



FP6-2005-IST-61-045410

MOBISERVE

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

**Deliverable:
MOBISERVE specification for open API's
D4.3**

Due date of deliverable: M9
Actual submission date: 02.08.2007

Start date of Project: 01 September 2006

Duration: 24 months

Responsible WP: Streamezzo

Revision:

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)		
Dissemination level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission Service)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	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	17/06/07	Starting template, split of combined deliverable
V0.2	20/06/07	Skeleton generation
V0.3	25/06/07	Version with STZ,FT, inputs
V1.0	01/08/07	Submitted version

0.3 Document data

Keywords	API TERMINAL ARCHITECTURE
Editor Address data	Name: Pierre Pleven Partner: Streamezzo Address: 21 BD Victor Hugo 75016 Paris France E-mail: pierre.pleven@streamezzo.org
Delivery date	

0.4 Distribution list

Date	Issue	E-mailer
01/08/07		Al_mobiserve_all@natlab.research.philips.com INFSO-IST-045410@ec.europa.eu

Table of Contents

0	DOCUMENT INFO	2
0.1	Author	2
0.2	Documents history	2
0.3	Document data	2
0.4	Distribution list	2
1	INTRODUCTION	6
2	TARGET LOGICAL TERMINAL ARCHITECTURE	7
2.1	Olympic Architecture.....	7
2.2	Components of the Olympic architecture.....	8
3	FLOW DIAGRAMS FOR USE CASES	9
3.1	Reminder of the use cases	9
3.2	Favorable P1 use cases	11
3.2.1	ALERTS FROM THE EVENTS ON THE AIR (1).....	11
3.2.2	POLLING (3).....	12
3.2.3	TIME-SHIFTING - SMART PAUSE (4).....	14
3.2.4	PROGRAM ASSOCIATED INFORMATION (5)	15
3.2.5	USER FEEDBACK (11).....	18
3.2.6	ESG (20).....	19
3.2.7	RICH MEDIA ENABLE ESG (21)	21
3.2.8	RICH MEDIA FUNCTION SUPPORT (22)	21
3.2.9	LOCAL (RE)PLAY (31).....	21
3.2.10	USUAL MOBILE PHONE FUNCTIONS WHILE WATCHING TV (32) 21	
3.2.11	INTERACTIVITY WITH THE TV PROGRAMS (34)	21
3.2.12	EMERGENCY NOTIFICATION (35).....	22
3.2.13	RELAY FROM DVB-H TO WIFI (36)	23
3.1.14	HIGH QUALITY VIDEO TRANSFORMING FROM MOBILE TO TV (37)	24
3.3	Favourable P2 and P3 use cases	25
3.3.1	PROGRAM INDEPENDENT DATA (6)	25
3.3.2	MULTI-LANGUAGE (8)	25
3.3.3	SELECTION OF PREFERRED COMMENTATORS (9).....	26
3.3.4	ONLINE SHOPPING - ASSOCIATED WITH PROGRAM (10).....	26
3.3.5	LIVE PROGRAM (14).....	27
3.3.6	DOWNLOAD - SCHEDULED (15).....	27
3.3.7	PVR (16).....	28
3.3.8	USER PROFILE AND PREFERENCE - USER INITIATED (17)	28

3.3.9	ACCESS TO INTERNET THROUGH RICH MEDIA INTERFACE (23) 29	
3.3.10	VOD (29)	30
3.3.11	INTELLIGENT ESG PRESENTATION (30)	30
3.3.12	EXCITING CLIPS AVAILABLE WHILE WATCHING THE LIVE TV PROGRAM – P3 (33).....	30
3.3.13	INDOOR NON-INTERRUPTION (40).....	32
3.3.14	LOCATION DEPENDENT CONTENT (41)	32
3.3.15	32	
4	COMPONENT DATA MODELS	33
4.1	Content handling data model	33
4.2	Metadata database data model.....	34
4.3	IPDC Notifications data model.....	35
5	COMPONENT COMMUNICATION AND API'S	36
5.1	Content handling interface	36
5.2	Metadata database interface	37
5.3	Notification IPDC interface.....	37
5.3.1	IPDC STACK.....	37
5.3.2	IP DATACAST NOTIFICATIONS	38
5.3.3	METADATA DATABASE COMPONENT	39
5.3.4	CONTENT HANDLING COMPONENT	39
5.3.5	ISAP DATA RECEPTION.....	40
5.3.6	ISAP FEEDBACK HANDLING	40
5.3.7	ISAP SYNCHRONIZATION.....	41
5.3.8	ISAP INTERACTIVE SERVICE MANAGEMENT	41
5.3.9	MEDIA PLAYER / RECORDER AND CODEC	42
5.3.10	IP DATACAST ESG PRESENTER	43
5.3.11	RICH MEDIA ENGINE.....	43
5.3.12	DATA MODEL PRESENTER	43
5.3.13	APPLICATION.....	43
6	APIS	44
6.1	ISAP API'S	44
6.1.1	INTRODUCTION.....	44
6.1.2	ISAP APIS' CATEGORIES	44
6.1.3	ISAP API'S DEFINITION	45
6.2	Rich media API overview	46
6.3	Rich Media API Reference.....	47
6.3.1	CLASS CRMENGINEINTERFACE	47
6.3.2	CONSTRUCTOR & DESTRUCTOR DOCUMENTATION.....	47
6.3.3	CONTROL FUNCTIONS DOCUMENTATION	48
6.3.4	DOM API OPERATIONS DOCUMENTATION	52

6.4	Class MRMEngineObserver	57
6.4.1	MEMBER FUNCTION DOCUMENTATION	57
6.4.2	MEMBER ENUMERATION DOCUMENTATION.....	59

1 Introduction

This document describes at API level 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.

2 Target logical terminal architecture

In Mobiserve, two terminal architectures are discussed, Olympic Architecture and Advanced Architecture. Only said Olympic architecture will be ported to the target terminal and detailed API's are described for this architecture

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, and the Java part, not ported will not be described at API level.

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.

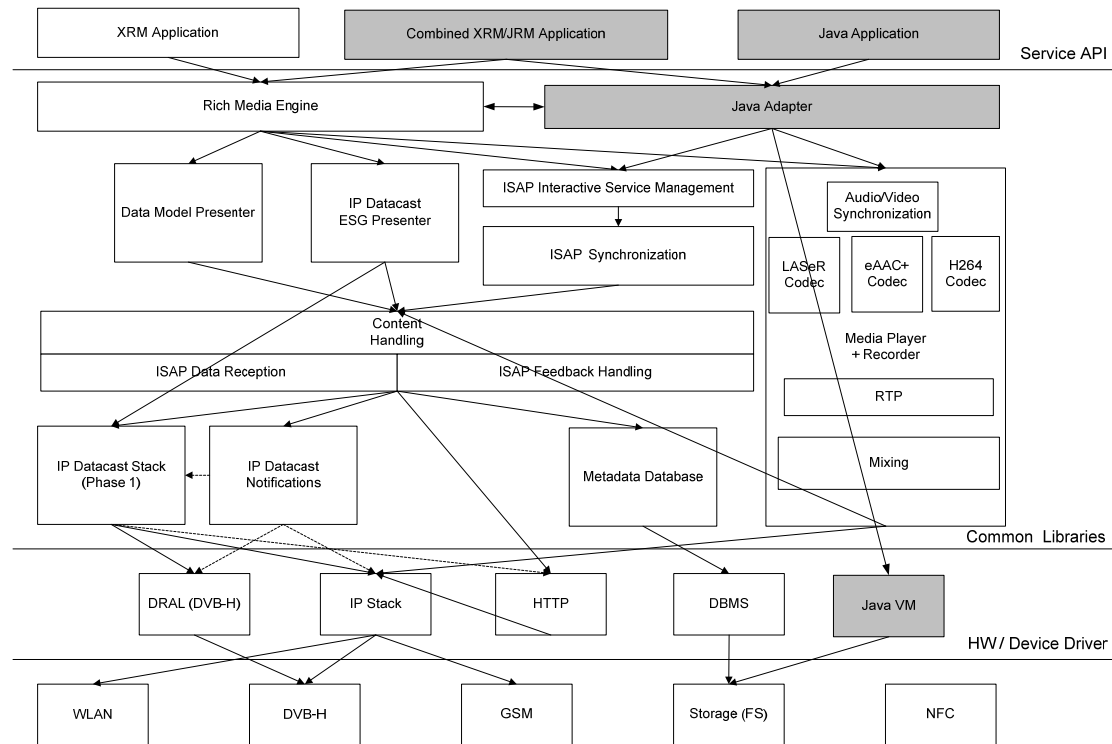


Figure 1 Mobiserve Architecture Diagram

Grey boxes are part of the advanced architecture and will not be described in this document

2.2 Components of the Olympic architecture

As per minutes of Beijing workshop the different modules have been attributed to the following partners

- Rich Media Engine :Streamezzo (T4.2)
- Data Model Presenter : Streamezzo(T4.2)
- IP Datacast stack : S3 (T4.2)
- IP Datacast Notifications : Philips Eindhoven(T4.2)
- Metadata Database : Philips Eindhoven (T4.2)
- ESG presenter : Streamezzo (T4.2)
- ISAP Interactive service Management France Telecom (T4.3)
- ISAP synchronisation France Telecom (T4.3)
- Content Handling Philips Eindhoven (T4.3)
- ISAP Data Reception France Telecom (T4.3)
- ISAP feedback handling France Telecom (T4.3)
- AV Player NXP (T4.4)
- RTP stack NXP (T4.4)

3 Flow diagrams for use cases

3.1 Reminder of the use cases

As per Deliverable 1.3, for full detail please refer to the original document

Use case Number	Use Cases	Description	Global Feedback from partners		
			Estimated implication on cost of the equipment (low, medium, high, unknown)	Estimated R&D cost (low, medium, high, unknown)	Recommendation to implement it (favourable, no opinion yet, not favourable)
No 1	Alerts from the events on the air.	Alert on noticeable events such as an interested tennis match might be reaching the end of match (match points) and offer the possibility to the user to switch directly to that channel. The alerts could be also grouped by interest. This kind of service could be subscribed.	low	medium	favourable
No 2	Zoom (Multi-Angle)	The TV program can be enjoyed from different angle such as focusing on the interested player.	medium/high: Multi-angle capability	high	not favourable
No 3	Polling	Collection of the end users' feedback (e.g. voting) or play (e.g. betting).	low	low	favourable
No 4	Time-Shifting (Smart Pause)	The ability to record the actual viewed program while switch to another activity such receiving a phone. As such, the user would not loose any piece of the viewed program.	medium/high: time-shifting capability	high for terminal low to medium for other modules	favourable
No 5	Program associated information	The TV program related information is broadcasted along with the associated TV program. For instance, the description about the teams participating to the football match. Inside this information, there may be some links (URL) which allows user to get further information on a specific interested issue like a player biography.	low	low for equipment... high for content	favourable
No 6	Program independent data	Broadcast of the program independent data such as weather, traffic	low	low	favourable P2
No 7	Stadiums Guide (with MAP)	To be guided on the way to go inside and outside the stadiums	Yes: Positioning capability (e.g. GPS)	medium	not favourable
No 8	Multi-Language	Multi-lingual for the TV program as well as the broadcasted information	low	medium	favourable P2
No 9	Selection of preferred commentators	While viewing a competition, one may have the possibility to choose the preferred commentators (sportscasters)	low	low	favourable P2

No 10	Online shopping (associated with program)	TV program triggered online shopping. That is, the goods are shown in the TV program, and the online shopping of those goods is suggested.	low	low for equipment... high for content	favourable P2 (needs a web site)
No 11	User feedback	User may want to express his/her opinion regarding the viewed programs or other suggestions.	low	medium	favourable if same mechanism as use case 3
No 12	Online Gaming	Game download via broadcast channel, local or online play and online ranking.	low	medium-high	not favourable (risky)
No 13	Multi payment modes	Different payment modes may be offered to the users, namely, by theme (for instance football or formula 1), pay-per-view, daily, weekly, monthly, etc.	low	medium-high	not favourable
No 14	Live program	Repurpose (retransmission) of the live TV program	low	low	favourable P2
No 15	Download (scheduled)	Any kind of data could be scheduled to be downloaded via the broadcast channel.	low	medium high	favourable P2
No 16	PVR	Based on the usage profile stored on both the mobile device and the home entertainment system, all relevant and potentially interesting services (Audio/Video and applications) can be stored in the PVR system.	medium/high: per capabilities	high	favourable P2
No 17	User profile and preference (user initiated)	User might be able to define his/her profile and preference.	low	medium	favourable P2

No 28	Breach-of-the-rule detection	The end-user device detects automatically potential breach-of-the-rule situations and highlights the scene and gives a comprehensive description. The user can compare the device decisions to those of the referee.	medium/high (3D analysis capabilities, breach of rule detection capabilities)	very high	not favourable
No 29	VOD	Video on Demand	medium/high (VOD)	medium	favourable P2
No 30	Intelligent ESG presentation	Channels of interest to a customer can be grouped together at the end-user device.	(see above)	medium	favourable P2 (mixed between use case 21 and 25)
No 31	Local (replay)	This means the possibility offered by the terminal to slow down the motion and freeze. It can be applied to both the live TV program and recorded one.	medium/high (PVR capabilities)	medium-high	favourable
No 32	Usual Mobile phone functions while watching TV	Still being able to use the usual mobile phone while enjoying the TV program.	medium/high (native phone function)	medium	favourable
No 33	Exciting spots (clips) available while watching the live TV program	For instance, during a basketball, the content provider cuts the interesting moments (clips), such as a nice shoot, along with the competition and notifies the user whenever these clips are available	medium/high (VOD)	high	favourable P3
No 34	Interactivity with the TV programs	Being able to influence the broadcasted program's progression, whether pre-recorded or live	medium	high	favourable

No 18	Automatically usage statistics data collection (system initiated)	The usage statistics such as which programs have been seen by whom, for how long, at what time, etc.	low	high	not favourable
No 19	Diary	Record one's own daily experiences: what did I see? What is my opinion? Etc.	low	high	not favourable
No 20	ESG	Electronic Service Guide	medium/high (IP Datasat Stack)	low medium	favourable
No 21	Rich Media enable ESG		(see above)	low high	favourable
No 22	Rich Media function support	AV control, Media library, VoD	low	medium high	favourable
No 23	Access to Internet thru Rich Media interface		low	low	Favourable P2
No 24	Overlay information of choice		low	high	not favourable
No 25	Top list recommendation	Based on the profile and the preference of the user.	low	medium high	not favourable
No 26	Content upload	User might want to upload content in order to let other people share his/her experience.	low	medium high	not favourable
No 27	3D scene analysis	A 3D model of the current game situation may be present to the user. By performing gestures with the pen on the touch screen, the user is able to rotate and zoom into the scene. Important details like offside (football game), defense assignment (basketball game) will be highlighted.	medium/high (3D analysis capabilities)	very high	not favourable

No 35	Emergency notification	Emergency events which are needed to be broadly announced to the public such terror, accident, earthquake, ...	low	medium	favourable
No 36	Relay from DVB-H to WiFi	-reception of DVB-h programs relayed by WiFi network - Seamless mobility from DVB-H to WiFi network for the same program	low	medium	favourable
No 37	High quality video transforming from Mobile to TV	The image/video on the mobile is sometimes unsuitable to be shown on the big screen due to the poor resolution and quality.	high	high	favourable
No 38	Content sharing in the community	Sharing your collection with those friends who have common interest.	low	high	not favourable (same as 26)
No 39	Emergency notification (same as n35)	Emergency events which are needed to be broadly announced to the public such terror, accident, earthquake, ...	low	medium	favourable
No 40	Indoor non-interruption	Seamless mobility to WiFi network when DVB-H signal is weak.	medium/high	high	favourable P2
No 41	Location dependent content	While the main program may still come from DVB-H, based on the location, what happened around could be also enjoyed such as more views on a match when you are at the stadium.	medium/high (positioning)	high	favourable P2

