Deliverable 3.6 Interface and Presentation Engine version 2

Daniel Ockeloen

Pieter van Leeuwen

14th October 2013

Work Package 3: LinkedTV interface and presentation engine

LinkedTV

Television Linked To The Web

Integrated Project (IP)

FP7-ICT-2011-7. Information and Communication Technologies

Grant Agreement Number 287911

Dissemination level[1]	*PU*
Contractual date of delivery	*30th September 2013*
Actual date of delivery	*14th October 2013*
Deliverable number	*D3.6*
Deliverable name	*Interface and Presentation Engine version 2*
File	*LinkedTV_D3.6.docx*
Nature	*Report*
Status & version	*Version 1.0*
Number of pages	*28*
WP contributing to the deliverable	*WP3*
Task responsible	*Noterik*
Authors	*Daniel Ockeloen, Pieter van Leeuwen* *Noterik*
Other contributors	*Matei Manca, Fabien Grisard* *UMONS*
Reviewer	*Katarina Stanoevska-Slabeva* *University of St.Gallen*
EC Project Officer	*Thomas Küpper*

[1] • PU = Public
• PP = Restricted to other programme participants (including the Commission Services)
• RE = Restricted to a group specified by the consortium (including the Commission Services)
• CO = Confidential, only for members of the consortium (including the Commission Services)

Table of Contents

1 Introduction 4

1.1 Related LinkedTV deliverables 4

1.2 History of the document 4

1.3 Abbreviations and Acronyms 5

2 Functional Requirements 7

2.1 Project requirements 7

2.2 Scenario Requirements 7

2.3 Technical demonstrators 8

2.3.1 First technical demonstrator 8

2.3.2 Second technical demonstrator 9

2.3.3 Outcomes of the technical demonstrator 10

3 Presentation Engine Structure 12

3.1 Springfield Multiscreen Toolkit 12

3.2 Single and multiscreen a unified model 12

3.3 Multiple device types 14

3.4 Application development 15

3.5 LinkedTV platform data integration 17

4 External control interfaces 19

4.1 Remote API 19

4.2 Gain Interface 20

4.3 Gestural interface 20

4.4 Hardware development 25

5 Future development 26

6 Conclusion 27

7 References 28

1 Introduction

The functional requirements of the second version of the LinkedTV interface and presentation engine are a revised version of the first version of the document. While in the first version of the document the focus was on the scenarios and extracting the functionalities needed from those to implement the interface and presentation engine and delivering a single screen solution, the focus for the second year was more to continue implementation and extend to deliver a second screen solution.

This has resulted not only in an extended interface and presentation engine but in a multiscreen toolkit that allows flexibility for rapid development of different second screen applications. This fits with the needs for the scenario partners who all have created multiple scenarios. The multiscreen toolkit allows them to quickly create different multi screen application demos based on their scenarios.

This document will more focus on the ideas and mechanisms used in the interface and presentation engine while a more technical description will be given in D5.5 *LinkedTV front-end: video player and MediaCanvas API V2*.

1.1 Related LinkedTV deliverables

The design requires the consideration and input of most of the LinkedTV packages. However, D3.6 strongly relates to the following deliverables in particular:

- *D3.5 Requirements document for LinkedTV user interfaces (version 2)*
- *D5.4 Final LinkedTV integrating platform*
- *D6.1 Scenario descriptions*

1.2 History of the document

Table 1: History of the document

Date	Version	Name	Comment
2013/07/10	V0.0.1	Daniel Ockeloen, Pieter van Leeuwen, Noterik	Created initial document structure
2013/07/31	V0.1	Daniel Ockeloen, Noterik	Added initial structure for 3, 4
2013/08/13	V0.2	Pieter van Leeuwen, Noterik	Added initial structure for 1, 2
2013/09/02	V0.3	Daniel Ockeloen,	Added initial structure for 5, updated 3.1, 3.2

Date	Version	Name	Comment
		Noterik	
2013/09/17	V0.4	Daniel Ockeloen, Pieter van Leeuwen, Noterik	Updated 2, 3.3, 3.4
2013/09/23	V0.5	Daniel Ockeloen, Noterik	Updated 4.1, 4.2, 5
2013/09/24	V0.6	Pieter van Leeuwen, Noterik	Updated 1
2013/09/25	V0.7	Daniel Ockeloen, Pieter van Leeuwen, Noterik	Updated 3,5, Added 6
2013/09/26	V0.8	Daniel Ockeloen, Pieter van Leeuwen, Noterik	Revised 3.5 and 6, updated 4.3, 4.4 and 5, added images in 2, 3, 4
2013/10/03	V0.9	Katarina Stanoevska-Slabeva, University of St. Gallen	Review with comments
2013/10/09	V0.9.1	Daniel Ockeloen, Pieter van Leeuwen, Noterik	Updated with comments from review
2013/10/09	V0.9.2	Pieter van Leeuwen, Noterik	Updated with comments from Lyndon Nixon, Modul
2013/10/13	V1.0	Martha Merzbach	Section 4.3 given by UMONS included

1.3 Abbreviations and Acronyms

Table 2: Abbreviations

Abbreviation	Explanation
API	Application Programming Interface
GAIN	General Analytics Interceptor
HbbTV	Hybrid broadcast broadband TV
IR	Infrared

Abbreviation	**Explanation**
J2EE	Java 2 Enterprise Edition
JAR	Java Archive
JSON	JavaScript Object Notation
NIC	Network Interface Component
QR code	Quick Response code
REST	Representational State Transfer
SMT	Springfield Multiscreen Toolkit
War	Web Archive
XML	Extensible Markup Language

2 Functional Requirements

This section describes the functional requirements that were taken into account for the second version of the LinkedTV interface and presentation engine. These requirements have been split in the two following categories:

- Project requirements - The general requirements that are defined by the LinkedTV project in general.
- Scenario requirements - The scenario specific requirements.

As this document is an update from the first version and in the meanwhile only minor changes have been made to both the project and scenario requirements we won't go into much details about the requirements as in D3.4 *LinkedTV Interface and Presentation Engine version 1*. We assume knowledge of the D3.4 deliverable and will only give a brief description of the most important features and changes since this deliverable.

2.1 Project requirements

In general the project requirements have remained the same since last year. The most important requirement for the project in the second year in respect to the interface and presentation engine was the realization of a second screen demonstrator. This in contrast to the first year where the focus was to provide a single screen demonstrator using the first version of the interface and presentation engine.

As the LinkedTV platform in the second year has been integrated to an end-to-end platform another project requirement was the integration with the other components of the platform. Entities and recommendations from the platform have to be displayed in the interface using the presentation engine while the presentation interface have to provide data that can be used for the personalization of the recommendations. The personalization will influence the entities and recommendations from the platform that the interface will display.

2.2 Scenario Requirements

The different scenarios created by partners have not changed much since last year. However several meetings with these partners have altered and matured the scenarios in such a way that there is now a more common and refined view on the scenarios. To achieve a better understanding of the possibilities for the scenario partners two technical demonstrators have been created. These demonstrators showed the possibilities from a technical point of view so that the scenario partners could take these into account. The first technical demonstrator is described in section 2.3.1 and focussed mainly on using the gestures that could be used while the second technical demonstrator, described in section 2.3.2, was a more evolved version from the first technical demonstrator.

