



STREP

**FP6-2005-IST-61-045410**

**MOBISERVE**

**New mobile services at big events using DVB-H  
broadcast and wireless network**

**Deliverable:**

**Terminal System Architecture Specification, for Olympic trial  
D4.1**

Due date of deliverable: M6  
Actual submission date: 27.04.2007

Start date of Project: 01 September 2006

Duration: 24 months

Responsible WP: Streamazzo

Revision:

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)		
Dissemination level		
<b>PU</b>	Public	X
<b>PP</b>	Restricted to other programme participants (including the Commission Service)	
<b>RE</b>	Restricted to a group specified by the consortium (including the Commission Services)	
<b>CO</b>	Confidential, only for members of the consortium (excluding the Commission Services)	

## 0 DOCUMENT INFO

### 0.1 Author

Author	Company	E-mail
Pierre Pleven	STZ	pierre.pleven@streamezzo.org
Philipp Steckel	TU-BS	steckel@ifn.ing.tu-bs.de
Marius Spika	TU-BS	spika@ifn.ing.tu-bs.de
Jianzhong LI	FTB	ljianzhong.ext@orange-ftgroup.com
Joep van Gassel	PRLE	joep.van.gassel@philips.com
Patrick Niessen	NXP	patrick.niessen@nxp.com
Vojkan Mihajlovic	PRLE	Vojkan.mihajlovic@philips.com

### 0.2 Documents history

Document version#	Date	Change
V0.1	17/04/07	Starting template, split of combined deliverable
V0.2	20/04/07	Skeleton generation (TU-BS)
V0.3	25/04/07	ISAP module descriptions added (FTB)
V0.4	26/04/07	Content Handler descriptions added (PRLE)
V0.95	25/04/2007	Addition STZ+NXP
V0.97	27/04/2007	Addition NXP
V0.99	27/04/2007	TU-BS additions and final document (TU-BS)
V1.0	27/04/2007	Final revisions (TU-BS)

### 0.3 Document data

Keywords	
Editor Address data	Name: Pierre Pleven Partner: Streamezzo Address: 21 BD Victor Hugo 75016 Paris France E-mail: <a href="mailto:pierre.pleven@streamezzo.org">pierre.pleven@streamezzo.org</a>
Delivery date	

### 0.4 Distribution list

Date	Issue	E-mailer

## Table of Contents

<b>0</b>	<b>DOCUMENT INFO</b>	<b>2</b>
0.1	<b>Author</b> .....	<b>2</b>
0.2	<b>Documents history</b> .....	<b>2</b>
0.3	<b>Document data</b> .....	<b>2</b>
0.4	<b>Distribution list</b> .....	<b>2</b>
<b>1</b>	<b>INTRODUCTION</b>	<b>5</b>
<b>2</b>	<b>TARGET LOGICAL TERMINAL ARCHITECTURE</b>	<b>6</b>
2.1	<b>Olympic Architecture</b> .....	<b>6</b>
2.2	<b>Components of the Olympic architecture</b> .....	<b>8</b>
2.2.1	IPDC STACK .....	8
2.2.2	IP DATACAST NOTIFICATIONS .....	9
2.2.3	METADATA DATABASE COMPONENT .....	9
2.2.4	CONTENT HANDLING COMPONENT .....	10
2.2.5	ISAP DATA RECEPTION .....	10
2.2.6	ISAP FEEDBACK HANDLING .....	11
2.2.7	ISAP SYNCHRONIZATION.....	11
2.2.8	ISAP INTERACTIVE SERVICE MANAGEMENT .....	11
2.2.9	MEDIA PLAYER / RECORDER AND CODEC .....	12
2.2.10	IP DATACAST ESG PRESENTER .....	13
2.2.11	RICH MEDIA ENGINE.....	13
2.2.12	DATA MODEL PRESENTER .....	13
2.2.13	APPLICATION.....	14
2.3	<b>Advanced Architecture Optional Components</b> .....	<b>14</b>
2.3.1	JAVA VM .....	14
2.3.2	JAVA ADAPTER.....	14
<b>3</b>	<b>HARDWARE PROTOTYPES</b>	<b>15</b>
3.1	<b>DVB-H Phone General Introduction</b> .....	<b>15</b>
3.2	<b>Hardware Specifications</b> .....	<b>17</b>
3.3	<b>Software Sub-system</b> .....	<b>18</b>
3.3.1	LIMITATIONS .....	19
3.3.2	PDA BASED PROTOTYPE .....	19
3.3.3	TERMINAL DEVICE .....	19
3.3.4	RF RECEPTION.....	19
3.3.5	ESG INTERPRETER / AV PLAYER.....	20
3.4	<b>Rich Media Prototype</b> .....	<b>20</b>
3.5	<b>Roadmap to the Olympic demo terminal</b> .....	<b>22</b>

## **4 REFERENCES**

**24**

# 1 Introduction

This document describes the system architecture of the MOBISERVE end user terminal which is envisaged to be demonstrated during the Olympic trials 2008 in Beijing.

All relevant architectural layers from the hardware to the interactive applications are in the scope of this deliverable. As far as detailed descriptions are concerned the major embedded components are described in this document. Nevertheless, components in the area of Rich Media technologies will be described in detail in Delivery 4.2 [1] together with backgrounds, which lead to decisions for particular solutions implemented in work package 4, will implement.

In addition, a first hardware prototype is presented in chapter 3, as well as a tentative roadmap to the form factors necessary for the trials.

In a first step, this document presents a common logical terminal architecture that is derived to a detailed software architecture describing individual components and interfaces later on. All architecture developments reflect the various properties and requirements of the target terminal platform, including novel concept of ISAP that ensures tight synchronization between media from different sources on terminal display.