No 42	Location dependent application	At the airport, everything about your flight, in the shopping center, price comparison; in a coffee shop, selection of the programs best fit your disposal time and profile.	medium/high (positioning)	high	not favourable (out of scope)
No 43	Mobile TV to home computer	No conflict with your love ones provided that different programs can be shown simultaneously on different devices.	unknown	unknown	not favourable (out of scope)
No 44	Mobile TV to laptop at backyard	No hassle with messy cables and go wherever you want in your home	unknown	unknown	not favourable (out of scope)

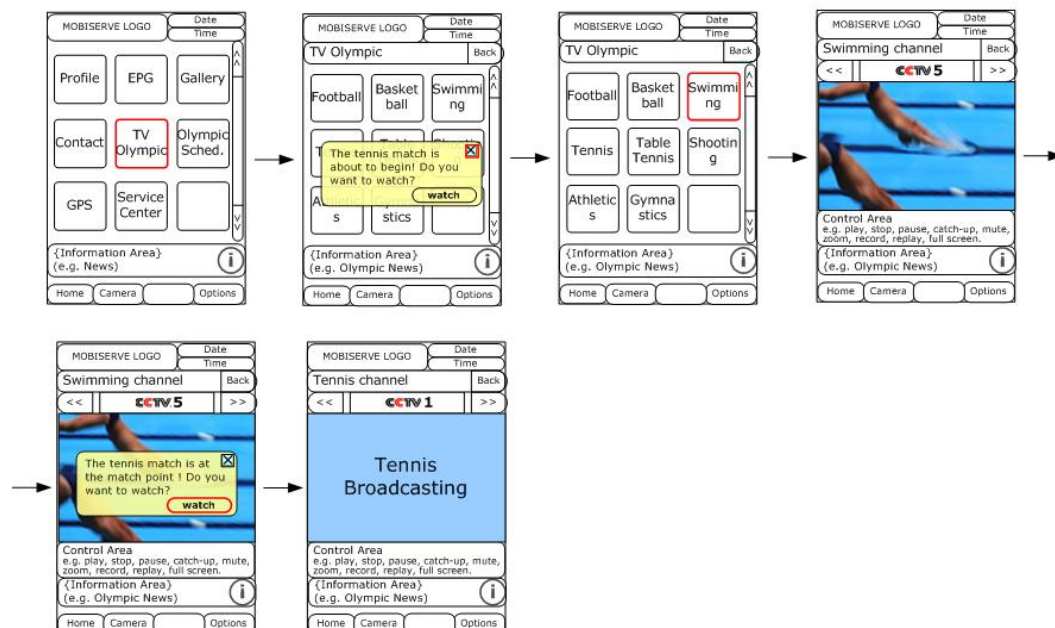
3.2 Favorable P1 use cases

3.2.1 Alerts from the events on the air (1)

Description: Alert on noticeable events such as an interesting tennis match might be reaching the end of match (match point) and offer the possibility to the user to switch directly to that channel. The alerts could be also grouped by interest. This kind of service could be subscribed.

Mobiserve – Alerts for coming events

Last Edited By Fran, Mar 28, 2007

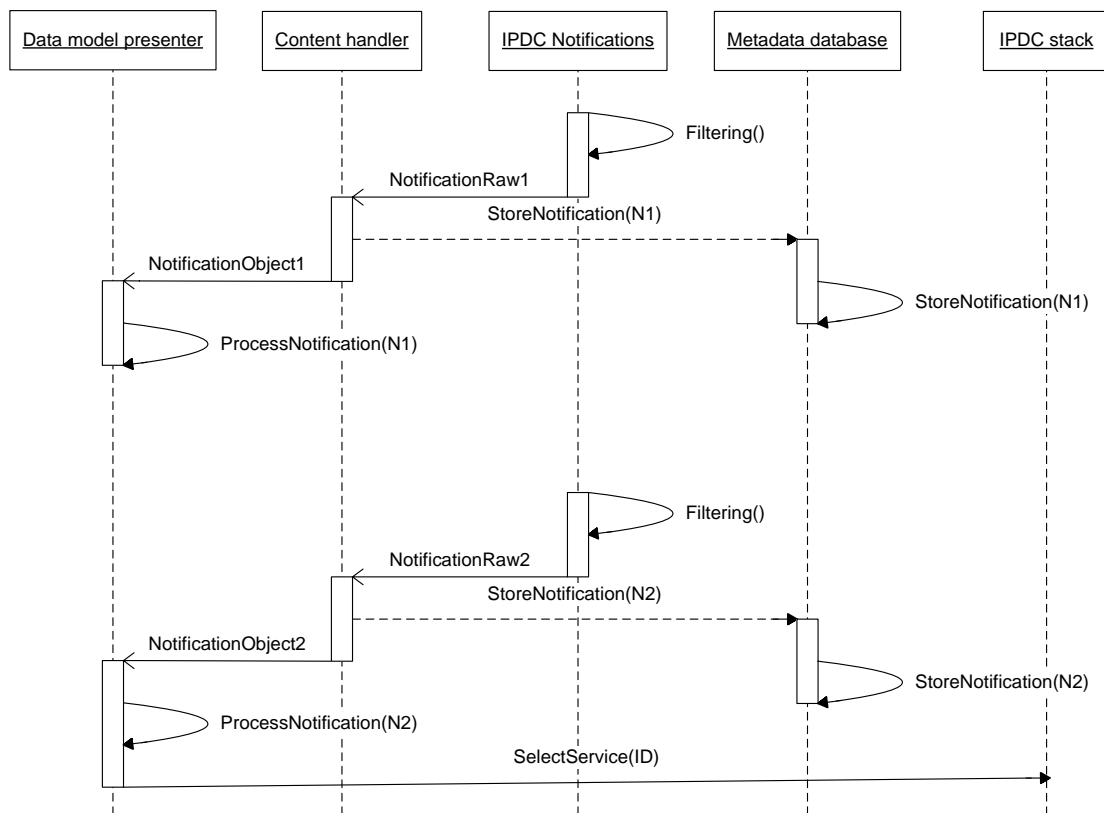


Preconditions: A mechanism for subscription to notification messages must exist. Also the option for selecting the notification filtering criteria should be supported.

Remarks: Content handling is not involved in the processing if the storage of notification messages is not applied (denoted in dashed lines).

The following sequence chart misses the next actions from the scenario: selection of TV Olympic option, selection of swimming channel, starting the tennis broadcast channel (these actions should be added by STZ).

The sequence chart describes the push mechanism initiated by the notification message.



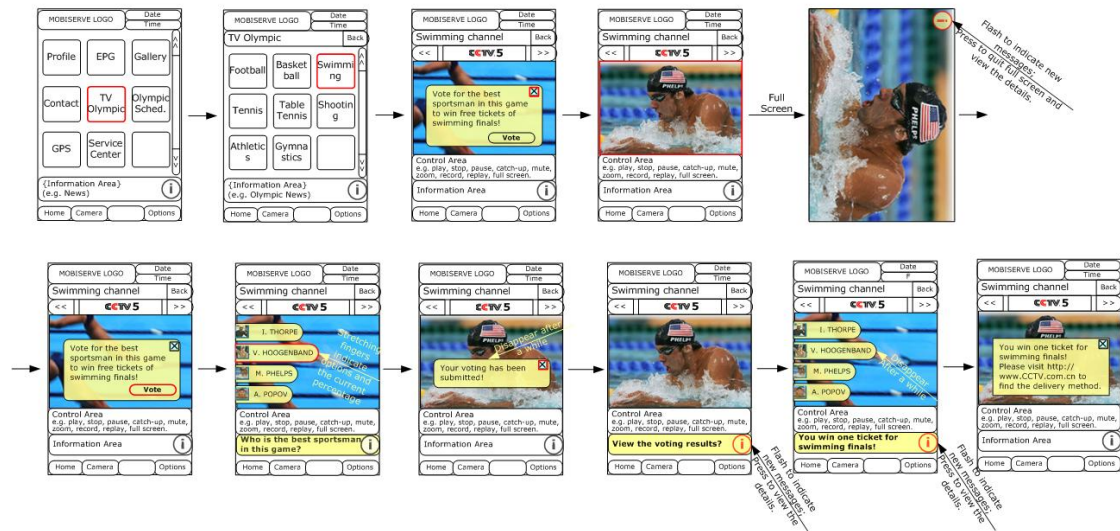
Q: What is the format of the notification message content: a) RME specific format, i.e., notification message as a transport protocol for RME formatted data; b) arbitrary XML format with the action code (watch); c) ...

3.2.2 Polling (3)

Description: Collection of the end users' feedback (e.g. voting) or play (e.g. betting).

Mobiserve - Polling

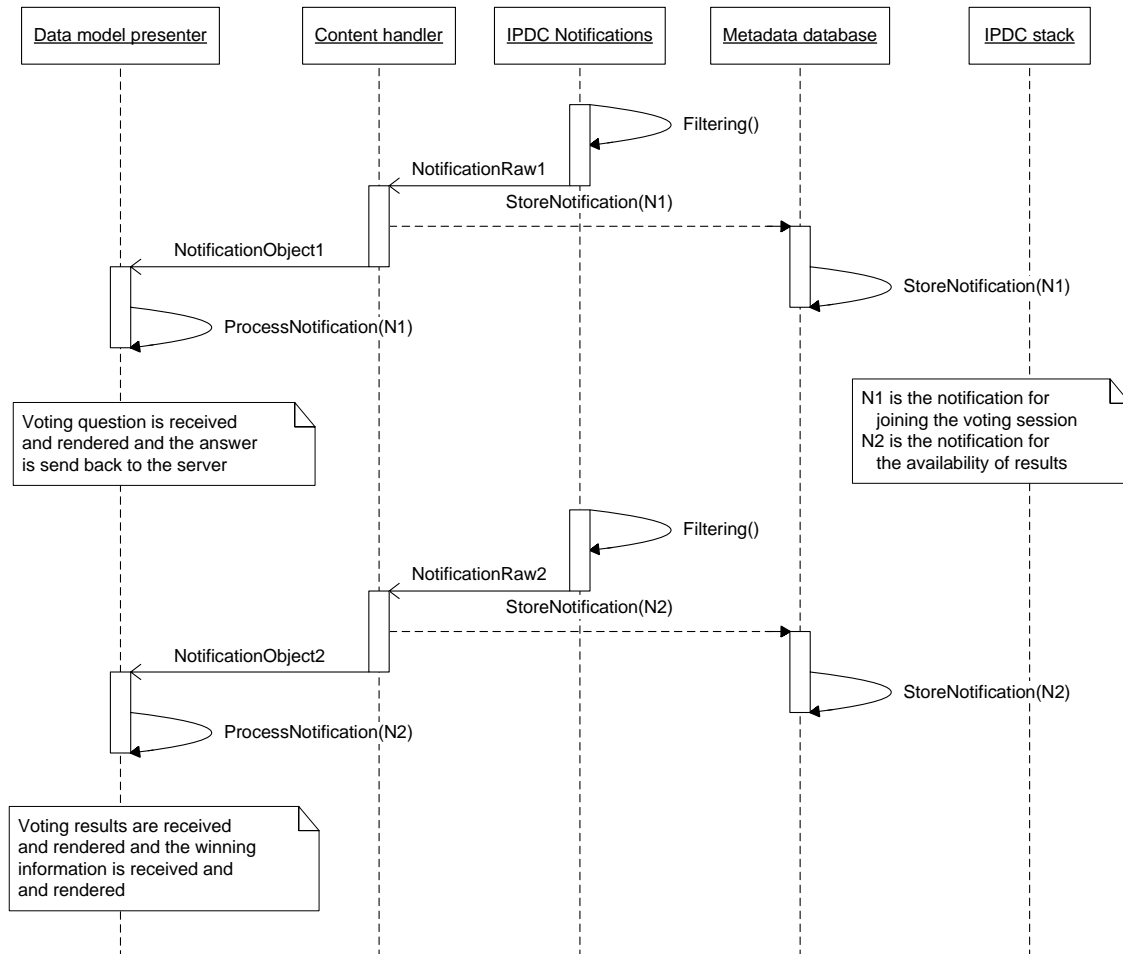
Last Edited By Fran, Mar 28, 2007



Preconditions: A mechanism for subscription to notification messages must exist. Also the option for selecting the notification filtering criteria should be supported.

Remarks: Content handling is not involved in the processing if the storage of notification messages is not applied (denoted in dashed lines).

The following sequence chart misses the next actions from the scenario: selection of TV Olympic option, selection of swimming channel, actions when rejecting the first notification message, full screen option, rendering the voting question, voting results, and winning information. These actions should be added by STZ, except for the rendering of the first notification message, which is a sub-part of the chart presented below.



Q: Two options are possible in obtaining the voting information (fingers with athletes): a) voting info is contained in the notification message (for format issues see previous Q); b) notification message points to a RME session used to render the voting info. The same question is valid for presenting the results.

Q: What is the backward channel for sending the vote and who is going to support this with the server side operation? What is the mechanism (forward channel) for sending the voting confirmation message and the information about winning the tickets?

3.2.3 Time-shifting - Smart pause (4)

Description: The ability to record the actual viewed program while switching to another activity such as receiving a phone call. As such, the user would not loose any piece of the viewed program.

Remarks: This use case is missing detailed description.

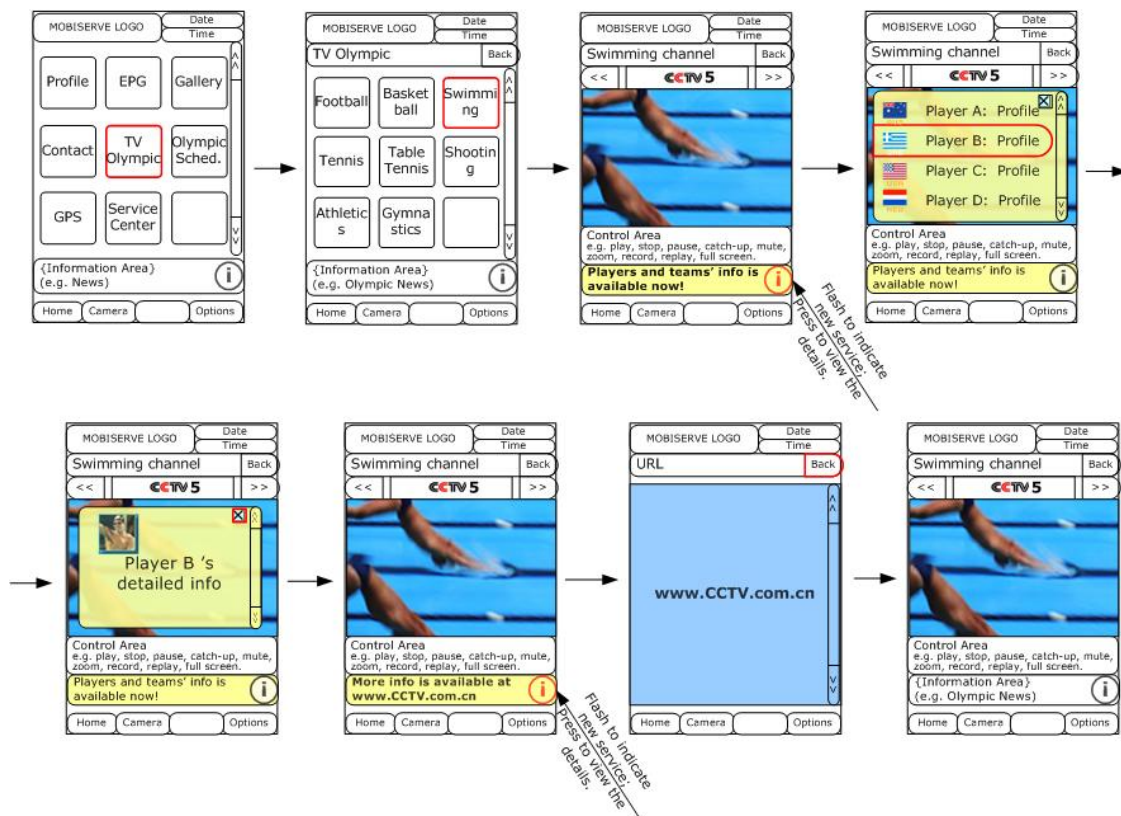
Q: Who will implement time shift buffer? What player is going to be used? Should this action involve content handling?