This has resulted in a common demonstrator of the LinkedTV player for the second year that will be used by all the scenario partners. In the meanwhile tools have been created that allow

technical partners in collaboration with the scenario partners to rapidly create and experiment with different interfaces to see how their ideas work out and can even be used for user testing as the tools create fully working multiscreen demonstrations.

2.3 Technical demonstrators

To provide scenario partners with a better understanding of what's currently all possible with second screens given the current technique two technical demonstrators have been created and were showed and discussed with them.

2.3.1 First technical demonstrator

The first technical demonstrator was created using the "Tussen Kunst & Kitsch" content provided by AVRO. It shows one video playing on the mainscreen and allows a second screen to connect (Image 1) and to control the mainscreen. Users would be able to do the normal video control actions like play, pause, move backwards and forwards using touch model interactions like swipe, pinch and tab. It's possible to swipe to a different screen with chapters overview that could be used to navigate. We also experimented with sharing between the mainscreen and second screen in that users would be able to swipe a moment from the mainscreen to the second screen and swipe a timepoint back from their second screen to the mainscreen (Image 2). The core of this demonstrator was to test the new network layers and concepts, basic touch based interfaces on tablets and other mobile devices and to provide a base for some of the user feedback needed for further development.

Image 1: Connecting a smartphone to the main screen using a QR code in the first technical demonstrator.

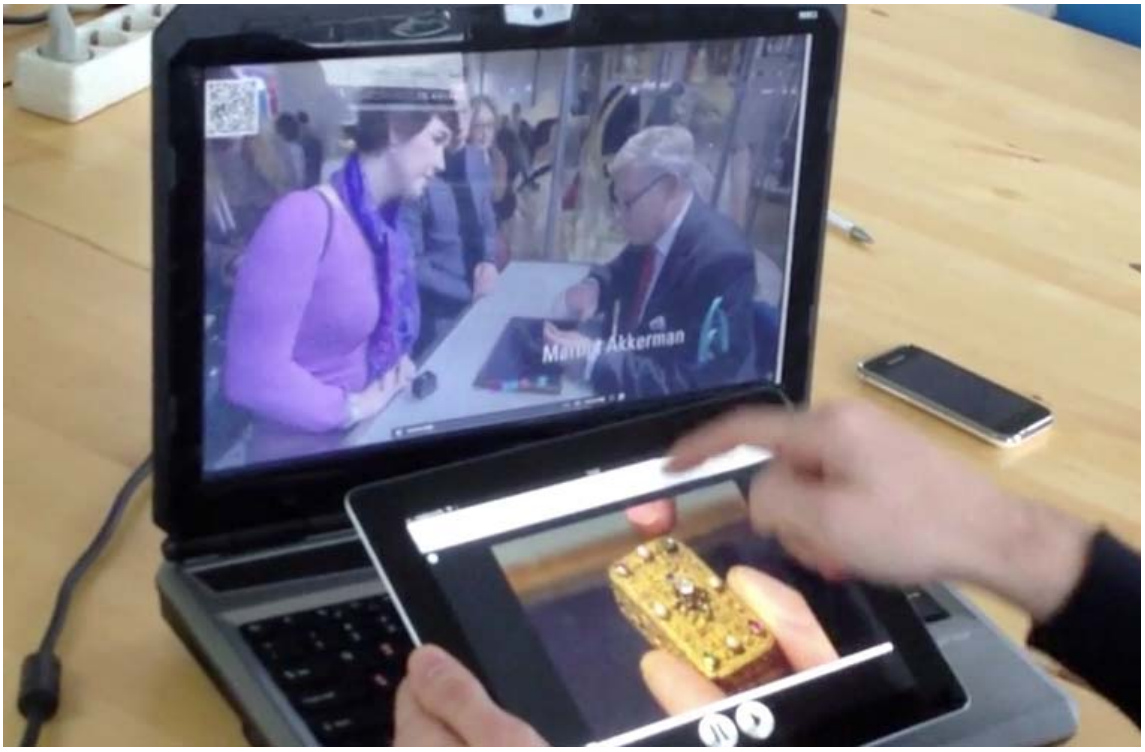

Image 2: Swiping a selected chapter from a tablet to the main screen in the first technical demonstrator.

2.3.2 Second technical demonstrator

The second technical demonstrator was more complete and has evolved into the second version of the interface that will be used as demonstrator for the second year review of LinkedTV. The demonstrator used open content in the form documentary called “Everything is a remix” [1]. The focus was on testing how to display time based information on the second screen (Image 3). A basic implementation of a user system with notifications was implemented. The notifications allow users to share fragments to other connected viewers and the mainscreen. It’s also possible to bookmark fragments that can be shared with other connected users (Image 4). We extended and implemented more adaptive features in the toolkit and the demonstrator reflects this with several proofs of concepts e.g. informational changes when rotating devices from horizontal to vertical mode and/or going from single screen to multiscreen while watching. This demo has been presented at the EuroITV 2013 demo session[2] and has been discussed during several other meetings related to second screen and HbbTV.

[2] Enriched Multiscreen TV Experience
http://www.noterik.nl/wp-content/uploads/2013/08/Enriched_Multiscreen_TV_experience.pdf

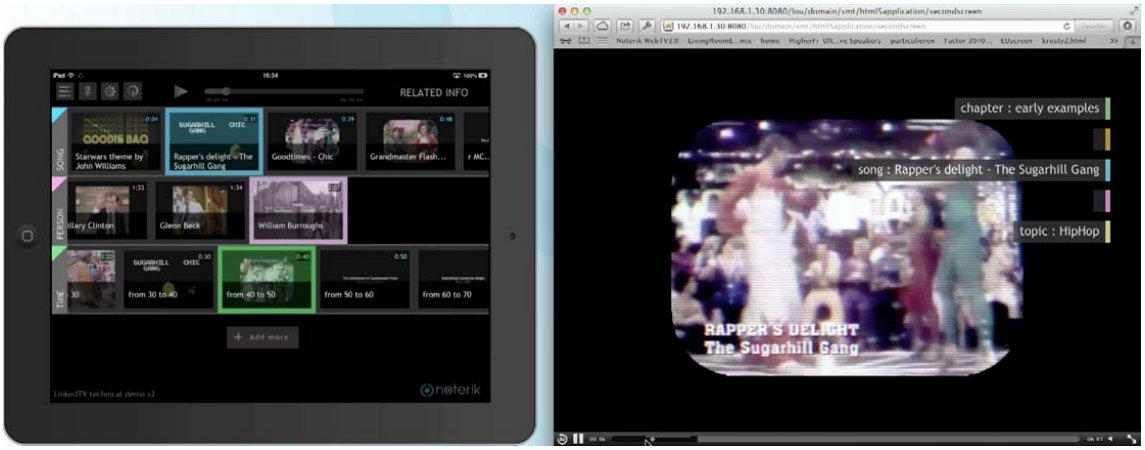

Image 3: A second screen linked with the main screen shows more information about current entities in the video.

