November, 06, 2008

Anlage 1

Projektendbericht

Feasibility Study

ORF MultiText on UPC MediaBox

Submitted By:

BearingPoint INFONOVA GmbH
Seering 6
8141 Unterpremstaetten/Graz
Austria

Version 1.0

Authors

Franz Kollmann
BearingPoint Infonova GmbH
Seering 6
8141 Unterpremstätten/Graz

franz.kollmann@bearingpoint.com

Heribert Amtmann
BearingPoint Infonova GmbH
Seering 6
8141 Unterpremstätten/Graz

heribert.amtmann@bearingpoint.com

Contents

1. Initial Situation and Goals

1.1 Initial situation

Since the introduction of terrestrial digital television (DVB-T) in Austria in October 2006, ORF provides an interactive service called "MultiText" based on the Digital Video Broadcasting Multimedia Home Platform (DVB MHP). The so called "Kabelmux" enables cable network providers to receive and distribute digital ORF services. Among digital ORF TV and radio programs this multiplexed signal contains also ORF MultiText related data streams including DVB-J applications, content data, layout and configuration files. To use and display the MultiText, the digital receivers in the households must comply with DVB MHP 1.1.2.

UPC intends to provide the ORF MultiText to its TV subscribers using the UPC MediaBox. However, the UPC MediaBox platform is based on the proprietary OpenTV architecture and DVB MHP is not supported. MHP middleware and OpenTV middleware are completely different technologies, so there is no interoperability at all. Therefore, an immediate use of the ORF MultiText service as contained in the "Kabelmux" is not possible.

1.2 Goals

UPC decided to investigate the complexity and practicability of providing the ORF MultiText services on the UPC MediaBox.

BearingPoint supported this project by contributing a technical feasibility study. The main goal of the study is to identify and evaluate possible strategies in order to enable the ORF MultiText in the UPC MediaBox environment. Among technical details of the variants, an overview of the advantages and disadvantages, as well as their technical complexity are summarized in the study. The study results are used as a basis for subsequent UPC internal decision-making processes.

It should be remarked that access to technical documentation about OpenTV is restricted and only high level information on the target system (see next section) has been provided by UPC to BearingPoint. Because of its prior experiences in the area of interactive TV, BearingPoint was able to identify and describe five technical variants for providing MultiText content on UPC MediaBoxes. However, a calculation of arising costs related to each variant has been dropped. Instead it was decided to compare the resulting variants with each other in terms of complexity, feasibility, and costs.

2. Current Knowledge about the Target Environment

In this section we want to sketch out our current level of information and understanding about the OpenTV technology [1], which is used in the MediaBox. It should be noted that useful technical reports and documentations about OpenTV are not freely accessible. In a diploma thesis [2] some technical insights into OpenTV (Core 1.0) are given. To clarify several open issues, we have generated a list of questions, which we have sent to the UPC team (see Section 2.2).

2.1 OpenTV

OpenTV is an operating environment for interactive television. Despite the name, OpenTV is a proprietary middleware available under commercial licences to set-top box manufacturers and application developers. OpenTV applications are ANSI-C programs that use the OpenTV API. The compiled program codes are referred to as O-code, which can be interpreted by the OpenTV virtual machine. The general architecture is illustrated below.

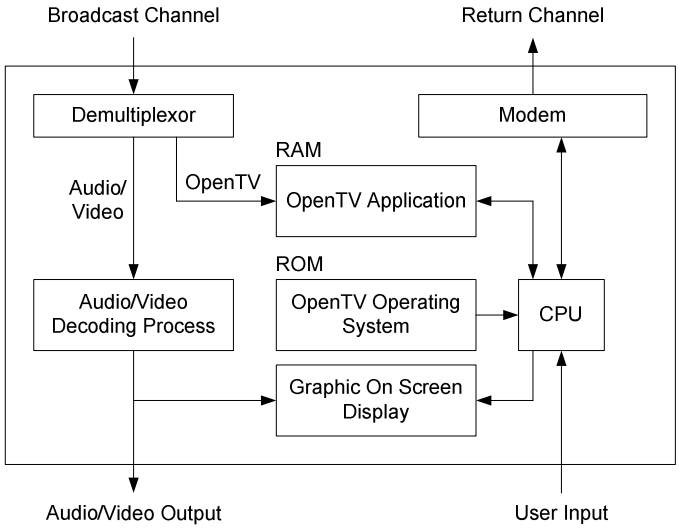

Figure 1: Receiver components

The OpenTV architecture consists of several modular components, including the OpenTV operating system, the OpenTV virtual machine and OpenTV applications. The static part of the OpenTV operating system is persistently stored in the ROM of the receiver. Programs are loaded into the RAM at their execution time. The OpenTV virtual machine interprets the OpenTV programs. The demultiplexer separates the application-specific data from the video and audio data and relays the data to the responsible components. Video and audio material is processed by the graphic on screen display.

OpenTV applications are event-driven, i.e. they process messages and respond to events. Every OpenTV application must implement a main message loop, which receives messages that are stored in a single message queue (see figure below). The main loop decides what needs to be done in order to respond to the incoming messages. Principally messages can be processed by the main loop or they are dispatched to other modules. The GUI elements are called gadgets, which are created from templates called classes. They are structured in a gadget tree where the order of the call activation is specified.

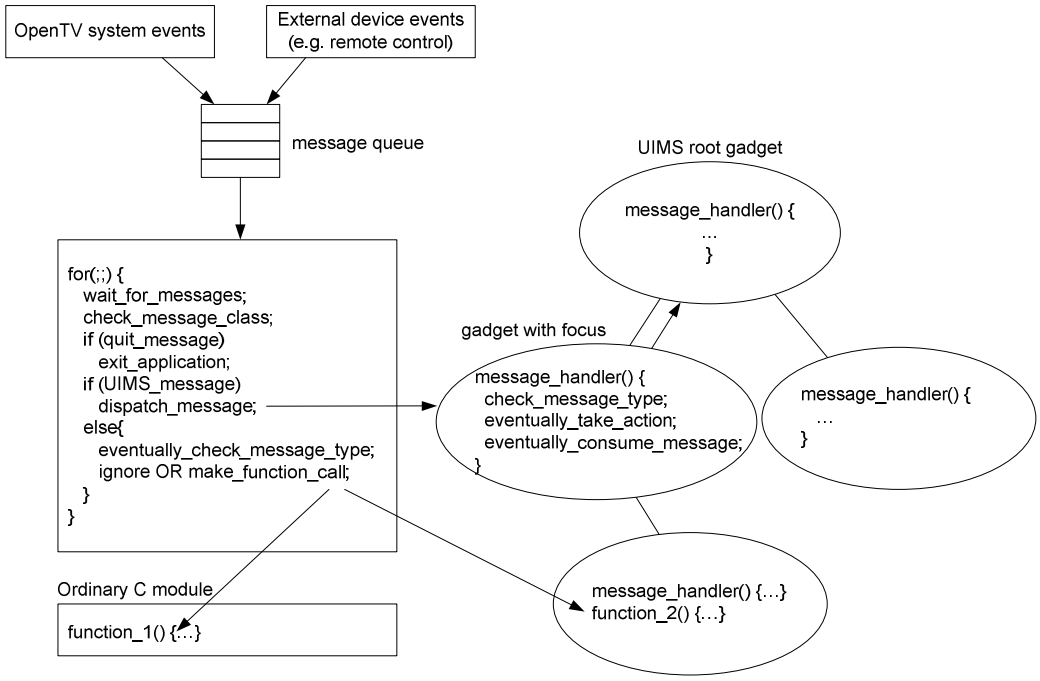

Figure 2: The OpenTV event model

OpenTV is based on ANSI-C. Although OpenTV tries to mimic object-oriented behaviour, it is not object-oriented. A concurrent execution of several applications is not supported: only one process can be executed at each time. When a new application is signalled the currently running application is asked to quit.

The latest version of the middleware is OpenTV Core 2.0, which is backward compatible to O-code applications that use former OpenTV versions. Core 2.0 comes along with many features like HTML Package, MHP Package, OpenTV Bluestreak Package for Flash, OpenTV Participate, OpenTV Browser and others. In cooperation with Alticast the development of the OpenTV MHP Package was initiated. However, to our knowledge, the initiative was not finished.

2.2 Evaluation of the current situation

As an important base for further considerations, the current interactive application offering on the UPC MediaBox has been examined as a first step. The idea was to draw conclusions on the technical platform and on the technologies used to implement the current application offering. Three application scenarios of relevance for further considerations are shortly summarized here.

