AMBIENT ASSISTED LIVING, AAL

JOINT PROGRAMME

ICT-BASED SOLUTIONS FOR ADVANCEMENT OF OLDER PERSONS' INDEPENDENCE AND PARTICIPATION IN THE "SELF-SERVE SOCIETY"

D4.1

Mobile Client

Project acronym:	**GeTVivid**
Project full title:	**GeTVivid - Let's do things together**
Contract no.:	**AAL-2012-5-200**
Author:	**EVISION**
Dissemination:	**Public**

TABLE OF CONTENTS

TERMINOLOGY & ABBREVIATIONS

ACS	Appointment Coordination System.
AJAX call	An asynchron HTTP(S) call, implies no reload of the browser window content (no blank screens or error codes). The transferred data is encoded as JSON or XML.
API	Application Programming Interface. It specifies how applications interact with each other.
CSS	A language to define the style of HTML elements (<tags>).
Git / SVN	Programs that help to store different versions of files on servers and clients.
GUI	Graphic User Interface
HbbTV	Hybrid broadcast broadband Television.
HTML	A markup language to structure information.
HTTP	A protocol used by browsers and web applications to get or post information in the world-wide-web.
HTTPS	Like HTTP, but the data is sent encrypted.
JavaScript	A programming language that runs on HbbTV-ready devices, HTML5 mobile applications and every modern web browser.
JQuery	A framework that allows to write browser-independent JavaScripts.
JSON	JavaScript Object Notation. Open standard that uses human readable text to describe objects for data exchange (usually) between client and server.
OAuth 2.0	Open Standard for Authorization 2.0. One server stores credentials, other servers or clients (services) may ask it, if the user is authenticated and authorized to use a certain service. The big idea: A user only need one account to use multiple services (e.g. log-in to Spotify with your Facebook account).
REST	Representational State Transfer, basically a mapping from a HTTP address (URL) to an object (e.g. calendar entry) or a process (like authentication or creation of something on the server-side).
SSF	Second Screen Framework
STB	Set-top-box
Wi-Fi	Wireless Fidelity
Wire Frame	A crude design sketch.

1. EXECUTIVE SUMMARY

1.1 Link with the objectives of the project

In accordance to the project proposal, the Mobile-Client is meant to be a “Second-Screen-Device” for the GeTVivid system running on a HbbTV set at the home of elderly people. In this role it will not only be a display screen, but more an important additional input and controlling device for the whole system. In order to reach this goal the mobile application need to work hand-in-hand with the application running on the television. In fact hand-in-hand means a synchronization between the HbbTV client and the Mobile-Client is necessary.

The Mobile-Client is conceptualized as hybrid application for mobile devices, using web-application techniques as core part of a native application. Using this technical approach, it is possible to keep the architecture design of the Mobile-Client analog to the conception of the HbbTV client and coincidently enhancing the usage of the existing Second-Screen-Framework for the synchronization.

The main tasks described in this document are T4.2 (development of Mobile-Client) and T4.3 (Integration with core platform). Requirements resp. functionalities are derived from T2.2 (user requirements investigation), T3.1 (system conception and architecture) as well as T4.1 (design of the mobile user interface). The purpose of this document is to give comprehensive insights regarding the conceptual approach of the Mobile-Client and the technical details of its implementation.

2. CONCEPTION

2.1 Requirements and conception

The mobile client is conceptualized as an additional input and output device for the GeTVivid platform. It is not meant to replace the traditional remote control of the television, especially not in standard TV operations, like for example changing the channels. But it should give the user a more convenient way of interaction with system than the remote control is able to provide.

2.1.1 Hardware

Because of their sophisticated hardware specifications, modern mobile devices like smartphones and tablets provide the most convenient way of system interaction for the user. Especially, the intuitive and reliable touch interaction promotes a maximum level of usability even for more complex controlling tasks, for younger people as well as for elderly people with mild impairments.

The first round of end user evaluations had shown that tablet devices with a display size of about 10 inch suites best for the tasks of GeTVivid. The devices are not too big and not too heavy, so the end user can handle them well. And the big size display together with a higher resolution provides more space for bigger icons and font-sizes, so all the functions and symbols can be recognized more easily. For this reason a display size of about 10 inch is highly recommended.

Beside this there are only two more hardware requirements, a running Wi-Fi connection and an onboard photo camera (that is needed to scan a QR-Code during the coupling process between TV and mobile client. The details of this coupling process will be explained in detail in Section 4.2.5.1.). But both features are mandatory for almost all relevant mobile devices available on the market.

2.1.2 Platform independency (neutral application)

For good acceptance and marketability towards the end user, it is mandatory that the mobile client app is available for a broad range of common mobile devices and not only limited to a specific hardware device. In order to address this requirement the mobile application is conceived as a “neutral” application. This means that the app should be available for most common mobile platforms on the market, like Android and iOS.

For the purposes of the development and evaluation process during the project time, the client is being optimized for devices with Android operating system (V5.x “Lollipop” or higher) because it is the most common “open” standard in mobile operating systems. But due to the “thin-client” architecture (i.e., a computer program that depends heavily on another computer (its server) to fulfil its computational roles), as described in the following document, it will be basically compatible with other versions of the Android operating system.

Due to the “hybrid” architecture it will be possible to adapted the most important core part of the application to other operating systems by replacing the platform depending, surrounding part by a native app, for other mobile operating systems, like iOS or WindowsMobile.

2.1.3 Concept decision

One of the mayor questions regarding the conception of the mobile client was: Is it necessary to build a “native app” for the desired target platform or is it better to use a web application or a hybrid concept for the mobile client? In the following the available possibilities are discussed and the resulting decision will be explained.

2.1.3.1 Native app

A “native app" is an app developed essentially for one particular mobile device and is installed directly onto the device itself. Users of native apps usually download them via app stores such as the Apple App Store or the Google Play Store.

The main advantages of native apps are:

- Direct access to the hardware resources of the mobile device, like for example camera or GPS
- Direct access to graphic processing and resources, necessary for applications with 3D graphic features
- Access to special input and controlling capabilities, like special gestures, hardware buttons or stylus, available only for some special devices

2.1.3.2 Web application

Web applications, on the other hand, are basically internet-enabled apps that are accessible via the mobile device’s web browser. They need not be downloaded onto the user’s mobile device in order to be accessed.

The main advantages of web apps:

- Using common web techniques like HTLM and CSS standards
- Independent from the operating system of the hardware device
- Not necessary to use or even buy a special kind proprietary development environment
- No app store authorization is needed

2.1.3.3 Hybrid application

Hybrid, by definition is anything derived from heterogeneous sources, or composed of elements of different kinds. A hybrid app is one that is written with the same technology used for websites and mobile web implementations, and that is hosted or runs inside a native container on a mobile device. So it is a kind of marriage between web technology and native execution.