In order to secure development road map, a first rich media demonstrator has been realized in order to test several application features in liaison with WP2 and WP5. During the development of the terminal architecture, a two-profile approach has been agreed on. The concept is to decouple the terminal software developments for the Olympic trials and research-oriented demonstrations on exhibitions and conventions. Accordingly, the "Olympic Architecture" is envisaged to be used for the trials during the Olympic games in Beijing whereas the "Advanced Architecture" is used during research-oriented dissemination activities. A roadmap toward terminal provisioning for the Olympic trial will be described.

## 2 Target logical terminal architecture

In this chapter, two terminal architectures will be presented, Olympic Architecture and Advanced Architecture. These two architectures represent two successive, distinct but highly related steps in the MOBISERVE project roadmap. Indeed, the overall terminal architecture can be thought as made of three parts, namely the basic components part, the Rich Media Part and the Java part. Thus, the Olympic Architecture is made of the basic components part and the Rich Media part, while the Advanced Architecture is simply made of these three parts. Consequently, the Advanced Architecture is in fact the extension (or enhancement) of the Olympic Architecture.

This distinction is mainly due to the following facts:

- The Rich Media is rather mature technology while Java technology envisaged in MOBISERVE project still represent some research aspects;
- The uncertainty to use the under developing terminal platform directly incorporated with the Java feature for Olympic trial;
- The tight Roadmap of the Olympic Games, thus the one of MOBISERVE project. Indeed, the project needs to start to implement the Olympic trial related sport service scenarios.

Therefore the terminal of the Olympic Architecture will be used for the Olympic trials while the terminal of the Advanced Architecture will be presented at demonstrations and exhibitions.

In this document, only the general functional description of different components will be given. Detailed description of the software components will be given in the deliverable D4.2 [1].

In section 2.1, the description of the Olympic Architecture will be given. The description of the Advanced Architecture will be given in Section 2.3.

### 2.1 Olympic Architecture

You'll find hereunder the description of each individual module and the main interfaces between them from top to bottom of the system architecture.

For improved readability we reproduce here the general overall figure, numbers refers to the paragraphs of detailed description.

The system architecture block diagram below shows the main modules of the MOBISERVE terminal. The modules are situated in roughly three different levels of the system architecture:

- **HW / Device drivers:** this level contains the software device drivers that allow the higher level components to abstract from the hardware of the platform.
- **Common Libraries:** this part contains mainly (third-party) off-the-shelf software libraries (e.g. shipped with the platform) used by higher levels of the MOBISERVE architecture.

- **Middleware:** this part contains the modules between the Common Libraries and the Service API where the ISAP and Content Handler sub systems are key components.
- **Application:** the Service API provides the abstraction used by the applications to implement the novel interactive rich-media service scenarios for the public visiting sports events.

The MOBISERVE project extends the current state-of-the art with respect to mobile terminals in particular the Middleware and Application levels of the system architecture.

The modules depicted in gray denote optional modules of the architecture that can be included in an instantiation of the Advanced Architecture as described earlier. Another instantiation, without these optional components, is used for the Olympic trials.

The Interactive Service Application Platform (ISAP), as discussed in more detail later in this section, consists of the following modules:

- ISAP Interactive Service Management
- ISAP Synchronization
- ISAP Data Reception
- ISAP Feedback Handling

As can be seen from the diagram below, the ISAP modules are tightly coupled with the Rich Media and Java components on the one hand and the Content Handler sub system on the other hand.

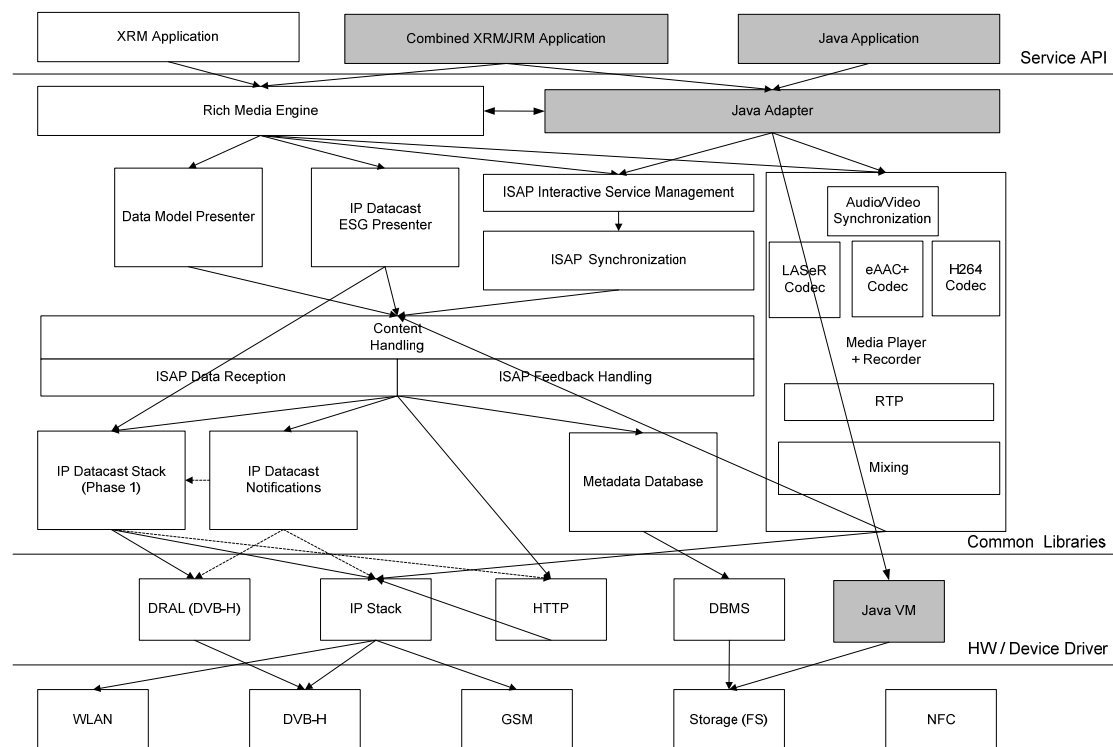


Figure 1 Mobiserve Architecture Diagram

## 2.2 Components of the Olympic architecture

High-level descriptions of each of the boxes is the following

### 2.2.1 IPDC Stack

IPDC Stack includes two main families of function, the ESG Engine and the Service Manager, this component is compliant to DVB-IP Datacast specifications. In Mobiserve this component will be targeted to S3 architecture.

