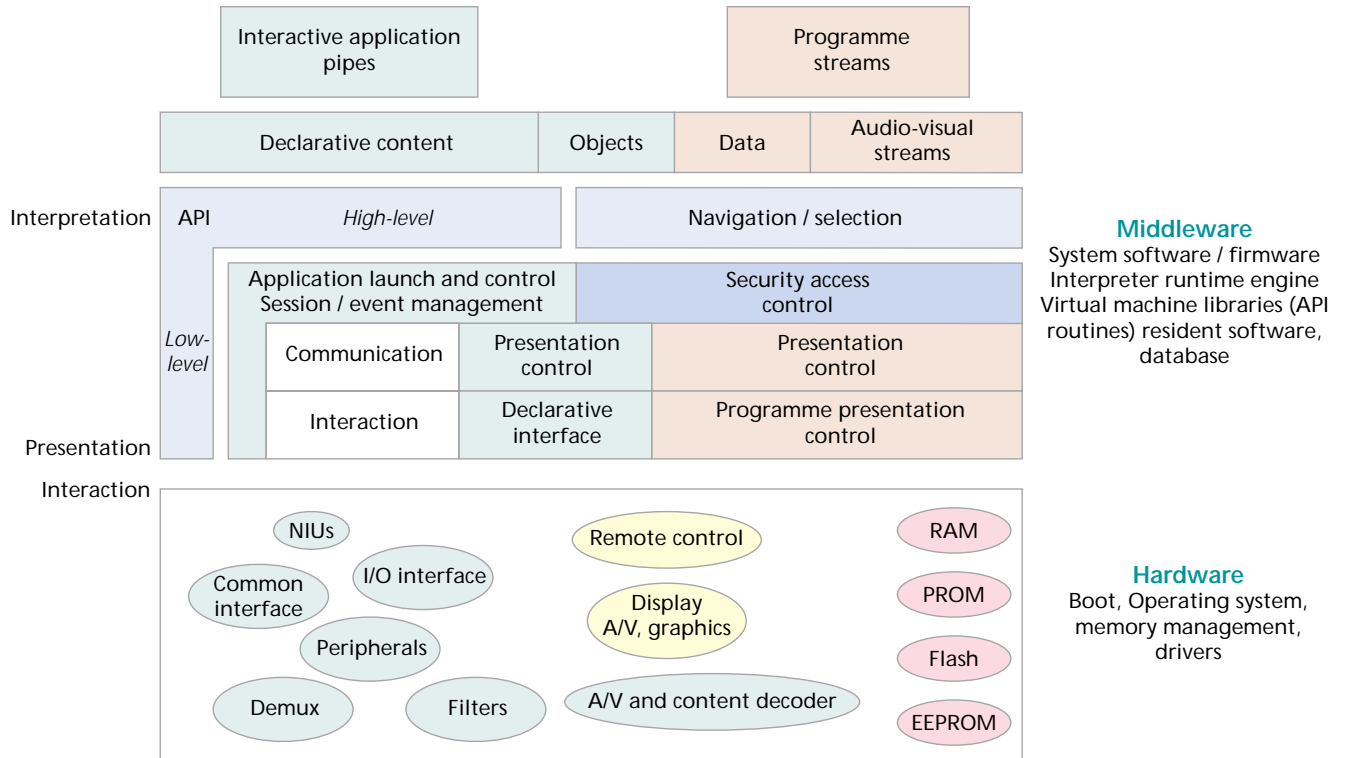
Different reference models have been defined for each MHP system currently in use. UNITEL used object-modelling tools to define the application classes and functionalities that would ultimately identify the hardware and software resources required by an MHP system. With this system, users would be able to access:

- enhanced broadcasting services;
- interactive broadcasting services;
- Internet services.

