A MHP RECEIVER OVER RT-LINUX FOR DIGITAL TV

J.C. López, C. López, A. Gil, J.J. Pazos, M. Ramos, R.F. Rodríguez
Depto. de Enxeñería Telemática, ETSE de Telecomunicación
Campus Universitario s/n, University of Vigo (Spain)

ABSTRACT
The growth of interactive digital TV heralds a seamless convergence of media, telecommunications, and information technology, offering viewers increasingly exciting and interactive programming. The need of standardization in this field led DVB to head the regulator project, defining standards for digital video broadcasting and interactive services in all transmission networks, and recently for a generic common interface (the MHP API) to enable interoperable applications to be downloaded from broadcast networks and executed on receivers with specific hardware and software implementations from any manufacturer. The software architecture of a MHP receiver consists of a multitask real-time OS that gives support to the middleware below the MHP API. This middleware consists of an application manager, the communications protocol stack and a Java VM together with suitable APIs. Conventional implementations of MHP receivers make use of a proprietary OS. In this paper, we present our experience designing a prototype based on an open platform over RT-Linux.

KEYWORDS

1. INTRODUCTION

There is no doubt that Television is still the most important mass media communication, strengthened by the arrival of the digital TV. Not only does digital TV enable to deliver many more programs with much higher quality than current analog TV, but the enhanced possibilities of interactivity make the technology increasingly attractive as well. On the other hand, although the Internet is growing exponentially and becoming another important mass media, the arrival of the digital television will trigger the integration between television and Internet.

The main problem in digital TV is that most of existing analog TV sets are totally incompatible with present TV standards. One evolutionary approach is the use of a special receiver between the digital TV network and the analog set, known as set-top-box (STB), which gives the suitable support for signal conversion and interactivity to the user. In order to get a successful integration of the new services in the TV market, it is necessary to standardize the technologies involved because only the compatibility guarantees permit to reduce costs and reach as many customers as possible. Nowadays, the main regulator body in this field is the DVB (Digital Video Broadcasting) consortium. Since the DVB project was established in 1993, it has defined a set of open standards for digital video broadcasting and interactive services in all transmission networks including satellite, cable and terrestrial.

Services and networks based on these standards are being widely deployed all over the world. Recently, the scope of the DVB project was extended to work on an ambitious generic, common API to enable interoperable applications to be downloaded from broadcast networks and executed on receivers with specific hardware and software implementations from any manufacturer. This standard is known as MHP (Multimedia Home Platform) (DVB Consortium 2001a, DVB Consortium 2001b). The core of the MHP specification is based around a platform known as DVB-J, which includes the Java Virtual Machine (JVM) as specified by Sun Microsystems. A number of Java packages (also called APIs) provide interfaces between the JVM and the features and functions of a DVB receiver and the networks to which they are connected. Content formats, network protocols and signaling are also specified.
In this paper, we present our experience designing and implementing a prototype of a MHP receiver based on an open platform over RT-Linux.

2. IMPLEMENTATION ISSUES

The software architecture of a MHP receiver (see Figure 1) consists of an operating system (OS) that gives support to the middleware elements placed below the upper MHP API: the application manager, the communications protocol stack and the JVM. Besides, between JVM and the MHP API, the aforementioned set of APIs (Sun Java, HAVi, DAVIC and DVB specific) must be included. On the other hand, a graphic user interface (GUI) must be provided in order that the user can interact with the system. This facility is usually called Home Navigator.

![Figure 1. Software architecture of a MHP receiver](image)

### 2.1. The operating system

Conventional solutions for the first-generation receivers are usually based on embedded real-time operating systems. These operating systems are traditionally designed according to two basic principles: fulfillment of real-time task requirements, because these systems are normally used in hard real-time applications, and adaptability to a specific application or type of applications (Berger 2001).

Nevertheless, this kind of operating systems are not suitable for the new devices, like our prototype, because apart from real-time requirements fulfillment, operating system services, typically only available on general-purpose operating systems, are also necessary. Until recently, there were no operating systems that fulfilled all these requirements.

Today, Linux approach seems a valid solution to us. Linux was not originally designed to be a real-time operating system. It was intended for desktop and server use, where there are no real-time requirements. So, Linux evolved as a general-purpose operating system. Nevertheless, the popularity and power of Linux is driving developers to try to use it in a growing number of systems, including embedded and real-time systems. Owing to its open source, Linux is quickly responding to the demands placed on it, both in terms of reduced size for embedded and in terms of improved real-time performance.

In this context, some solutions like RT-Linux (Barabanov 1997) appear, providing deterministic worse-case response-times fulfillment by means of a simple real-time kernel and running conventional Linux on it during the idle time. With this solution, the system can respond to events in real time and provides simultaneously the complete services of conventional Linux to the applications. The utilization of RT-Linux on our prototype provides us the following advantages:

- It is a proven, stable and robust operating system with a strong developer community.
- Open source code availability has the advantage of great design flexibility.
• The availability of different versions of middleware components (as, for example, different implementations of the Java VM), which allows to easily compare features and performances of these versions.

RT-Linux is a relatively simple kernel that manages and communicates real-time tasks, where conventional Linux is a low-priority task. We propose to use a simplified version of Linux that includes only the necessary elements (the protocol stack and the JVM among them) and implement the application manager like a high-priority real-time task that communicates with Linux through the kernel.