#### Functions of ESG Engine

- Provides controlled access to the ESG database
- Stores and maintains entries in the ESG database
- Service discovery
  - Sequential search of XML tree
  - Acquisition of complete XML service database
- Manages received XML pages
  - XML parsing
  - Binary decompression
  - Data decoding
- Controls ESG fragment acquisition
  - Requests File Transport Stack to find and deliver specific ESG data
  - Deals with updates

#### Functions of Service Manager

- Maintains a list of services currently available to user
  - May vary due to moving between DVB-H cells
- Provides functionality to start/stop services
- Supports a number of application features
  - Theme Filtering
  - Favourite Channel List
  - Parental Control
  - Service Locking

#### Interfaces

- **Initialisation API**
  - Interface for system boot-strapping
- **Middleware (MW) API**
  - Interface to be used by application developers
- **DRAL/TSC API**
  - Interface to DVB-H front-end hardware
- **OSAL API**
  - Interface to platform operating system
- **DRAFS API**
  - Interface to platform non-volatile storage (file system)
- **FILET API**
  - Interface to broadcast file interface

## 2.2.2 IP Datacast Notifications

Currently the DVB IPDC specifications specify how Audiovisual content and files are transported in a broadcast environment using respectively the RTP and FLUTE transport protocols. For short messages, also called Notifications, there is currently no suitable solution. Examples of such notification in the context of MOBISERVE are for instance sport score updates and traffic information. Currently such a Notification framework is specified within the DVB CBMS working group. The specification is expected to be finished towards the end of 2007.

The role of the Notification component is to receive the Notification messages from the broadcast channel and pass the Notification payload to other blocks that make use of them (in our case this is the content handling component). To realize this it will make use of the IPDC stack to select (Notification) services. This is the control part, offering the possibility to subscribe to Notification messages, the actual Notification messages itself are received through the IP (UDP) stack. Additionally the Notification block has the function to filter out messages that are not relevant to components that make use of Notification.

The actual interface will allow users of the component to subscribe and unsubscribe to certain Notification messages (possibly based on specific filter criteria). The Notification component will intern notify users of relevant Notification message (payloads). The API exposed by the Notification block is abstracted by the content handling which takes care of all data pipes.

## 2.2.3 Metadata database component

The purpose of the metadata database component is threefold. Firstly, it should provide efficient access to information. Secondly, it should enable efficient access to information required by content handling component to manage the data flow from and to lower-level and higher-level components. Thirdly, it should interpret the data from the underlying DBMS component and generate meaningful information for the content handling component or the one that can be understood by the rich media engine or by the application.

The interface used to communicate with a DBMS employed in the terminal should abstract away from the exact implementation of DBMS component. This can be achieved by, e.g., using abstract *object-like* components that can be ported on different DBMS implementations. Higher-level interface should enable the access to the metadata database information using a set of *get-like* and *set-like* methods. These methods should be an instantiation of abstract methods, i.e., access to new information or new DB structures should be possible.

The most important units of the metadata database (also reflected within the underlying DBMS), assuming that the focus is on big sporting events, i.e., Olympic Games (OG), are:

- Personal preferences unit - handles information about the user preferences, including preferences related to services provided by different channels.

- Content handling metadata unit - handles relevant information about the data flow from the user/application to device drivers and the other way around, as well as for the synchronization between the metadata database and other lower-level components (especially IPDC stack).
- OG schedule and athlete unit - handles relevant information about the OG that is of interest to the user. It includes events, athletes, etc., taking place/participating at the OG.

## **2.2.4 Content handling component**

The Content Handling (CH) component is in essence a control module that should provide transparent view on the information channels (low-level terminal components) for the application (Rich Media Engine). Therefore, the component should be able to assemble the information coming from different channels in the high-level readable format. The information flow is regulated by a content handling logic. Additionally, content handling logic should manage the communication between the components at the lower-levels.

Lower-level interface consists of a number of (sub-) interfaces. These interfaces should enable porting instantiations of different data models used in different information channels on the same content handling information flow. On the other hand, it should enable resolving the transformation of content handling information to the proper lower-level component format. This component should have a single interface providing information that can be used by higher-level components (and in particular the Data Model Presenter). The information is assembled from the information channels and formatted to effectively fulfill requests from the application and reach media components.

The units of the content handling component are:

1. Logical unit - responsible for resolving the issues with respect to high-level to low-level, low-level to high-level, and low-level to low-level component communication. This can be achieved using, e.g., formal rule specifications, easily extendible with new rules modeling new requests from higher-level components as well as information channels introduced at the lower-level.
2. Assembly unit – responsible for merging different types of (fragmented) information from the lower-level components into the (consistent) format recognized by the higher-level component. The assembly process is based on the logical unit rules and the format specifications.
3. Resolver unit – responsible for resolving the application requests and for forwarding these requests to the proper lower-level component(s).

## **2.2.5 ISAP Data Reception**

ISAP data reception component consists mainly in the data files reception at the terminal side sent by its counter part in the Head end, decoding of those data files and rendering them to the ISAP synchronization component.

The data files are received through FLUTE (or other transport protocols in case of Notification messages). Files intended for ISAP will be passed to the appropriate ISAP sub-modules, other data will be forwarded to the Content Handler.

Upon reception, these data files are decoded and separated into different application ready parts, namely the metadata part and application data part. The metadata part consists mainly of the timing information and the properties such as the nature (description, vote, alert, notification etc.), the MINE type, the relationship with the channel etc, while the application data consists of simply the data which may be directly used by the application layer components.