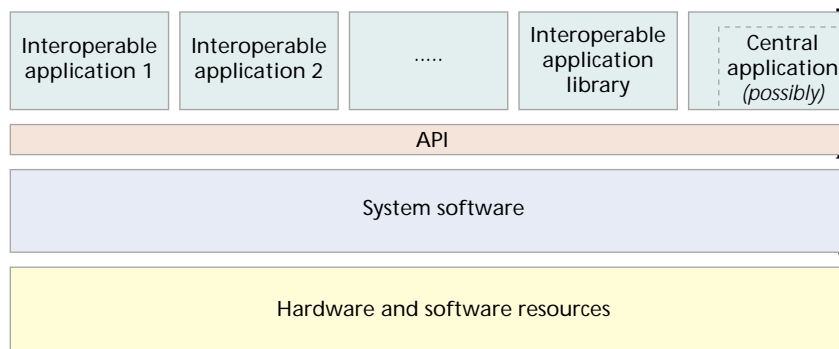
**Figure 1
UNITEL: MHP
hardware and
software
resources**



**Figure 2
DVB-TAM
reference model: system layers.**



**Figure 3
model: a
and middleware
streams.**



**Reference
possible API
or pipes and**

Fig. 1 shows just a UNITEL reference submitted to the group that is system modelling.

DVB-TAM

succeeded in defining a common generic reference model, which is shown in Fig. 2. The reference model shown in Fig. 3 is a combination of the models shown in Figs. 1 and 2.

The reference model shown in Fig. 3 allows the development of high-level APIs and applications, independent of the MHP system infrastructure.

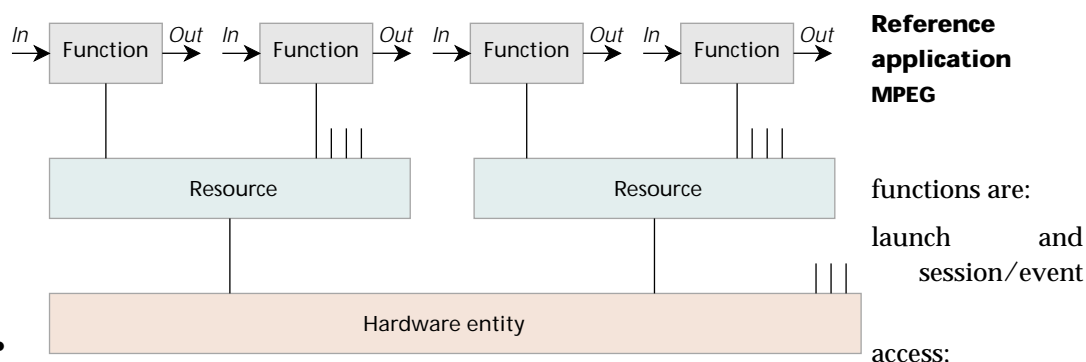
This model offers system modularity through the use of key interfaces. These interfaces will be able to maintain the stability of MHP systems as they evolve – both in terms of hardware and software. Backward compatibility will be supported to the largest possible extent, e.g. by using scalable applications.

Each application that is developed will need to comply sufficiently with the reference model to ensure cross-platform interoperability in a competitive environment. This should result in host platforms where the integrity of the application is protected, and its behaviour is stable and predictable (thus resulting in a high quality of service). The reference model must also define modes for data delivery, memory handling, object handling and instruction execution.

The reference model consists of five layers:

- application (content, script) and media (audio, video, subtitle) components;
- pipes and streams (see Fig. 4);
- the API and native navigation/selection functions;
- platform/system software or middleware, including the interactive engine, the run-time engine (RTE) or virtual machine, the application manager, etc.;
- hardware and software resources, and associated software.

Figure 4
model:
"pipes" and
streams.