3.2.4 Program associated information (5)

Description: The TV program related information is broadcasted along with the associated TV program. For instance, the description about teams participating at the football match can be offered. Inside this information, there may be some links (URL) which allows user to get further information on a specific interested issue like a player biography.

Mobiserve – Program related information

Last Edited By Fran, Mar 28, 2007



The following sequence charts miss the next actions from the scenario: selection of TV Olympic option, selection of swimming channel, resolving the URL, and activation of the “Back” button (these actions should be added by STZ). Also the notification about the availability of more information about the event is not covered as it is a subpart of the diagram presented below.

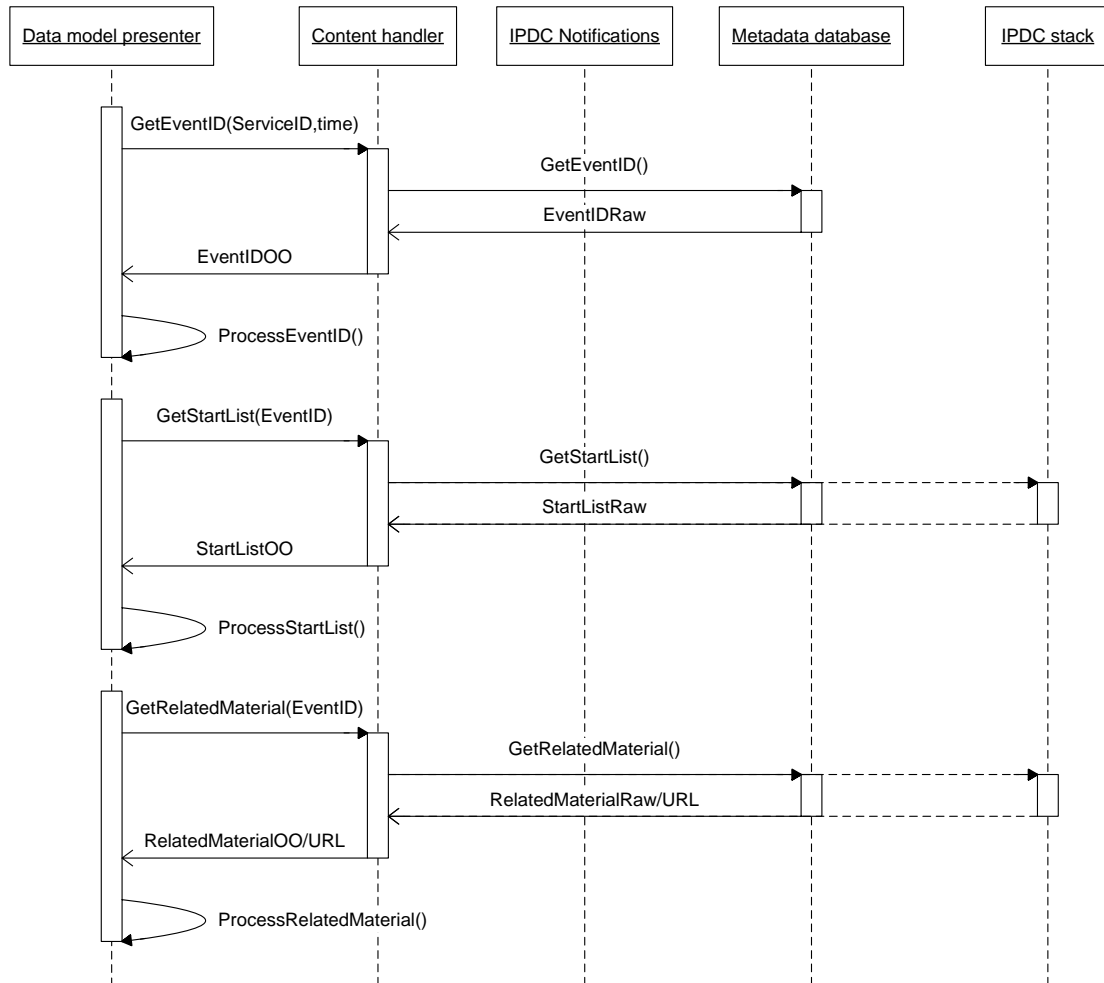
The sequence of actions for the swimming event can be triggered either by scanning the Olympic info within the metadata (pull mode) or by a notification message (push mode). In both cases additional info (start list and related material) can be retrieved

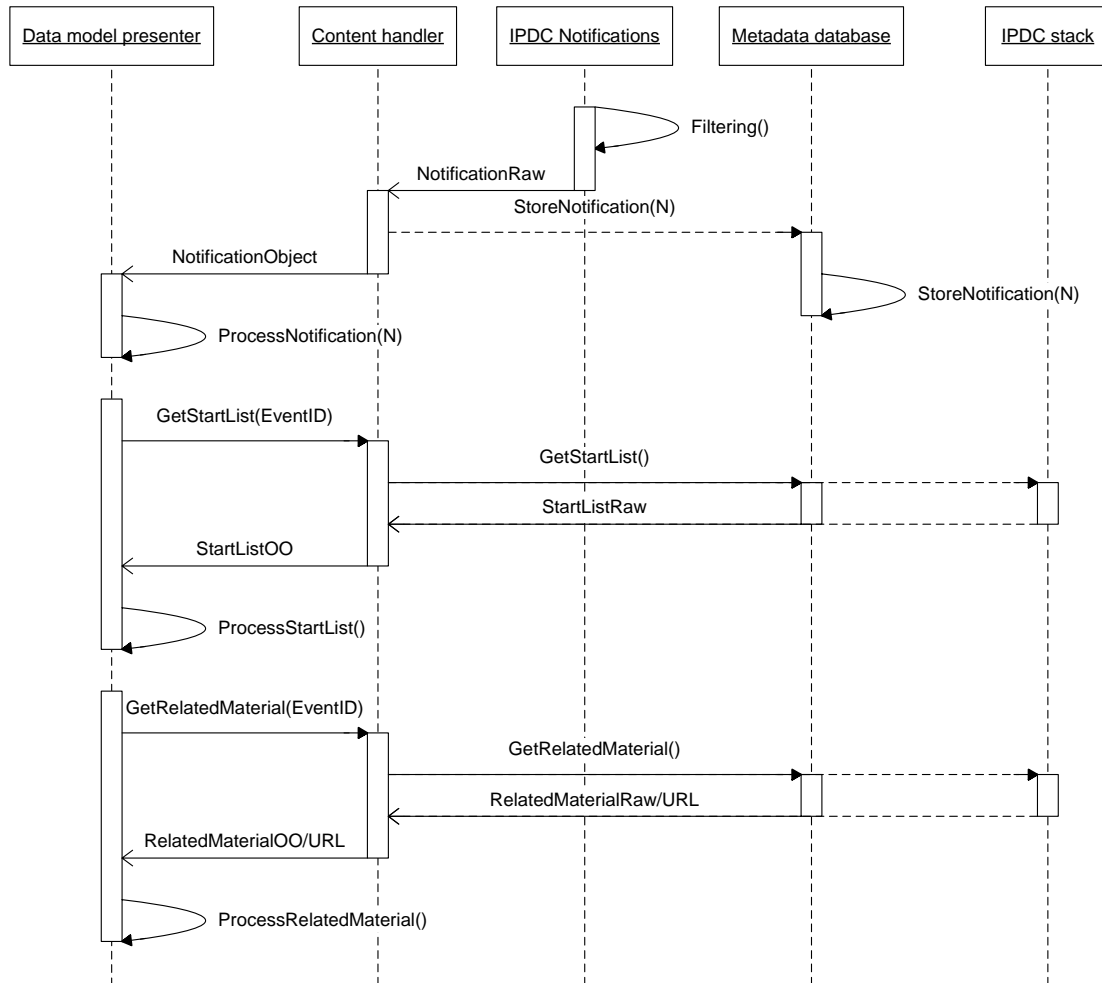
from different communication pipes: metadata database, IPDC stack, or bi-directional channel (URL).

For the pull case (first chart), the first action is resolving the current event (EventID resolving), while for the push case (second chart) the sequence of actions is triggered by the notification message).

Preconditions: (for the pull case:) Content handling should contain the indication that the metadata component contains the Olympics related data (schedule and athletes). The Olympics related data should be (optionally) pre-loaded into the metadata database using the metadata database model. Content handling should be able to query the metadata database over the Olympics related info and transform the answer into the data format understood by the data model presenter. (for the push case:) A mechanism for subscription to notification messages must exist. Also the option for selecting the notification filtering criteria should be supported.

Remarks: Content handling should be able to resolve the call message to the right communication pipe (depicted with dashed lines in the diagram).





Q: It is not clear how Olympic schedule is going to be linked to the ESG fragments, i.e., no global/unique IDs exist: a) ESG contains this information; b) special notification message with links; c) ... How is this going to be organized at the server/broadcaster side?

Q: Are websites, containing Olympics related info broadcasted? This is important as due to the number of subscribers per cell the usage of GPRS/UMTS may be a problem.

3.2.5 User feedback (11)

Description: User may want to express his/her opinion regarding the viewed programs or other suggestions.

Remarks: This use case is missing detailed description. It should involve STZ RME and ISAP feedback handling for sending the information via interactive channel.

Q: How is this action initiated?

Q: Besides ISAP part is there any additional role for content handling?

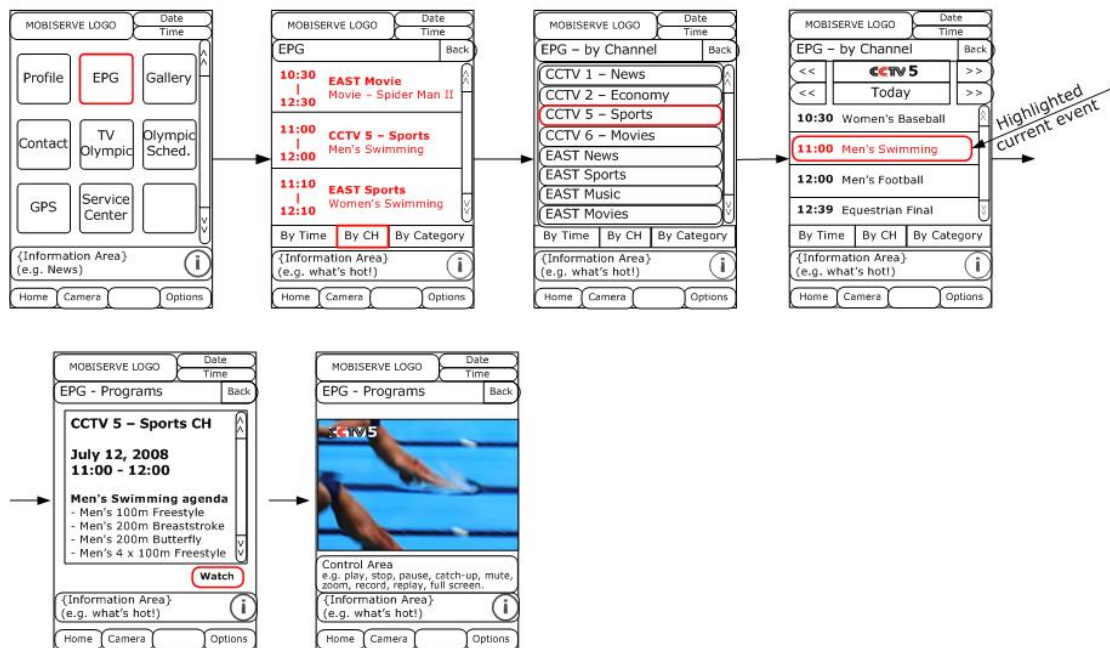
3.2.6 ESG (20)

Description: Electronic Service Guide

Remarks: These use cases will be resolved by the communication of STZ RME and ESG presenter with the IPDC stack.