The component is also responsible for the data files management, namely, the storage according to the properties of the data files and their deletion after their validation. This is essential for keeping the precious disk space and optimizing its usage.

The component is interfaced with the ISAP synchronization component and the interface would be the metadata and the id of the application data.

## **2.2.6 ISAP Feedback Handling**

ISAP Feedback Handling component consists mainly in, upon a valid feedback request, encoding the feedback request and sending it back to its ISAP counter part at the head end side. This module also takes care of back-channel traffic generated by the Content Handler module.

Instant Messaging (IM) is used for sending this feedback request in case of ISAP. For the Content Handler other transport mechanisms (as for instance standardized in IPDC) may be used as well.

## **2.2.7 ISAP Synchronization**

ISAP Synchronization component is mainly responsible for handling the timing information related both to received data and feedback requests.

Regarding the received data, timing information may consist of the following elements:

- When to use the data
- For how long
- When to end using the data

Concerning the feedback requests, ISAP Synchronization component should assure the time validity of the feedback. For instance, as a kind of feedback, a vote may have a deadline over passing it implies the invalidity of the vote.

## **2.2.8 ISAP Interactive Service Management**

ISAP interactive service management component is the central component for implementing interactive service applications. Serving as a bridge, it interfaces with the applications, the rich media engine, ISAP synchronization component and other software (web browser, etc.).

The functions of interactive service management module include:

- Store and manage information of downloaded interactive application, available on-line interactive services
- Execute commands from rich media engine or from applications. These commands could be to send some feed back, or to do some local actions including open a file, write in a file, etc.

- Distribute interactive service triggering messages received from different channel but forwarded by content handling module
- Receive and handle commands sent from ISAP synchronization component. These commands could be to display a certain file, to trigger actions of a certain application etc
- Maintain application sessions (channels) status, thus allow back and forward between sessions.

### **Interface:**

Interface between interactive service management and rich media engine or applications: feed back commands, local action commands

Interface between interactive service management and content handling module: file information updates; file existence detecting commands; notification message or other service triggering message; feed back commands

Interface between interactive service management and data synchronization module: action commands extracted from AV streams

Interface between interactive service management and other software: action commands

## **2.2.9 Media Player / Recorder and Codec**

The Media Player / Recorder and Codec is the part dealing with A/V RTP stream handling, A/V decoding and display. It has interfaces with: Rich Media Engine (1, 5) and IP stack (2). It also has internal interfaces (3, 4). Below gives specification on each block and interface.

Block specification:

### **Media Player + Recorder (RTP)**

#### 1. RTP stream treatment

It receives RTP stream according to Rich Media Engine command via interface 1 and realizes RTP protocol treatment. This includes RTP buffering, sorting and synchronization between related streams (audio, video).

#### 2. Audio/video decoding

It sends coded audio and video content to audio/video codec via interfaces 3 and 4 to be decoded.

#### 3. Recording

According to command of Rich Media Engine via interface 1, it records the decoded media(s) on local memory. (this functionality is not included according to WP1 scenario design)

### **A/V codec (eAAC+ and H264)**

Audio/Video codec are to be called by Media Player to decode audio/video content. Supported encoding formats are eAAC+ for audio and H264 for video.

### **LASer Codec**

LASer codec ensures the relationship between Rich Media engine and the audio and video flows.

### **Mixing**

It mixes decoded audio/video content for display. Video content from RTP stream is mixed with other items from the Rich Media Engine (controls, buttons...) to form the complete screen display and send the all to LCD display. Audio signal is sent to speak.

### **Interface specification:**

1. interface between Rich Media Engine and Media Player  
Calling message: create or active the Media Player.  
SDP: RTP stream receiving and decoding information in form of SDP.
2. interface between IP Stack and Media Player  
Untreated RTP stream, according to SDP information from interface 1.
3. interface between Media Player and eAAC+ codec  
eAAC+ encoded audio content
4. interface between Media Player and H264 codec  
H264 encoded video content
5. interface between Rich Media Engine and Mixing  
video mixing control: video display location and size  
audio mixing control: volume

### **2.2.10 IP Datacast ESG Presenter**

This module interprets data not from the IPDC stack but from the Content Handler ESG data format (the format should abstract away from the exact ESG/EPG format used in different channels).

### **2.2.11 Rich Media Engine**

The RME is a software component which ensures graphic rendering of the presentation layer of the multi-media application. This engine takes into account the graphic, audio and video elements, the fixed images or animations, and the texts with their police to compose the "scene" seen by the user. This one will be able to interact on the "scene" in using the interaction mechanism proposed on the terminal (keys, joystick, touch screen, stylus). The scene can be static or dynamic with the temporal dimension envisaged by the author.

The scenes can be transferred in the form of "streams" or locally be downloaded.

### **2.2.12 Data Model Presenter**

The Data Model Presenter module provides a strict separation between the data model provided by the Content Handler at the lower level interface and the RME interface (i.e. data presentation) at the higher level interface according to the Model View-Controller (MVC) architectural pattern. Data models abstracted in the MOBISERVE scenarios are for instance sports related information like event schedules, match results, athlete start lists and background information. The IP

Datacast ESG Presenter below can also be seen as a specialization of a Data Model Presenter module.

### **2.2.13 Application**

For each service, the consortium will develop an application which corresponds to the specifications in terms of drawing of screen, sequence of the screens and more generally in the rules of ergonomics specified in WP2

If these applications are developed in Rich Media, the author has a software tool of design and simulation (the Studio), which runs on a workstation of the type PC.

The application thus generated will be adapted to the target terminal according to its particular characteristics (size of the screen, physical ergonomics (function keys or not), and obviously embarked OS). This adaptation is facilitated by the presence of the RME (Rich Media Engine) which plays the part of a “middleware” specific to the “Rich Media” applications.