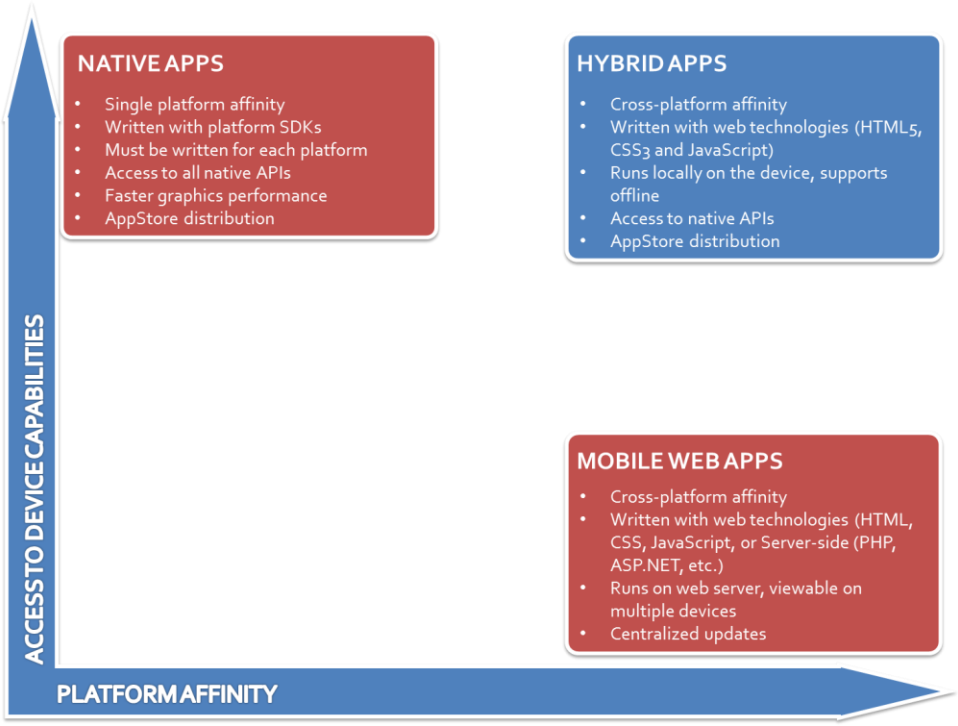

Figure 1: Native apps vs. mobile web apps vs. hybrid apps[1]

Hybrid apps use a web view control (WebView on Android, UIWebView on iOS) to present the HTML and JavaScript files in a fullscreen format, using the native browser rendering engine and not the browser itself. That means that the HTML and JavaScript used to construct a hybrid app is rendered or processed by the WebKit rendering engine and displayed to the user in a full-screen web view control, not in a browser. If necessary a web-to-native abstraction layer enables access to device capabilities that are not accessible in mobile web applications, such as the accelerometer, camera and local storage.

NATIVE vs. WEB vs. HYBRID: 7 FACTORS OF COMPARISON

KEY: CON | PRO | NEUTRAL

	NATIVE	HYBRID	WEB
COST	Commonly the highest of the three choices if developing for multiple platforms	Similar to pure web costs, but extra skills are required for hybrid tools	Lowest cost due to single codebase and common skillset
CODE REUSABILITY/ PORTABILITY	Code for one platform only works for that platform	Most hybrid tools will enable portability of a single codebase to the major mobile platforms	Browser compatibility and performance are the only concerns
DEVICE ACCESS	Platform SDK enables access to all device APIs	Many device APIs closed to web apps can be accessed, depending on the tool	Only a few device APIs like geolocation can be accessed, but the number is growing
UI CONSISTENCY	Platform comes with familiar, original UI components	UI frameworks can achieve a fairly native look	UI frameworks can achieve a fairly native look
DISTRIBUTION	App stores provide marketing benefits, but also have requirements and restrictions	App stores provide marketing benefits, but also have requirements and restrictions	No restrictions to launch, but there are no app store benefits
PERFORMANCE	Native code has direct access to platform functionality, resulting in better performance	For complex apps, the abstraction layers often prevent native-like performance	Performance is based on browser and network connection
MONETIZATION	More monetization opportunities, but stores take a percentage	More monetization opportunities, but stores take a percentage	No store commissions or setup costs, but there are few monetization methods

Figure 2: App concepts - pros vs. cons[2]

[1] Source: blogs.telerik.com/images/default-source/icenium-blog-posts/native-v-hybrid-png.png?sfvrsn=0

[2] Source: http://java.dzone.com/articles/state-native-vs-web-vs-hybrid

2.1.3.4 Conclusion: The mobile client as a hybrid app

Matching the pros and cons of the different app conceptions to the general requirements, it becomes clear that the hybrid approach fits nearly perfect for the desired specifications of the mobile client. The web application core stays platform independent, while the "shell", meaning the native app around the web core, can be easily adapted to other target platforms, like for example iOS or WindowsMobile.

Furthermore, with a hybrid conception it is possible to have consistent design, content and behaviour for mobile client as well as for the HbbTV client, because from a system architecture perspective HbbTV applications are also only a special kind of hybrid application. The core part of HbbTV applications is also basically a web browser with special profiles, connecting and using the corresponding web application. And as well there are some surrounding components dealing with the internal interaction between TV broadcast stream and the web content.[3]

And finally, a hybrid system conception actually allows the usage of the mobile client even without a HbbTV client, as a standalone solution in case that no connected TV is available. The different ways to use the mobile client will be explained in more detail in the following section.

2.1.4 Different ways of client usage

Since the mobile client is meant to be a "second-screen-device" for the GeTVivid platform running on an HbbTV set, it was necessary to define how a "second-screen-device" should behave, in terms of interaction between the mobile device and the television.

Some basic questions regarding the interaction between both clients had to be answered, like: What content is shown on the second screen and what content is on the TV? Should they have exactly the same user interface and functions, so that the appearance on both devises is mirrored? Or is it better to divide user interface and function between the devices? And in which way should they interact in this case, in order to reach the highest level of usability?

During the evaluation period of the project these questions were answered in accordance to one of the fundamental ideas of the project, namely using the television as the device elderly people are most familiar with, in order to transport additional functionality. According to this strategy, the whole concept of interaction should be clear, intuitive and as simple as possible. So three different ways of client usage were defined, starting with the most "traditional" way of interaction.

2.1.4.1 TV stand alone

This is the basic way of system usage, using the classic remote control of the television to control the GeTVivid application, the same way as in the system menus and other TV applications. This is probably the best way for new users to have a first contact with the new platform, because the user can use the remote control, which should be very familiar to him. It is sufficient for a simple experience of the system, like browsing through the application menus or reading existing entries, posted offers and demands of other users for instance. But it is pretty inconvenient for the more complex functions, like creating and editing new entries, especially when text input is required.

[3] For further details about the HbbTV client please refer to deliverable D3.1 and D3.2.