The main system

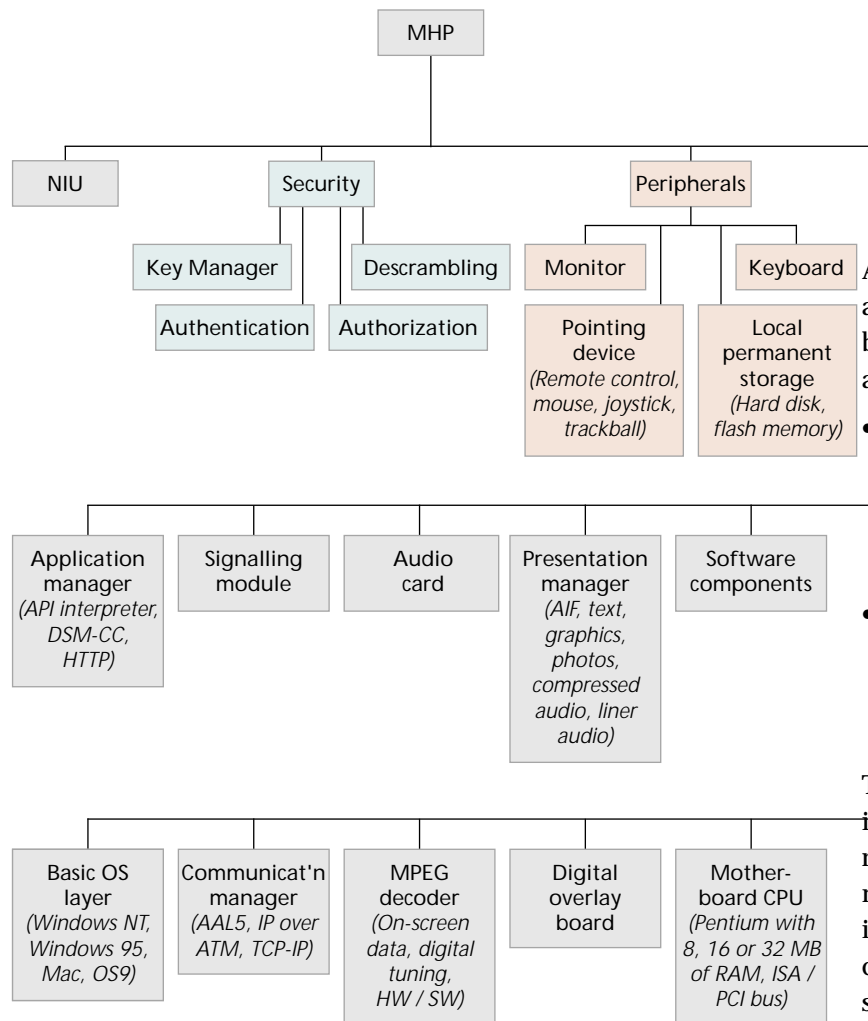
- application control, management;
- security and
- content loading;
- navigation and selection;
- declarative content and streams presentation control;
- communication and I/O control;
- signalling, bit transport, driver and management functions.

Applications

The predictable environment described by the reference model will readily allow applications to be authored and tested. Compliance with the reference model will ensure that applications execute properly, independent of the precise MHP implementation. The integrity of the look, feel and functionalities of each application will have to be ensured; the design of the original application provider must be preserved – irrespective of the platform implementation. It should be possible to design scalable applications that maintain compatibility across a range of receiver implementations.

DVB-TAM defines an application as a functional implementation of an interactive service which is realized as software modules. An application can also be seen as a set of organized functions that request activation of MHP hardware and software resources (see Fig. 5).

Figure 5



DVB-TAM: relationship between hardware entities, resources and functions.

An interactive application is basically built around:

- *application script* (which can be declarative and/or procedural)
- *content/scenes* (declarative interface and media streams).

The declarative interface is the representation of the man-machine interface. It can consist of graphics such as a background design, selection

buttons, still pictures, text etc. Each scene can comprise a set of other scenes, application objects and attributes. The pipes implement the interconnections between the scenes and concatenated functions.

Procedural applications, based on low-level functions and primitives, are used when very strong optimization is required at the host level (e.g. to minimize the platform footprint and maximize the use of the transmission resources). Procedural applications are generally platform-dependent and, hence, each one must be verified on the different host platforms.

Declarative applications use high-level functions and primitives. This allows us to define a platform-independent reference model which can verify whether such applications comply in terms of cross-platform compatibility and performance accuracy.

In reality, applications are neither fully declarative nor fully procedural. As an example, declarative applications can make use of procedural enhancements to improve their performance. This allows us to reduce the size of the application and to reduce its execution time by using routines written in executable code. Platform-independence is ensured by relying on embedded RTEs, virtual machines or other interactive engines. It is more difficult to achieve compliance of the compiled code routine libraries for different platforms, if they are not taken into account at the time of the platform design.

Applications are identified and signalled to indicate their availability, and an appropriate mode of access is presented to the user. Applications are launched automatically or by request. The application presentation can be nominal or down-sized (if scalable), thus maximizing the use of the available resources. Application management encompasses: interruptions, failures, priority modes and dynamic resource allocation. The application must release the system resources it has used, when quitting.

Application delivery mechanisms

Application script and content are grouped together in application objects which are converted into DSM-CC carousel objects. DSM-CC has been standardized by MPEG for the retrieval and transport of MPEG streams, and has been adopted by DVB. DSM-CC UU is the interface that allows us to extract DSM-CC carousel objects from the broadcast stream, or via an interactive access to a remote server.