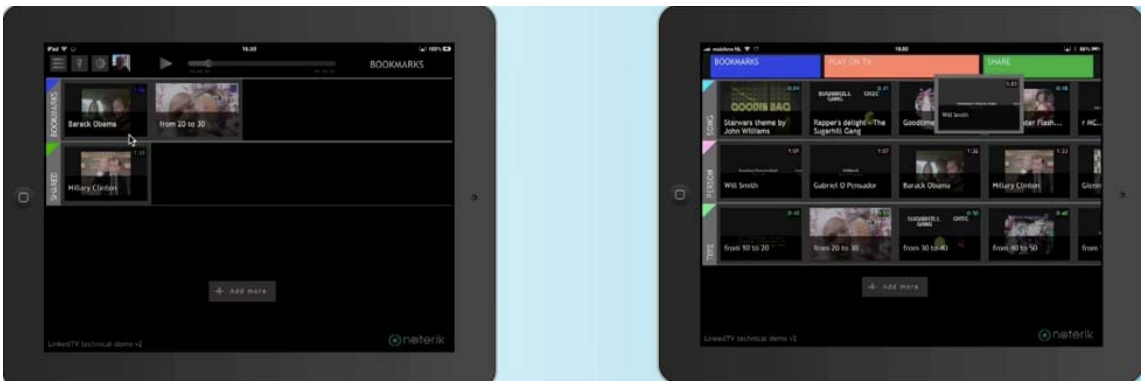

Image 4: Two connected users share their favorite fragments with each other using the social functions in the second technical demonstrator.

2.3.3 Outcomes of the technical demonstrator

The two technical demonstrators have led to a common view for the second year demonstrator of the LinkedTV player. The detected entities showed on the second screen will always be grouped into three different layers by dividing entities based on “who, what, where” to identify respectively persons, objects and locations (Image 5). While the video is playing on the mainscreen the entities in the three different layers on the second screen will be highlighted once the entity passes on the screen. If a user wants to know more about a certain entity the user can click the entity on his second screen and is presented with a short abstract and a larger image of the entity (Image 6). If the user wants to read more he can simply rotate his device. By means of this the user switches his second screen from a lean backward mode to a lean forward mode. Because of this the video stops playing allowing the user to read the entry. Once finished the user can rotate the device back and the video continues to play while only the abstract is displayed. Recommendations are displayed in the form of links that might be of interest for the user based on the selected entity.

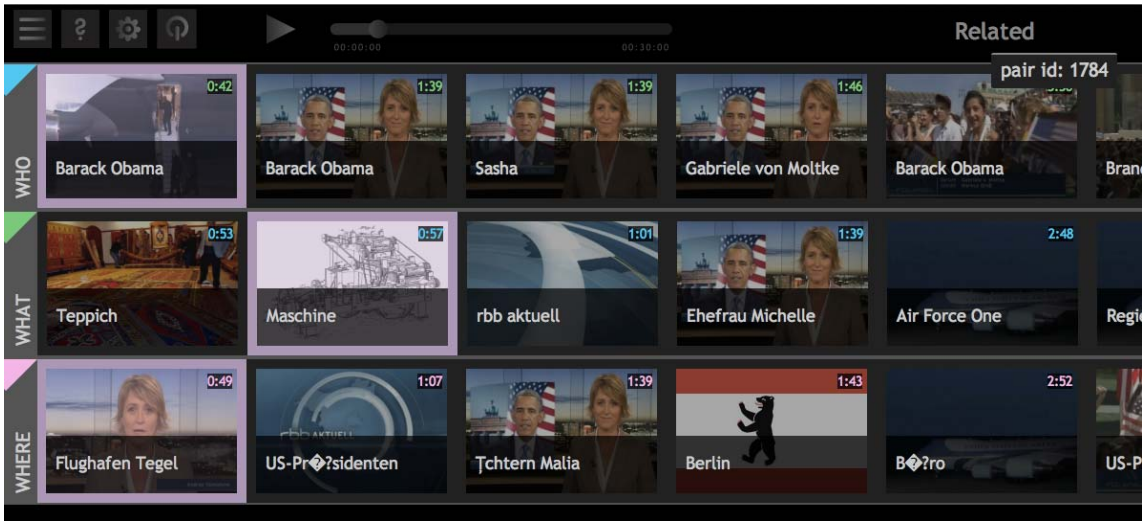

Image 5: The second screen showing detected entities of the video grouped by persons, objects and locations.

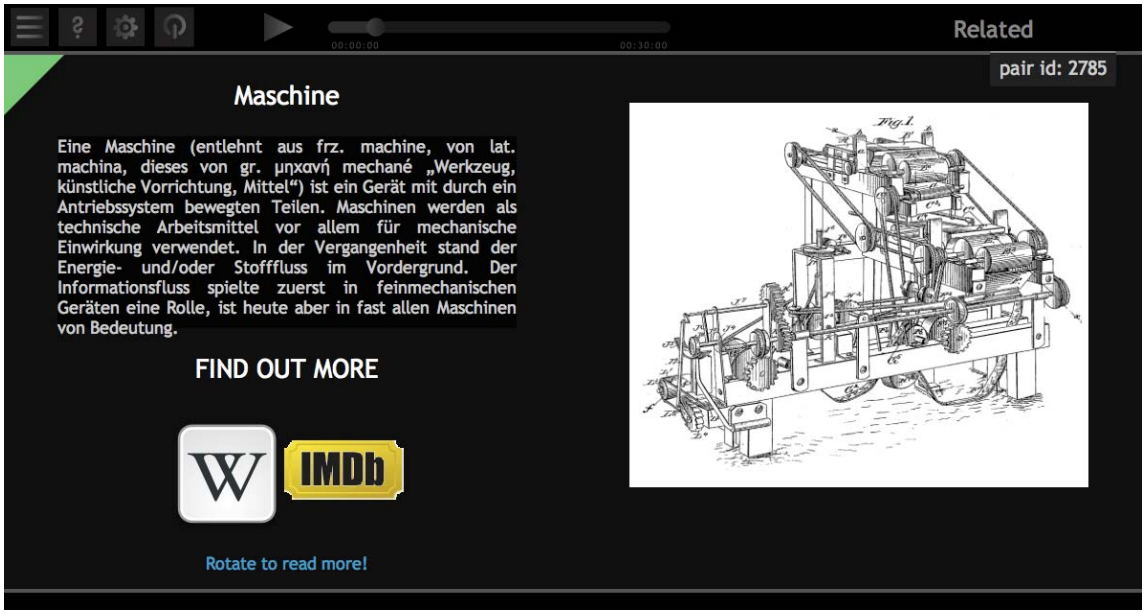

Image 6: The entity ‘Maschine’ is clicked on the second screen showing an abstract and an image of a machine.

3 Presentation Engine Structure

The main goal for the presentation engine was adding support for multiscreen applications. In the following sections we will describe how we have updated the presentation engine and started with the development of the multiscreen toolkit.

3.1 Springfield Multiscreen Toolkit

The presentation engine has evolved from last year's version into a new product called the Springfield MultiScreen Toolkit or SMT for short. In parts of the documentation we might refer to it as the toolkit or SMT. The toolkit has evolved as the project progresses and tries to resolve and answer the following problems and questions:

- How do we handle multiple formats (single/multiscreen, group viewing)?
- How do we handle multiple devices (phones, tablets, tv and HbbTV)?
- How can partners in the project be involved in the development?
- How can we host applications built using the SMT in the cloud?
- How do we display the information streams in different modes?

Within the toolkit we tried to resolve most of the problems outlined above and create a solid base for implementation of the final product in year three. As part of the building and testing process we created many small tests and examples of how to work with the toolkit. Describing all of them here would be too much and we will release them as part of the documentation of the toolkit but we have used several of the bigger ones for the technical demonstrators we made as part of this document to explain both concepts and show examples as described in section 2.3 and further. In the rest of this chapter we will expand on how we tried to solve each of the issues with concepts and examples.