2.1.4.2 Full synchronisation between the HbbTV and the mobile client ("mirroring")

For a convenient use of the system, especially for complex tasks, the mobile client should be the preferred solution. Regarding the interaction between both clients it is specified that functions, user interface and behaviour should be as similar as possible on both devices. This means also that there should be a full synchronization between the clients, so that each display is "mirroring" the other display and every input done with the one device is synchronised via the system to the other device. This principle way of interaction is intuitive and easy to understand, even for inexperienced users. So it can be avoided, that they get confused or distracted, when they switch from one device to the other.

2.1.4.3 Overlay display during TV broadcast

In case the application is not already running on the HbbTV, it should be possible to inform the users about news and updates or special events taking place on the platform by fading in a special text or symbol area during the regular TV broadcast stream. This overlay area may contain any kind of information, like calendar and reminder entries, as well as a method of direct interaction, like pushing a specific button to get further information.[4]

[4] For further details about this mode on the HbbTV client please refer to deliverable D3.2.

3. SYSTEM ARCHITECTURE

For a detailed overview over the whole GeTVivid platform regarding conception and architecture, please refer first to deliverable D3.1. This document is meant to focus on the mobile client and the interaction between backend and clients, resp. between the HbbTV and the mobile client.

3.1 Frontend-backend interaction

3.1.1 System overview without synchronization

The following figure shows the basic system architecture between backend and clients, first without synchronization.

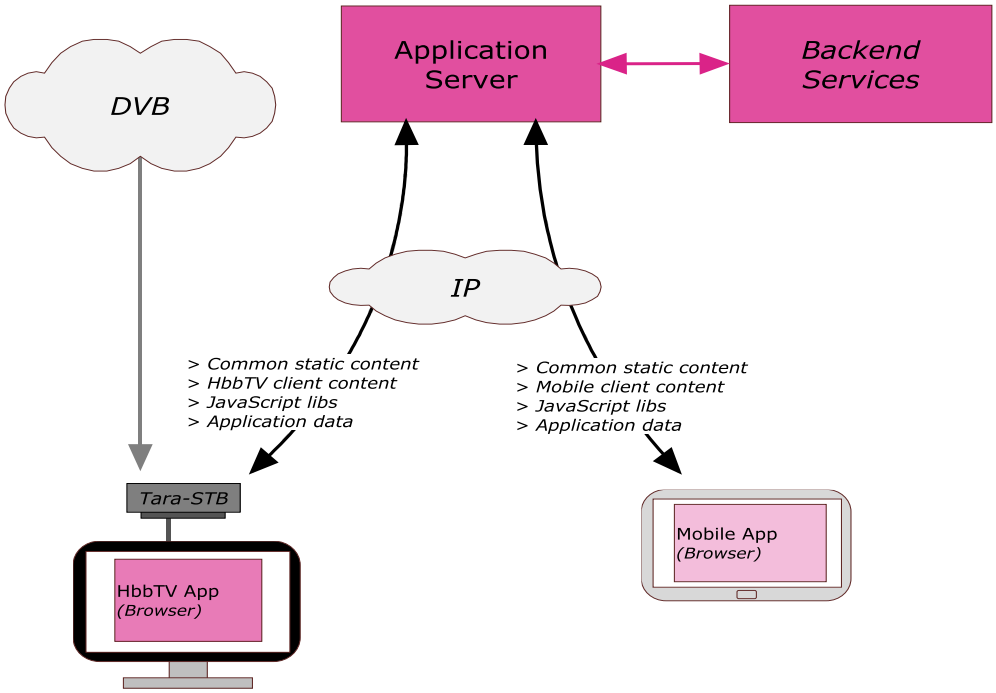

Figure 3: Basic architecture without SSF

On the TV the application is started via the TARA systems set-top-box, as described in Section 4.1 of deliverable D3.2.[5] On the mobile device the application is started like standard native app. Once started the application requests the necessary content from the "GeTVivid Application Server".

[5] GeTVivid deliverable D3.2 - TV Client (p. 12)

Mainly the content can be specified as:

- "Common static content" like the graphic components of the user interface, common CSS style sheets and the common JavaScript libraries.
- In addition to this common content, used by both clients, each client has some specific content, like special style sheets and JavaScripts, dealing for example with the different ways of input and control on both devices, classic remote control versus touch interface.
- And the application data itself, like for example user input data or a catalogue of the open offers and demands posted by users.

This way, both client applications have full access to the system, but both are working independently from each other. It is possible to work with the system on both devices at the same time but since they are not coupled via the Second-Screen-Framework there is no synchronization between the applications.

3.1.2 System overview with synchronization

In order to establish the mobile client as a real second screen remote control of the system some kind of synchronization process is necessary. Figure 4 shows the system with the implementation of the SSF as synchronization instance. The main working principal is still the same, but additionally both clients exchange messages about the status of the current application via the SSF Server. This way the applications on both devices are always at the same status, displaying exactly the same user interface on their screens. The exchange and synchronization process is explained in more detail in Section 4.2.5.2.

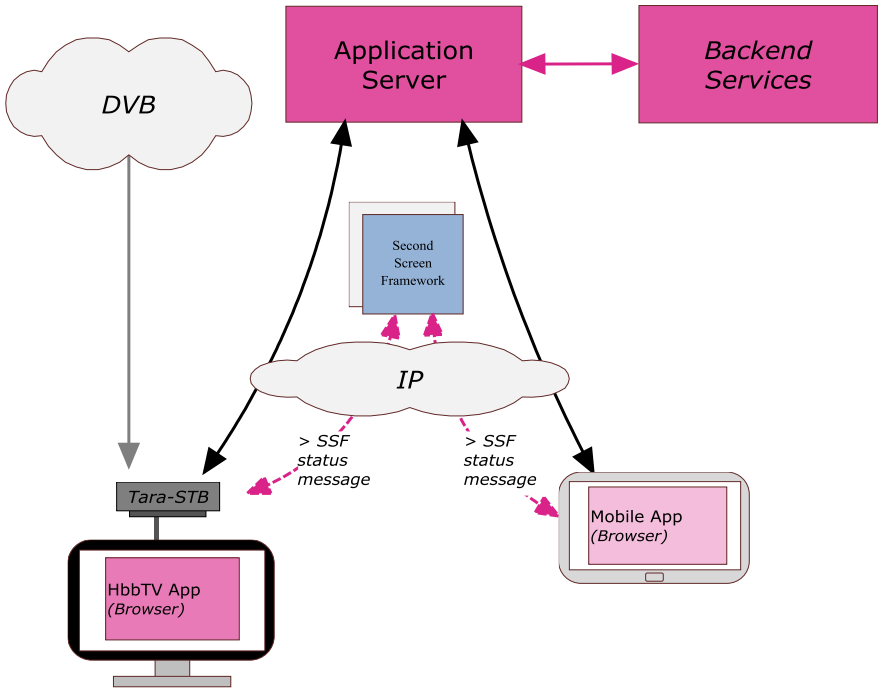

Figure 4: Basic architecture with SSF

3.1.3 Layer based overview

Figure 5 shows a layer view of the client and backend structure in order to have a clean view of the involved components and subcomponents.