(1) UPC interactive "Home" Portal

By pressing a dedicated button on the MediaBox remote control, the UPC interactive Home Portal is entered. From here the UPC branded interactive offering can be viewed. It consists of UPC branded and third party applications (information and entertainment).

From a technical point of view, the MediaBox tunes to a dedicated UPC interactive transport stream, when the remote control button is pressed. An OpenTV application is downloaded from the broadcast stream and executed in the local environment. The previously watched TV channel is not visible any more; instead an UPC interactive video is displayed full screen in the background. When exiting the UPC interactive application the MediaBox tunes to the previously watched TV channel.

(2) EuroNews Application – service bound application

On certain TV channels (so far it seems to be used on the EuroNews channel only) applications are bound to the particular TV channel. When the viewer switches to such a TV channel, a logo indicating the presence of an interactive application appears on the upper right of the TV screen. Technically, this is an autostart OpenTV application, loading in the background from the broadcast stream. After pressing a dedicated button (in the case of EuroNews it's the red button), the application content data is loaded and displayed on the screen. In this scenario the video stays visible and the application content is displayed as a graphic overlay.

(3) UPC Electronic Program Guide

UPC offers an Electronic Program Guide (EPG) to its subscribers. The technical implementation is comparable to the one described in (1). When pressing a dedicated EPG button, the MediaBox tunes to a certain transport stream and starts to download an EPG viewer OpenTV application and in addition the EPG metadata. It should be noted that EPG metadata is not read from DVB Service Information tables (EIT scheduled) but provided in some other format. The previously watched TV channel stays visible in case of the EPG application (minimized in corner of the TV screen). However, there is no "TV channel preview" functionality available, when selecting other channels inside the EPG application. So, the minimized TV window stays always the previously watched TV channel.

It should be noted, that no separate DOCSIS cable modem has to be used to connect the MediaBox to the IP network. All available MediaBox models have a DOCSIS cable modem on board. This is a major advantage since it's only one single (coax) cable which has to be connected to the box and the so called return channel functionality needs thus no separate handling from user side. However, the UPC MediaBox IP return channel is intended to be used for typically transactions only (e.g. NVOD ordering). That means, the available bandwidth is restricted, allowing only short messages between MediaBox and servers operated at the backend. Large volume transfers (e.g. IP based true VoD) is not possible in such a scenario

In order to gain more detailed technical information about open questions concerning the UPC MediaBox and used technologies, BearingPoint has formulated a list of questions, which were sent to the UPC team. The requested questions and the answers are listed below. Note that some of the questions were not answered.

1. Which OpenTV middleware version is used on the MediaBox?

   Pace SD & Thomson SD: Core1.1
   Thomson SD DVR: Core2.0
   Philips HD DVR: Core2.0

2. Are there any additional application execution environments used on the MediaBox beside the o-code interpreter (HTML/JavaScript, Java/MHP, Flash)? If this is the case, please specify the detailed features of the relevant execution environments.

3. Which from the above mentioned execution environments is used to implement the available UPC interactive products (especially for the regional news, weather, etc services and for the interactive games)?

   OpenTV

4. Please explain in which way the existing UPC interactive products (application and data) are delivered to the MediaBox. Is it a data carousel broadcast only, is the IP path involved?

   The current UPC interactive products are broadcasted as a PID-carousel. The IP path is only used for the return path of the OpenTV-Applications.

5. We understand that regional content is transferred to the UPC Headquarter first. Can you please describe the content data format for this and how the data is modified (transcoded) before multiplexing (that is the data format as it is used for the existing interactive products).

6. Is it possible to add a regional interactive product to your existing portfolio? In particular, is it possible to define a new graphical layout that represents the ORF MultiText?

7. Are there any restrictions on the return channel (e.g. bandwidth, limited IP range, etc.)?

Our STB's are configured in special IP-Ranges. These IP-Ranges are only used by the STB's.

8. Is it possible to load interactive applications and/or content data for the applications via the return channel?

9. Does the middleware support the remote frame buffer protocol or similar technologies?

10. Is it possible to link interactive applications to certain TV channels, so that these applications can be launched on those specific TV channels only?

    Yes. The PID-carousel and Open-TV DVB-Descriptors has to be added to the PMT of that special Service.

11. Is it possible to embed live video into an interactive application (that is can the video be resized and then be displayed as part of the application)?

    Yes.

12. Can you please explain the functionality of the "Mistral" subsystem?

3. ORF MultiText

In the following section a brief overview of the main components of the ORF OK MultiText is given. Notice that only a selection of the supported features is presented.

The ORF OK MultiText, which is based on the Flexible Unified Service Environment (FUSE), is a kind of "interactive multimedia browser" for TV set-top boxes. FUSE is a Java-based architecture that is fully MHP 1.1.2 compliant and was developed by BearingPoint for the Austrian television provider ORF. It enables to display multimedia contents like news, program information, special topics, advertisements and others on the televisions. Additionally it provides interactive services like voting or chat.

Since 2006 the ORF has been broadcasting MultiText data per DVB-T enabling configured DVB-T receivers to process the information so that the contents can be displayed on the television accordingly.

3.1 Overall architecture

FUSE is built upon the DVB MHP middleware (version 1.1.2). Generally, it consists of a launcher and an engine.

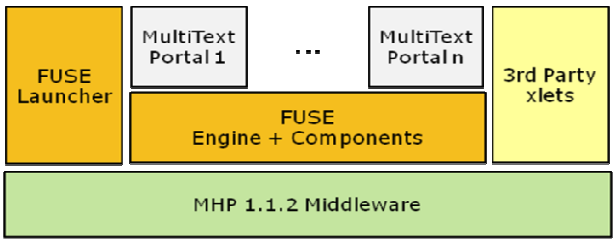

Figure 3: MultiText architecture

3.2 FUSE Launcher

The FUSE Launcher is signaled as "autostart" and it starts when the set-top box is switched on. Its task is to assign special buttons of the remote control to portals, to monitor the buttons, to display the ORF OK logo, and to initialize the FUSE Engine with the start portal.

3.3 FUSE Engine

The FUSE Engine represents the core of the MultiText. Among many other tasks, it displays the contents and reacts on user input. The contents of the MultiText are structured by so called "portals".

According to the MHP specification, the displayed objects are divided into several layers. Usually the background layer contains an MPEG I-frame. Alternatively a background color can be defined. The video layer displays the live TV and can be resized or cropped. The graphic layer represents the front layer and it can display banners or pictures (JPG, GIF or PNG format). The MultiText enables to split the graphical layer into further subdivisions.

3.3.1 Portals

Portals represent a central element in the FUSE engine. The contents and appearance of portals are specified by content and layout XML files. When the MultiText is started by the launcher (initiated by pressing the OK-Button), the Start Portal (see figure below) is displayed.

© ORF 2008

Figure 4: Start Portal

The Start Portal serves as "entrance hall" of the MultiText. Essentially it shows an index of the portals, lists current news headlines, presents advertisements, informs about actual and next TV program as well as about the current date and time, and it displays the live TV in a separate box. As any other portal, the appearance and structure of the start portal can be adjusted (per XML files). The colored buttons provide a possibility to ease the user navigation. They can also be used to access often used features, which can be assigned to the tabs according to one's preference.

The ORF offers separate portals for news, weather, sport, special topics, advertisements, and games. The following figure illustrates the potential of the portals and it shows some possible applications.

© ORF 2008

Figure 5: Special Portals

3.3.2 Components

In the following section an overview of the supported FUSE features is given. The overall functionality of FUSE is fairly large. A detailed description would go beyond the scope of this feasibility study. We therefore limit the demonstration to a selection of the FUSE features.

The following figure exemplarily illustrates a page in the News Portal. In this example several FUSE components are used. Details of the main components in the News Portal are given below.

© ORF 2008

Figure 6: News Portal

- **Live TV:** The live TV can be included in the MultiText. Its size and position is scalable (e.g. downsizing by 1/8). Additionally it can be cropped and shaped.
- **Image:** The Image component positions and displays images (GIF, JPG, PNG). As a special feature, FUSE provides Rollover Images, which are displayed when the associated items are focused. Dependant on the focused items, the Rollover Images may change.
- **Navigation:** FUSE provides a line-based concept to display navigation entries. The items are hierarchically structured whereby the hierarchy can principally be at any granularity. When the menu items exceed the size of the intended displaying area, items are masked out. Arrows indicate the existence of further items (see figure below), which are displayed when navigated to the corresponding directions.