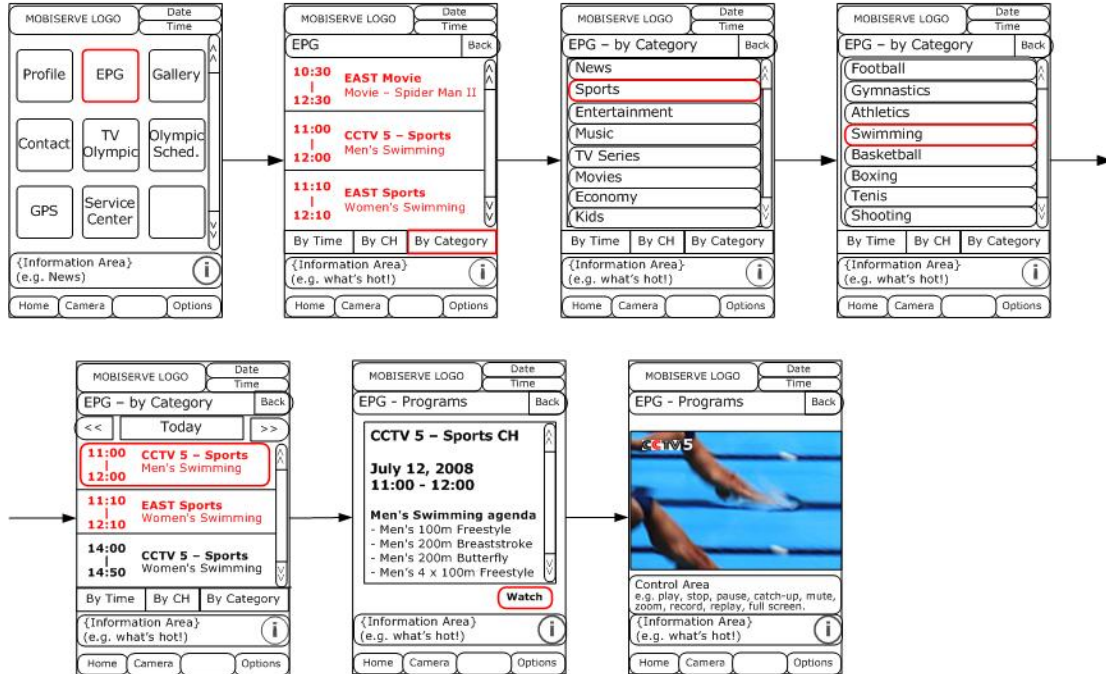
EPG – by channel

Last Edited By Fran, Apr 5, 2007



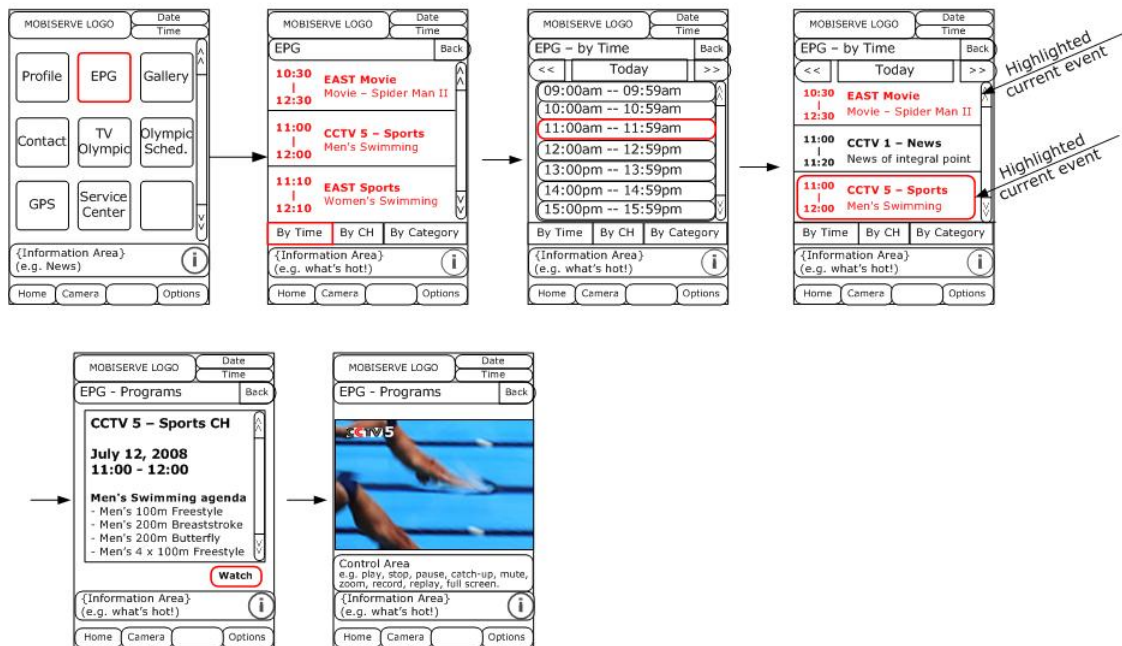
EPG – by category

Last Edited By Fran, Apr 5, 2007



EPG – by time

Last Edited By Fran, Apr 5, 2007



3.2.7 Rich Media enable ESG (21)

Description: ???

Remarks: This use case is missing the description. It will probably be resolved by STZ RME in communication with the IPDC stack (as the name suggests).

3.2.8 Rich Media function support (22)

Description: AV control, Media library, VoD.

Remarks: This use is missing the description. It will probably be resolved by STZ RME communication with the A/V component.

Q: Should it involve content handling in resolving the access to the media library or VoD server?

3.2.9 Local (re)play (31)

Description: This means the possibility offered by the terminal to slow down the motion and freeze. It can be applied to both the live TV program and recorded one.

Remarks: This use case is missing the description. It is similar to the use case 4 (Time shifting – smart pause).

Q: Who will implement slow down (replay) buffer? What player is going to be used? Should this action involve content handling (gallery)?

3.2.10 Usual Mobile phone functions while watching TV (32)

Description: Still being able to use the usual mobile phone while enjoying the TV program.

Remarks: This use case is missing detailed description. It will probably not require any action from the higher level components (implemented by NXP).

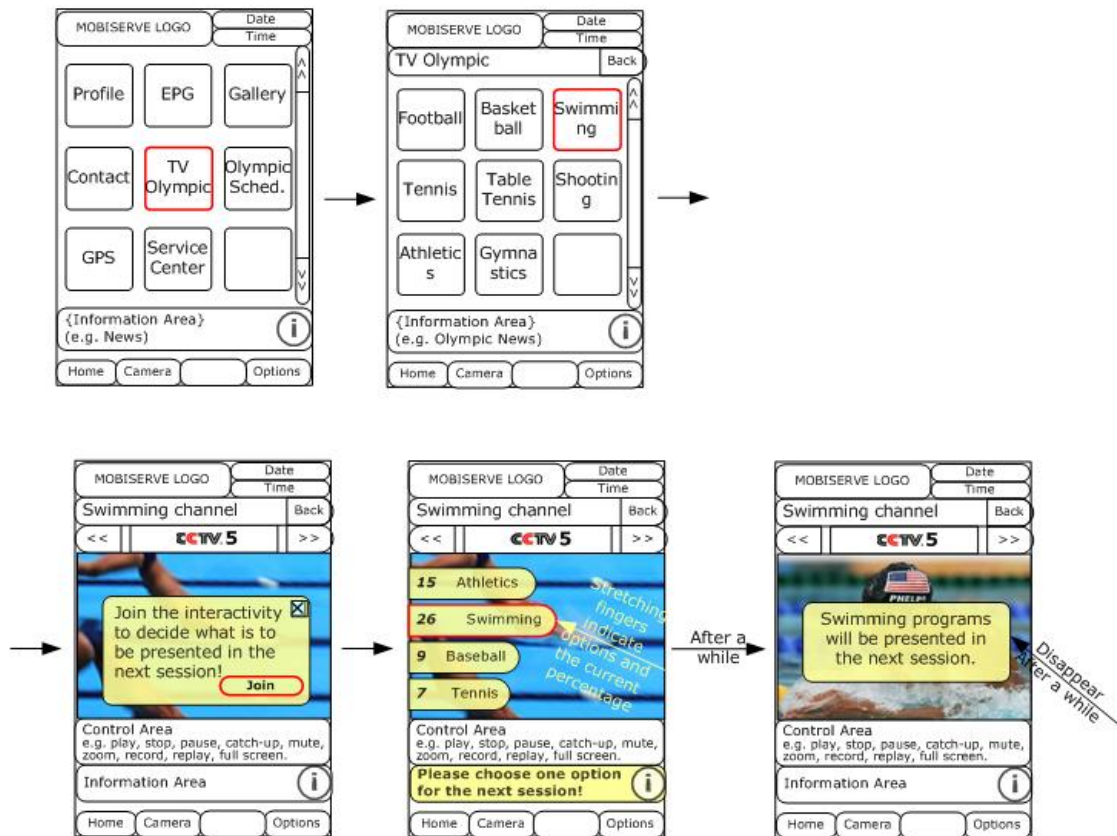
3.2.11 Interactivity with the TV programs (34)

Description: Being able to influence the broadcasted program's progression, whether pre-recorded or live.

Remarks: This use case is a special case of the use case 3 (Polling) It is either based only on STZ RME, or it can include the notification mechanism.

Mobiserve - Interactivity

Last Edited By Fran, Apr 9, 2007



3.2.12 Emergency notification (35)

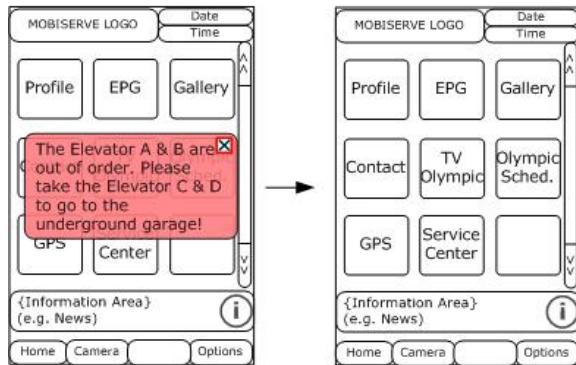
Description: Emergency events which are needed to be broadly announced to the public such as terror, accident, earthquake.

Preconditions: A mechanism for subscription to notification messages must exist.

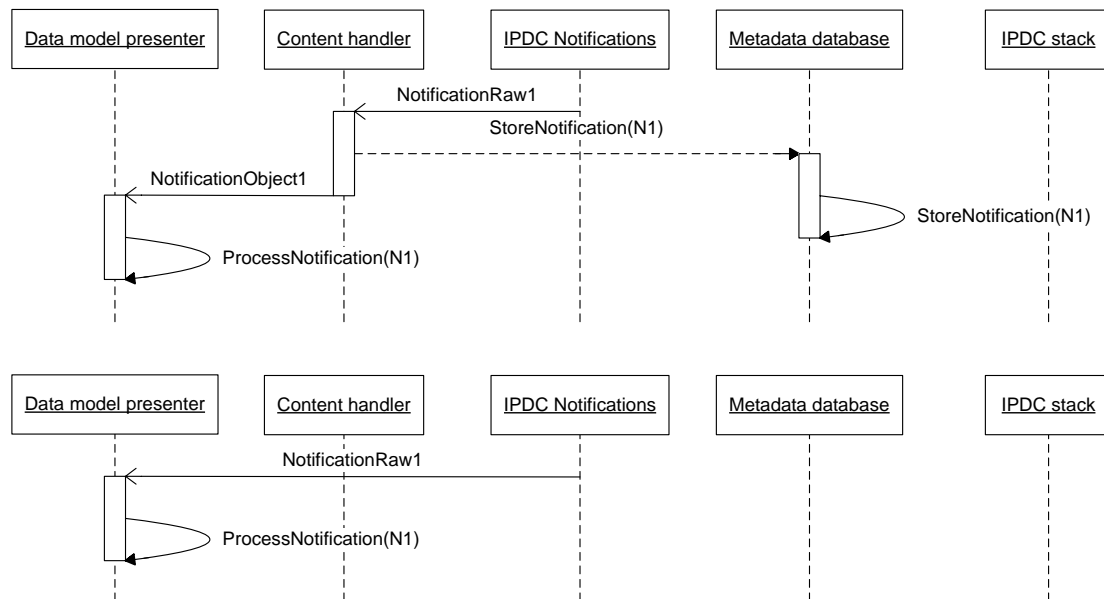
Remarks: Content handling is involved in the processing if the notification messages is stored for later display (terminal is in "stand by").

Mobiserve – Emergency notification

Last Edited By Fran, Apr 3, 2007



Q: Should content handling be involved or not?



3.2.13 Relay from DVB-H to WiFi (36)

Description: Reception of DVB-H programs relayed by WiFi network and seamless mobility from DVB-H to WiFi network for the same program.

Remarks: This use case is missing detailed description. It will probably be implemented by Thomson China.

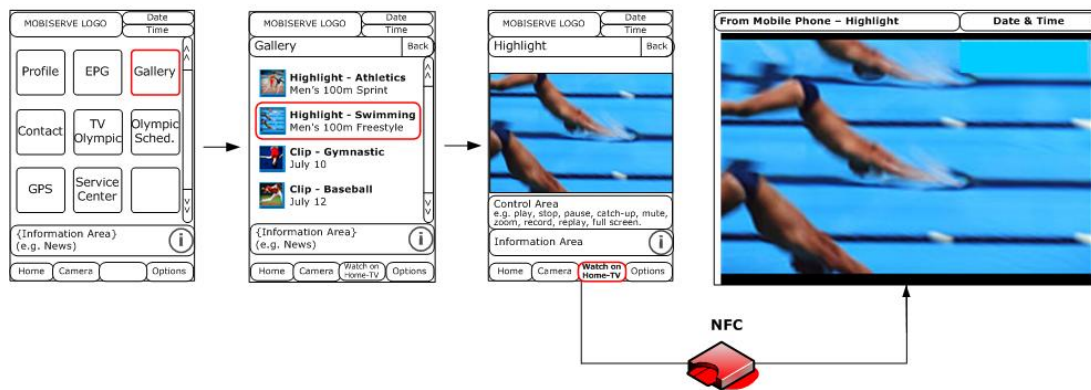
Q: Is this going to be in the trial or a separate demo?

3.1.14 High quality video transforming from Mobile to TV (37)

Description: The image/video on the mobile is sometimes unsuitable to be shown on the big screen due to the poor resolution and quality.

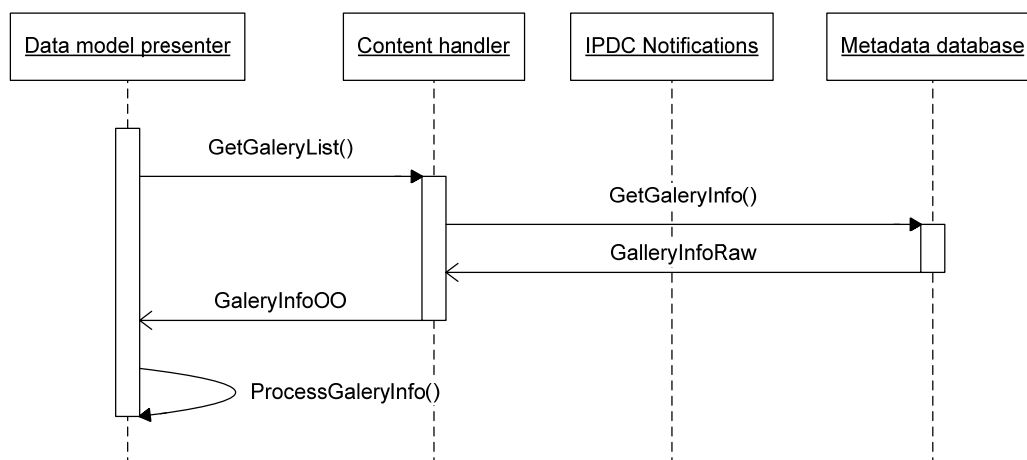
Mobiserve – Connect to TV set

Last Edited By Fran, Apr 6, 2007



Preconditions: A mechanism for entering the data necessary for the connection to the TV set and the transmission of the video.

Remarks: Content handling might not be needed if STZ RME is accessing directly the Gallery file structure, and handles the information for mobile to TV connection.



Q: Is this going to be in the trial or a separate demo?

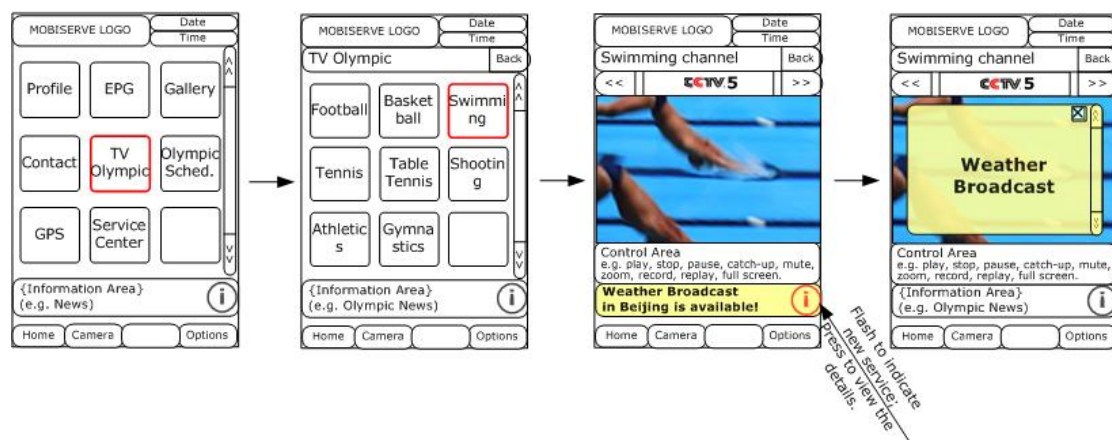
3.3 Favourable P2 and P3 use cases

3.3.1 Program independent data (6)

Description: Broadcast of the program independent data such as weather, traffic.

Mobiserve – Program independent information

Last Edited By Fran, Apr 10, 2007



Preconditions: A mechanism for subscription to notification messages must exist. Also the option for selecting the notification filtering criteria should be supported.