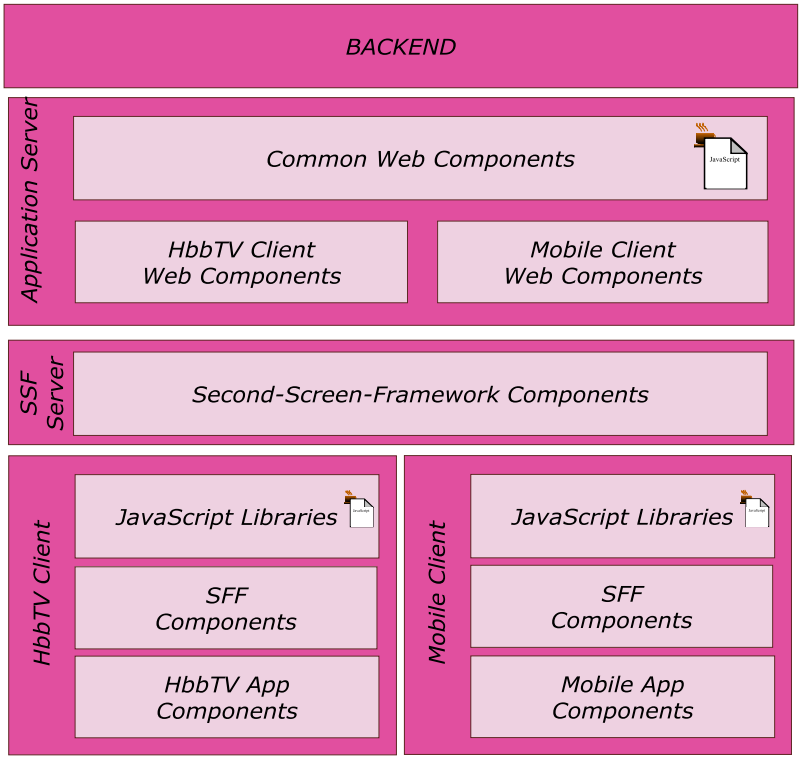

Figure 5: Layer based overview

4. MOBILE CLIENT COMPONENTS

4.1 Architecture overview

The high-level architecture of the mobile client and its most important components is shown in the following layer diagram.

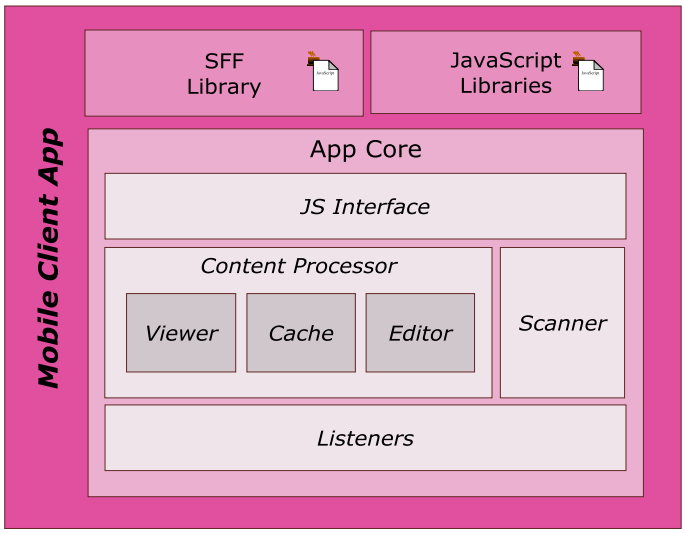

Figure 6: Layer diagram of the mobile client

The input **Listeners** are hardware connected components responsible for catching all input activities on the device. Physical input activities of the user are translated by the operating system into so called “Events”. The corresponding input Listener of the App is listening to this input “Events”, like for example `onTouch()` or `onClick()` and processes them according to the application logic.

The **Content Processor** is a browser-like component dealing with the web content, in terms of viewing, caching and editing. It represents the “web application core” as it is mentioned in Section 2.1.3.4. In Android this standard component is called `android.webkit`.

The **SSF Library** provides the API necessary for the coupling and the message exchange between the clients. The principal function of this component and the synchronization process is explained in Section 4.2.5.2.

JavaScript libraries are used for the implementation of all advanced functions of the web application. The `client library` developed by the project partners provides the API for the communication with the backend, like requesting and storing user data. But JavaScipts (JS) are also used, to create functions inside the mobile app itself, for example the validation of user input data (check of mandatory fields) is implemented in JavaScript. The **JS Interface** component is used to communicate the app core with the JavaScript libraries, for example enable a JavaScript function to start the scanner. **Scanner** is the component responsible to scan QR-Code from TV to start SSF coupling.

4.2 GeTVivid - Web application

Basically the mobile application is free of the strict limitations of the HbbTV client, as they are specified so far in the HbbTV V1.1 standard. Nevertheless it sticks to some of this limitation, for example no usage of HTML5 and no other script languages rather than JavaScript. The reason for this decision is that the mobile client, as a second-screen-device for the TV, should have an equal appearance to the user. Furthermore both clients need to provide a synchronized interaction. To meet this requirement it is necessary to keep the conception and architecture as well as the used technologies on both clients similar. The web application for the mobile client is build with standard web development techniques like HTML (V4.01) and CSS. Additional functions are realized in JavaScript.

The web application hosted on the server (demo.getvivid.eu/mobile) provides all functions and content for both clients. But in order to prevent any hassle for the elderly people with the complicate usage of a standard web browser on the mobile device, the web client is embedded in a native Android app. This way the mobile client appears - despite of its web application core - as a “normal” App, providing the common “app-like” user experience.

4.2.1 Mobile client and HbbTV client - A collaborative project

Since both system clients follow the same hybrid approach, it is possible to share common components, like most of the graphic content, all backend libraries and some JavaScript libraries, containing common functionality. While this approach establishes the necessity of a very synchronise collaborative way of development for both clients, there are still some differences between them, due to the different hardware resources and usability concepts of both devices.

4.2.2 Design of the graphic user interface

The design of the graphic user interface took place under the lead of the project partner PLUS. The basic concepts and designs were elaborated during several collaborative workshops. The results were worked out in a sophisticated stile guide. The stile guide describes all important information needed for the technical implementation of the graphic user interface. Due to the differences between TV and mobile client it is divided in two slightly different versions, one for each client. Following some examples to elucidate the major differences in the GUI implementation.