## **2.3 Advanced Architecture Optional Components**

As described above, the research-based advanced architecture may be regarded as a superset of the Olympic trial-based architecture. Therefore, it contains all modules of the latter one. The focus of the advanced architecture lies on the Java execution environment that incorporates with the Rich Media Engine. As the components “Java Adapter”, “Combined XRM/JRM Application” and “Java Application” are related to the middleware specification they will be described in-depth in Deliverable D4.2 [1]. This section refers to the lower layers of the “Advanced Architecture” such as “Java VM”. In addition a short description of the “Java Adapter” will be given. A detailed description is located in Deliverable D4.2 [1].

### **2.3.1 Java VM**

A Java Micro Edition based Virtual Machine (JME VM) provides the core functionality of the advanced architecture. Thus all applications using Java rely on the VM. Within a Java development process the Java compiler compiles source code into a specific byte code. Afterwards this byte code is interpreted and executed by the JVM.

### **2.3.2 Java Adapter**

The Java Adapter represents the interface between native-based and Java-based software components. As an integral part of the Application Middleware its detailed description is located in deliverable D4.2 [1].

## 3 Hardware prototypes

Two terminal prototypes are proposed by this prototyping activity: DVB-H phone and PDA based prototype. Decision to port the complete stack to a given architecture is still under discussion, the different terminal hardware described below should not be considered as final. At this stage of the project we have

1) The NXPP1.0/NXPP1.0 solution that is contributing to analyze various options in architecture but still miss hardware decoding facilities for a good picture quality in broad cast to mobile environment

2) The PDA based demo that has been used to demonstrate concepts and demo applications of Rich Media in DVB-H environment, details have been given in WP5 reporting but the overall architecture is Windows Mobile 5.0 based with an existing in the market available SDIO DVB-H module.

### 3.1 DVB-H Phone General Introduction

The DVB-H terminal prototype is the NxPP1.0 provided by NXP. The appearance of the terminal is shown in Figure 2.

The NxPP1.0 phone prototype has been shown to partners at the January 2007 Workshop in Beijing. At this workshop, the reception of live DVB-H with the NxPP1.0 was shown to the partners, using the software subsystem described in section 3.3. In addition to the partners already having access to this development platform (being PREA and PRLE), four other partners received a development kit at the January 2007 workshop to get started with their development.

The NxPP1.0 terminal is equipped with the following interfaces:

- 2.9" QVGA display with Touch Screen
- USB OTG interface
- MiniSD Card slot
- Stereo Headset with MIC input
- 4 directional buttons with Select key
- Docking connector (Charging + USB)



Figure 2 Appearance of the NxPP 1.0

The NxPP 1.0 can provide the following functions for users.

**Multimedia:**

- Decode and display video formats H.264
- Playback MP3 audio
- Display pictures
- Store video clips on SD or MMC Flash memory card
- Provide a graphical user interface to users, to access the functions

**DVB-H:**

- Receive and display DVB-H transmissions carrying data compressed using the H.264 format
- Select different DVB-H channels

**GSM/GPRS:**

- Enter the PIN code for the SIM
- Register to a GSM network
- Accept and participate in an incoming GSM call
- Initiate an outgoing call

**WLAN:**

- Connect to a WLAN network

- Perform a voice over IP (VoIP) call via WLAN
- Download pictures and video clips from a server via WLAN

### 3.2 Hardware Specifications

The above functions of the NxPP 1.0 are supported by a Multimedia Application Engine (AE) called PNX4008, and three sub systems which are controlled by the AE. Refer to Figure 3,

- DVB-H hardware sub-system. It is controlled by the AE. The control functions include carrier frequency and transport stream selection. The DVB-H HW sub system performs TS de-multiplexing, MPE decoding and FEC processing. Data from the DVB-H HW subsystem, which is a stream of IP packets, is transferred to the AE.
- Cellular sub-system contains a GSM/GPRS module supporting normal phone functions
- Connectivity sub-system includes a wireless LAN module

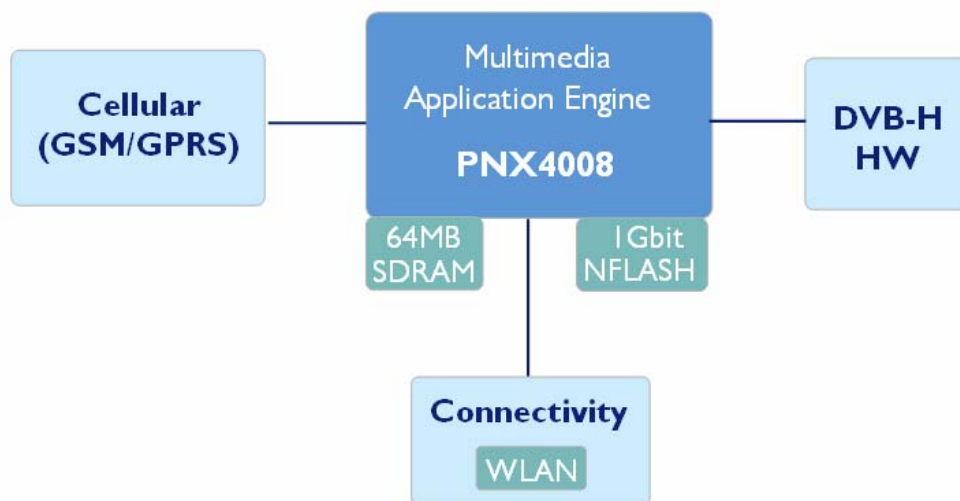


Figure 3 Block diagram of the hardware system of the NxPP 1.0

The PNX4008 AE is based on an ARM9 RISC processor, which is accelerated by dedicated video and audio engines. It delivers optimum performance with very low power consumption. Refer to Figure 4, the CPU subsystem combines a 208-MHz ARM926-EJ processor with dedicated 32k instruction and 32k data caches. The video engine provides high-performance support for still and video images. It includes an encode acceleration JPEG decoder, an MPEG-4/H.263 video encoder/decoder, and a graphics accelerator. The audio engine combines a fully dedicated audio DSP with integrated stereo DACs and ADCs to provide sound capabilities. The audio DSP can operate independently from the ARM CPU and can reference embedded RAM memories for full audio flexibility.