3.2 Single and multiscreen a unified model

Instead of having different solutions for single screen and second screen applications we developed a concept where an application is independent of how many screens are involved and can in fact dynamically react to changes in the amount and types of screens attached to the application. In concept the idea is simple but we found out it requires people to make shift in the way they think about how they view these types of applications.

Writing distributed applications (Image 7- left), like second/multiscreens applications, is always more difficult. This is understandable since most of the teaching materials and tools programmers have used over the last decades focussed on non-distributed forms. Also it's more natural for programmers to think about single user applications instead of thinking about applications being used by multiple people at the same time. So what we try to do is make a distributed multiscreen application look like a non-distributed application (Image 7 - right). We do this by running the application on the server and introducing the idea of

screens. The framework will do all the complex work to make it behave distributed while the developer can think in the more traditional development model.

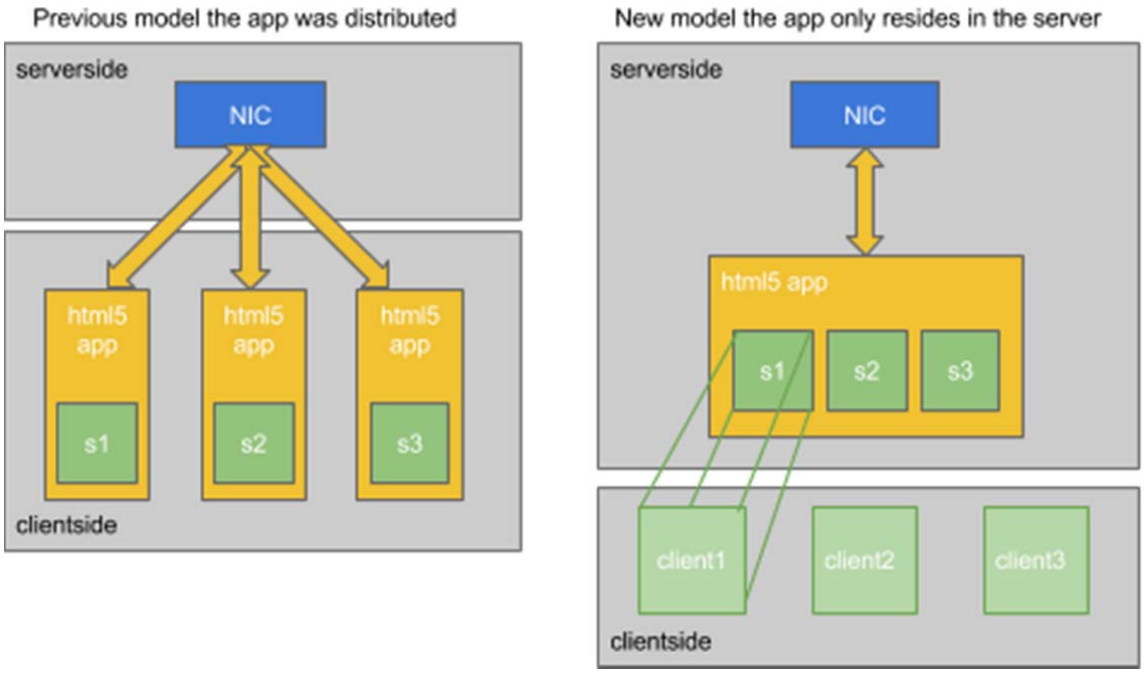

Image 7: Distributed application (left), server side application with screens (right).

This sounds like a slight change but has huge effects on both what we can do and the technical implementation since now the application is able to see and interact with all the users in a unified way.

From the developers point of view this now means that there is only one application that is always running (even if there nobody is watching) and that screens, users and devices changes are all just signals to this running application that it can and has to react on. This way it becomes possible for an application to grow from providing content from one screen to many when more users start watching and adapt to what each screen is capable of (Image 8).

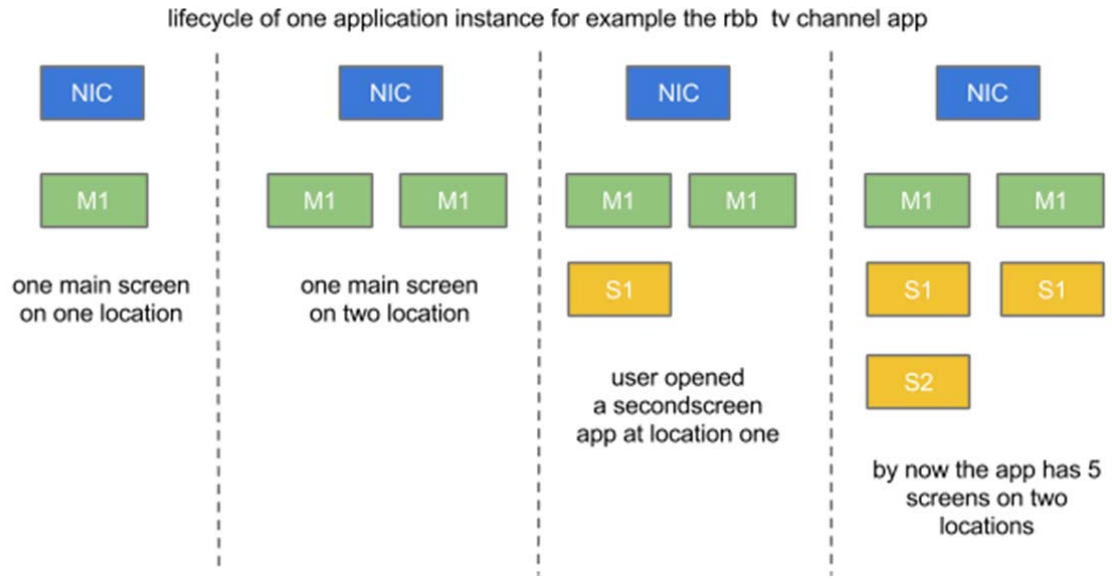

Image 8: Lifecycle of an application instance where users and devices are joining.

Creating this unified model for the developer does of course mean that the underlying framework (the toolkit) has to resolve many technical communication and synchronization problems since in reality the application is not fully running inside the server but involved many HTML5 browsers at clients, HbbTV clients, remote controls and even gestural interfaces. We will expand on this later in this document but from the unified application view this level of abstraction is actually the goal developers should not have to worry about how it's done but should be able to rely on the framework to resolve it.

3.3 Multiple device types

It became clear during the development of the scenarios and during the course of the project that the ability to support mobile devices and tablets is now becoming an integral part of the project. To keep the system as flexible as possible we decided to focus on HTML5 based applications with a downscaling to HbbTV specs 1.5 and possibly 2.0 for the third year of the project [2]. But within this limitation we again feel it's important that we can have a unified view for the developer and that the framework handles as much as possible in matching the device capabilities to our internal model.

Traditionally one would use adaptive/responsive design using stylesheets to adapt to different screen sizes and browser. And we use this on the lower levels inside the HTML5 clients but we also need a more generic concept on the application level. From the applications point of view each client (mostly browsers) are just seen as screens with capabilities and this is the abstraction level we want to support.