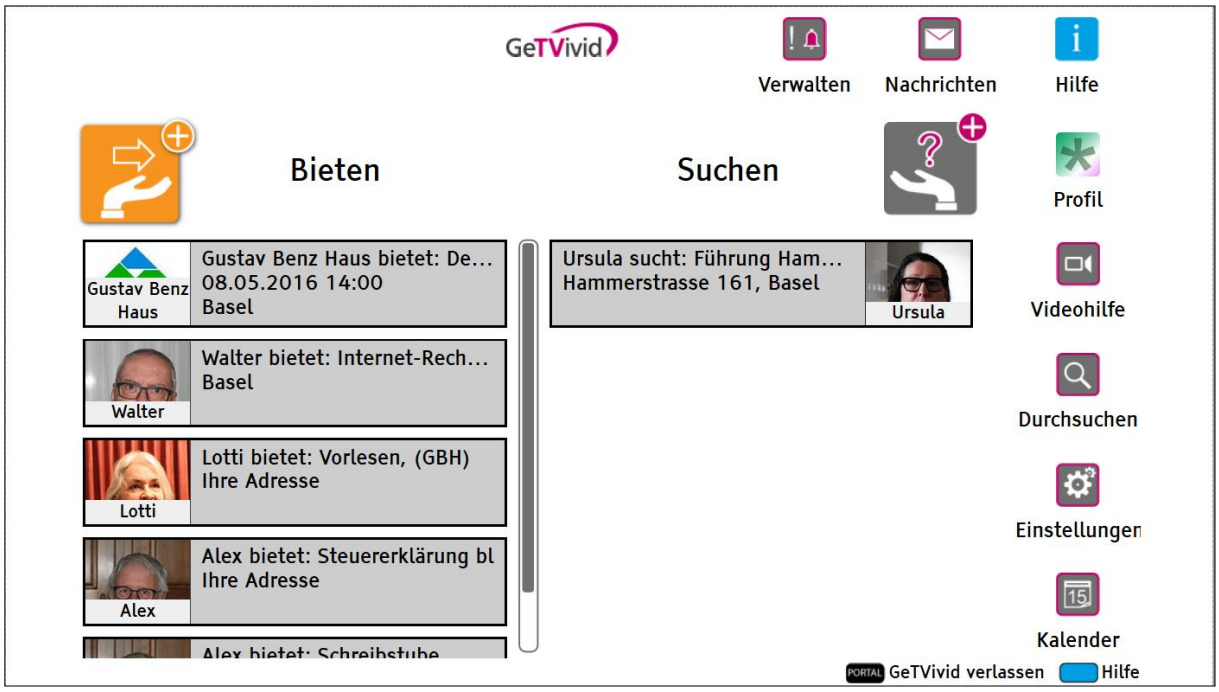

Figure 7: Start screen of the HbbTV client

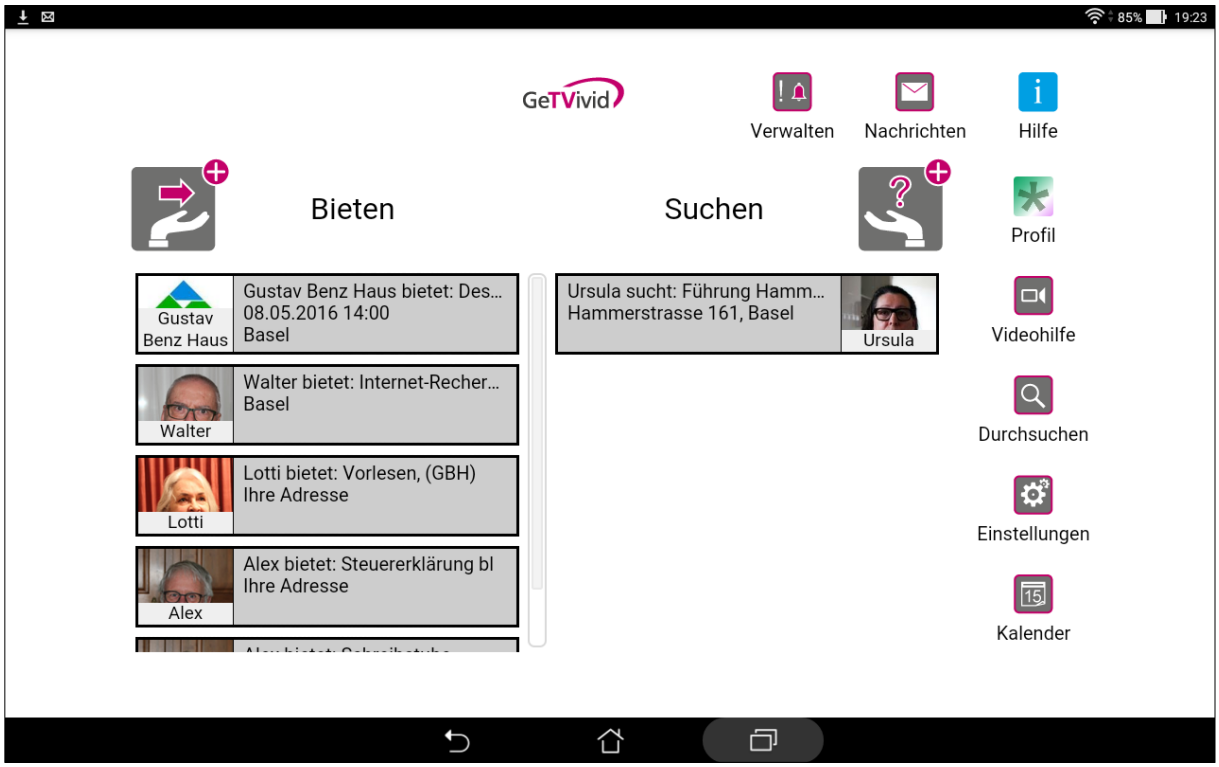

Figure 8: Start screen on the mobile device

Basically all pages have a slightly different format on both devices due to the different display resolutions and aspect ratios of the screens (1280x720/16:9 on HbbTV vs. 2560x1600/16:10 on mobile) and the “scene footer”, which shows up at the bottom of all pages on the TV client, is not implemented for the mobile version, because this is one of the TV specific implementations not applicable for the mobile device.[6] Anyway the GUI is designed in a manner that this differences are not much attracting the attention of the user in order to reach a “mirror-like” effect on both devices especially while working on both devices in parallel.

A major difference between the clients UI is the highlighting of `onfocus` GUI elements and the corresponding `KeyEvent` handlers on the TV. In order to emulate a cursor which can be navigated by the arrow key of the remote control. Because of the direct touch interaction of the mobile device this kind of highlighting is not needed. Therefore this difference requires a completely different kind of implementation for all GUI actions. For details of the implementation please refer to Section 4.2.3.

The same is applicable for all kinds of text or number inputs. Both clients using completely different types of keyboards. For the technical implications of this difference, please refer to Section 4.2.3.3. The following screen shots of both clients will exemplify the differences in the appearance of the user interface.