Figure 7: Navigation

- **Content Container:** The Content Container displays contents, which are divided into pages (see figure below). For large amount of data, the contents might be divided into subpages. The current subpage is indicated by the so called Paginator. Within the subpages, one can navigate to previous and following subpages (if existent).

Figure 8: Content Container with its XML structure

A page is primarily characterized by a headline and the contents (of the Content Container), which refer to pages and subpages. The listing in the figure above gives an insight into the data structure of the Content Container. The layout is defined in a separate layout file. Tables are also supported.

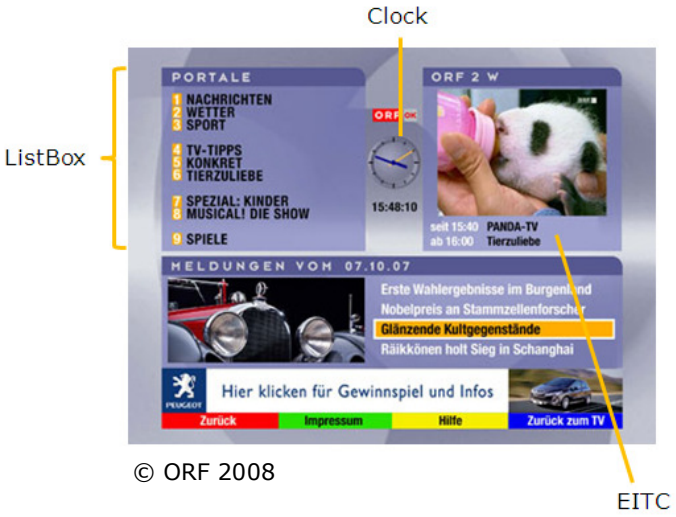

Figure 9: List Box, Clock, and EITC

- **List Box:** The List Box displays contents as a list of items. The items can also be represented as links. Instead of text items, it is also possible to use images. Additionally the items can be enumerated, which enables to select an item by pressing its corresponding numeric key.
- **Clock:** The Clock displays the current time (analog or digital).
- **EITC:** The EIT content informs about the current and next TV program.

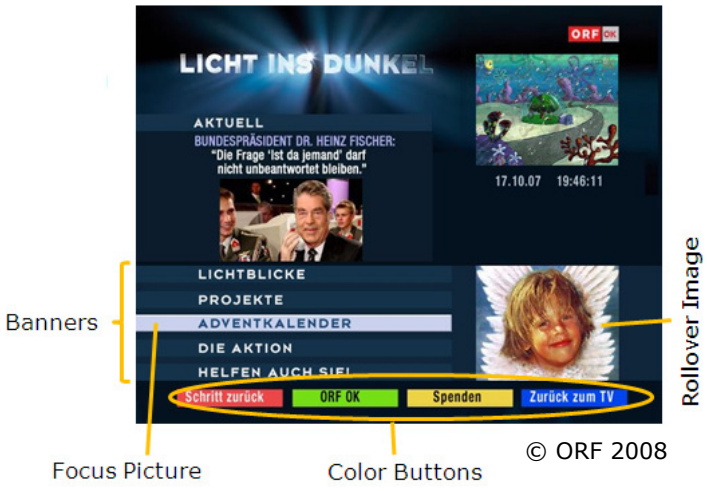

Figure 10: Banners, Focus Picture, Color Buttons, and Rollover Images

- **Banners:** Banners represent a possibility to link to external applications. Hereby a Focus Picture can be defined, which is displayed when the relevant item is focused.

- **Color Buttons:** The Color Buttons, which correspond to the colored buttons on the remote control, allow a quick and easy navigation (e.g. back).
- **Progress Bar:** The Progress Bar illustrates a graphical representation of the elapsed and remaining time of a video or audio. The appearance can also be adjusted.

Figure 11: Progress Bar

- **Entry Box:** It allows users to enter text inputs (e.g. by the remote control). To support special characters, a virtual keyboard can be used (see below).

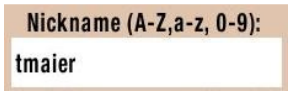

Figure 12: Entry Box

- **Virtual Keyboard:** The Virtual Keyboard allows users to enter text inputs by using the arrow keys and the OK button. Buttons are defined in the corresponding XML file.

3.3.3 Services

FUSE also supports web-based services. We exemplarily describe the Voting service and the Chat service.

- **Voting:** FUSE provides a web-based Voting service. Multiple-choice questions and also "single shot" votes are supported. The messages (vote, query, answers) are exchanged per SOAP.

- **Chat Service:** The FUSE platform offers a chat service that enables a text based communication between users. The functionality encompasses user registration, user authentication, the generation of text messages, and the facility to send the generated messages.

Figure 13: Chat Service

Messages are usually supervised by a moderator. Sent messages are transmitted to a chat server. A chat room moderator inspects the messages and he or she decides whether or not they are displayed in the chat room.

4. Realization Variants

Purpose

The aim of this section is to give a compact overview of possible variants that enable to use the ORF MultiText on receivers that are based on non-MHP compliant devices (like OpenTV). The presented strategies are compared with each other concerning their feasibility, functionality, and expected costs.

Nomenclature

In this work we use the term "MultiText" to refer to the MultiText application and "MultiText data" denotes the data, which are used by the MutiText for displaying and rendering. The latter usually contains content files, layout files, and images. These files are included in the transport stream, which is broadcasted by the ORF.

Assumptions

The following feasibility study assumes that the UPC MediaBox referred to as "receiver" is used as target device. Concerning the software environment on the receivers, several assumptions are met. In some scenarios an HTML browser is assumed as presentation engine. However, this does not mean that we limit our considerations to HTML browsers. Since too little information about the presentation engine was provided, we have decided to describe the situation when HTML is used, because this is one of the OpenTV's supported formats. Note that it is unreliable to estimate the complexity of solutions based on proprietary systems, when relevant documentation is not accessible. In this context essential questions need to be clarified before the strategies can be tackled. However, possible solutions are described and compared with each other, whereby the advantages and disadvantages of the approaches are listed.

Distribution strategies

How to distribute the MultiText contents is a question that needs to be clarified in advance. Basically two possibilities are considerable. For the first variant the relevant data of the MultiText is embedded in the transport stream (embedding variant) and is broadcasted by the service provider (see left scenario in the figure below).

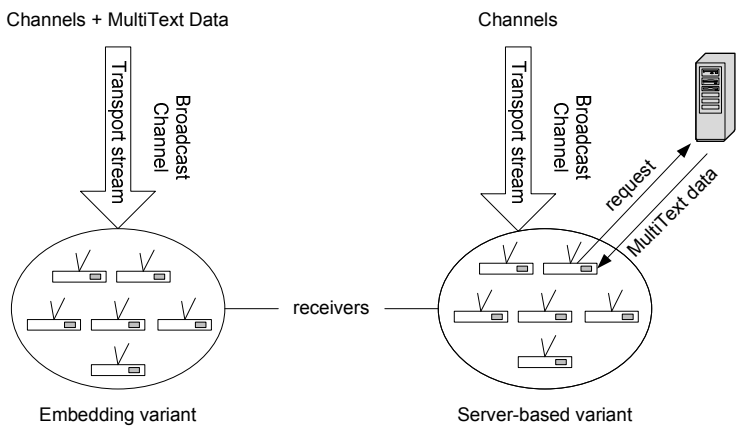

Figure 14: Embedding into transport stream versus requesting from server

In this case the contents need to be packaged and included in the transport stream. From the answers to our questions it is not clear how this can be realized in the concrete case. For a practical realization this issue needs to be clarified. In particular: Is it feasible to integrate the MultiText contents in the transport stream? If so, how can the packaging process be realized concretely?

Another way of offering MultiText contents for the receivers is to locate the corresponding MultiText data on a separate server, which the receivers acting as clients can access (server-based variant). The server can get the MultiText data either via transport stream (broadcast channel) or the server retrieves the MultiText data from the CMS. Note that the server can be placed anywhere as long as the server can be accessed by the boxes and the server can access the MultiText data (either from the broadcast channel or from the CMS output).

Apart from interactive services like chat or voting services, in the embedding strategy a concurrent utilization of MultiText services does not influence the operational reliability of the services. Hereby the MultiText data is broadcasted and hence it can be retrieved from the transport stream.

In a server-based variant the MultiText data is offered by a separate server. Once a MultiText service is started on the receiver or when it is refreshed, the desired MultiText contents are requested from that server. A server-based hosting of MultiText data requires a separate communication channel. As a prerequisite, the boxes must provide a return channel.

Interactive services like chat and voting services require an additional server. In the server-based variant, a server that offers the MultiText data is already used. Therefore, it might be considerable that the interactive services are included in the MultiText data server so that these services are offered from one source.