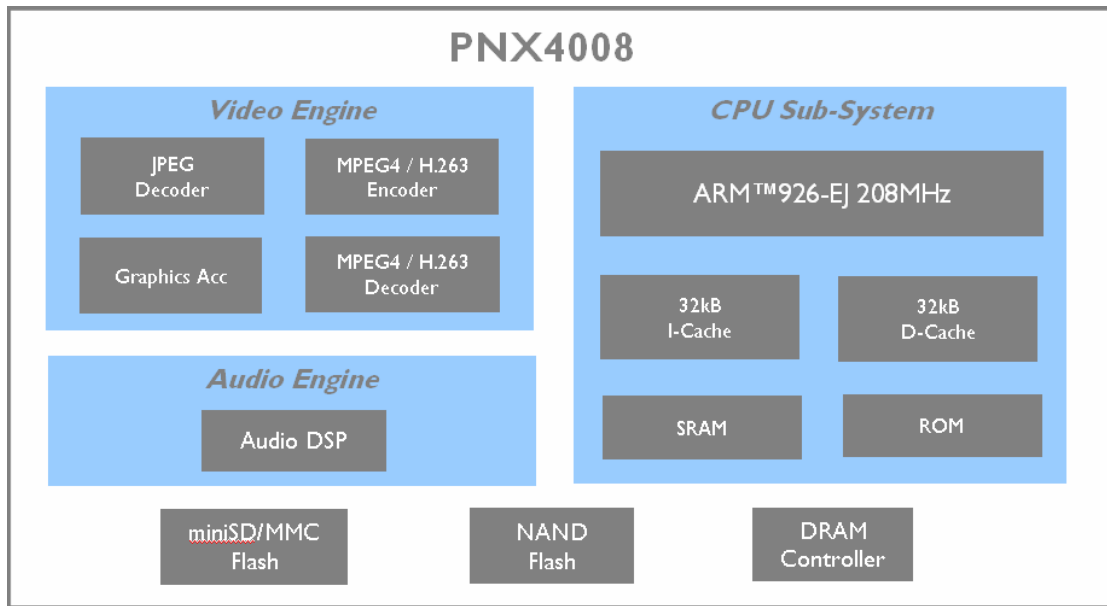


Figure 4 PNX4008 Application Engine

### 3.3 Software Sub-system

To support TV-on-mobile applications, a software sub-system is developed for the NxPP 1.0. Refer to Figure 5, an embedded Linux is running on the ARM CPU of the PNX4008 application engine. A Driver Resource Abstraction Layer (DRAL) running on top of the embedded Linux acts as a unified interface between the DVB-H hardware sub-system of the NxPP 1.0 and the DVB-H middleware, which is also called IP Datacast stack. The RTP video player get input as RTP packages with H264 video as the payload from the DRAL, unpack the RTP packages in the RFC3984 compliant way, decode the H264 video and show it in the Graphics User Interface (GUI).

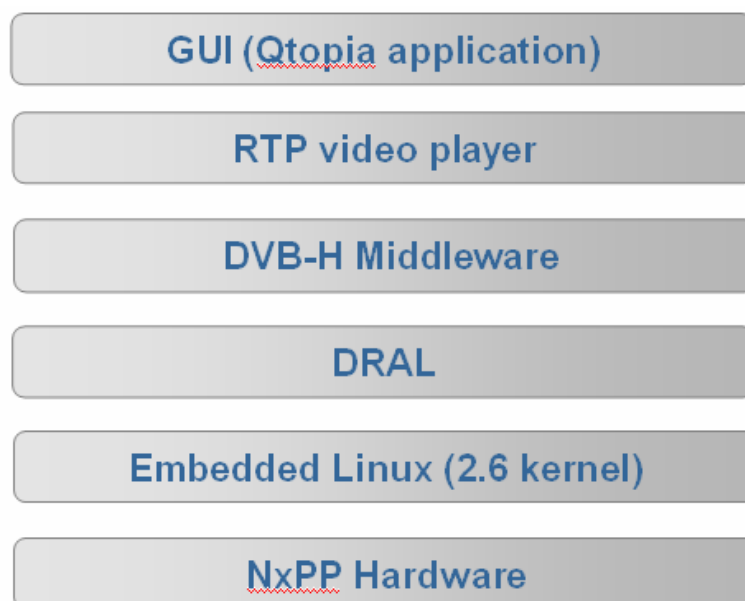


Figure 5 software sub-system supporting TV-on-mobile applications

## **DRAL**

The DRAL is responsible for handling the IP and section data flow from the DVB-H HW module and for maintaining the resources used by the DVB-H middleware. The resources are section and IP filters and the contexts on which they run. For example, the DVB-H middleware uses the specified DRAL API to control the tuner, the demodulator and the demux during a service scan to get appropriate information about the available services and/or networks.

The DRAL ensures a smooth data flow between the middleware and the DVB-H HW module.

## **DVB-H middleware**

The DVB-H middleware used in the software sub-system is the onHandTV™ middleware provided by Silicon & Software Systems Ltd. (S3), which is the world leading provider of Mobile DTV handheld solutions. The onHandTV™ is a fully DVB-H/IP Datacast compliant software stack. It is architected to deliver the same core TV features such as ESG and Interactivity across multiple networks using preconfigured Operator Adaptation Packages (OAPs). For further information, please visit [www.s3group.com](http://www.s3group.com).

### **3.3.1 Limitations**

Regarding DVB-H related TV-on-mobile applications, the NxPP1.0 currently has some limitations:

- Since the video engine of the PNX4008 only supports H.263 hardware decoding, the RTP video player can only partially use it for the H264 decoding. Therefore, the NxPP1.0 can't support H264 QVGA with a refresh rate of 25 fps.
- The audio engine of the PNX4008 is not dedicate for AAC, so the NxPP1.0 currently can't support AAC audio decoding if it is decoding H264 video in the mean time.