Figure 9: Keyboard of the HbbTV client

[6] For details please refer to Figure 11 and Section 4.2.3.2.

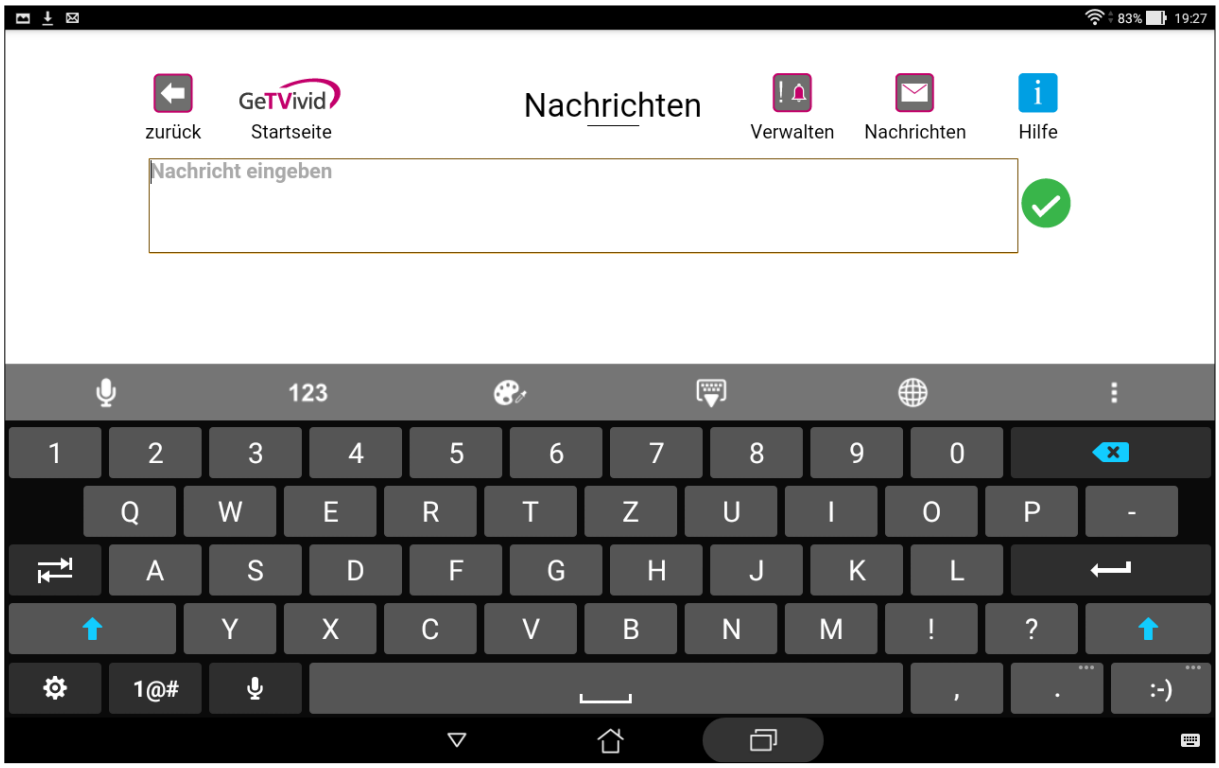

Figure 10: Keyboard of the mobile client

4.2.3 Specific implementations for the mobile client

Most of the technical implementation of the mobile client are inherited from TV implementation. An example of this inheritance is the implementation of dialogs, as described in Section 4.2.4 of deliverable D3.2[7]. The only difference between these two implementations on mobile device and TV is that in mobile the focus of icons is excluded. On the other hand in other parts of implementations the differences deviate from each other. The following sections will explain these major implementation differences for the mobile client in contrast to the inherited implementations of the TV.

4.2.3.1 Content processor - WebView

The first major differences between TV and mobile application is, how to view the web pages and resources from the server on the client. As already mentioned in Section 4.1, the native android `WebView` component as part of the `android.webkit` is used to load and display the `index.html` page of the GeTVivid application, once the app is started on the mobile device, whereas on the TV a specific HbbTV browser is used for this task. Although both browser components basically do the same, `WebView` is yet free of the strict limitations defined by the standard HbbTV 1.1 standard. Therefore it was necessary to build the whole GeTVivid application within these strict HbbTV limitations and adapt it afterwards to more liberal mobile browsers.

[7] See GeTVivid deliverable D3.2 - TV Client (p. 22)

4.2.3.2 General interface interaction and navigation

While general the conception of the navigation itself, namely `SceneManager`, is inherited from the HbbTV implementation, the way of interface interaction within the scenes (remote control vs. touch display) is also one of the major differences between the two clients. On the mobile device touch interaction is used to control the hole application, whereas on TV the navigation is achieved by using the arrow keys of the remote control. In technical terms, this means on the mobile the `onclick()` JavaScript function is used, whereas the `doclick()` function is used in TV , which is part of specific HbbTV JavaScript libraries.

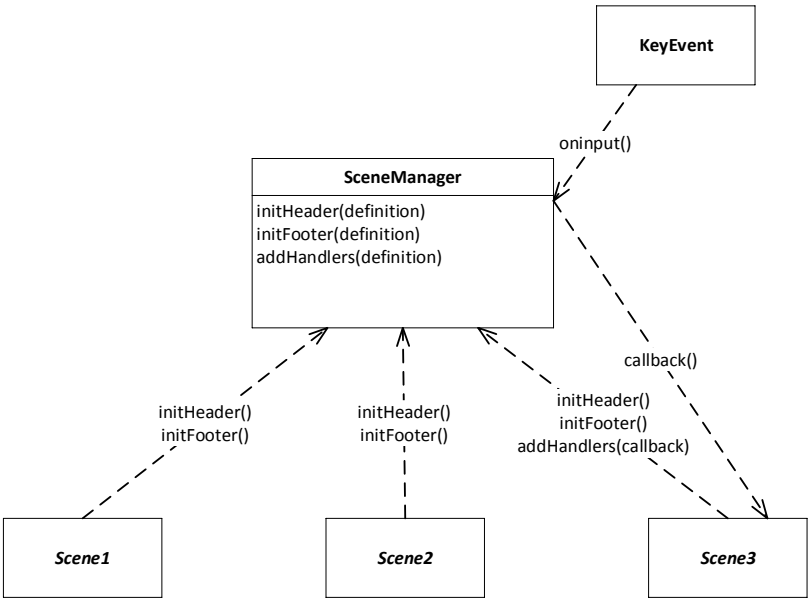

Figure 11: SceneManager implementation[8]

4.2.3.3 Virtual touch keyboard

In the mobile client the native virtual touch keyboard is used whenever the user is asked to enter text or numbers. This is not the case in TV. Since there is no such a feature available, an individual application keyboard was implemented for the TV to satisfy the GeTVivid specifications. This major difference entail different ways of implementation for data inputs, which will be explained in the following sections.