We distinguish between the following screen capabilities:

- **size** (pixel count and physical size)
- **rotation** (is the user holding it in landscape or portrait)
- **stylesheet level** (what options for visual styles do we have)
- **passive or active** (is it just a screen or also provides mouse, touch input)
- **keyboard support** (is there a fixed or optional keyboard)
- **brand specific features** (special buttons for example)

Each of the screens works with these capabilities as almost a contract (interface) between it and the system. Communication from and to these unified screens is either handled by the framework as part of the abstraction concept (so for example if a state of a button changes in the application the framework will change it on all the screens needed) or is done manually using a simple message back to the server side of the application (Image 9).

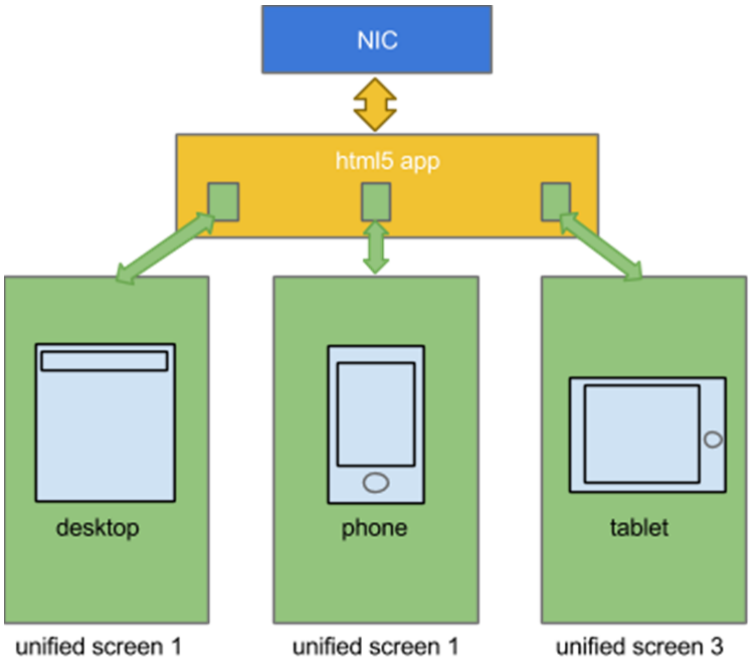

Image 9: Communication from the application to connected unified screens.

Using these methods we currently support all HTML5 based desktop browsers, android phones and tablets as well as Apple (iOS) phones and tablets. We also did some first tests with Condat in the area of HbbTV support to see if some of the basic abstraction concepts also work for the needed integration in third year.

3.4 Application development

One of the biggest problems we ran into over the last 12 months is the wish for a more open development process for the frontend applications. This was partly because of the shared responsibilities in the area of concept design and partly because many of the partners also

have a keen interest in doing second screen and touchscreen interfaces. Lastly and not unimportant we are getting into a phase of the project where we need to supply answers about future business models (Work Package 8: *Markets, Business Models and Exploitation Strategies*) and up to now it was unclear how we could extend some of the applications beyond the scope of the project.

To enable this we continue the concept of splitting the framework and applications, where as outlined above the framework takes away as much of the groundwork as possible to keep developing for these complex environments. As a side effect it became clear that it would now be possible for partners to also create applications without being involved in the development of the framework and more important the hosting platform we provide for these applications. The design we followed is the traditional model used for applets, servlets, restlet and other programming systems. The name we picked as base class is 'html5application' (Image 10).

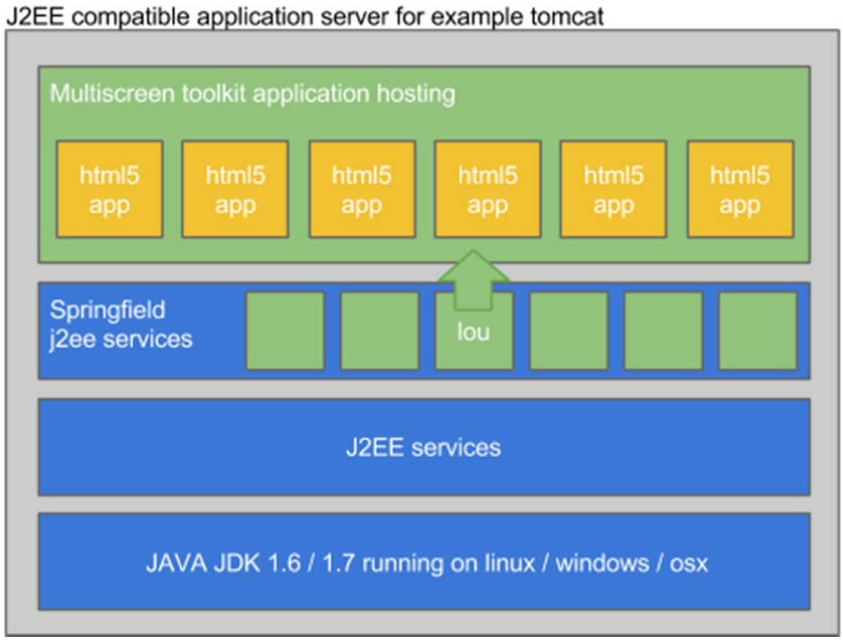

Image 10: J2EE compatible application server overview.

With most of these programming concepts also comes a development model, tools for packaging, distribution and hosting. Clearly it would be out of the scope of this project to invent such a tool chain for our applications so we decided to co-opt a well known model that people already know and use in the community and added some special parts to make it work for us. The system we picked was the J2EE [3] programming model that is already used for the Springfield [4] system itself. Developers can program using any J2EE toolchain (we use Eclipse and Tomcat/Apache) use the normal compile and deploy tools (using the jar and war standards). The main difference is that unlike normal wars that need to be deployed to a J2EE application server you now deploy the application to a newly designed deployment area inside a Springfield cluster (Image 11). Partners can still decide in the future to host their own Springfield cluster but they now also have the much easier and better understood

model of just writing applications and deploying them on the already available cluster inside the LinkedTV project.