Expected amount of transmitted data

A question that is relevant for all variants is the following: How much data is expected to be transmitted? To answer this question, it is necessary to understand the structure of the MultiText data. The MultiText data is divided into portals (e.g. news, sport, weather). Each portal consists of content files and layout files, which are XML files. Besides content and layout, the MultiText data contains also image files (JPG, GIF, PNG). As one might expect, the overall amount of data particularly depends on the size of the images. For the MultiText data transmission, the contents and layout files are zipped. Additionally the images are zipped and transmitted. As a rough orientation, the following table estimates the expected sizes of the resulting zipped files (rounded values refer to a broadcasted transport stream). Note that, according to the size of transmitted contents and images, the amount of data varies. Additionally the background images (e.g. MPEG I-frames) and further images (e.g. logos etc.) need to be considered. As a rough estimation, the amount of data for these images can be estimated by 300 kB.

Portals	**Content & Layout** (in kB)	**Images** (in kB)	**Total** (in kB)
Start	20	80	100
News	50	400	450
Sport	30	300	330
Weather	20	—	20
"Konkret"	80	20	100
Total	200	800	1,000

Table 1: Estimated amount of data for the MultiText portals (MultiText data example)

Like the classic portals, the data amount for further portals primarily depends on the data sizes of the images. Besides the main portals like news, sport and weather, the ORF provides special portals, which are adapted according to topics of current interest (e.g. Konkret, Urlaub etc.). Obviously, the amount of data increases with the number of portals. At the time of writing, the ORF broadcasted six MultiText portals (besides the start portal).

As already mentioned, these values must not be seen as absolute values. They should only help to roughly estimate the expected amount of data, which is approximately required for the MultiText.

4.1 Variant 1: Receivers with MHP plug-ins

An obvious solution to run the ORF MultiText services in an OpenTV environment is to extend the existing software stack of the receivers by corresponding MHP plug-ins. The idea behind this concept is to establish a coexistence of OpenTV and MHP applications. As a precondition, the MHP middleware needs to be available on every receiver (next to the OpenTV environment).

Architecture

On the one hand the receiver must support the OpenTV platform and its applications while on the other hand it should offer the features of the original FUSE architecture, which is based on the Java MHP middleware. OpenTV requires an OpenTV virtual machine and the MHP middleware assumes to have a Java virtual machine. Both must be running on the receiver.

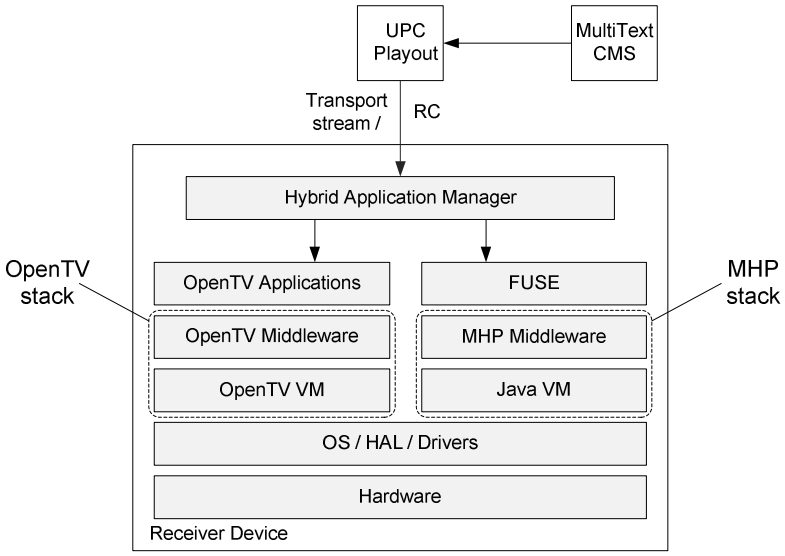

Figure 15: OpenTV receivers with MHP plug-ins

For this hybrid approach, the receiver must be able to handle applications for both software technologies. Thus, a hybrid application manager is needed that can switch between the relevant environments. To enable services that require a feedback channel, an interactive server is required.

Pros and Cons

This approach represents an obvious way to integrate the MultiText in OpenTV-powered boxes. In comparison to other variants the efforts of a realization are expected to be low. No development costs (on application level) are necessary, because the existing MultiText application is used. Hence this solution provides the original MultiText functionalities and preserves the intended appearance of the MultiText. Principally services with user interactions like chat and voting portals will be supported (require an appropriate server). In case of a MultiText update, new program versions need to be transmitted to the receivers. A possibility to update the MultiText on the receiver is to broadcast the updates via transport stream. Then listening boxes are able to install the updates individually.

To our knowledge, previous ambitions to integrate both environments did not succeed. Hence this variant would be a pioneering work in this field whereby experiences from similar projects are missing. Since every device must cover both environments (OpenTV and MHP require different environments), the software stack of the devices is substantially enlarged. The activation of the environment requires a "hybrid application manager".

Pros	Cons
no development costs for MultiText (on application level)	no mature coexistence currently known
high support of functionalities	large software complexity on the boxes
original appearance	MHP licenses expected
relatively unproblematic maintainability	

Table 2: Pros and cons of the MHP plug-in variant

Remarks

The MHP variant requires that the existing software on the receiver is extended. Therefore, charges for licenses are expected. In particular the license costs for MHP application and MHP software stack might be necessary. In cooperation with Alticast, OpenTV has developed the OpenTV MHP Package, which should allow the execution of MHP applications. However, the package was developed for MHP version 1.0.3 (MultiText requires MHP 1.1.2). According to our information, this project was not finished.

4.2 Variant 2: Native OpenTV implementation

Another approach to enable the MHP-based MultiText services on OpenTV-based receivers is to implement the MultiText for the OpenTV-based receivers from the scratch. However, a one-to-one mapping of the existing FUSE architecture, which is currently implemented upon MHP and is runnable on certified MHP boxes, is considered to be highly complex. Efforts for an implementation, which fulfills the original requirement of the FUSE architecture, would be very high. Therefore, a realistic aim will be characterized by a development of a customized OpenTV application, which incorporates a subset of the FUSE functionalities. When also interactive services like chat, online shopping or voting portals are required, a handling of the user management (resp. receivers) is additionally necessary.

Architecture

In this scenario the FUSE architecture (MultiText) is implemented as native OpenTV application. The existing software of the receivers is extended by the newly implemented MultiText application. Interactive services like chat and voting require a return channel and they communicate with an appropriate interactive application server.

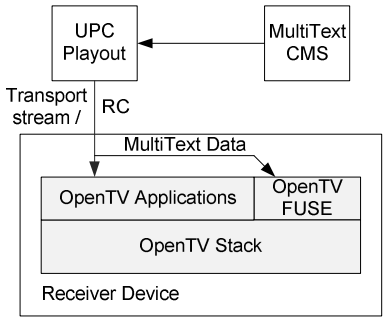

Figure 16: Native OpenTV variant

Pros and Cons

Since the services are implemented as native ANSI-C programs, this variant will probably achieve best performance results in comparison to other variants. However, in order to estimate the functionality and complexity on a more detailed level, the OpenTV architecture needs to be studied in detail. This requires having detailed technical specifications.

A crucial disadvantage of this variant is characterized by the efforts and costs it will require: time and development costs are expected to be huge. The OpenTV version of the Mutitext is implemented from the scratch. It will probably take a long time to develop the required functionalities so that the new software will run in a stable and robust manner on OpenTV-powered boxes. An original mapping of the FUSE functionalities is unrealistic, because OpenTV and MHP are totally different environments. Furthermore the boxes need to be extended by additional software. The appearance of the MultiText services depends; it might differ from the original and intended design. It should be noted that the described approach is based on a customized concept for Austria, representing a proprietary solution. Consider the case when the MultiText needs to be updated. A modification of the MultiText (implemented in Java) causes also an update of the OpenTV version (written in ANSI-C) of the MultiText. For a synchronization both versions need to be updated, which augments the efforts.

Pros	Cons
high performance and flexibility	high development costs
high functionality principally possible	additional software on the box required
	MultiText updates cause also updates of OpenTV application

Table 3: Pros and cons of the native OpenTV variant

4.3 Variant 3: Remote framebuffer protocol

A different approach following the thin client principle can be realized by the remote framebuffer protocol. Hereby the MultiText services are executed on a server, which transmits the screens of the MultiText sessions to the clients. The data is transmitted by the remote framebuffer protocol. Per user a separate MHP instance is running on the server. The clients receive the frames and display them on their output screens. As a prerequisite, framebuffer clients must be installed on receiver devices.