Remarks: This use case is a special case of the use case 3 (Polling) and include the notification announcing the availability of weather broadcast and the weather broadcast information (notification or RME format).

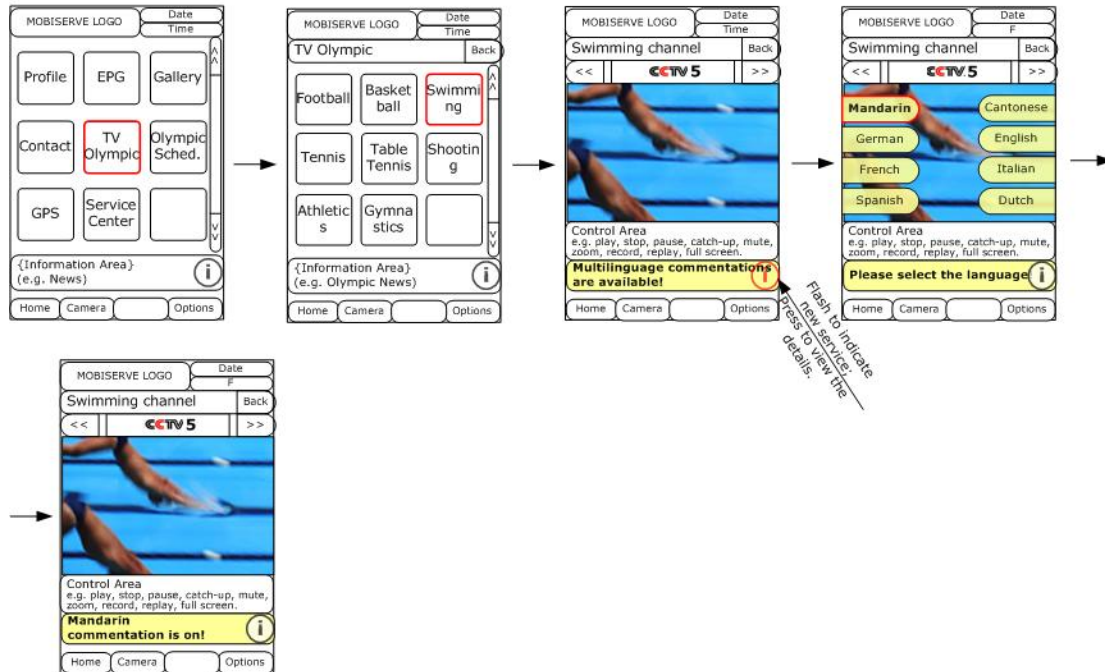
Q: Who is going to provide this service?

3.3.2 Multi-language (8)

Description: Multi-lingual for the TV program as well as the broadcasted information.

Mobiserve – Multilingual

Last Edited By Fran, Apr 2, 2007



Preconditions: A mechanism for subscription to notification messages must exist. Also the option for selecting the notification filtering criteria should be supported.

Remarks: This use case is a special case of the use case 3 (Polling) and include the notification announcing multilingual coverage and the selection of offered options that result in tuning to a different audio stream.

3.3.3 Selection of preferred commentators (9)

Description: While viewing a competition, one may have the possibility to choose the preferred commentators (sportscasters).

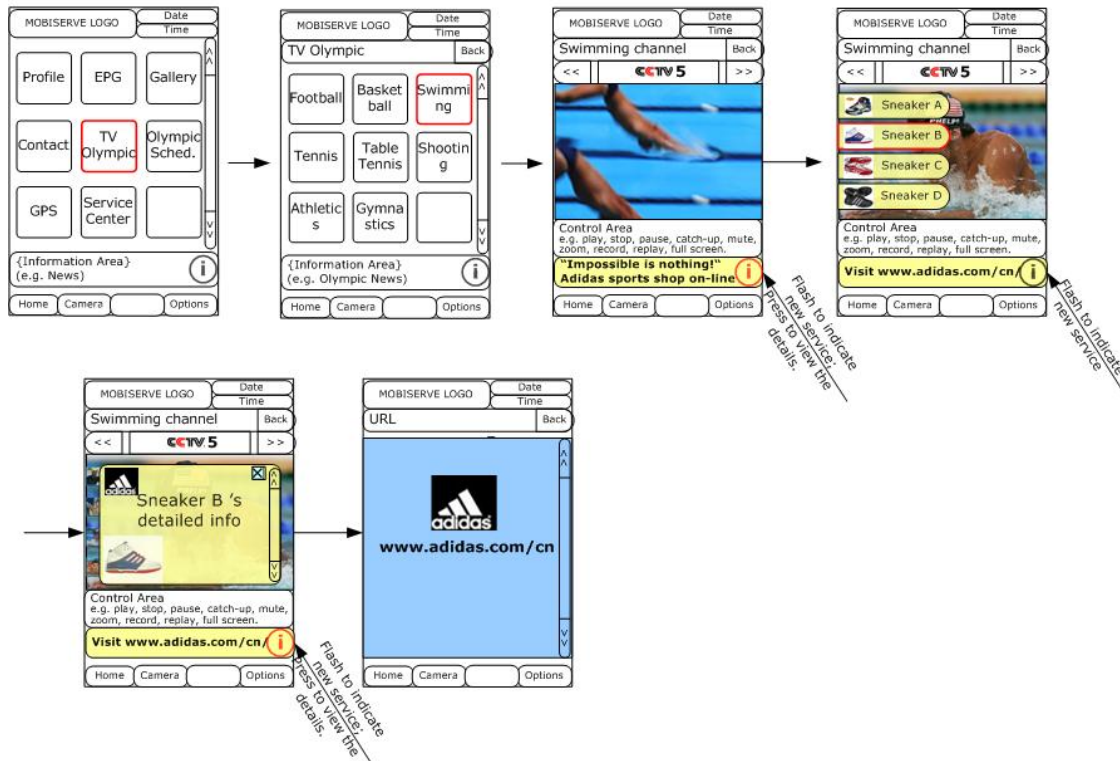
Remarks: This use case is missing detailed description. This use case looks similar to use case 8 (Multi-language), i.e., it is most probably a special case of the use case 3 (Polling).

3.3.4 Online shopping - associated with program (10)

Description: TV program triggered online shopping. That is, the goods are shown in the TV program, and the online shopping of those goods is suggested.

Mobiserve – Online ad.

Last Edited By Fran, Apr 5, 2007



Preconditions: A mechanism for subscription to notification messages must exist. Also the option for selecting the notification filtering criteria should be supported.

Remarks: This use case is a special case of the use case 3 (Polling) and include the notification for the Adidas add and additional info on the webpage, and presenting the offers and details about the products (Adidas sneakers).

3.3.5 Live program (14)

Description: Repurpose (retransmission) of the live TV program.

Remarks: This use case is missing detailed description. It will probably contain the announcement of the retransmission (notification) that can be followed.

3.3.6 Download - scheduled (15)

Description: Any kind of data could be scheduled to be downloaded via the broadcast channel.

Remarks: This use case is missing detailed description. It will probably include the content handling for tuning to the right channel (broadcast at the specific time) and for storing the data on the device (including metadata). This use case is similar to use case 33 (see below).

3.3.7 PVR (16)

Description: Based on the usage profile stored on both the mobile device and the home entertainment system, all relevant and potentially interesting services (Audio/Video and applications) can be stored in the PVR system.

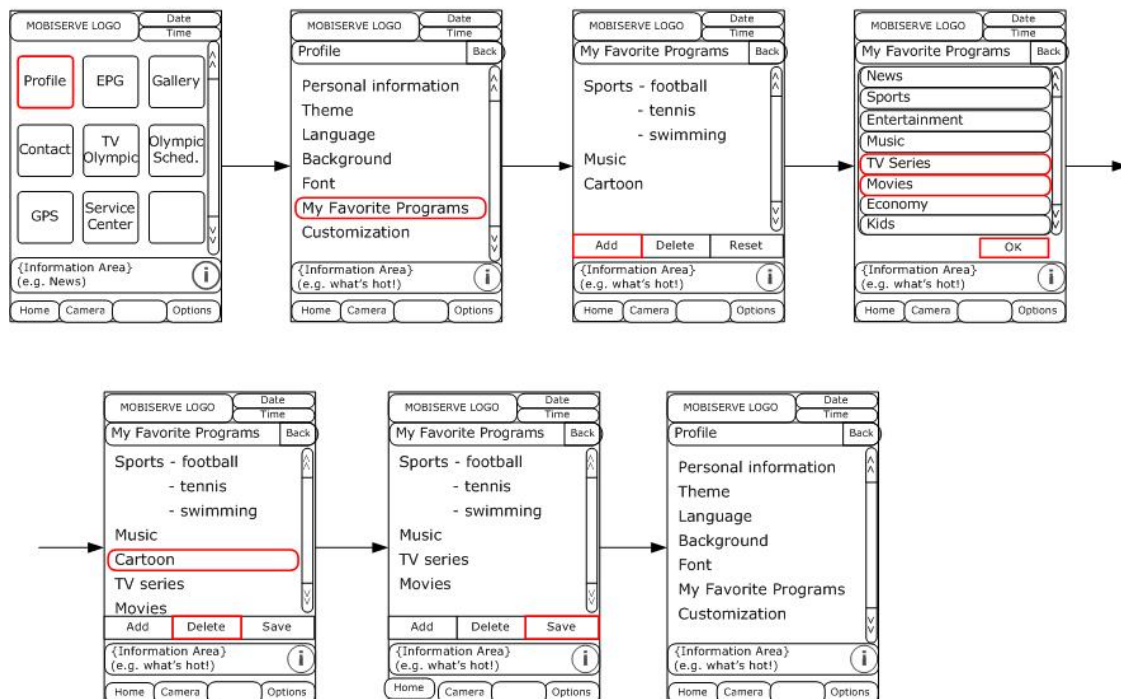
Remarks: This use case is missing detailed description. It will probably include the content handling. Content handling would need to contain the consistent set of data describing user profile that can initiate (in what way?) the PVR functionality.

3.3.8 User profile and preference - user initiated (17)

Description: User might be able to define his/her profile and preference.

Mobiserve – Set preferences

Last Edited By Fran, Apr 5, 2007



Preconditions: In case this is organized via the metadata component, content handling would need to provide the interface for querying the profile in the metadata component and forwarding it to the data model presenter. Data model presenter would

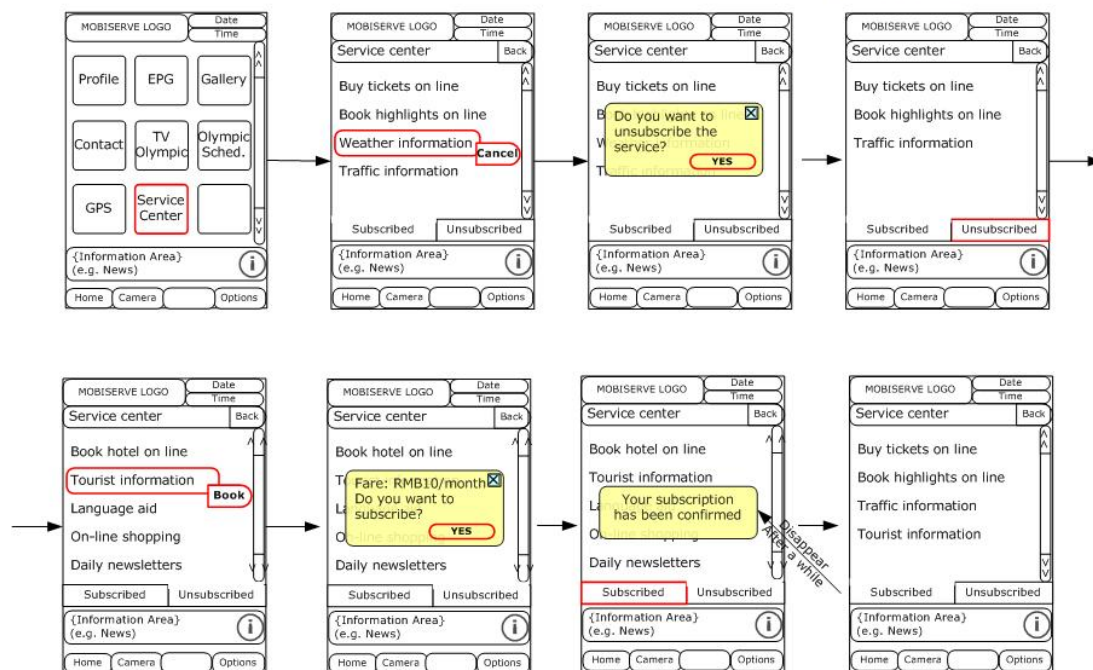
have to support presenting of this information. Additionally the metadata component would have to support the instantiation of the data model with user related information.

Remarks: Two options exist for the implementation of this use case. One is that RME contains a key-value pairs for the user preferences that can be set by the user. The other is that the data is stored within the metadata component and application and content handling have to support user preferences setup.

Service center use case would probably be supported via STZ RME as it involves interactivity. However, it also includes the subscription to notification messages and setting the notification filtering criteria. This will involve the content handling component.

Mobiserve – Service center

Last Edited By Fran, Apr 6, 2007



Q: What are the responsibilities of the content handling with respect to storing the user profile and the notification message setup is not clear. Also what part of these scenarios is supported by STZ engine?

3.3.9 Access to Internet through Rich Media interface (23)

Description: ??

Remarks: This use case is missing detailed description.

3.3.10 VOD (29)

Description: Video on Demand.

Remarks: This use case is missing detailed description.

3.3.11 Intelligent ESG presentation (30)

Description: Channels of interested to a customer can be grouped together at the end-user device.

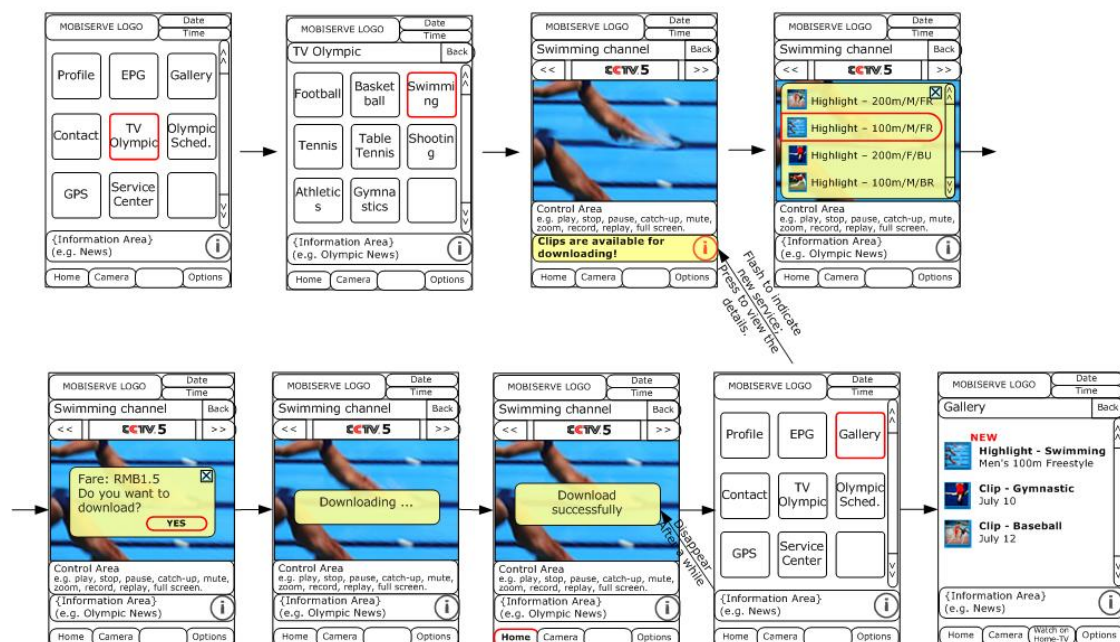
Remarks: This use case is missing detailed description.