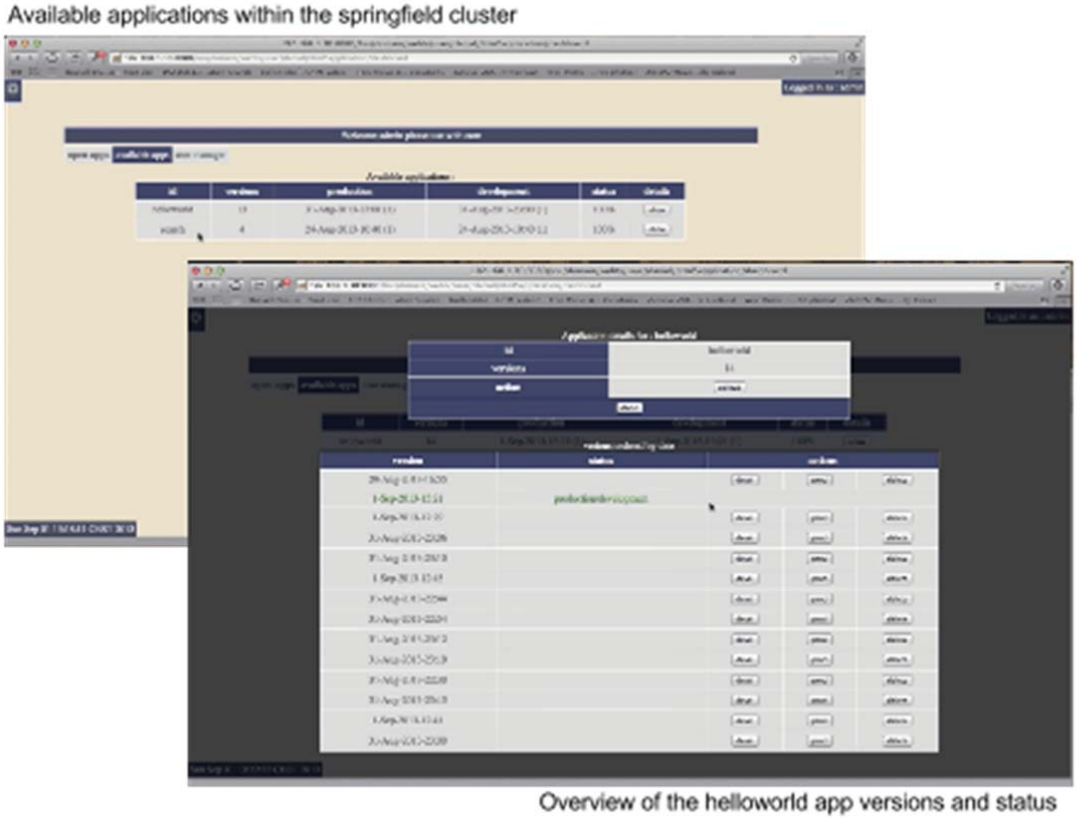

Image 11: The package manager to deploy applications within the Springfield cluster.

3.5 LinkedTV platform data integration

The LinkedTV platform provides the presentation engine with data in the form of detected entities and recommendations that can be used to enrich a video. However results will vary per user since personalization is applied to the results (Work Package 4: *Contextualisation and Personalisation*). This provides the most relevant and interesting entities and recommendations per user based on a user profile. The user profiles are created and updated using the feedback from the LinkedTV player to match the users' interests. Communication between the LinkedTV platform and the presentation engine is done through a REST interface that can provide XML or JSON output. This data is not directly used by the presentation engine but first parsed by the display database to convert it to a suitable format for the presentation engine. The presentation engine can filter results based on the interface, for example a smartphone might show less results than a tablet due to the size of the screen. To abstract the personalization from the presentation engine the API call to retrieve the data is the same with or without personalization applied (e.g. when a user has never logged in) is the same. Feedback from the LinkedTV player about user actions are send back to the LinkedTV platform using the GAIN interface, for more information about this see section 4.2.

This completes the integration loop between the LinkedTV player, including the interface and presentation engine, with the LinkedTV platform (Image 12).

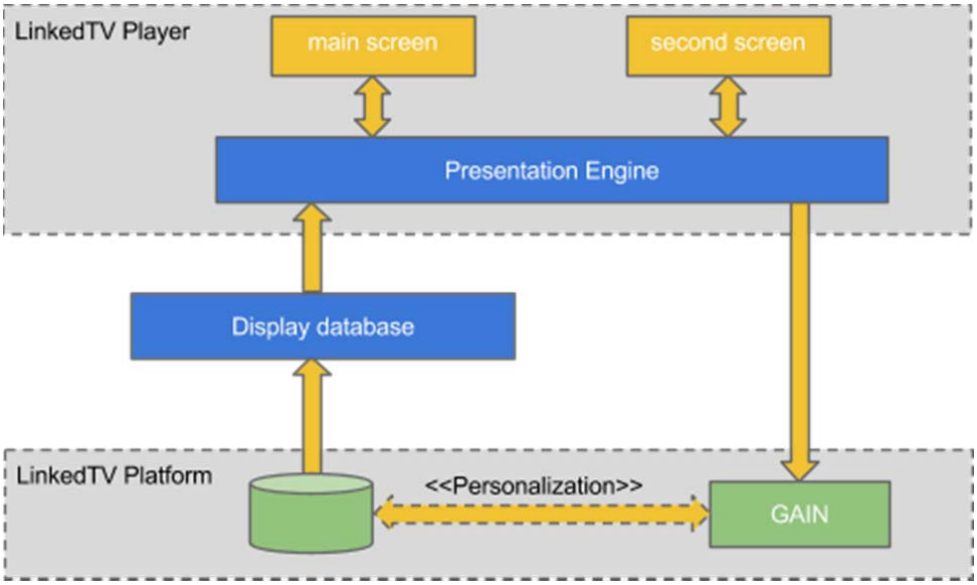

Image 12: Platform data integration loop between the LinkedTV Player and the LinkedTV platform.

4 External control interfaces

With a unified application view most of the communication and control interfaces are now abstracted into the framework and in most cases don't require work from the designers and developers. In this chapter we talk about the area's left where we do still need to interface from a software or hardware point to the outside and what we have developed so far.

4.1 Remote API

The remote API allows any external program to send messages to our applications and in that way control the mainscreen using simple commands. We have implemented and supplied JavaScript code and a working HTML5 example for the partners. The HTML5 example shows an image of an old fashioned remote control (Image 13) so it can also be used during demonstrations when the hardware versions (see 4.4) are not available.

Image 13: Remote control interface to simulate remote communication with an application.

4.2 Gain Interface

The second external interface that is not part of the NIC directly is the event interface to GAIN. The main goal for the connection to the GAIN system is to deliver all the signals needed for the personalization system (Work Package 4: *Contextualisation and Personalisation*). This is done by allowing all the components within an application to send events to the gain interface. Internally this means signals about what is happening to the mainscreen (play, pause, stop,...), user events (joining and leaving,...). We also allow external components using the remote API to use the player as a proxy to deliver its events to the GAIN interface (now planned to be used by the gesture/Kinect interface).

To be able to test the different events and user models (Work Package 4) we also implemented an event simulator (Image 14). This is an application mode build into each application that allows you to send any type of event and values to the GAIN interface. This will be used to test the effects of signals to Work Package 4 on the content that is being shown in the interfaces.

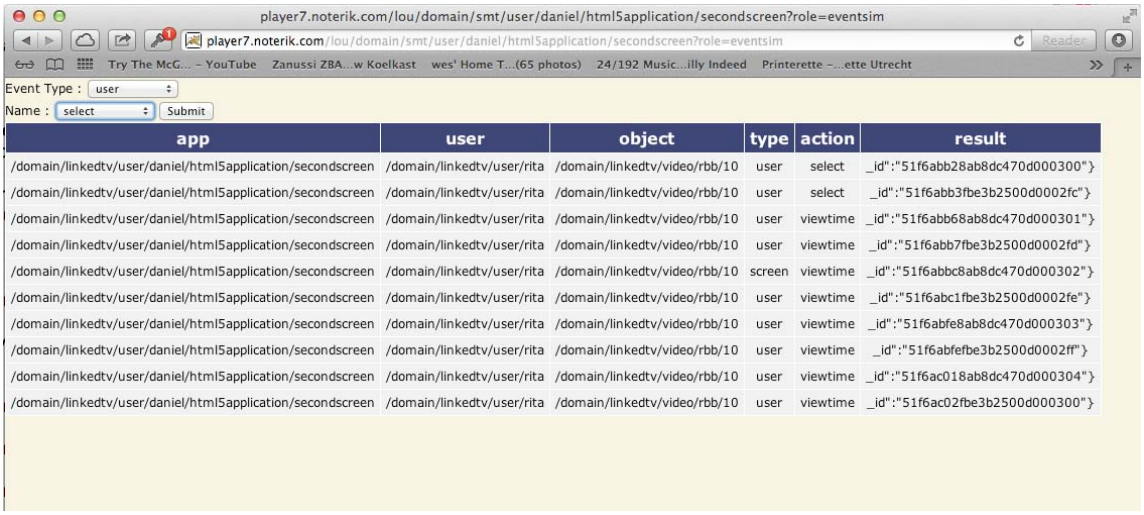

Image 14: Event simulator.