Architecture

In this variant the server executes a MultiText instance for each user separately. The current screen contents (frames) are sent to the receiver when requested. As a prerequisite, the receivers must contain framebuffer clients, which are able to receive and display the frames.

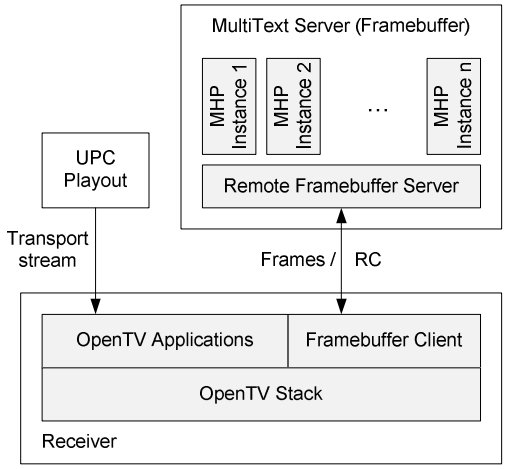

Figure 17: Framebuffer variant

When a user starts (resp. refreshes) the MultiText on the receiver, the desired data is requested from the MultiText server (remote framebuffer server). The server transmits the requested frames to the receiver, which displays them on the local screen. The frames are transferred by the framebuffer protocol. For interactive services like chat and voting, the relevant components communicate with an interactive application server. In contrast to the other variants, an integration of MultiText contents in the transport stream does not make sense in the given architecture.

Pros and Cons

An advantage of this concept is that, concerning MultiText services, no adaption is necessary. Besides, the clients do not need to make intensive computations. They only need to display the frames, which they receive from the server. The efforts for MultiText updates are low, because only the MultiText on the server needs to be adjusted. The appearance of the MultiText on the output device is (nearly) true to original, because the clients receive the presentation data of the original MultiText screens. Many MultiText services are supported (the MultiText of each user runs on the server); in particular interactive services like chat and voting will be included.

A serious disadvantage of the remote framebuffer variant is its scalability in context of the number of users: according to the number of users, the server has to manage a huge number of MHP instances. The server must be accordingly prepared such that it can cope with a multitude of users. Therefore, redundant services/servers may need to be installed to ensure the stability of the system. In critical cases, the number of concurrent users might be limited to a certain constant number so that a denial of services is prevented. The remote framebuffer variant presumes a communication channel providing a high bandwidth. Besides the running costs for the necessary servers, a framebuffer client must be installed on every receiver. Hereby an integration of the framebuffer clients into the middleware is necessary. Besides running costs for the servers, software license fees including MHP licenses (for the software on the server) are expected.

Pros	Cons
No FUSE adaption	scalability
thin clients	server resources
low efforts for MultiText updates	high bandwidth required
original appearance	FB-client on STB necessary
	box must provide a return channel

Table 4: Pros and cons of the remote framebuffer variant

Remarks

Note that a framebuffer-based approach can be used for miscellaneous applications (not only for the MultiText). Several technologies are developed that offer solutions for transporting framebuffers to remote clients. A possibility to realize the framebuffer variant is to utilize the Virtual Network Computing (VNC), which is based on the remote framebuffer (RFB). Further developments based on similar concepts are initiatives like Microsoft Remote Desktop Protocol (RDB), Citrix Presentation Server, Sun Ray, X11 (for a summary see [3]). VNC is a client-pull technique, i.e. the server send updates only after a client has requested them. In a VNC application for distance learning where the teacher remotely showed how to write emails (34 seconds presentation), a group of researchers reported a peak data rate of 4.32 Mbps at a resolution of 1152x882 resulting in 3.1 MB of transmitted data [4]. A further group of researchers have tested several implementations for different scenarios [5]. In particular they tested the scenarios of typing a character (single keystroke), scrolling text (44 lines in a 160x316 pixel area), filling a screen region (a 216x200 pixel area filled red) and downloading and displaying a 37 KB GIF image (320x240). For instance their VNC Linux client, a 172 KB program that needed less than 300 KB main memory, required a data load of 0.4 KB for typing a letter, 1.2 KB for the filling test, 3.2 KB for the text scroll, and 84.3 KB for the image test. The latency for the image test was around 600 ms and the latencies for the other tests were less than 100 ms. For further details on the test see [5]. It is argued that these technologies are not well suited for transmitting multimedia-intensive workloads over wide area networks. Note that these test suits should only act as a rough orientation to estimate the data payload.

The stability and operability of this variant depends on the number of users. Hence the following questions and issues should be clarified in advance:

- How many users are expected?
- How much resources (e.g. redundant servers) need to be mobilized?
- Which remote framebuffer product is suitable?
- How can the live television be integrated into the received MultiText screen on the box?
- Software fees including a magnitude of MHP licenses

4.4 Variant 4: Transmitting screenshots

The screenshot variant represents another way to transmit the current contents of the MultiText services from a server to the receiver devices. The idea of this concept is to create a screenshot from every MultiText page (and all its possible navigation statuses) and store them on the server. When a user wants to access MultiText contents, the corresponding receiver sends a request to the server, which returns the desired screenshot of the service.

Architecture

The system consists of the following components: the MultiText screen capturer, the receiver device, the playout center, and the MultiText CMS. Additionally an interactive application server is required for handling interactive services like chat and voting.

The MultiText screen capturer periodically inspects the MultiText data with all their references to subdivided contents and produces a screenshot per MultiText page. Together with the screenshots the navigation data (links to references, links to previous and next pages etc.) is encapsulated in documents that are offered by a server. The boxes request relevant documents from the server and use the navigation information to link the pages accordingly.

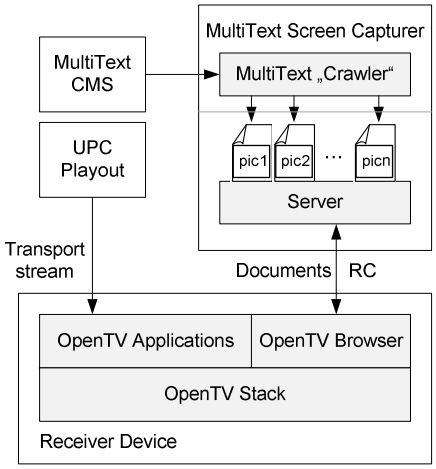

Figure 18: Screenshot variant

In order to reduce the data transmission, one might consider splitting the screen into several parts (see remarks below). Then only changed parts have to be updated. When HTML is used, the screen can be divided into frames so that only relevant frame contents need to be transmitted.

Pros and Cons

In contrast to others, this variant benefits from a license-free and lightweight construction: no new software (on application level) needs to be installed on the boxes. The costs and complexity of this approach are estimated to be somewhere between low and medium. Hereby the MultiText screen capturer needs to be implemented. The stability can be estimated as high. Only some software adaptations are necessary on the receivers. Received data (that includes the screenshots) is rendered by the existing presentation engine (e.g. HTML browser) of the receivers.

Regarding the functionality, this approach may have some restrictions depending on the supported features of the presentation engines on the receivers. When the MultiText has to be updated, the MultiText on the server needs to be adjusted. In comparison to others, the efforts for updating the MultiText are considered to be medium. Interactive services like chat and voting will require an additional user management. Assuming that enough resources are given, in terms of the number of users the scalability of this variant is expected to be at a moderate level.

Pros	Cons
low costs and resources	restricted functionality
thin clients	boxes must provide return channels
high stability (standard technology used)	MultiText updates cause also updates of the screenshot capturer
original appearance	

Table 5: Pros and cons of the screenshot variant

Remarks

The required amount of transmitted data depends on the size of the portals. In the beginning of this chapter the expected number of pages for the main portals is estimated. In the following table the estimated numbers of MultiText pages (screens) of the main portals are listed. For a rough estimation, the News Portal covers around 130 pages with 20 different navigation statuses. The Sport Portal encompasses a number of pages, which is usually less than that of the News Portal. As a rough estimation one might expect 110 pages (plus 15 navigation statuses). The Weather Portal might consist of approximately 50 pages (plus 20 navigation entries).

Portals	Number of Content Pages	Number of Navigation Entries
News	130	20
Sport	110	15
Weather	50	20

Table 6: Number of pages and navigation entries of the main portals (example)

To compute the total number of pages, also the possible focus statuses need to be considered: depending on the focus, per page several statuses are possible (see figure below).