4.2.3.4 Input fields

Although the actual processing of data inputs is the same on both clients, makes the use of different keyboards a different kind of graphical representation necessary. So the diagram in Figure 17 of deliverable D3.2 is valid also in mobile implementation, except that the `show`, `addCharcter`, `showKeyboard` functions are not implemented on mobile.

[8] From: GeTVivid deliverable D3.2 - TV Client (p. 11) - Please note that the `initFooter()` functions are not implemented for the mobile client.

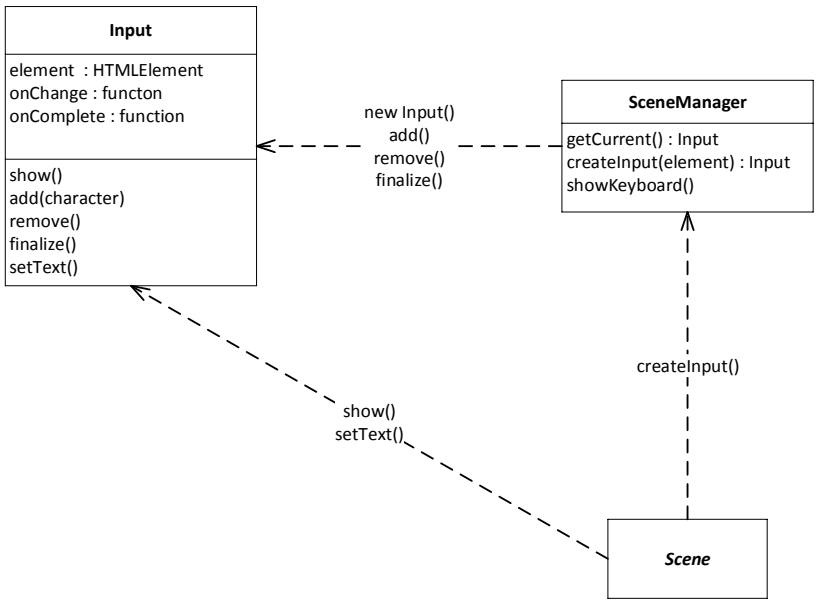

Figure 12: Input method implementation using facade pattern[9]

On the mobile client, text input fields are realised as separate popup windows containing only the input field and an action button to close the window. The action button is also to start processing the text entered. Each input field is registered once it is created to listen to incoming SSF messages coming from TV to mobile. In difference the TV depends on `Hide` function of the keyboard to start processing the text to the other side.

Similar to the TV there are also two different screens used for inputs on the mobile client, one for `text` and other for `date`, with date input realized as a dropdown list of possible numbers resp. dates. The following two figures will give an example of both input methods.

[9] From: GeTVivid deliverable D3.2 - TV Client (p. 22)

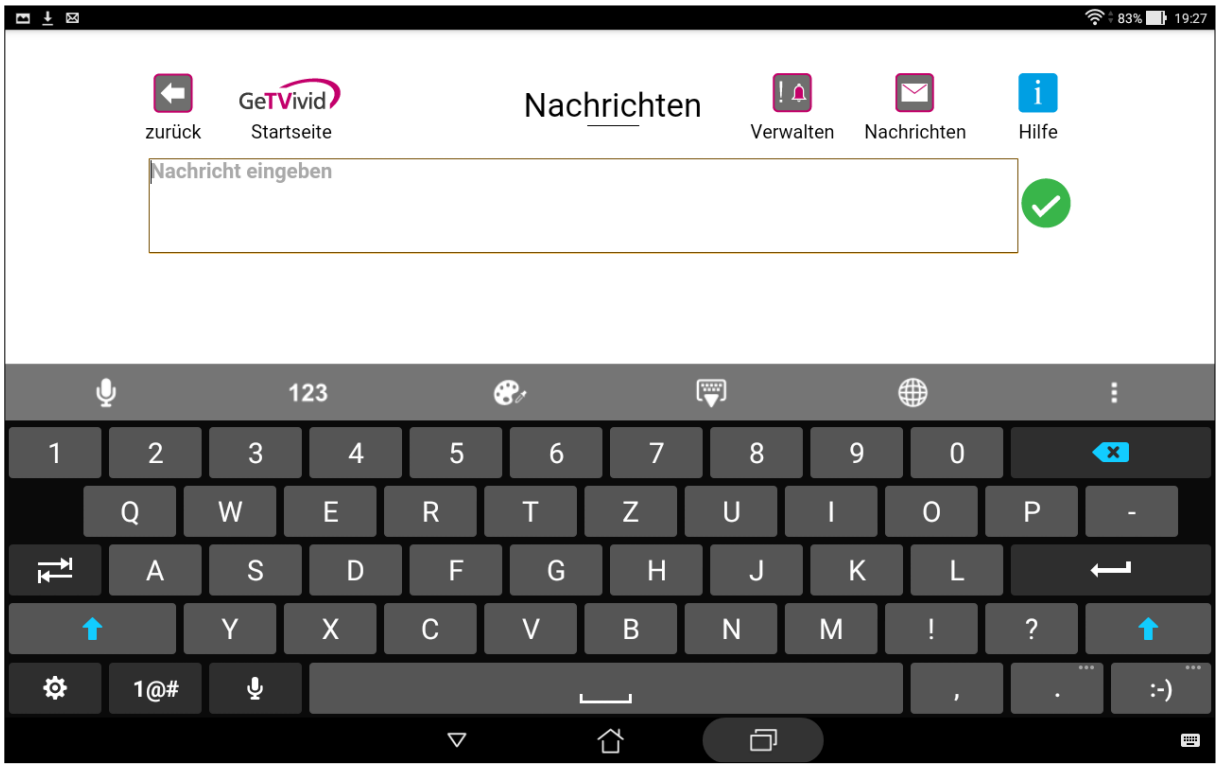

Figure 13: Input box and native keyboard of the mobile client

Figure 14: Date input of the mobile client

4.2.3.5 Scrolling

Scrolling of `DIV` item lists on TV is described in detail on Section 4.2.5.1 and 4.2.5.2 of deliverable D3.4.[10] While the implementation of `DIV` item lists is basically the same on the mobile client, the scrolling of the lists items is implemented in a completely different way by using the CSS feature `overflow-y: scroll`. This feature allows the user to scroll all lists within the app by the common touch gestures. The following code examples illustrate the way of implementation of scrolling for the mobile client.

The items that need to be scrolled have the CSS class `.mobileframe` which allows them to be scrolled by touch action:

```
.mobileframe {
    overflow-y: scroll;
    display: none;
    display: block;
}
```

The following html code shows how to scroll messages within the *organizer* section of the application:

```
<div id="organizer-messages">
    <div class="items mobileframe" onscroll="scrollMobile(this);"></div>
    <div class="hidden-spaceless scroll-bar">
    <div class="scroll-thumb"></div>
    </div>
</div>
```

The event `onscroll` will be cached whenever a list is being scrolled and it will call the `scrollMobile()` function to update the look and feel of the list in the same way as on the TV.