4.3 Gestural interface

Method

There are a lot of possible sensors which allow performing gesture recognition.

On one hand, several wearable sensors are available, as accelerometers or gyroscopes. The data they provide does not need a lot of processing before using gesture recognition algorithms, and most of the time, filtering is enough. On the other hand, some sensors use standard or RGBD cameras and provide classical RGB images or/and 3D depth maps and 3D clouds of the scene. In this latter case, the acquired data has to be processed to extract the shape of the user and follow his gestures.

For the LinkedTV project, we chose an RGBD sensor. This sensor provides, in addition to classical RGB images a depth map of the scene which describes the objects position related to the one of the camera. An example of depth map is displayed in Image 15.

Image 15: RGBD camera depth map. Clear pixels are closer to the camera than dark ones. Post processing on this depth map let us extract the viewer silhouette (in cyan) and the viewer skeleton (red dots linked by red lines).

The use of an RGBD sensor is also in line with the fact that the same sensor is also used to extract context and interest information (see deliverable D4.4). The idea is to use only one RGBD sensor to extract context, interest and gestures from the viewers. Currently we need to use two different sensors (one for interest and one for context and gestures) because the head direction extraction needs the current cameras to be not further than 1 meter from the viewer (see D4.4) while the context camera needs a global view of the scene and this thus located on the TV. Nevertheless, the new generation RGBD cameras (like the Microsoft Kinect 2 which will be available in 2014) will allow us to get interest and emotional cues even when the camera is far from the viewer like in the typical living rooms (2-3 meters of distance with the viewers).

For the current developments we used an Asus Xtion depth sensor which is low cost, not wearable, and is able to scan the whole scene. Furthermore, it comes with OpenNI software and the Primesens drivers that allow the find users in the scene and to track their virtual skeletons in 3D (see Image 15).

Among the lots of existing algorithms already used for gesture recognition (Gesture Follower [5], DTW [6], [7], etc.), we chose here the simplest approach: the descriptive method. F. Kistler (from Augsburg University, Germany) developed the Full-Body Interaction Framework (FUBI) [8], an open-source framework that uses a Kinect-like RGBD sensor and that has been successfully used in many situations [9], [10], [11], [12]. The main difference

between this method and the others is the learning phase. In other approaches, we have to teach to the system how to interpret the gesture by giving it a basis of examples. With the descriptive method, the user has to learn how to perform the gesture and to do it according to the initial description.

Description of FUBI

« FUBI is a framework for recognizing full body gestures and postures [for several users] in real time from the data of a depth sensor integrated using OpenNI or the Kinect SDK » [8]. The developer defines the gestures either directly in C++ classes or in an XML file (the latter solution being more flexible as modifications can be done and reloaded while the program is running). The gestures consist in some boolean combinations of basic elements, function of the time. These basic elements are the relative position (right hand above head), orientation (left arm oriented front) or linear movements (left hand moves to the left at a minimum speed of 800mm/s) of the skeleton joints. They are updated at each newly acquired frame and give a binary outcome. These binary values are combined in different states, during a defined period of time. All the states make a kind of pipeline, and if the gesture is performed in the order of this pipeline, within the correct timings, it is detected.

Referents

For this first prototype, a set of 16 commonly used command were selected, inspired by Vatavu [13], [14]. According to Wobbrock et al. terminology [15], theses commands are called *referents*.

This list of referents should cover all the functions needed to control a TV in a basic use, like navigating into the media, setting the volume, interacting with menus and asking for help. They are presented in Table 3.

Referent	Function / Description
Play	Start to play the media
Pause	Pause the playing
Stop	Stop to read the media
Next	Next media/channel
Previous	Previous media/channel
Volume value	Set the volume value
Mute	Mute the volume
Open menu	Pop up the menu
Hide menu	Close the currently opened menu
Yes	Answer to system questions
No	

Referent	Function / Description
Help	Ask for help
Add bookmark	Add a bookmark on the currently played media
Remove bookmark	Remove a bookmark on the currently played media
Lock / Unlock	Pass over controls / accept controls
Focus	Get the system attention

Table 3: Referents used for LinkedTV in the first prototype of gestural control, inspired by [13]

We opted for a limited set of referents for two reasons. The first one is the same as proposed by Vatavu [13]: « The number of gesture commands people can remember for effective use should be limited in order not to increase cognitive load. More individual commands would translate into complex designs with a similar puzzling effect […] Also, menus represent a viable option to group other, less frequent, commands » [16]. The second one is linked to the gesture recognition method we use: more gestures could lead to interaction between them and unwanted detections.

To limit the interactions between gestures, we added a Focus command, a gesture to be performed before most of the other commands to get the attention of the system. If no gestures have been detected after 2.5 seconds, the focus is lost and all the new gestures are ignored until the focus gesture is performed. The TV can be locked or unlocked to prevent any gesture performed in front of the system to be interpreted as a command. It is the same idea as Focus command but in a more restrictive way. Only gestures which need to be done immediately like “bookmark” do not need to be initiated using the focus gesture.

Implementation

For flexibility reasons, the gesture description has been implemented in an XML file. Most of the gestures were inspired by [14]. In this experiment, people were told to imagine gestures to match each referent, although the referents were not exactly the same as in our case. After some experiments we agreed on this set of gestures, some of them are used for different referents, depending on the context.

According to FUBI implementation, there are different types of gestures. Postures are static gestures that have to be maintained for a certain period of time to be detected. Linear movement are a simple movement performed at a certain speed (we chose 1m/s for normal speed and 2m/s for fast speed, indicated as quickly in Table 4). Combinations are complex gestures which need more than one linear movement to be described. Dynamic postures are like postures but one of the joint is moving and its position, relatively to other joints, is translated into a continuous value (e.g. to control a continuous parameter, such as volume).

Referent	Type of gesture	Context	Gesture description
Play	Posture	Focus ON	The right hand stay stable 40 cm in front of the torso for at least 2 seconds
Pause			
Stop	Posture	Focus ON	Arms crossed for at least 0.5 seconds
Next	Linear movement	Focus ON	Right hand moves to right quickly
Previous	Linear movement	Focus ON	Right hand moves to left quickly
Volume value	Dynamic posture	Focus ON	Left arm vertical and right hand near from it, volume = 1 when the right hand is at the same height as left hand and volume = 0 when the right hand is at the same height as left elbow
Mute	Posture	Focus ON	The left hand stay stable 40 cm in front of the torso for at least 2 seconds
Open menu	Linear movement	Focus ON	Both hands move up quickly
Hide menu	Linear movement	Focus ON	Both hands move down quickly
Yes	Posture	Menu opened	Same as play
No	Posture	Menu opened	Same as stop
Help	Posture	Focus ON	Both hands near head
Add bookmark	Combination of linear movements	-	Right hand stay stable 40cm in front of the torso for at least 0.3s and then moves up normally
Remove bookmark	Combination of linear movements	-	Right hand stay stable 40cm in front of the torso for at least 0.3s and then moves down normally
Lock / Unlock	Combination of linear movements	-	Left hand above head moves left normally, then left hand above head and moves right normally
Focus	Combination of linear movements	-	Draw a circle with one hand in any direction

Table 4: Description of the gestures implemented for each of the referent.