In this way, the design and implementation of the MHP receiver limit to the set of APIs over the existent JVM, that is, the MHP API, the application manager and the navigator application.

3. THE MHP API

This standard covers many aspects, from tuning a transport stream in a network interface to displaying adequately provider contents. For this reason, and especially for compatibility, the interface for the applications must be wide.

So, the MHP standard includes APIs with various origins, e.g. the HAVi standard (HAVi 2000) that defines the graphic interface or the appendixes from the DAVIC standard (DAVIC 1999) related to low-level functions into the STB, like transport stream tuning or conditional access.

However, the APIs that arouse most interest, owing to their novelty, are those specifically defined for MHP, included into the package `org.dvb`. This package provides the two novel basic functions included in the MHP standard: the ability to run remote applications and provide interactivity to the user.

The package `org.dvb.lang` enables us to obtain bytecode files of remote applications from an objects carousel arriving into the transport stream. This feature is essential since it enables the TV set to be more than a simple audio and video player, although we have noticed that applications should be simple enough due to the cyclical nature of the carousel that slows down obtaining the associated objects. Nevertheless, we have verified that the implementation of a cache mechanism reduces this latency.

But perhaps the most important package is `org.dvb.event`, as it enables the applications and the rest of system elements to receive events from the users providing the interactivity. This is the key function in the new television. This package defines one fundamental entity, the event manager, which receives all the user events and distributes them among the applications. This API provides compatibility with the standard event mechanism of `java.awt` and defines a new mechanism for applications that do not use `awt`. The API also defines two ways for applications to access to the events: exclusive and shared access. However, while the exclusive access is clearly justified in the MHP standard, the shared access is confusing since for a user it is not intuitive that one button pressing is going to be sent to several applications at the same time. For this reason, we argue for implementing a focus in such a way that several applications can request the same events, but they will be only received by the application previously selected by the user, that is, the application that has the focus in each moment.

As we shall see in the next section, another important feature of the API `org.dvb.event` is that enables the user to modify the life cycle of the applications, that is, participate in the processes of loading, running and finishing applications.

4. THE APPLICATION MANAGER

The application manager is another novel entity present in the MHP standard. This entity manages the life cycle of the applications in the system. The standard also defines the life cycle of the services, but this does not add anything new since it is the classic concept of the TV channel change, although improved.

The life cycle of the applications introduces the key element that brings the TV set closer to the computer world, smoothing the way for the integration. This life cycle (see Figure 2) is defined in the Sun Java TV standard (Sun Microsystems 2000); hence its similarity to that of a Java applet, with little changes in the definition of the states that bring it nearer the television environment where it will be executed.
On the other hand, this cycle favors the adaptability to the typical running environment of the STB, where various elements can act on its life cycle. The application manager is the entity responsible for receiving the orders that will act on the applications. It is responsible for initiating, running, pausing and destroying MHP applications. The manager makes use of the API \textit{Xlet} to communicate with the applications. The state changes of the applications (notified to the manager through the interface \textit{XletContext} of the API \textit{Xlet}) can be caused by its internal working (for example, after doing its job) or come from external sources. The contents provider can use the signaling into the transport stream for modifying the life cycle of an existing application, adding new applications or removing other. The manager must monitor the stream to detect these signaling changes. On this matter, we could observe that a more efficient implementation would consist of using a listener interface to the system responsible for decoding and managing the transport stream. In our opinion, a possible improvement would be that the provider sent a signal when an application gives up being available in the system. This would avoid the manager having to keep a copy of the previous signal and check what applications are present in the system every time a new signal is received.

Another application in the system can also acts on a given application by means of the \textit{application manager}, making use of the API \textit{org.dvb.application} (Application Listing and Launching), but only if it has the corresponding permissions.

Finally, as we said in the previous section, the user can also modify the life cycle if the application has defined user events. However, in this case, we think that the most suitable decision is that the communication between the user and the application manager is through the \textit{event manager} and that in turn through the \textit{Home Navigator} also using the API \textit{org.dvb.application}. Thus, the navigator conceals the application manager and its details from the user.

5. CONCLUSIONS

The software architecture of a MHP receiver consists of a multitask real-time operating system that gives support to the middleware elements placed on top of it and below the MHP API: the application manager, the communications protocol stack and the JVM together with a set of various APIs. Conventional implementations of the receiver make use of a proprietary OS, whose disadvantage lies on the need of developing all the middleware for it.

In this paper, we present our experience designing and implementing a prototype of a MHP receiver based on an open platform over RT-Linux. This decision allows us to reduce developing costs since an important part of middleware (protocol stack and JVM) is already available and, on the other hand, the computational capacity of the processors is higher and higher while the price is decreasing. So, the rest of the paper focuses on some aspects about the experience in designing and implementing the set of APIs over JVM, the event manager, the application manager and the navigator.

REFERENCES


DAVIC, 1999. \textit{DAVIC Specification 1.4.1 part 9}.

DVB Consortium, 2001a. \textit{Multimedia Home Platform (MHP) 1.0.1}.

DVB Consortium, 2001b. \textit{Multimedia Home Platform (MHP) 1.1}.


627