These limitations will be solved with the future terminal prototypes.

### **3.3.2 PDA based prototype**

The PDA based terminal prototype is composed of 3 parts: terminal device, RF reception, and ESG interpreter / AV player.

#### **3.3.3 Terminal device**

The terminal prototype uses 2 PDA models: HP iPAQ and Dell x51v. These 2 models of PDA support both WiFi connection and SDIO interface, which is used for DVB-H reception card.

#### **3.3.4 RF reception**

In this prototyping, both DVB-H and WiFi are included.

- WiFi reception  
WiFi connection is provided by the embedded WiFi reception module in the 2 chosen PDA.
- DVB-H reception  
Commercially available Receivers in SDIO form are used for PDA terminals.

### 3.3.5 ESG interpreter / AV player

Two suits of ESG interpreter / AV player are available to illustrate different services.

- Thomson solution

The Thomson experimental solution combines EyePhone, the ESG interpreter, and Thomson Player, the AV player to provide terminal side user interface.

EyePhone reads the pre-downloaded ESG data stored locally and provide interfaces for user to choose channels and to view program list (EPG). Figure 6 illustrates EyePhone interfaces.



Figure 6 EyePhone Interface

Despite of the local ESG approach chosen in this prototyping, EyePhone is also capable to handle on-air broadcasted ESG data.

- Streamezzo rich media engine

The Streamezzo rich media engine is terminal side UI software. It interprets rich media data and provides rich media interface to the user. It has its own A/V player.

### 3.4 Rich Media Prototype

Streamezzo is proposing a contribution to the short-term demo in the integration of Rich Media in the demo. This PDA based demo with commercially available SDIO module has been shown in two occasions in Shanghai (Jan 19, 2007) to the consortium partners and in Beijing during the advisory committee meeting (Jan 22, 2007)

The general setup of these demos (detailed in WP5 reporting) reads as follows.

It is, in the first prototyping activities, a standalone demo system from the physical point of view.

The architecture of DVB-H broadcast to mobile is well known and the figure below highlight the necessary evolution for injection of Rich media uses cases such as alerts, additional information, quiz, votes and access to VOD portals.

Particular attention is given to the Rich Media modules:

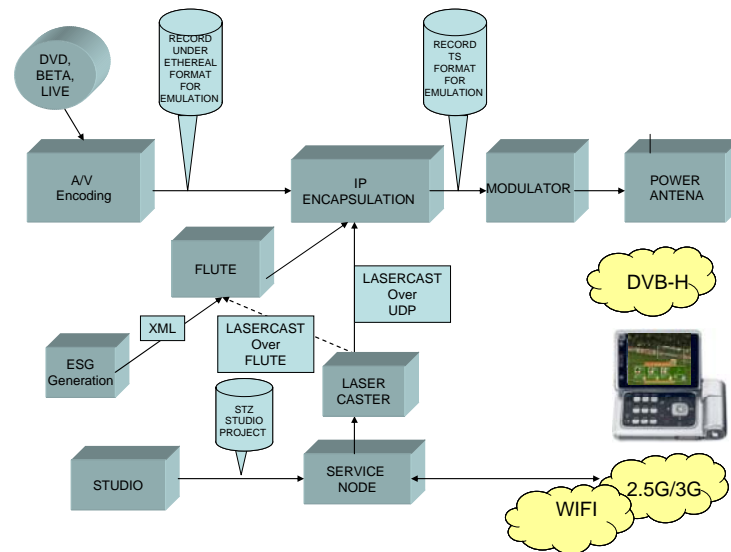


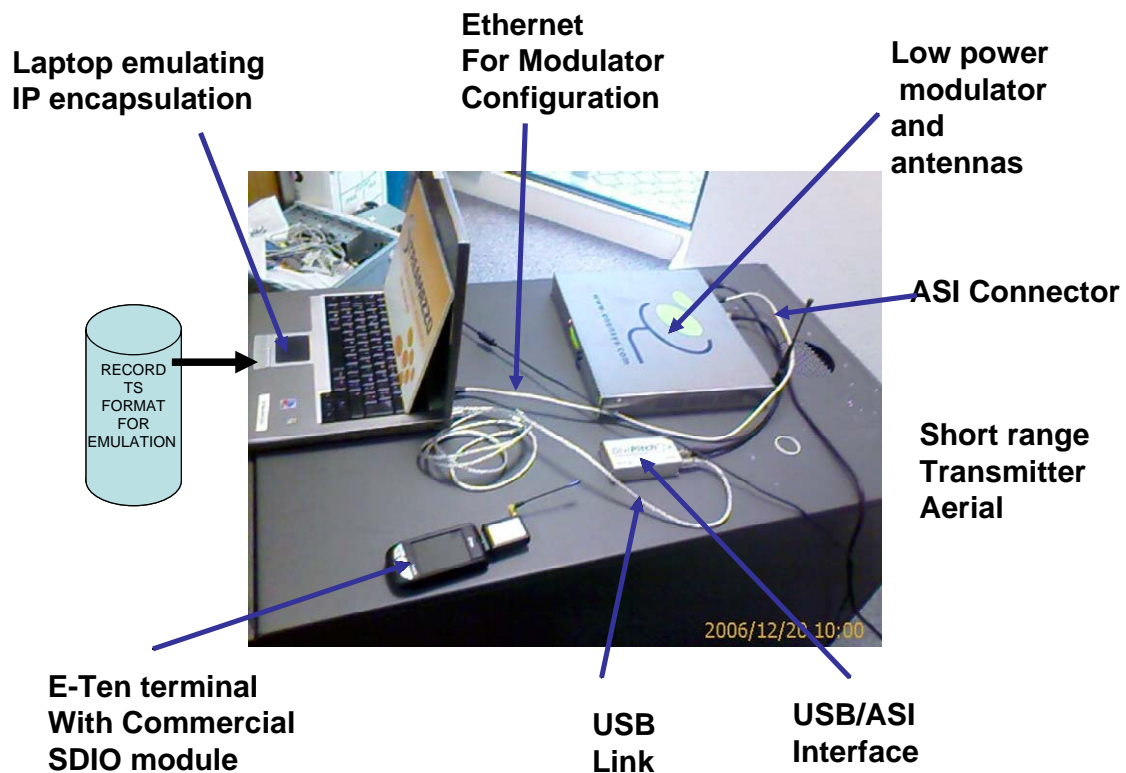
Figure 7 Rich Media Demonstration Architecture