As it will be used very often, the focus gesture should be easy to remember and to perform. We chose to implement it as a circle, drawn with the right or the left hand in any direction. The only restriction is to start it from the top.

Each time a gesture is recognized, a message is sent to the AttentionTracker system (see D4.4). The AttentionTracker packs the message by using the websockets protocol and send

it to the test player built by UEP partner which is controlled by those gestures. Some controls (play/pause, etc…) are fattened by the test player with the video ID and time and forwarded to the GAIN component (UEP partner). For the next year, the messages will be sent to LOU (interface server of partner Noterik) which will command the LinkedTV player on one side and forward fattened messages to GAIN.

4.4 Hardware development

One of the results of last year's LinkedTV interface and review process is that we found out how limiting it was to use a remote control that is specific to a device. In the case of the first demonstrator we used an Apple remote that had to be used in combination with an Apple laptop (and not even all models worked). This lead to the request if we could not use more internet based solution and that we could use more realistic remotes. After some research we found a company that sold an IR-extender solution that we could reprogram to act as a gateway to our platform (Image 16).

Image 16: IR-extender that allows physical remote controls to communicate using HTTP requests.

These small boxes (about 5x5cm) connect directly to the internet and are programmed to send messages to our applications directly. The result is that when a partner gives a demo anywhere in the world he can use a normal browser/tv/tablet as the main (single) screen and attach this box to the internet and use a normal IR remote control. When putting the mainscreen into fullscreen mode on a tv or beamer this gives you a very realistic solution that can even be used to simulate a HbbTV setup. This last element is important for the user trials since not all the partners and countries have access to real HbbTV hardware.

5 Future development

In the third year of the project we need to focus more on the integration role between the different modes of use that have been researched towards an unified demonstrator. One of the main goals of the LinkedTV project is to enrich video fragments and will have come to full realisation when all work packages interconnect their systems. With Work Package 1: *Intelligent hypervideo analysis* and Work Package 2: *Linking hypervideo to Web content* providing the content Work Package 4: *Contextualisation and Personalisation* and Work Package 5: *LinkedTV platform* make the editorial and personalised filtering available in the platform. What we have learned over the last two years is that there is a second way you can interpret LinkedTV in that we found out that in real world scenarios you also need to supply this new content in a linked way. With linked we mean that a user's viewing location will most likely not be one static device but an environment where multiple devices that are linked and changed over time with users, laptops and mobile devices coming and going into the viewing area.

In the area of interface design we felt we also had to be actively involved in the creation of tools to be able to do some of the research and user testing. The era of paper prototyping is over. It just doesn't provide enough realistic feedback on new types of interfaces that have to be experienced to provide good feedback for testing (touch and gestural). So in close cooperation with Work Package 5 we will continue defining concepts both on the tool chain building and on the output side.

Using the co-design (with Work Package 5) toolchain we will based on Work Package 6: *Scenarios* create the final version of the LinkedTV interface (v3) that will merge the work from year one (single screen), year two (multiscreen) and the last not yet included area HbbTV. The end result we will aim for is that this version (v3) will be a unified app that adapts to all these needs and enhanced the experience for all the views watching a program on a personal and group level.

6 Conclusion

After the first year of the LinkedTV project where we delivered a single screen solution it was decided that the focus for the second year would be towards a second screen solution. During this second year we came to the conclusion that in real world scenarios it's much harder to clearly separate the different usage models. This is true for enduser concepts where we found that within one household it's most likely we have a mix of mainscreens, second screens and other input devices that result in a mix of user models (members of the family joining/leaving). At the same time we found out that for the development process tools also needed to be improved for us to be able to work in partnership with other project members. This resulted in the creation of a new the Springfield Multiscreen Toolkit which we discussed in this deliverable. As a result we feel we are well prepared for year three where we will work with the new tools we have developed to create a unified demonstrator that is able to adapt in innovating ways for single screen, second screen and HbbTV systems. And that this demonstrator will be delivered in a way that shows how to package it up and deploy to a cloud based system in a realistic sellable model.

7 References

[1] Everything is a remix, http://everythingisaremix.info

[2] HbbTV specification, http://www.hbbtv.org/pages/about_hbbtv/specification.php

[3] Java EE, http://www.oracle.com/technetwork/java/javaee/overview/index.html

[4] Springfield Framework, http://www.noterik.nl/products/webtv_framework/

[5] F. Bevilacqua, R. Muller, N. Schnell, F. Guédy, J. Lambert, A. Devergié, A. Sypniewski, B. Zamborlin et D. Glowinski, «Gesture Follower,» chez International Conference of New Interfaces for Musical Expression (NIME 07), New York, 2007.

[6] F. Bettens et T. Todoroff, «Real-Time DTW-based Gesture Recognition External Object for Max/MSP and PureData,» chez 6th Sound and Music Computing Conference (SMC 2009), Porto, 2009.

[7] N. Gillian, Gesture Recognition for Musician Computer Interaction, PhD Thesis, Belfast: Queen's University Belfast, 2011.

[8] F. Kistler, «FUBI- Full Body Interaction Framework,» 2011 - 2013. [En ligne]. Available: http://www.informatik.uni-augsburg.de/lehrstuehle/hcm/projects/tools/fubi/. [Accès le Mars 2013].

[9] F. Kistler, D. Solfrank, N. Bee et E. André, «Full Body Gestures Enhancing a Game Book for Interactive Story Telling,» chez International Conference on Interactive Digital Storytelling, ICIDS '2011, Vancouver, 2011.

[10] F. Kistler, B. Endrass, D. I., C. Dang et E. André, «Natural interaction with culturally adaptive virtual characters,» Journal on Multimodal User Interfaces, vol. 6, n° %11-2, pp. 39-47, 2012.

[11] K. Janowski, F. Kistler et E. André, «Gestures or Speech? Comparing Modality Selection for different Interaction Tasks in a Virtual Environment,» chez Tilburg Gesture Research Meeting (TiGeR 2013), Tilburg, 2013.

[12] C. Frisson, G. Keyaerts, F. Grisard, S. Dupont, T. Ravet, F. Zajéga, L. Colmenares-Guerra, T. Todoroff et T. Dutoit, «MashtaCycle: on-stage improvised audio collage by content-based similarity and gesture recognition,» chez 5th International Conference on Intelligent Technologies for Interactive Entertainment (INTETAIN), Mons, 2013.

[13] R. Vatavu, «User-Defined Gestures for Free-Hand TV Control,» chez 10th European Conference on Interactive TV and Video - EuroITV'12, Berlin, 2012.

[14] R. Vatavu, «A Comparative Study of User-Defined Handheld vs. Freehand Gestures for Home Entertainment Environments,» Journal of Ambient Intelligence and Smart Environments, vol. 5, n° %12, pp. 187-211, 2013.

[15] J. Wobbrock, M. Morris et A. Wilson, «User-defined gestures for surface computing,» chez CHI'09, New York, 2009.

[16] G. Bailly, D. Vo, E. Lecolinet et Y. Guiard, «Gesture-aware remote controls: guidelines and interaction technique.,» chez ICMI'11, New York, 2011.