Figure 19: Different focus statuses of the same page

When three focused components are used, the number of pages is tripled. In this case, the server needs to generate and offer 130 x 20 x 3 pages resulting in 7,800 pages for the News Portal. When we add the Weather Portal and the Sport Portal we have a total of 15,750 pages. For the screenshot variant this means that whenever a user goes through the navigation items, a full screen is requested from the server. Apart from the high amount of transmitted data, a smooth display can be problematic; especially when the menu items are quickly changed.

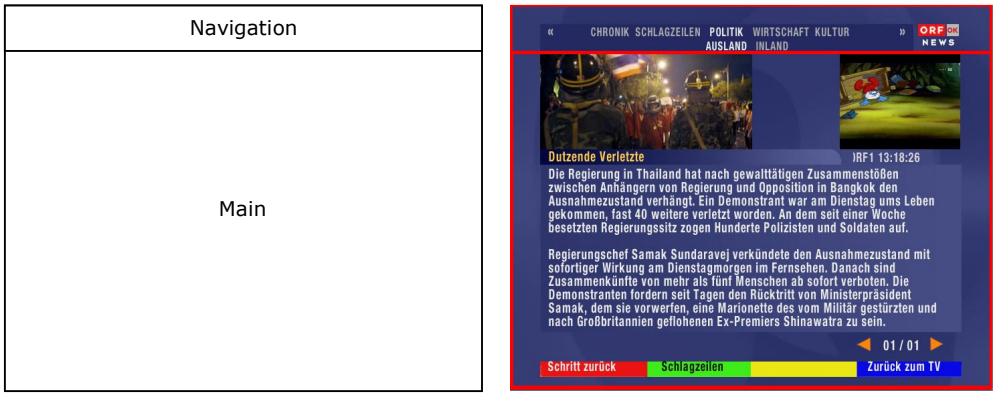

Figure 20: Splitting the screens into areas

Running through the menu items does only affect the appearance of the navigation bar. Therefore, an obvious improvement is to separate the navigation area from the rest of the screen (see above). Then, when a menu item is changed, only a small portion of the screen, namely the navigation area, has to be refreshed. Hence the boxes only need to request the current navigation part.

However, the screen splitting approach must be supported by the used presentation engine of the boxes. In case of HTML, frames can be used to divide pages into several areas.

Suppose that the navigation area can be separated from the full screen. According to the data given by the estimation, the News Portal, the Sport Portal, and the Weather Portal will require a total of 290 pages and furthermore 55 pages for the navigation of these portals. Additionally the different appearance for the focus must be considered. Screenshots in standard TV resolution are required, which is 720 x 576 pixels (if the page is divided, the resolution is accordingly reduced). When JPEG is used, the file size depends on several factors (inter alia the degree of its compression). For a rough estimation, one might expect a file size less than 100 kB for a JPEG in the given resolution that covers the main part of the screen. In comparison to the picture of the main part, the JPEG picture representing the navigation will require only a small fraction (approximately less than 10 kB).

4.5 Variant 5: Transcoding to OpenTV's presentation engine

Another possibility to provide MultiText in OpenTV environments is to implement a MultiText transcoder. Its task is to inspect the MultiText data and convert them so that the presentation engine of the OpenTV-based receivers can use them. For instance, if an HTML browser is supported, then the MultiText can be translated into and represented as HTML files. No new software components (on application level) need to be implemented for the receivers, because the MultiText data is converted so that it can be displayed by existing presentation engines.

Architecture

This approach is based on a server called "MultiText transcoder", which processes the broadcasted MultiText data. The MultiText data consists of XML files, which specify the layout, navigation, and contents; it may also contain multimedia contents like pictures and audio files. The main task of the transcoder is to convert the MultiText data into the format of the used presentation engine. When HTML is used as the desired target presentation language, then the translated data can be displayed by an appropriate HTML browser. As in other server-based variants described in this feasibility study, the transcoding server can be located anywhere as long as the server can be accessed by the boxes and the server can access the MultiText data (either from the broadcast channel or from the CMS output).

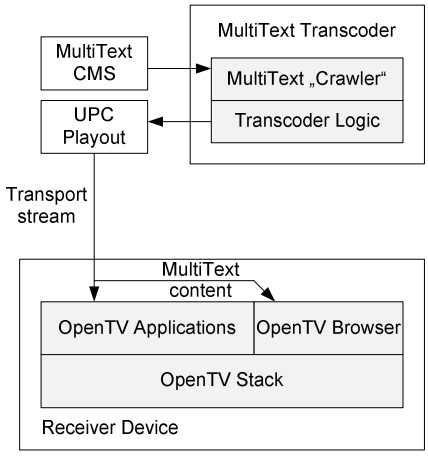

Figure 21: Transcoding concept

Pros and Cons

A main advantage of this variant is that receivers do not need to be supplemented with additional programs or services. The receivers use their default presentation engine (e.g. OpenTV HTML browser) to display the converted MultiText portals. Compared with other variants, the costs are expected to be lowest. A realization of this approach is expected to run in a stable manner, because the existing standard modules (OpenTV features) are used. In addition to this, changes and updates of the MultiText can be implemented on the server and, hence, the clients do not need to be reconfigured.

The appearance of the MultiText (page layout, colors, positioning etc.) depends on the possibilities that are provided by the presentation engine of the receiver devices. Also the functionality is up to the target platform. In principle, interactive services like chat and voting are possible but require a handling of user sessions on the server. When some MultiText services have to be modified are when some services are added, the transcoding logic needs to be adapted. In comparison to the other variants, the efforts for the adaptations are estimated to be moderate.

Pros	Cons
no box updates	different appearance
no license fees	restricted functionality
low costs and low resources	MultiText updates cause also updates in the transcoder
high stability (standard technology used)	

Table 7: Pros and cons of the transcoding variant

Remarks

Referring to Table 1, the sizes of the main MultiText portals are roughly estimated as 450 kB for the News Portal, 330 kB for the Sport Portal, and 20 kB for the Weather Portal. The main parts of these file sizes are caused by the images. When the transcoding variant is used, a similar amount of data is expected in order to support these portals on the boxes.

5. Comparison

According to the functional scope, a comparison of the presented variants is given in the first part of this section. In the second part the complexity of the variants is evaluated from several viewpoints.

5.1 Functional scope

In this section the possible variants are summarized and compared with each other. The following table gives an overview of the corresponding approaches with their supported features.

	MHP Plug-in	Native OTV	Framebuffer	Screenshot	Transcoding
Animations	X	X	O[1]	—	O[3]
Stream events	X	O[2]	X	—	—
3rd party xlets	X	X	X	—	—
Virtual keyb.	X	X	X	O[3]	O[3]
Audio clips	X	X	—	O[4]	O[4]
Live video	X	X	—	O[5]	O[5]
Dynamic data	X	X	X	O[6]	O[6]
Navigation	X	X	X	O[3]	O[3]

Table 8: Functional comparison of the proposed variants

X … supported

O…conditionally supported

—… not supported

[1] Principally possible; a smooth display is questionable
[2] Expected to be possible, but not verified
[3] Dependant on the browser; this requires additional programming (adjustments)
[4] Box specific
[5] Dependant on the presentation engine
[6] Polling mechanism required

Annotation:

Animations:	animated navigation, animated clock, prompting cursors etc.
Stream events:	control events for boxes (triggers)
3rd party xlets:	other applications
Virtual keyboard:	support of virtual keyboard
Audio clips:	playing audio files
Live video:	displaying live TV
Dynamic data:	content updates
Navigation:	selecting menu items, back- and forward navigation

MHP variant

In the MHP plug-in strategy the software of the boxes is extended so that the original FUSE architecture can directly be used on OpenTV-powered boxes. This allows supporting the original functionality and appearance.

Native OpenTV variant

When a native OpenTV application is considered, it is difficult to estimate which features will be supported. The reason for this is that useful technical documentation about OpenTV is insufficiently freely available. In the native variant, the application must be built up from the scratch. Hence it is principally possible to implement desired features (though maybe at high costs). However, several questions need to be clarified; in particular concerning processing stream events, handling live videos in the OpenTV environment, and supporting dynamic data; it is principally assumed that the features live TV and dynamic data can also be realized in the OpenTV environment.

Remote framebuffer variant

The remote framebuffer variant executes the MultiText on the server and transmits the screens to their clients. However, the transmission of moving images and video streams are not recommendable. Apart from the resulting magnitude of transmission data, a smooth display of videos on the boxes is questionable. Nevertheless a transmission of short and small animations is feasible. Besides animations, the remote framebuffer protocol does usually not transmit audio data.