3.3.12 Exciting clips available while watching the live TV program – P3 (33)

Description: For instance, during a basketball, the content provider cuts the interesting moments (clips), such as a nice shoot, along with the competition and notifies the user whenever these clips are available.

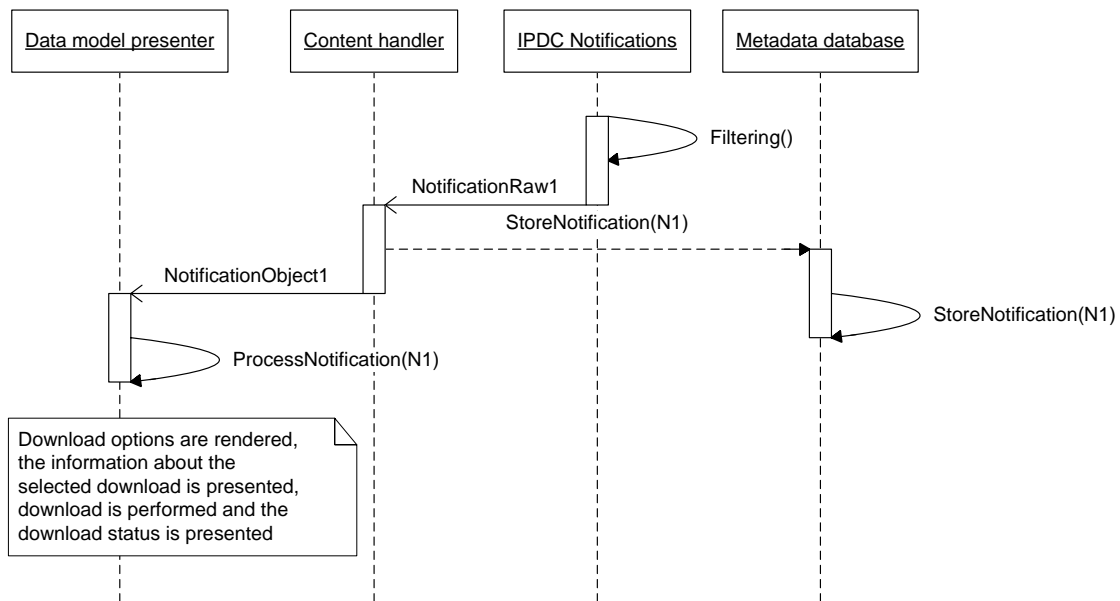
Mobiserve – Clip Download

Last Edited By Fran, Apr 10, 2007



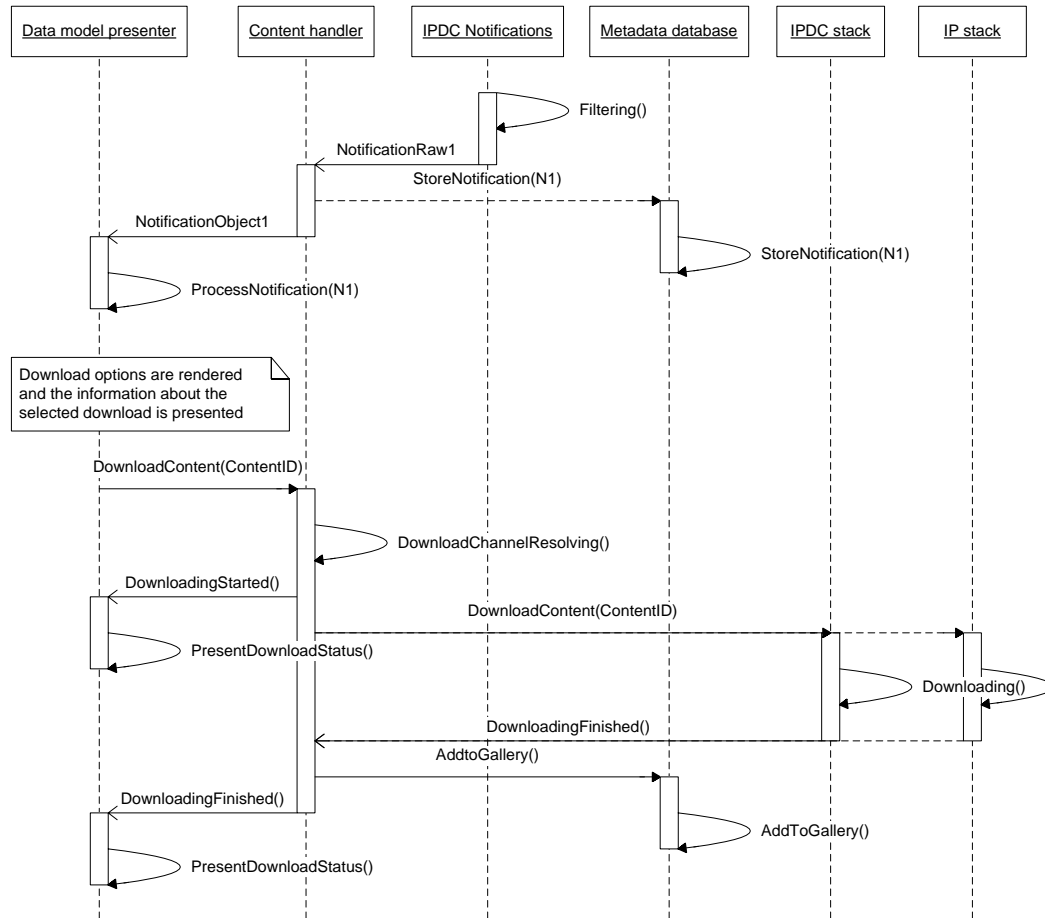
Preconditions: A mechanism for subscription to notification messages must exist. Also the option for selecting the notification filtering criteria should be supported. (for second chart:) Content handling should be able to resolve the downloading channel (broadcast, unicast) used for clip download.

Remarks: This is the variant of the use case 3 (Polling), the only difference is when the YES button is clicked. Then content handling has to resolve (based on the content of the download description) which channel to use for the download: a) to tune to the right DVB-H stream; b) to start HTTP request over GPRS/UMTS.



Q: How is this downloading supported at the server/broadcaster side?

Q: How is rendering of “Downloading ...” and “Download successful” messages organized: a) complete control from STZ RME; b) content handling sending messages to RME of the rendering status; c) ...



3.3.13 Indoor non-interruption (40)

Description: Seamless mobility to WiFi network when DVB-H signal is weak.

Remarks: This use case is missing detailed description. It resembles the use case 36.

3.3.14 Location dependent content (41)

Description: While the main program may still come from DVB-H, based on the location, what happened around could be also enjoyed such as more views on a match when you are at the stadium.

3.3.15

Remarks: This use case is missing detailed description. It will include the option to choose between different camera angles (achieved using RME) and tuning to video stream coming from different camera.

4 Component data models

4.1 Content handling data model

The content handling data model should be a simplification of the data models of components at the lower level: metadata data model, IPDC notification data model, sport event info data model, etc. The main difference is that it should focus only on the information that can be useful for the user. It should disregard all the information that is needed for the data flow at lower levels.

Notification

Notification data model should be a sublimation of the IPDC notification (metadata notification) data model:

- PayloadElements (type, encoding, dataFile, textMessage, ...)
- ReferencingElements (esgRef, objectRef, streamRef, dataFileRef, ...)
- RenderingElements (renderingTime, renderingPosition, ...)
- LifeCycle&TimingElements (timeStamp, lifetime, priority, serviceTime, ...)
- ...

Sport event info

Sport event data model should closely resemble schedule and athletes' data model from the metadata component:

- EventInfo (ID, startTime, endTime, venue, sport, referee, ...)
- Athletes (number, name, statistics, linkToAthleteInfo, ...)
- Results (finalScore, standings, teamStatistics, ...)
- AthleteInfo (name, gender, height, weight, birthData, nationality)
- PersonallInfo (nickname, birthplace, club, coach, ...)
- Results (event, date, place, result, description)
- Competition (discipline, sport, ...)
- AdditionalInfo (webLink, ...)
- ...

Personal preferences

This data model should closely resemble personal preferences data model in the metadata component (might not be needed):

- PersonallInformation(name, location, phone, ...)
- DeviceSetup (theme, language, background, font, ...)
- FavouritePrograms (programType, programs, ...)
- Gallerery (caption, location, description, mediaType, sharing, ...)
- Contacts (name, location, phone, ...)
- HomeConnection (...)
- ...

ESG

ESG data model should be a sublimation of the CBMS stack (S3 stack, metadata ESG, or DVB-H CBMS ESG) data model, containing only the information relevant to the user (probably not needed):

- Program (...)
- Acquisition (...)
- Service (...)
- ServiceBundle (...)
- ...

4.2 Metadata database data model

This section gives a brief overview of the data that will be kept in the metadata database component.

Notification

Notification data model should be a subpart of the Notifications IPDC data model (see Section 3.2). Therefore, it should be organized based on the following components:

- GenericElements (ID, sequenceNumber, encodingType, version, sessionId, ...)
- PayloadElements (type, encoding, dataFile, textMessage, ...)
- ReferencingElements (esgRef, objectRef, streamRef, dataFileRef, ...)
- RenderingElements (renderingTime, renderingPosition, ...)
- LifeCycle&TimingElements (timeStamp, lifetime, priority, serviceTime, ...)
- Update&StatusElements (type, time, statusType, statusInfo, ...)
- ...

Olympic schedule

Olympic schedule data model should be organized based on the following components:

- MainInfo (ID, startTime, endTime, venue, sport, referee, ...)
- Athletes (number, name, statistics, linkToAthleteInfo, ...)
- Results (finalScore, standings, teamStatistics, ...)
- ...

Athletes

Athlete's data model should be organized based on the following components:

- MainInfo (name, gender, height, weight, birthData, nationality)
- PersonalInfo (nickname, birthplace, club, coach, ...)
- Results (event, date, place, result, description)
- Competition (discipline, sport, ...)
- AdditionalInfo (webLink, ...)
- ...

Personal preferences (including profile, gallery, contacts and home connection)

This data model might not be necessary if the personal preferences are stored as the key-value pairs and directly accessed by the RME. In case this information is stored in the database, the personal preferences data model should be organized based on the following components:

- PersonalInformation(name, location, phone, ...)
- DeviceSetup (theme, language, background, font, ...)
- FavouritePrograms (programType, programs, ...)
- Gallerery (caption, location, description, mediaType, sharing, ...)
- Contacts (name, location, phone, ...)
- HomeConnection (...)
- ...

ESG

It is assumed that the RME will directly access the CMBS stack and therefore this data model is optional. In case it is used, ESG data model should be organized based on the DVB-H IPDC ESG components:

- EsgMain (...)
- Acquisition (...)
- Content (...)
- Purchase (...)
- PurchaseChannel (...)
- ScheduleEvent (...)
- Service (...)
- ServiceBundle (...)

4.3 IPDC Notifications data model

The IPDC notifications data model should closely follow the format of the DVB-H CBMS Notification message format. The format will be published in the near future, and the components presented below depict the types of data that will be part of the notification format:

- GenericElements (ID, sequenceNumber, encodingType, version, sessionId, ...)
- PayloadElements (type, encoding, dataFile, textMessage, ...)
- ReferencingElements (esgRef, objectRef, streamRef, dataFileRef, ...)
- RenderingElements (renderingTime, renderingPosition, ...)
- LifeCycleTimingElements (timeStamp, lifetime, priority, serviceTime, ...)
- Update&StatusElements (type, time, statusType, statusInfo, ...)
- ...

5 Component communication and API's

This section details the communication between components in the terminal architecture. It gives a first look into the function calls that can be expected when one component asks or forwards the information to the other component. These calls are based on the analysis of the MOBISERVE use cases and flowcharts given in Section 2.

5.1 Content handling interface

Content handling contains the functionality for the logical coordination between the data flow within the component, as well as calls for managing and querying data stored in databases at the lower levels in the architecture.

Data flow control:

- ScheduleNotificationDisplay(Notification*)
- resolveEventIDCall(EventInfo)
- resolveStartListCall(EventID)
- resolveRelatedMaterialCall(EventID)
- resolveGalleryListCall(GalleryID)
- resolveDownloadContentCall(ContentInfo*)
- ...

Data management:

- storeNotification(Notification*)
- deleteNotification(ID)
- storeInGallery(Data*)
- deleteInGalleryData(DataID)
- ...

Querying:

- acquireNotification(NotificationID)
- acquireEventIDfromCH(EventInfo)
- acquireEventIDfromIPDC(EventInfo)
- acquireEventIDfromWeb(EventInfo)
- acquireStartListFromCH(EventID)
- acquireStartListFromIPDC(EventID)
- acquireStartListFromWeb(EventID)
- acquireRelatedMaterialFromCH(EventID)
- acquireRelatedMaterialFromIPDC(EventID)
- acquireRelatedMaterialFromWeb(EventID)
- acquireContentFromCH(ContentID)
- acquireContentFromIPDC(ContentID)
- acquireContentFromWeb(ContentID)
- ...

5.2 Metadata database interface

This interface consists of GET and SET methods that either store data in the DBMS or query the data that is already in the DBMS.

Get methods:

- getNotification(NotificationID)
- getStartList(EventID)
- getRelatedMaterial(EventID)
- getGalleryInfo(GalleryID)
- ...

Set methods:

- setNotification(Notification*)
- setGalleryInfo(GalleryElement*)
- ...

5.3 Notification IPDC interface

Notification IPDC interface is focused on filtering and encoding the messages into the format that is readable by content handling:

- filterMessage(Notification*)
- encodeMessage(Notification*)
- sendNotification(Notification*)

5.3.1 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

5.3.2 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.

5.3.3 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.

5.3.4 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).

5.3.5 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.

5.3.6 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.

5.3.7 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.

5.3.8 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 form 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

5.3.9 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

5.3.10 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).

5.3.11 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.

5.3.12 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.

5.3.13 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.

6 APIs

6.1 ISAP API'S

6.1.1 Introduction

At the terminal side, ISAP provides 4 functional components in order to accomplish the ISAP related functionality. They are as follows:

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

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.

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.

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

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.).

Only the last component, namely ISAP interactive service management (ISM) component, will provide public APIs to interface the MOBISERVE terminal components other than the 4 ones mentioned above (those of ISAP).

6.1.2 ISAP APIs' categories

The ISAP APIs can be classified into two categories: active (ISAP_ISMA) and passive (ISAP_ISMP). One may easily understand that the Active ones are of the ISM component initiative and the Passive ones are of the ISAP related application initiative.

It is worth mentioning that the ISAP related applications are time sensitive. Since the timing is managed by ISAP components, ISM components will activate the Active APIs when it is time to do so. Consequently, when the ISAP related applications receive the call from the active APIs, they should proceed without delay.

The category of Active APIs is mainly made of two APIs. They are:

- ISAP_ISMA_ProcessFile
- ISAP_ISMA_ExpireFile

The API `ISAP_ISMA_ProcessFile` asks the ISAP related application to process promptly the indicated file.

The `ISAP_ISMA_ExpireFile` invalids the indicated file and asks the ISAP related application to promptly stop processing it.