DSM-CC carousel objects allow one or more application objects to be carried in one module of the data carousel. Objects can be arranged in modules, in order to optimize the performance and use of memory. DSM-CC also includes compression tools to format the application objects and carousel modules, and mechanisms to ensure the secure downloading of the carousel objects.

Definition of the API

DVB-TAM has defined an API as a set of high-level functions, data structures and protocols which represent a standard interface for platform-independent application software. It uses object-oriented languages and it enhances the flexibility and re-usability of the platform functionalities.

An application describes a set of objects according to the definition of high-level APIs. It defines the interface (via the interactive engine) between the applications, and the software and hardware resources of the host. The primitives that are embedded in the application objects are interpreted, and the resources that are requested by the corresponding declarative and procedural functions are activated. The interpreter is an executable code.

UNITEL identified the following API requirements:

- *Openness*: it should be specified in such a way that it can be used in the implementation of other interfaces.
- *Abstraction*: it should not expose its implementation. It should also hide all aspects of the underlying software and hardware.
- *Evolution*: it should be flexible and easily extendible.
- *Scalability*: it should be hardware-independent in order to take advantage of future improvements in hardware and of the characteristics of different hardware implementations. The API itself can be updated or complemented by, for example, adding new libraries (e.g. procedural extensions) using download mechanisms.

According to the application format, low-level and/or high-level APIs will be used to deal, respectively, with procedural and declarative functions:

- *Low-level* APIs are more procedural and tend to access low-level procedural functions. The API interprets the application function or primitive but also knows how to activate the resources.
- *High-level* APIs are more declarative. The higher the level of abstraction declaration (i.e. the hiding of the system implementation), the stronger is the system independence. The API interprets the application function or primitive but does not need to know how the corresponding resources will be activated.

The specification of an open API should lead to the embedding of this hardware-independent facility within DVB receivers.

DVB-MHP has stated that the API should:

- support applications that are locally stored as well as those that are down-loaded in either real time or non-real time;
- preserve the “look and feel” of the application;
- enable access to databases (e.g. DVB-SI);
- allow room for competition among implementers.

An open and evolutionary (modular, portable, flexible, extendible) API is vital for the implementation of platforms in an unfragmented horizontal market. This will allow different content and service providers to share different implementations of compliant platforms.

Navigation/selection

The API can also be used by resident programmes such as the embedded navigator function that allows a first level of navigation when the receiver is switched on. APIs can also be used to manipulate streams and to enable basic functions such as channel/programme hopping or “zapping”.

The navigator can also be implemented in executable code, in which case it does not need to use the API and its interpreter. In the DVB-TAM model (*Fig. 3*), the navigator has consequently been placed at the same level as the API to enable boot access to pipes and streams.

The basic navigator should:

- list all the programmes available, without discrimination;
- allow user-friendly access to these programmes by offering appropriate shortcuts (e.g. specific remote control buttons).

Enhanced navigation can then be provided by means of electronic programme guides, possibly including such enhanced facilities as user profiles and bookmarks.

Application launch and control

The application launch function, and the application and presentation control functions, provide the facilities to run an application. The application code may be already resident in the STU or it may be obtained via a session to a remote server. After loading, the application is launched and execution is transferred to the new code.

It is the application manager's responsibility to:

- check the code and data integrity;
- synchronize the commands and information;
- adapt the presentation graphic format to suit the platform display;
- obtain and dispose of the system resources;
- manage the error signalling and exceptions;
- initiate and terminate any new sessions;
- allow the sharing of variables and contents;
- conclude in an orderly and clean fashion.

Security functions

DVB has defined the following security requirements (although the security model itself has not yet been defined):

- The API should be accompanied by a system which incorporates a common security model for the applications and data. It should enable full compatibility between the signals transmitted by the different broadcasters and content providers.
- The security model should include a description of the procedures and entities that must be put into place to support the associated secret management issues. It should be independent of CA systems. The MHP API should give access to CA functions, if and when required.

Among the important security aspects to be addressed are (i) machine protection against abusive requests for system resources (e.g. excessive demands on memory) and (ii) protection against non-authorized access to data (e.g. private data).