Screenshot variant

The screenshot version and the remote framebuffer variant share some common ideas. In both approaches the MultiText is processed on a server whereby the boxes are supplied with the corresponding screens. In contrast to the remote framebuffer strategy, the screenshot variant does not execute a MHP instance per user separately. It only offers screenshots of the current MultiText pages and delivers them on request. Therefore, the functionality range is limited. The transmission of animations would increase the data load significantly and a smooth display of the animations based on the screenshot approach is hardly possible. When the box is used as presentation engine only, stream events and 3rd party xlets are not supported. The integration of stream events is principally possible but its support requires additional software on the boxes. When users need to be handled individually, user sessions are required. The ability to include features like virtual keyboard and playing audio clips depends on the functionalities of the presentation engine. When a browser is used as presentation engine, it can be considered to implement these features in JavaScript. The possibility of playing audio clips depends on the audio players the box is providing. Similarly the integration of live videos is up to the ability of including live videos in OpenTV's presentation engine. The use of dynamic data requires a polling mechanism. Also the navigation may be restricted; this depends on to which extent the navigation can be realized on the presentation engine.

Transcoding variant

In the transcoding variant, the MultiText data is converted so that it can be displayed on the presentation engine of the receivers. The functional scope of this strategy is limited. It depends on the features and the functionalities, which the receivers are offering, i.e. features that are not provided by the presentation engine are not supported. In particular it is expected that stream events (although principally possible) and third party xlets will not be included. Assuming that the target language is HTML, JavaScript can be used to realize (limited) animations, virtual keyboard, and page navigation. Analogous to the screenshot variant, the utilization of dynamic data requires a polling mechanism on the box, the support of live videos depends on how live videos can be included in the utilized OpenTV presentation engine, playing audios requires box-specific audio players, and an individual processing of user-dependent data presumes to have user sessions.

5.2 Evaluation

According to our estimations, we evaluate the proposed variants from several viewpoints and describe potential problems. The rated issues are divided into categories; namely costs, operating, and functionality. The rating is performed by numbers from 1 to 5, whereby low numbers are better than high numbers. To emphasize the rating, the numbers are highlighted by the colors dark green (1), green (2), orange (3), red (4), and dark red (5). We describe the rating of each category separately and summarize the overall score afterwards.

5.2.1 Costs

In the following table the variants are rated in terms of cost factors. According to our estimation, the transcoding variant is likely the cheapest variant. Note that the summary assumes that all costs factors are equally rated. Therefore the summarized values should only act as a rough estimation.

COSTS	MHP Plug-in	Native OTV	Framebuffer	Screenshot	Transcoding
Development costs	5	5	3	2	1*
Maintainability costs	1	5	1	3	2
Operational costs	1	1	4	2	2
MHP license costs	4	1	3	2	1
Total	**11**	**12**	**11**	**9**	**6**

* In case when HTML is used as target language

Table 9: Comparison of arising costs

Development costs

- MHP plug-in variant: The complexity of the MHP plug-in version depends on the feasibility of a concurrent execution of OpenTV and MHP applications. It should be remarked that efforts to include MHP services in OpenTV did not succeed. Since one cannot take advantage of a finalized practical realization, we expect a very high development complexity.

- Native OpenTV variant: A native OpenTV implementation of the MultiText is expected to require a very high amount of development costs, because the MultiText must be designed and implemented from the scratch.
- Remote framebuffer variant: In the remote framebuffer variant a separate MHP instance is running per user. Costs concerning the server farm, which will be necessary to ensure to handle a multitude of users, are necessary. In comparison to the other versions, we expect medium-sized development costs.
- Screenshot variant: The screenshot approach is estimated to have a moderate complexity (here rated with 2): a special server-sided software component needs to be designed that periodically inspects the MultiText data, generates screenshots of the MultiText pages and produces pages that include these screenshots. Additionally information about the navigation (e.g. how pages are linked) is necessary.
- Transcoding variant: In terms of development costs, the transcoding approach will likely be the most favorable strategy, provided that an adequate target language (like HTML) is used. The main efforts are given by the implementation of the translation logic on the server.

Maintainability costs

- MHP plug-in variant: When MultiText features must be added or changed, the software on the boxes needs to be updated. A software update can be realized by including the program updates in the transport stream.
- Native OpenTV variant: The maintainability of the native OpenTV variant is considered as difficult, because updates of the MultiText (implemented in Java) need to be "translated" in OpenTV (based on ANSI-C) and the boxes need to be updated with the new version.
- Remote framebuffer variant: The maintainability of the remote framebuffer variant is considered to be unproblematic, because a new version of the MultiText needs to be updated on the server only.
- The screenshot variant: In case of a software update, the MultiText on the server needs to be refreshed. An update may also cause an adaption of the MultiText crawler, which inspects and generates the images of the MultiText pages. In relation to the other variants, the maintainability costs are considered as medium.
- In the transcoding approach the transcoding logic needs to be adjusted. The complexity of this adjustment is rated with 2.

Operational costs

Noteworthy operational costs are necessary in those variants, which need additional servers. The remote framebuffer variant and the screenshot approach offer MultiText services on a server. Also in the transcoding strategy a server, which executes the transcoding operations, is considerable. For these variants operational costs are expected. The operational costs for the remote framebuffer variant can become costly, because hereby several servers may be necessary.

MHP license costs

- MHP variant: In the MHP variant freely accessible channels (free-to-air TV) are not charged. Generally, if MHP-based services from third party providers are "relayed", then usually no MHP license fees are necessary.
- Native OpenTV variant: For the native OpenTV approach no MHP licenses are required, because the resulting program will use the OpenTV middleware (instead of MHP).
- Remote framebuffer variant: In the remote framebuffer strategy, for each user a MHP instance is running. Hence costs for the MHP software arise whereby the software license must include a magnitude of users. For the MHP stack on the server a MHP license may also be required.
- Screenshot variant: In the screenshot variant a MHP license for the server is necessary.
- Transcoding variant: For the transcoding variant no license fees are necessary.

5.2.2 Operating

The second category reflects operating issues. One aspect covers the scalability. Hereby the impact of an increase of users is reflected. Another issue addresses the adaption efforts of the boxes in order to comply with the corresponding strategy. In this category, the native OpenTV and the transcoding variant achieve best results.

OPERATING	MHP Plug-in	Native OTV	Framebuffer	Screenshot	Transcoding
Scalability	1	1	5*	2*	1
Receiver adaption	5	1	3	1	1
Total	**6**	**2**	**8**	**3**	**2**

* Corresponding variant requires a server

Table 10: Comparison concerning scalability and receiver adaption

Scalability

- MHP plug-in variant, OpenTV variant, and transcoding variant: When the MultiText data is embedded in the transport stream, the MultiText data is locally available for every box. Hence a concurrent use of the MultiText has no impact on the availability of the services (provided that no interactive services like chat and voting services are used). The embedding strategy is practically considerable for the MHP plug-in, for the native OpenTV approach, and for the transcoding variant.
- Remote framebuffer variant: Consider the case when countless users concurrently use MultiText services. Since an MHP instance per user is created, a concurrent use of services has direct consequences on the computational efforts on the server. To ensure the availability of services, several servers (server clusters) may be necessary.
- Screenshot variant: In a server-based distribution as given by the screenshot approach, desired images (plus navigation information) are transmitted from the server to the boxes. In comparison to the remote framebuffer variant, only a single MHP instance is running on the server (independent on the number of users). Thus, the situation is expected to be less precarious: countless requests may influence the server's response time.

Receiver adaption

- MHP plug-in variant: A realization of the MHP plug-in variant requires that the boxes are equipped with an appropriate MHP version. Additionally the MultiText must be available on the boxes. Regarding the size of the used software, this approach requires a very high amount of resources on the boxes. A way to update the box software is to embed the relevant software in the transport stream and broadcast it so that every box can retrieve and install the software locally. Additionally the software on the boxes needs to be configured such that the relevant software stack can be activated.

- Native OpenTV variant: In the native OpenTV variant the new application uses the existing OpenTV stack. Since the software is based on the OpenTV stack of the existing boxes, no further software stacks and adaptations are necessary.

- Remote framebuffer variant: In the remote framebuffer approach it is assumed that the boxes are equipped with corresponding framebuffer clients. Hence framebuffer clients need to be installed on the boxes. They may also be configured. In comparison to the other variants, the adaption efforts are rated as medium.

- Screenshot variant and transcoding variant: In these approaches the MultiText services are displayed by the OpenTV presentation engine. The software on the boxes does not need to be revised. No or only marginal adjustments are expected; however, when features have to be included that are inherently not supported by the corresponding variant, some configurations and programming (particularly concerning the presentation engine) might be necessary.