The category of Passive APIs is mainly made of two APIs. They are:

- `ISAP_ISMP_ChannelSwitch`
- `ISAP_ISMP_Feedback`

The API `ISAP_ISMP_ChannelSwitch` informs the ISM component that the ISAP related application is changing the channel and asks it to send back the files related to the new channel to be processed.

The API `ISAP_ISMP_Feedback` informs the ISM component that the ISAP related application had made his feedback (on a previously ISM component sent feedback request file) and asks it to handle.

6.1.3 ISAP API's definition

`void ISAP_ISMA_ProcessFile` (int event_id, char* file_name, TYPE type, MimeType type, int channel)

Parameters:

int event_id: unique numeric id which will help to identify the event between ISAP and ISAP related applications.

Char* file_name: the data file to be processed.

TYPE type: defining the nature of the file, whether it consists of an alert, a public notification, a description, an emergency, a polling, etc. and the ISAP related application will process the file accordingly.

MimeType type: indicating the data file MIME type which will help the ISAP related application to arm the ad hoc file processor.

Int channel: indicating the event related channel.

`void ISAP_ISMA_ExpireFile` (int event_id, char* file_name, int channel)

Parameters:

int event_id: unique numeric id which will help to identify the event between ISAP and ISAP related applications.

Char* file_name: the data file to be processed.

Int channel: indicating the event related channel.

void **ISAP_ISMP_ChannelSwitch** (int old_channel, int new_channel)

Parameters:

Int old_channel: indicating the current channel that the ISAP related application is tuned on.

Int new_channel: indicating the new channel that the ISAP related application is tuning on.

void **ISAP_ISMP_Feedback** (int event_id, char* file_name, int channel)

Parameters:

int event_id: unique numeric id which will help to identify the event between ISAP and ISAP related applications.

Char* file_name: the data file containing the feedback.

Int channel: indicating the event related channel.

6.2 Rich media API overview

The *Streamezzo Rich Media Engine* works on graphical and audio scenes that are loaded through local files or remote servers. The Streamezzo RME reads the content at a specified URL and builds the graphical representation of the scene in an off-screen bitmap.

A custom application that uses the Streamezzo RME SDK creates the component with the desired size and issues requests for opening URLs. The custom application is responsible for responding to some of the component's events. The most important event requires the custom application to actually draw the off-screen buffer on the screen.

The Streamezzo RME control API is a two-ways API that relies on the use of two classes. The first one provides a set of control APIs to the main operations of the *Streamezzo Rich Media Engine*. The second one provides an observer framework to enable the application to react on the various browser events.

The two classes are:

- **CRMEngineInterface**: This class is directly used by the custom application to issue request to *Streamezzo Rich Media Engine*. This class must be instantiated once to access various calls detailed in §6.3
- **MRMEngineObserver**: This class must be overridden by the custom application (or one of its components) to receive events from the *Streamezzo Rich Media Engine* detailed in §6.4.

A Rich Media enabled application can also affect the behavior and the rendering of the *Rich Media Engine*. This API, the DOM API, is opened to get and set some attribute values among the LAsER scene graph. Using the DOM API is restricted to complex services for which the standard RME coverage is not enough. Using the DOM API also needs a tight synchronization between LAsER service development team and Rich Media enabled application development team.

6.3 Rich Media API Reference

6.3.1 Class CRMEngineInterface

CRMEngineInterface is the class that defines the control interface to the *Streamezzo Rich Media Engine*.

CRMEngineInterface is declared in the “RMEngineInterface.h” header file.

CRMEngineInterface exposes the API to:

- open URL thus enabling the reading of content,
- draw the off-screen bitmap on an on-screen window,
- pause and resume the playback of contents,
- adjust the volume of audio play-back, including mute audio volume,
- get some contents metrics,
- adjust some of the *Rich Media Engine* parameters,
- decode service specific parameters (static API),
- get and modify the attributes of the identified nodes in the currently played scene (aka DOM API).

CRMEngineInterface also exposes some operations to redirect keyboard and pointer events to *Streamezzo Rich Media Engine* to enable the actual user interactivity.

Most operations of *CRMEngineInterface* will start asynchronous processes on the *Streamezzo Rich Media Engine*. By implementing the *MRMEngineObserver* detailed in §6.4, the Rich Media enabled application will receive events, including redraw events and specific application layer data, from the *Streamezzo Rich Media Engine*, thus enabling the implementation of specific operations on some event occurrence.

6.3.2 Constructor & Destructor documentation

CRMEngineInterface()

CRMEngineInterface constructor.

The constructor of instances of *CRMEngineInterface* class. A call to the second phase constructor can be issued after this call.

~ *CRMEngineInterface* () [virtual]

CRMEngineInterface destructor.

The destructor of instances of *CRMEngineInterface* class.

BOOL construct(HDC hDC, HWND hWnd, SIZE sizeScreen, LPCWSTR lpszInstallDir, LPCWSTR lpszContentDir, LPCSTR lpszSystemLanguage, UINT nCacheSize)

CRMEngineInterface second phase constructor.

The second phase constructor of instances of *CRMEngineInterface* class. It performs the final construction of the object. It provides the *Streamezzo Rich Media Engine* all the necessary data to run. A call to this operation is necessary after a call to the constructor operation and before issuing any call to any other operation.

Parameters:

hCD: The graphic context that is necessary to build the off-screen bitmap.

hWnd : The main window handle that may be used to create children windows.

sizeScreen: The size of the *Streamezzo Rich Media Engine* graphical element. It defines the size of the off-screen bitmap and thus the size of the *Streamezzo RichMedia Engine* on the custom application.

lpszInstallDir: The installation directory for the custom application. The *Streamezzo Rich Media Engine* will use this directory for temporary and cached files. The provided string must be ended with a '\ ' character.

lpszContentDir: The content directory. It will be used by the *Streamezzo Rich Media Engine* to resolve local relative URIs (Ires). For most custom applications this directory is the installation directory or the directory where the home page is installed (top.stz). The provided string must be ended with a '\ ' character.

lpszSystemLanguage: The system language (ISO639-ISO3166. ex : en-US). An empty string can be provided.

nCacheSize: The RME cache size. Streamezzo recommends to use the default size of 500KBytes.

Return value:

A boolean that specifies if *Streamezzo Rich Media Engine* has been initialized.

void DestroyL ()

RMEngineInterface DestroyL operation.

The destroy operation of instances of *CRMEngineInterface* class. The operation cancels all pending requests to the *Streamezzo Rich Media Engine* and releases all allocated data. Only a call to the instance destructor can be safely issued after this call.

void setRMEngineObserver (MRMEngineObserver& oObserver)

RMEngineInterface setRMEngineObserver operation.

The setRMEngineObserver operation sets the observer of the *RMEngineInterface*, enabling the custom application to receive events from the Streamezzo Rich Media Engine. See header file for MRMEngineObserver class for details.

Parameters:

oObserver: an object that implements the *MRMEngineObserver* interface. This will usually be the custom application itself or one of its components. Note that an observer shall always be set.

void setContentDirectory(LPCWSTR lpszContentDir)

RMEngineInterface setContentDirectory operation.

The setContentDirectory operation updates the content directory for the instance of *RMEngineInterface*.

Parameters:

lpszContentDir: the new content directory. The restrictions and recommendations are the same as for ConstructL operation.

6.3.3 Control Functions documentation

Note that the audio volume is negotiated between the scene and the application. The scene may request volume changes: these changes must be validated by the Rich Media enabled application. The application can also choose to affect the audio volume.

TRMSceneType getCurrentSceneType ()

RMEngineInterface getCurrentSceneType operation.

The getCurrentSceneType operation makes the *Streamezzo Rich Media Engine* return the type of the currently played scene.

Return value:

A *RMSceneType* that specifies the current format. Note that the only available format using the RME SDK 4.19 is STZ_SCENE: LAsER CD.

void pause ()

RMEngineInterface pause operation.

The pause operation makes the *Streamazzo Rich Media Engine* pause in its current state. The graphical and audio rendering of the current content are paused. It issues a call to the resume operation to resume the processing of the current scene. When a new URL or a setPanel operation is called, the *Streamazzo Rich Media Engine* is automatically resumed.

void resume ()

RMEngineInterface resume operation.

The resume operation makes the *Streamazzo Rich Media Engine* resume from its current state. The graphical and audio rendering of the current content is resumed from its current state.

void adjustAudioVolume (int *nAudioVolume*)

The adjustAudioVolume operation.

The adjustAudioVolume operation makes the RME modify the audio volume level. The volume is confined between 1 and 10.

Parameters:

nAudioVolume: an integer between 1 and 10.

void setMute (BOOL *bMute*)

The setMute operation.

The setMute operation makes the *Streamazzo Rich Media Engine* produce or not the audio output. Setting the volume as mute does not affect the preset audio volume level, e.g. when the mute mode is unset, the previous audio level shall be set.

Parameters:

bMute: a boolean that sets or not the mute mode

void setOfflineMode (BOOL *bOffline*)

The setOfflineMode operation.

The setOfflineMode operation makes the *Streamazzo Rich Media Engine* refuse any network connection if it is called with false value. When offline mode is set and the scene requires a network connection, the *Streamazzo Rich Media Engine* will anticipate connection failure and notify the observer with the suitable code (see *MRMEngineObserver* class for details).

Parameters:

bOffline: a boolean that specifies if network connection is disallowed.

void getVersion (LPWSTR* *ppszVersion*) const

The getVersion operation.

The getVersion operation makes the *Streamazzo Rich Media Engine* return its version as a string literal. Caller is responsible for deleting this string (use free()).

Parameters:

ppszVersion: pointer to a string that will contain the RME version as a string literal

BOOL getApplicationName(LPCWSTR& *sApplicationName*)

The getApplicationName operation.

The getApplicationName seeks for the current application name and returns its content as a string pointer.

Parameters:

sApplicationName: pointer to a const string that will contains the application name as a string literal

Return value:

A boolean that specifies whether the application name is defined.

BOOL getApplicationVersion(LPCWSTR& sApplicationVersion)

The getApplicationVersion operation.

The getApplicationVersion seeks for the current application version and returns its content as a string pointer.

Parameters:

sApplicationVersion: pointer to a const string that will contains the application version as a string literal

Return value:

A boolean that specifies whether the application version is defined.

void setPanel (LPCWSTR lpszText)

The setPanel operation.

The setPanel operation makes the *Streamezzo Rich Media Engine* build a simple content that displays the provided text. The background is white, the font is small and black.

Parameters:

lpszText: the text to display in the simple content.

void openURL (LPCWSTR lpszURL, *TRMSceneType* nSceneType)

The openURL operation.

The openURL operation makes the *Streamezzo Rich Media Engine* resolve the provided URL, open and play the content at the provided location. A valid URL for the *Streamezzo Rich Media Engine* Interface is:

- http://\$web path\$: for remote content on web servers
- \$path/file name\$: for access to local files ex: c:\top.stz
- lres://\$file name\$: for access to local files in the content directory set in ConstructL operation ex: lres://top.stz

Parameters:

lpszURL The URL to open

nSceneType the scene type of the content to open. This an optional parameter, if not provided the *Streamezzo Rich Media Engine* tries to identify the scene type according to the file extension if any.

BOOL onKeyPressed(UINT nCode)

The onKeyPressed operation.

The onKeyPressed operation makes the *Streamezzo Rich Media Engine* handle the provided key event. The user application must send the events to the *Streamezzo Rich Media Engine* to have scene interactivity happen. The codes VK_F1 and VK_F2 must be sent to RME for emulating respectively left soft key and right soft key.

Parameters:

nCode: the character or Windows key code

Return value:

A boolean that specifies whether the key has been processed.

void onPointerEvent(UINT msg, DWORD lParam)

The onPointerEvent operation.

The onPointerEvent operation makes the *Streamezzo Rich Media Engine* handle the provided key event. The user application must send the events to the *Streamezzo Rich Media Engine* to have scene interactivity happen.

Parameters:

Msg: the windows message (WM_LBUTTONDOWN or WM_LBUTTONUP)

lParam: the data associated with the window message (see windows API)

void enableZoomAndPan(BOOL bEnable)

The enableZoomAndPan operation.

The enableZoomAndPan operation makes the *Streamezzo Rich Media Engine* handle user interaction for navigation/selection or zoomAndPan. This functionality is only available for SVG based scene.

Parameters:

bEnable: A boolean that specifies if ZoomAndPan is enabled or not.

void drawOffScreenBitmap(HDC hdc, const POINT& ptDest, const BOOL bOptimizedRedraw)

The drawOffScreenBitmap operation.

The drawOffScreenBitmap operation makes the *Streamezzo Rich Media Engine* draw its off-screen buffer in the provided HDC. This operation should be called when receiving a redraw event via the *MRMEngineObserver*. It can also be called when the custom application view that contains the *Streamezzo Rich Media Engine* display has been invalidated. The OptimizedRedraw indicator specifies if the optimized redraw based on invalidation regions can be used. This flag must be set to FALSE when the Rich Media window is redrawn because it has been invalidated by an external component, for example a menu bar or overlapping window. When using a TRUE OptimizedRedraw operation.

Parameters:

hDC: the device context where to draw the off-screen bitmap

ptDest: the top left point where to start drawing the off-screen bitmap

bOptimizedRedraw: a boolean that specifies if the optimized drawing can be used

void drawOffScreenBitmap(BYTE *oScreen, const POINT& ptDest, const BOOL bOptimizedRedraw, int nXPitch, int nYPitch, int nHeight)

The drawOffScreenBitmap second form operation.

The drawOffScreenBitmap second form operation behaves like the first one except that it writes the graphic data directly in the provided memory segment. This operation might be called when using direct screen technologies like Game API's.

Parameters:

oScreen: the raw buffer where to write

ptDest: the top left point where to start drawing the off-screen bitmap

bOptimizedRedraw: a boolean that specifies if the optimized drawing can be used

nXPitch: the horizontal pitch for destination memory

nYPitch: the vertical pitch for destination memory

nHeight: the height of destination memory

int getTenTimesFramerate()

The getTenTimesFramerate operation.

The getTenTimesFramerate operation makes the *Streamezzo Rich Media Engine* compute and return the graphical frame rate as ten times its value (to avoid floating point operations). A -1 returned value means the frame rate can't be computed because there is not enough data to compute any average.

Return value:

Ten times the graphical frame rate as an integer.

int getSceneTime()

The getSceneTime operation.

The getSceneTime operation makes the *Streamezzo Rich Media Engine* compute and return the time for the current content. The time is measured in milliseconds. A -1 value means that no content is currently being played.

Return value:

The scene time in milliseconds or -1 if scene time cannot be computed.

6.3.4 DOM API operations documentation

void setFontProperty(const BYTE nSTZFontStyle, const WORD nSizeSpec, const BYTE nStyleSpec, LPCWSTR sFileName, LPCWSTR sFontName)

The setFontProperty operation.

The setFontProperty operation enables the overriding of default fonts identified by LAsER CD values.

Parameters:

nSTZFontStyle: the LAsER CD font specification

nSizeSpec: the size to apply for the new specification

nStyleSpec: the LAsER CD style to apply for the new specification

sFileName: the name of the file that contains the font

sFontName: the font name to apply for the specification

BOOL decodeCustomCommand(LPCWSTR sCmd, LPWSTR& sCommandType, UINT& nParamsNb, LPWSTR *&oParams, LPWSTR *&oValues)

The decodeCustomCommand operation (static operation).

The decodeCustomCommand operation shall be used when processing custom application commands. The decodeCustomCommand operation extracts the following information from the given custom command: command type, parameters and values; all returned strings and strings arrays must be freed.