Middleware

The possibility of implementing the API by means of middleware is directly related to the application format (whether declarative or procedural) and the use of either low-level or high-level APIs. Each middleware implementation will be tailored for optimum use by the host platform.

There can be different ways of implementing the interactive or run-time engine which, in general, is required to support the following:

- the script and content interpreters;
- the libraries;
- the event manager (remote control and other devices, user actions, markers, timers, the handling of error conditions);
- the loader.

Depending on the API used, the RTE offers low-level interfacing with the system hardware and software resources. The RTE may call up resident programmes which can use a native platform-dependent interface to improve the system performance and to diminish the operational constraints (e.g. the size of the downloaded application objects) at the declarative application level. The RTE is executable code, adapted to each platform and aligned with the reference model.

The virtual machine can be used to emulate declarative interfaces but it is generally used to run procedural functions (e.g. complex calculations, information and text processing, data extraction) or resident programmes that enhance the declarative interface of the application.

The use of run-time engines and virtual machines allows the API to support the platform independence of applications.

Hardware and software resources

The Multimedia Home Platform must be user-friendly. A minimum set of peripherals includes a display, a pointing device and, optionally, a keyboard and local internal/external permanent storage. The connection of these peripherals should be on a “plug and play” basis.

Internal resources in the MHP receiver include the front-end, demux, decoders, filters, a common interface, a communication interface, a CA system, memory and associated drivers.

DVB has currently defined three profiles. These require a minimum of 1 Mbyte of Flash-ROM and 1 Mbyte of DRAM, up to a maximum of 16 Mbytes of Flash-ROM and 32 Mbytes of DRAM, coupled with a CPU speed from 20 mips to more than 100 mips. It is sometimes specified that, for example, 70% of CPU time should be devoted to run the applications, with the remaining 30% being used for the system management.

Stored in ROM are the following:

- the API interpreter;
- the libraries;
- the run-time engine and/or virtual machine;
- the loader,
- the system tools;
- the file system;
- the firmware;
- the operating system (boot-up, memory management, task scheduler, resource identification, alarms and timers, resource locking);
- the drivers;

- the navigator.

The use of Flash memory allows a limited number of revisions to be downloaded. Flash memory can be partitioned in order to reserve memory segments for different memory uses and, for example, to refresh selectively only part of this memory.

The applications delivered via the DSM-CC carousel are stored in RAM. RAM is also used for video/audio/data decoding and buffering, for dynamic platform management (e.g. process queues, stacks), for data, and for persistent storage of data such as application variables.

The basic system configuration and factory settings are usually stored on EEPROM (using less than 10 Kbytes of memory).

Migration and future operational issues

Migration is primarily the process by which a population of receivers based on proprietary software systems are all converted to a population of MHP receivers which use the common DVB-MHP system and, particularly, the API. According to DVB-MHP: *“the migration process will be initiated when service providers have begun to offer services in a format that is compatible with the MHP solution”*.

DVB receivers already make use of a large number of common elements including the modulation and multiplexing schemes, MPEG-2 audio and video, the DSM-CC UU interface and protocols, the Common Interface (for conditional access and other uses) and the DVB-SI.

Nevertheless, a number of elements differ between implementations:

- the mechanisms which combine application script and code, data and contents into application objects;
- compression tools;
- the format of procedural functions;
- libraries (e.g. procedural extensions, graphics);
- data carousels or other cyclic data delivery mechanisms;
- down-loading procedures and tools;
- memory allocation and management (e.g. application queues and garbage collection);
- interactivity;
- the formats of the variables;
- security procedures.

DVB has requested that, in a multi provider/multi-application environment, the MHP solutions should be based on the separation of data. This will enable different authorized applications to use this data (if in a common data format), particularly as different applications can be implemented to accomplish the same task. It will be possible to reserve part of the data for specific applications.

At the system level, migration should be considered carefully in order to achieve the largest possible use of the DVB-TAM API. This will help to maintain receiver portability and mobility, particularly for digital terrestrial broadcasting where the limited number of programmes is another reason to support solutions which favour a horizontal retail market. The consumer will probably not invest in several receivers if the added content value is limited.

Migration will not be “easy”. It will require substantial effort and collaboration, e.g. to maintain backward compatibility with currently-deployed plat forms.