5.2.3 Functionality

The third classification reflects functionality aspects of the variants. First we evaluate the grade of FUSE functionalities that are supported by the described variants. Second we evaluate to which extend the appearances of the described variants are expected to vary from the original MultiText. Finally we estimate the expected latencies for the different variants.

FUNCTIONALITY	MHP Plug-in	Native OTV	Framebuffer	Screenshot	Transcoding
FUSE Functionality	1	2	2	3	3
Appearance	1	3	2	2	3
Latency	1	1	3*	2	1
Total	**3**	**6**	**7**	**7**	**7**

* depends on the grade of the service utilization

Table 11: Comparison concerning functionality

Functionality

- MHP plug-in variant: In the MHP plug-in strategy the original FUSE platform is installed on each box separately. The functionality of this variant encompasses the same features as provided by the original FUSE architecture. Hence its functionality is expected to be very high.
- Native OpenTV variant: The functionality of the native OpenTV approach depends on the OpenTV environment, whose functionality is principally assumed to be high.
- Remote framebuffer variant: The remote framebuffer variant is expected to provide a high grade on functionalities. Since every user is handled by a separate MHP instance on the server, dynamic data, stream events, virtual keyboard and navigation are supported. However, the transmission of videos is not recommended and audios are generally not supported by the remote framebuffer protocol.
- Screenshot variant and transcoding variant: In these variants the MultiText is processed so that the presentation engine of the receivers can display the MultiText contents. Several

features are not supported; some of them require a client-side implementation. A detailed list of supported functionalities of the variants is given in Section 5.1.

Appearance

- MHP plug-in variant: Since the original MHP MultiText is executed on the boxes, the original appearance of the MultiText is provided.
- Native OpenTV and transcoding: It is expected that in the native OpenTV strategy and in the transcoding variant the MultiText appearance differs from the original one, because in both variants the MultiText data is processed so that it complies with OpenTV's presentation engine.
- Remote framebuffer variant and screenshot variant: The MultiText data is displayed as screenshots from an original FUSE application. The appearance is expected as nearly original.

Latency

- MHP plug-in variant and native OpenTV variant: The latencies of the MHP plug-in and the native OpenTV variant will be negligible (MultiText is retrieved from the transport stream).
- Remote framebuffer variant: In the remote framebuffer variant the response time can become problematic: intensive concurrent use of MultiText services will require high server efforts and high data payloads. This may also increase the latency.
- Screenshot variant: In the screenshot variant the transferred data consists of images. When well considered (e.g. splitting screens), the latency is expected to be low.
- Transcoding variant: The latency in the transcoding variant is expected to be unproblematic. Less transmission data is expected than necessary in the previous two variants.

5.2.4 Overall comparison

Referring to the categories, the following table reflects the average rating of the described variants. Hereby the average values of the categories are used: for each category the sum of the values is divided by the number of investigated items in the corresponding category. Analogous to the average category values, the total values represent the average values of all three categories.

AVERAGE SCORES	MHP Plug-in	Native OTV	Framebuffer	Screenshot	Transcoding
Costs	2.75	3.00	2.75	2.25	1.50
Operating	3.00	1.00	4.00	1.50	1.00
Functionality	1.00	2.00	2.33	2.33	2.33
Total (Average)	**2.25**	**2.00**	**3.03**	**2.03**	**1.61**

Table 12: Average scores

Note that the scores are based on equally weighted values. According to one's sight and preference, the values (of each category respectively of each item in the categories) need to be weighted separately. Here only the situation is presented when all categories are equally important.

According to our evaluation, the transcoding variant achieves an overall score of 1.61 and is ranked on the first place. The native OpenTV variant and the screenshot variant have nearly the same overall score. The MHP plug-in variant achieves a score of 2.25 and is ranked on fourth place. On final place we have the remote framebuffer variant, which has a score of 3.03 according to our ranking.

6. Summary

The aim of this work is to describe and evaluate possible strategies to activate the MultiText on UPC boxes. Generally the strategies can be divided into three main classes: either the MultiText presentation engine is included on the box or the main part of the MultiText presentation engine is executed remotely or the MultiText content data is transcoded to a format that can be interpreted by another presentation engine. The MHP plug-in variant and the native OpenTV variant fall into the first category while the remote framebuffer variant and the screenshot variant are based on the second approach. In the transcoding strategy the MultiText content data is converted to another presentation format (third category). In the following we briefly summarize the variants.

6.1 MHP plug-in variant

An obvious approach to enable MultiText on boxes based on non-MHP compliant platforms is to extend the boxes with the MHP (1.1.2) stack so that the MultiText data stream could be used as is. However, in this variant the software on the boxes would be significantly increased. Furthermore the boxes must be able to switch between the two software stacks. To our knowledge attempts to extend OpenTV with an MHP stack did not succeed.

6.2 Native OpenTV variant

Another possibility is to implement a MultiText presentation engine from scratch in order to fit in the existing software stack of the boxes. The costs for a new development of the MultiText presentation engine as an OpenTV O-code application would be very high. Several issues in this variant remain questionable and, in case this variant gets selected despite of its potential high costs, the questions need to be addressed in more detail.

6.3 Remote framebuffer variant

A further realization strategy is based on the remote framebuffer approach, where the screen contents (frames) of the corresponding MultiText instances are transferred from the server to the boxes. However, for each box a separate MultiText instance is required. Therefore the availability of the services may become a serious problem. This variant requires that a framebuffer client is installed on each UPC MediaBox.

6.4 Screenshot variant

Instead of transferring frames, the transmission of screenshots can be considered. In the screenshot variant from each MultiText page a screenshot is produced. The produced pages are offered by a server, which the boxes can access. In case that no HTML browser exists on the box, adaptations of the STB's software stack are necessary.

6.5 Transcoding variant

The final strategy of this feasibility study is characterized by a conversion of the MultiText content data into a data format that can be processed by the UPC MediaBox. Assuming that such a presentation engine is already available respectively the MediaBoxes can be upgraded by means of a software image update, the MultiText content data is rendered by the presentation engine. This strategy has several advantages. Apart from marginal adaptations, no additional software needs to be installed on the boxes. The stability of this variant is expected to be high. In comparison to the other variants the complexity and costs are lower, although, of course, the costs of implementing and maintaining the transcoding process must be considered.

A. References

[1] OpenTV http://www.opentv.com

[2] M. Fagerqvist, A. Marcussen: Application and System Migration from OpenTV to DVB-MHP (Diploma Thesis), University of Lulea, Finland, 2000

[3] S. Linecker: Television Remote Framebuffer Protocol (Bachelor work), University of Vienna, Austria, 2007

[4] S. F. Li, Q. Stafford-Fraser, and A. Hopper: Frame-buffer on Demand: Applications of Stateless Client Systems in Web-based Learning, 1999

[5] J, Nieh, S. Yang and N. Novik: A Comparsion of Thin-Client Computing Architectures, Technical Report, Columbia University, 2000

B. Appendix – Some Technical Details for the Transcoding Variant

In the following section we identify the main tasks and sketch out some technical details for realizing the transcoding variant. As a prerequisite, the used target language must be well known. In particular we consider that HTML (using CSS style sheets and JavaScript) will be used as target language. Note that the support of some features may depend on the used HTML dialect. This issue needs to be investigated in advance.

A MultiText page consists of two XML files, the content file and its corresponding layout file. While the content files only contain content information, like which pictures to use or which text to display, the layout files contains information about the background image, the size and position of the content elements, their used fonts and colors and the focus traversals. Using HTML as target format, layout information may be converted into CSS style sheets.

Starting with the initial page (the MultiText entry page), the crawler will inspect all the content files recursively. It will build an optimized map of all reachable pages and convert their contents to the target format, preserving the link structure of the original MultiText pages. The output of the transcoding process will be a set of linked HTML files with their CSS style sheets and supplementing JavaScript functions.

Notice that some MultiText images possibly have to be processed so that the HTML Browser can display them. For instance, MultiText uses MPEG I-Frames as background images and colors them using a semi-transparent rectangle overlaying the original image. If such image formats or techniques are not valid for the target platform, an image processor has to convert the pictures into a valid picture format.

For interactive components like the animated navigation, virtual keyboard, entry boxes or animated images, JavaScript has to be used. The idea is to implement a common JavaScript library which is used by the generated HTML pages. Features like navigating between subpages on the content container will be executed locally and will not require a full HTML page request. Thus, the server communication over the return channel will be minimized and the system response time will be faster.