Parameters:

sCmd: the custom application command to decode

sCommandType: the command type (e.g. the path of the URI)

nParamsNb: the number of parameters (and of values)

oParams: an array that contains the command parameters

oValues: an array that contains the command values

Return value:

A boolean value that specifies if the command has been successfully parsed.

BOOL getNodeActivity(LPCWSTR sDef, BOOL& bActivity)

The getNodeActivity operation.

The getNodeActivity seeks for a given node in the graph and returns a boolean value corresponding to its activity.

Parameters:

sDef: the node identifier

bActivity: the activity of the node

Return value:

A boolean value that tells whether the targeted node has been found, and thus if the returned activity boolean value is valid.

BOOL setNodeActivity (LPCWSTR sDef,const BOOL bActivity)

The setNodeActivity operation.

The setNodeActivity seeks for a given node in the graph and sets the activity to the provided activity value.

Parameters:

sDef: the node identifier

bActivity: the activity of the node

Return value:

A boolean value that tells whether the targeted node has been found, and thus if the provided value has been set.

BOOL startNode(LPCWSTR sDef)

The startNode operation.

The startNode seeks for a given node in the scene graph and sets the start time to now. This functionality is only available for LASeR CD scene.

Parameters:

sDef: the node identifier

Return value:

A boolean value that tells whether the targeted node has been found, and thus if it has been started.

BOOL getTextContent (LPCWSTR sDef, LPTSTR& sContent)

The getTextContent operation.

The getTextContent seeks for a given Text node in the scene graph and returns its content string pointer.

Parameters:

sDef: the Text node identifier

sContent: the Text node content

Return value:

A boolean value that tells whether the targeted node has been found, and thus if the returned text content is valid.

BOOL setTextContent(LPCWSTR sDef, LPCTSTR sContent)

The setTextContent operation.

The setTextContent seeks for a given Text node in the scene graph and fills the string with a given content if found.

Parameters:

sDef: the Text node identifier

sContent: the Text node content

Return value:

A boolean that tells whether the targeted node has been found, and thus if the provided text content has been set

BOOL getTextColor(LPCWSTR sDef, RGBTRIPLE& color)

The getTextColor operation.

The getTextColor seeks for a given Text node in the scene graph and returns the color of a text node.

Parameters:

sDef: the Text node identifier

sColor: the Text node color

Return value:

A boolean value that tells whether the targeted node has been found, and thus if the returned color is valid.

BOOL startNodeAfter(LPCWSTR sDef, int nStartDelay)

The startNodeAfter operation.

The startNodeAfter seeks for a given node in the scene graph and sets its start time accordingly with the specified delay. The delay is specified in milliseconds. This functionality is only available for LAsER CD scene.

Parameters:

sDef: the node identifier

nStartDelay: the delay in milliseconds

Return value:

A boolean value that tells whether the targeted node has been found, and thus if its start time has been set.

BOOL getTextContent (LPCWSTR sDef, LPTSTR& sContent)

The getTextContent operation.

The getTextContent seeks for a given Text node in the scene graph and returns its content string pointer.

Parameters:

sDef: the Text node identifier

sContent: the Text node content

Return value:

A boolean value that tells whether the targeted node has been found, and thus if the returned text content is valid.

BOOL setTextContent(LPCWSTR sDef, LPCTSTR sContent)

The setTextContent operation.

The setTextContent seeks for a given Text node in the scene graph and fills the string with a given content if found.

Parameters:

sDef: the Text node identifier

sContent: the Text node content

Return value:

A boolean that tells whether the targeted node has been found, and thus if the provided text content has been set

BOOL getTextColor(LPCWSTR sDef, RGBTRIPLE& color)

The getTextColor operation.

The getTextColor seeks for a given Text node in the scene graph and returns the color of a text node.

Parameters:

sDef: the Text node identifier

sColor: the Text node color

Return value:

A boolean value that tells whether the targeted node has been found, and thus if the returned color is valid.

BOOL setTextColor(LPCWSTR sDef, const RGBTRIPLE& color)

The setTextColor operation.

The setTextColor seeks for a given Text node in the scene graph and sets its color if found.

Parameters:

sDef: the Text node identifier

color: the Text node color

Return value:

A boolean value that tells whether the targeted node has been found, and thus if the provided color has been set.

BOOL getTextFont(LPCWSTR sDef, unsigned char& font)

The getTextFont operation.

The getTextFont seeks for a given Text node in the scene graph and returns the font it uses.

Parameters:

sDef: the Text node identifier

font: the used font

Return value:

A boolean value that tells whether the targeted node has been found, and thus if the returned font definition is valid.

BOOL setTextFont(LPCWSTR sDef, const unsigned char nFont)

The setTextFont operation.

The setTextFont seeks for a given Text node in the scene graph and sets its font.

Parameters:

sDef: the Text node identifier

nFont: the used font

Return value:

A boolean value that tells whether the targeted node has been found, and thus if the provided font definition has been set.

Note: Both getTextFont and setTextFont use the following bitmasks to define the fonts used by the *Rich Media Engine*. This definition corresponds to the Streamezzo format font specification. The font bitmask definition is:

```
// the face of the font on the 2 first bits
#define FS_SYSTEM          0x00
#define FS_PROPORTIONAL  0x01
#define FS_MONOSPACE     0x02

// the style of the font on the 2 next bits
#define FS_PLAIN          0x00
#define FS_BOLD           0x04
#define FS_ITALIC        0x08

// the size of the font on the 2 next bits
```

```
#define FS_NORMAL          0x00  
#define FS_SMALL          0x10  
#define FS_LARGE          0x20
```

BOOL getShapeColor(LPCWSTR sDef, RGBTRIPLE& oColor, unsigned char &nTransparency)

The getShapeColor operation.

The getShapeColor seeks for a given Shape node in the scene graph and returns the color it uses.

Parameters:

sDef: the Shape node identifier

oColor: the used color

nTransparency: the used transparency

Return value:

A boolean value that tells whether the targeted node has been found, and thus if the returned color definition is valid.

BOOL setShapeColor(LPCWSTR sDef, const RGBTRIPLE& oColor, const unsigned char nTransparency)

The setShapeColor operation.

The setShapeColor seeks for a given Shape node in the scene graph and sets the color used by a shape node.

Parameters:

sDef: the Shape node identifier

oColor: the used color

nTransparency: the used transparency

Return value:

A boolean value that tells whether the targeted node has been found, and thus if the provided color has been set.

BOOL setAudioMedia(LPCWSTR sDef, LPCWSTR sFileName)

The setAudioMedia operation.

The setAudioMedia set the audio media that corresponds to an Audio node. The operation loads a local .amr file from the device and creates a media stream that is linked with the provided identified audio node. Operation is limited to AMR files.

Parameters:

sDef: the Audio node identifier

sFilename: the complete path to the .amr file

Return value:

A boolean value that tells whether the targeted node has been found, and thus if the provided .amr clip has been inserted in the scene.

BOOL setBitmapMedia(LPCWSTR sDef,LPCWSTR sFileName)

The setBitmapMedia operation.

The setBitmapMedia set the bitmap media that corresponds to a Bitmap node. The operation loads a local bitmap file from the device and creates a media stream that is linked with the provided identified bitmap node. Only available for LAsER CD scenes.

Parameters:

sDef: the Bitmap node identifier

sFileName: the complete path to the bitmap file

Return value:

A boolean value that tells whether the targeted node has been found, and thus if the provided bitmap has been inserted in the scene.

BOOL setAVMediaURN(LPCWSTR sDef, LPCWSTR sURN)

The setAVMediaURN operation.

The setAVMediaURN operation sets the URN of the media relative to the provided Audio or Video Node. The URN can be an absolute path to a local file or a RTSP URN. When the Audio/Video is allready playing a media, this media reference is removed with the new one.

Parameters:

sDef: the audio or video def node name

sURN: the URN to the media.

Return value:

A boolean value that tells whether the targeted node has been found, and thus if the provided URN has been set as reference of the provided audio or video node

BOOL setAnimateParamsCallBack(LPCWSTR sDef, Win32AnimateParamsCallBack oCallBack, void *pCustomInfo)

The setAnimateParamsCallBack operation.

The setAnimateParamsCallBack operation sets a callback function to an Animate node. This callback function will be then called at the node composition. This function allows customizing the time position of the designated Animate node (set the callback function at 0 to disable the customization).

Parameters:

sDef: the Animate node identifier

oCallBack: the callback to set

pCustomInfo: the complementary information for callback context

Return value:

A boolean value that specifies if the target node has been found, and thus if the custom animate callback has been installed.

6.4 Class MRMEngineObserver

MRMEngineObserver is the class that must be overridden in order to receive callback events from the *Streamezzo Rich Media Engine*.

onRMEngineEvent is called with parameters that describe the event that the *Streamezzo Rich Media Engine* is reporting. The supplementary information field contains data or a pointer to data depending on the reported event.

MRMEngineObserver is declared in the "RMEngineObserver.h" header file.

6.4.1 Member Function documentation

void OnRMEngineEvent (TRMEvent oEvent, const void *oInfo) [pure virtual]

OnStzPlayerEvent operation.

The onRMEngineEvent operation must be implemented by MRMEngineObserver sub classes to handle events from the *Streamezzo Rich Media Engine*. The *Streamezzo Rich Media Engine* is reporting events by calling the onRMEngineEvent operation. The

onRMEngineEvent operation is called with parameters that describe the event that occurred. The supplementary information field contains data or a pointer to data depending on the reported event.

When receiving the **ERMONFrameReady** event, the observer must update the graphic window with the offscreen buffer data by calling the *RMEngineInterface* drawOffScreenBitmap operation.

When receiving the **ERMONPrepareCnx** event: not applicable to this platform. Complementary information data is empty.

When receiving the **ERMONBuffering** event, the *Rich Media Engine* signals that it has begun bufferizing a scene. Complementary information data is empty.

When receiving the **ERMONPlaying** event, the *Rich Media Engine* signals that it has begun playing a scene. Complementary information data is empty.

When receiving the **ERMONCommand**, The complementary information field will contain a pointer to a literal string (LPCWSTR) that is the application layer command. The custom command can be decoded by using the decodeCustomCommand operation.

When receiving the **ERMONLoadComplete**, The complementary information field will contain a pointer to a string object that specifies a local file name where streamed data has been saved. If no data has been saved, the complementary information field is null. The user application is responsible for deleting or moving the file for future use. Note that the file name will always be null with Windows CE RME SDK.

When receiving the **ERMONError** event, the *Rich Media Engine* signals that it has encountered an error playing a scene. Complementary information data contains one of the error codes enumeration (see §0).

When receiving the **ERMONLaunchCnx** event: not applicable to this platform. Complementary information data is empty.

When receiving the **ERMONSoftKeys**, the complementary information is a boolean value that specifies if the content handles the phone soft keys. In this case, the standard phone soft keys behavior must be overridden and these events must be sent to the *Rich Media Engine*.

When receiving the **ERMONQuit**, the *Rich Media Engine* signals that the content requests the application to quit. Complementary information data is empty.

When receiving the **ERMONWAP**, the complementary information is a pointer to a LPCTSTR object that contains the URL to open the WAP browser with.

When receiving the **ERMONBacklight**, the complementary information is a boolean value that specifies if the content requests the screen backlight to be forced on/off.

Parameters:

oEvent: the player event

nInfo: complementary data for some states/events

LPWSTR onTextInputQuery(LPCWSTR lpszTitle, LPCWSTR lpszDefault) [pure virtual]

The onTextInput operation.

The `onTextInputQuery` operation must be implemented by *MRMEngineObserver* subclasses to allow *Streamezzo Rich Media Engine* getting user text input. The application should open a modal dialog to query user input.

Parameters:

lpszTitle: the title for the text input dialog

lpszDefault: the result string (also default string when entering the operation)

Return value:

The string entered by the user. *Streamezzo Rich Media Engine* will free the memory block of the returned value.

6.4.2 Member Enumeration Documentation

enum TRMError

TRMError enumerates the *Streamezzo Rich Media Engine* possible errors.

Enumeration values:

ERMSuccess: no error has been met...

ERMErrGeneral: the RME has encountered an error which has not been categorized

ERMErrPanic: the RME has encountered a fatal error

ERMErrSystemRes: the RME has encountered an error with system resources

ERMErrNotFound: the RME tried to open a non-existing URL

ERMErrArgument: the RME has encountered an invalid argument

ERMErrCorrupted: the RME is not able to read the content at the provided URL

ERMErrNotAvailable: the RME has requested an unavailable resource

ERMErrNotSupported: the RME has attempted to use an unsupported feature

ERMErrCompletion: the RME has attempted to perform an operation which is complete

ERMErrTimeOut: the RME has encountered a time out

ERMErrMalformedURI: the RME has encountered a malformed URI

ERMErrDNS: the RME has encountered a DNS error

ERMErrRequest: the RME has encountered an error when connecting to a server

ERMErrOpenResource: the RME is unable to open a local or remote resource

ERMErrConnectRefused: the RME does not accept the replying server

ERMErrHttpHeader: the RME has received an invalid http header

ERMErrHttpRequest: the RME has encountered request failure

ERMErrHttpResponse: the RME has received no response or an invalid HTTP response

ERMErrHttpRedirect: the RME has encountered an invalid HTTP field for redirection

ERMErrPredictiveNetError: the RME has foreseen a future network error (e.g. because of offline mode)

ERMErrNetConnection: the RME has encountered an error when establishing a network connection

ERMErrNetConnectionCancel: The user has canceled the network connection request

enum TRMEvent

TRMEvent enumerates the *Streamezzo Rich Media Engine* events types

Enumeration values:

ERMONFrameReady: the RME has redrawn its offscreen bitmap. The user application should get the *Streamezzo Rich Media Engine* off-screen buffered bitmap and draw it immediately on the screen through a call to *CRMEngineInterface* drawOffScreenBitmap operation.

ERMONPrepareCnx: *not applicable to Window RME SDK* - The *Rich Media Engine* is ready to issue a network connection. The custom application can use this event to force the use of a peculiar access point by opening it. Note that *Streamezzo Rich Media Engine* HTTP stack uses direct connections by default. The user application can force the *Streamezzo Rich Media Engine* HTTP stack to use a HTTP proxy by specifying its address and port number.

ERMONBuffering: the RME has begun bufferizing a scene.

ERMONPlaying: the RME has begun playing the current scene.

ERMONCommand: the RME has received some application layer data that should be handled by the rich media enabled application.

ERMONLoadRequest: the RME starts to open an URL. The complementary information contains the URL.

ERMONLoadComplete: the RME has finished loading the last AU of the content.

ERMONError: the RME has encountered an error. The complementary information is an error code. See TRMError for details

ERMONLaunchCnx: *not applicable to Window RME SDK*.

ERMONSoftKeys: the RME signals that the Rich Media application should send (or should no more send) soft keys events to the RME instead of raising the system menus. This event is sent when Rich Media content overrides the system menu. The Rich Media application should discard the handling of soft keys when an error has been raised by the RME since it cannot guarantee that the content menu can be raised safely. The custom argument is a boolean that specifies if trapping the soft keys is requested or not. The application must consider that the trapping of soft is off until it is explicitly requested.

ERMONQuit: the RME signals that the Rich Media application should quit now. This event is sent when the rich media content completely implements the service MMI and thus has a "quit" button.

ERMONWAP: the RME requests the opening of WAP content with the local WAP browser. The complementary contains the URL to open.

ERMONBacklight: the RME signals that the Rich Media content requests the application to force the device screen backlight on or off.

ERMONNextTrack: the RME signals that the current media playlist goes to next track. This event is used by the RME and should not reach the application.