The wide use of a common API will raise new operational issues. There will be significant changes in the modes of operation of the service providers who, currently, target well defined and proven receiving platforms. In order to

accommodate different implementations of platforms, all using a common API, we will have to follow certain generic guidelines:

- applications will have to be down-loadable and should not rely on persistent storage when the application is not active;
- common libraries (procedural extensions, graphics etc.) and resident programmes should be embedded in order to limit the size of the application;
- application, data and declarative interfaces should be organized in accordance with common generic schemes;
- the same data carousel object format should be used, and the same mechanisms should be applied for the delivery of these objects over the streams being broadcast;
- common compression schemes should be adopted;
- similar start-up and closing application procedures should be used;
- the amount of re-inscriptible Flash memory that is available should be defined.

MHP platforms will evolve and will be able to support more complex and, hopefully, scalable applications. These may require further API extensions. Future evolution should certainly aim at increasing the degree of commonality of these system elements and procedures. This should contribute towards increasing the cost-effectiveness of the system. It should also help to increase the lifetime of the equipment.

Acknowledgements

The Author would like to thank the DVB-MHP and DVB-TAM members who patiently agreed to describe their respective systems, and also Philippe Bridel (France-Telecom/CCETT) who developed the UNITEL architecture and reference model.

Bibliography

- [1] G. Luetteke (Philips): **The DVB Multimedia Home Platform**
MUST'98, May 1998.
- [2] ITU-R document 11A/107-11: **Examples of API structure**
ITU/EBU, April 1997.
- [3] DVB document TAM 029 rev. 5: **TAM Reference model.**
- [4] M. Echiffre et al. (CSELT): **MHEG-5 – Aims, concepts, and implementation issues**
IEEE 98.
- [5] J. van der Meer and C.M. Huizer (Philips): **Interoperability between different interactive engines for digital television, problems and solutions**
Philips, June 1997.
- [6] DAVIC 1.4 specification – Part 9
- [7] **An API and an Operating System for Interactive Digital TV Decoders**
OpenTV, January 1998.
- [8] N. Birch (S&T): **The UK MHEG profile as a response to DVB TAM RSD1**
UK Digital Television Group, April 1998.
- [9] UNITEL Deliverable 3.1: **Characterisation and specification of the architecture reference model**
CCETT on behalf of the UNITEL Consortium, December 1997.

And in this issue ...

[10] A. Mornington-West: **MHEG-5 and Java – the basis for a common European API?**
EBU Technical Review, No. 275, Spring 1998.

Abbreviations

API	Application programming interface
A/V	Audio / video (visual)
CA	Conditional access
CCETT	(France Telecom's) <i>Centre Commun d'Etudes de Télédiffusion et de Télécommunications</i>
CPU	Central processing unit
DAVIC	Digital Audio-Visual Council
DRAM	Dynamic random access memory
DSM-CC	(ISO) Digital storage media – command control
DSM-CC UU	DSM-CC, user-to-user
DVB	Digital Video Broadcasting
DVB-SI	DVB - Service Information
EEPROM	Electrically-erasable programmable read-only memory
I/O	Input/output
ISO	International Organization for Standardization
JAVA	Programming language for the WWW (developed by Sun Micro-systems)
MHEG	(ISO/IEC) Multi- and Hyper-media coding Experts Group
MHP	Multimedia home platform
mips	Million instructions per second
MPEG	(ISO) Moving Picture Experts Group
NIU	Network interface unit
ROM	Read-only memory
RTE	Run-time engine
TAM	(DVB) Technical issues Associated with MHP



Jean-Pierre Evain graduated from ENSEA, Cergy-Pontoise (near Paris), in 1983. His first employment was with CCETT in Rennes and, in 1992, he moved to Geneva to join the EBU Technical Department as a Senior Engineer.

Currently, Mr Evain works in the EBU division called "New Systems and Services" and is a member of various BMC project groups. He represents the EBU in various international consortia and in European collaborative projects.

Concerning his MHP and API activities, Jean-Pierre Evain is the project Manager of the CEC DG3 UNITEL project. He chaired the MHP Launching Group that eventually led to the transfer of MHP activities to DVB. He launched the ICT-SB project group on MHP harmonization. He is now the Secretary of the DVB-MHP requirement group and is actively following the DVB-TAM and DAVIC activities on the API. He also represents the EBU and UNITEL in the ETSI-MTA (Multimedia Terminals and Applications) group.