Studio for the authoring of rich media elements , taking into account graphics “assets” such as images, text and animation and generating the rich media scenario according to the storyboard. Service Node is central and will manages streaming of Rich media elements according to inputs received from the terminal connected with 2;5G 3G or WIFI bidirectional network . Service Node is also injecting Rich Media elements inside the broadcast stream by the means of the Laser caster that interfaces with IP Encapsulator. Synchronisation can be obtained up to the second. Rich media is injected directly over UDP into the IPE or through the FLUTE. Terminal application will present Rich Media elements coming from Broadcast stream Connected feed. This application is also able to present ESG elements if present in a standard format inside the stream.

Special attention is also given to some useful intermediate formats that can be used to store contents in order to implement incremental strategies of integration and avoid immobilizing an expensive end to end chain when doing partial tests.

First intermediate format is the storing of ethereal content obtains from Encoder and accepted by IP Encapsulator, this allows replaying video independently from some time expensive configurations.

Other intermediate format is obtained in output of the IP encapsulation, by storing the Transport stream with or without Rich media elements. This format is taken for the prototyping activities and is very useful for the test of terminals as well as for independent demos performed with “head end” simulator as described below:



### 3.5 Roadmap to the Olympic demo terminal

NXP is conducting activities on this roadmap.

The roadmap runs as follows

#### Q1/2007

- Experimentation with NXPP form factor for network and lower layer developments
- Rich Media demo based on PDA with SDIO DVB-H module

#### Q2/2007

- Porting of main software components on the TV hot plug development board
- Two options are currently under discussion as depicted below., development boards are available to the relevant partners
- Discussions have been initiated with S3 for the IPDC stack integration.

#### Q3/2007

- Final choice of porting option (A or B) (see Figure 9 and 10)

#### Q4/2007

- The porting of the various components should be achieved
- Decision on form factor can be made
- Service development can start

**Q1/2008**

- Form factors made available
- Integration of coexistence antenna from Task 4.4
- Service can be tested

**Q2/2008**

- Olympic trial
- Assessments

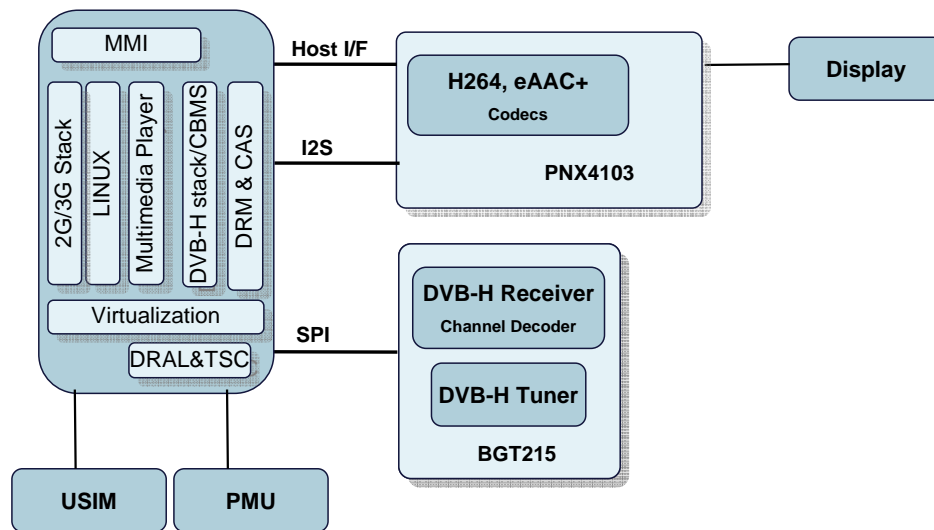


Figure 9 Option A

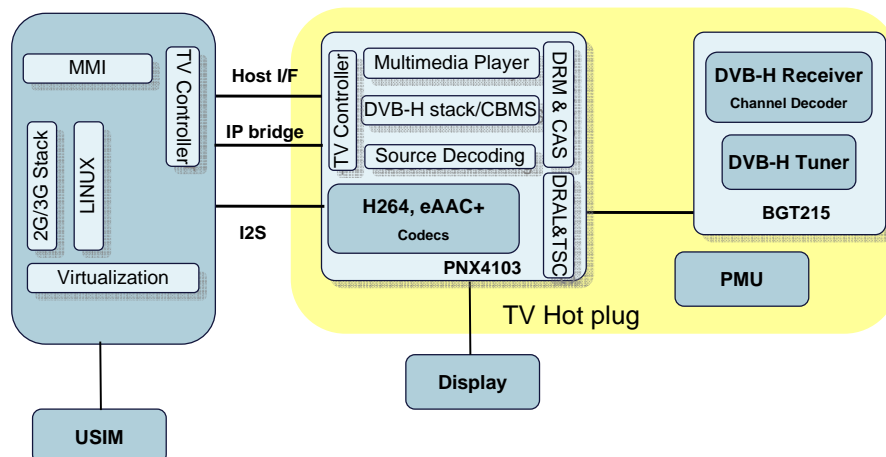


Figure 10 Option B

## 4 References

- [1] MOBISERVE Deliverable 4.2: Terminal System Architecture Specification, for Olympic trial