Following code shows the mobile adaption of `scrollThumb` and `scrollBar` as described for the TV client in deliverable D3.2.[11]:

```
if(element.scrollHeight <= element.clientHeight) {
    if (getvivid.util.hasClass(element, "mobileframe"))
    {
        getvivid.util.removeClass(element, "mobileframe");
         element.removeAttribute("onscroll");
    }
    hide(scrollBar);
} else {
    show(scrollBar);
    if (!getvivid.util.hasClass(element, "mobileframe"))
    {
        getvivid.util.addClass(element, "mobileframe");
        if (!amount || (amount && amount != -1)) {
            element.setAttribute("onscroll", "scrollMobile(this)");
        }
    }
```

[10] See GeTVivid deliverable D3.2 - TV Client (p. 26ff)

[11] See GeTVivid deliverable D3.2 - TV Client (p. 26ff)

```
    var rate = scrollBar.clientHeight / element.scrollHeight;
    var thumbHeight = rate * element.clientHeight;
    var thumbMargin = Math.max(0, rate * element.scrollTop - 8);
    scrollThumb.style.height = thumbHeight + "px";
    scrollThumb.style.marginTop = thumbMargin + "px";
}
```

4.2.3.6 JavaScript Interface

The JavaScript interface is the interface between the core of application implemented in `Java` and the additional `JavaScript` code used, for example initializing the scanner as decried in the next section.[12] Once the user press the scanner icon in the app a `JavaScript` function will be called which will initialize the scanner implemented in `Java`. This way the JavaScript interface establishes a kind of tunnel between the two components. The same interface is also used to connect the native Android back-button to specific JavaScript functions, like "go back to previous point of the application". The usage of the native menu-button function is implemented in the same way. In general the JavaScript interface is always used whenever to access the peripheral hardware resources of the mobile device.

4.2.3.7 Scanner

The scanner is an exclusive implementation of the mobile client. It is a native Java implementation used to scan QR-Code from the TV screen to start the SSF coupling. The Scanner is initialized using the JavaScript interface, as explained in the previous section. Once the scanner component is initialized it can be handed as part of the app itself in order to use the back camera of mobile device to scan and process the QR-Code from the TV.

4.2.4 Backend communication

For the implementation of the backend communication there are no specific differences between the clients. Both are using the same libraries `client library` provided by PLUS - used to access web service provided by the Amiona backend for instance to access database (inserting/editing user offers)

4.2.5 Synchronisation of the clients

4.2.5.1 The coupling process

For the coupling of the both clients, mobile and TV, the Second-Screen-Framework framework provided by IRT is used. A common JavaScript library is used in both clients to access the framework.

The Process of coupling starts once the user press `coupling` function on TV which shows a unique QR-Code containing information about the device as well as URL to connect to. On the other hand, within the mobile client the user starts the scanner by clicking scanner icon, which initialize the scanner responsible to scan the QR-Code displayed on TV. Using the scanned data the mobile client will be connected through the SSF server with the TV. The SSF server stores the connection between the two devices by ID. Once the connection is established both client can start exchanging messages via the SSF. These messages are used to keep the clients synchronized.

[12] Refer to Figure 6: Layer diagram of the mobile client

4.2.5.2 The synchronization process

Once the coupling is established the SSF server is able to forward messages from one device to the other. The messages communicated between the two clients are of the following types: `PATH`, `SELECT` and `INPUT`.

- The message of type `PATH` informs the other client about the pages opened.
- The `SELECT` message type is to inform the other client once an item within an opened page is selected or unselected, for example the implementation of a radio button.
- The `INPUT` type is to synchronize text entered by the user to the other client, for example text fields.

Assuming that the GeTVivid platform is running on both devices, the following description will illustrate the synchronization process for the message type `PATH`.

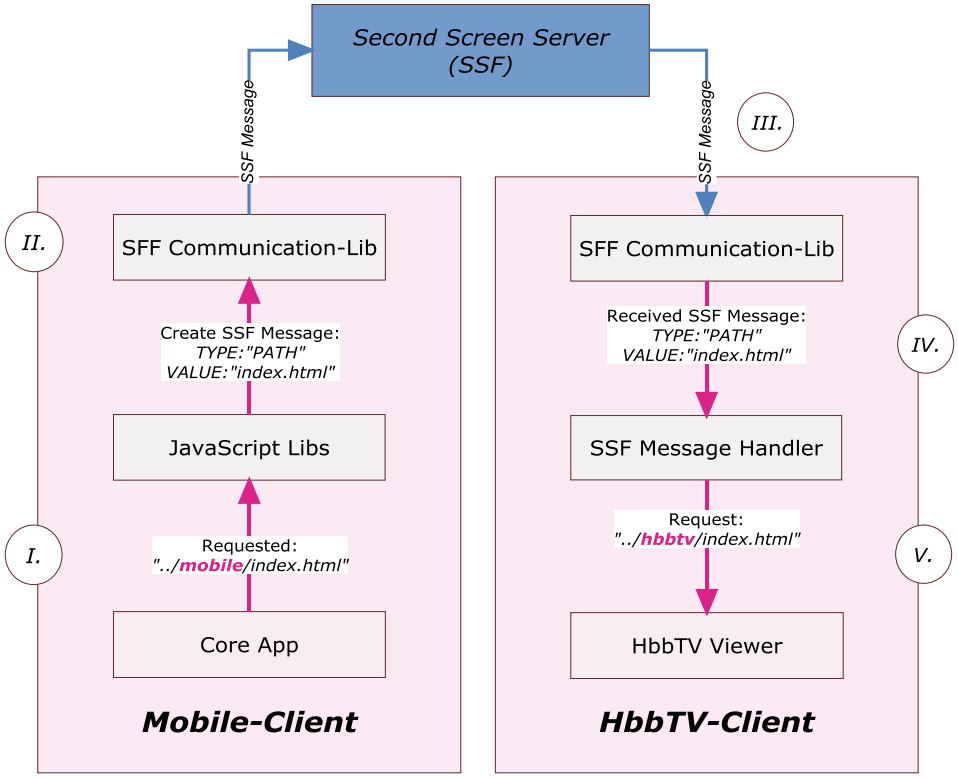

Figure 15: Synchronization process mobile-to-TV

Starting with a common input, like pushing the GeTVivid logo (“Home-Button”) on the mobile device for instance. This input navigates from any point of the application back to the starting page. In terms of a web application this means, the mobile client will request the “`../mobile/index.html`” page from the GeTVivid application server.

Beside this standard action, a SSF message of type `PATH` and value `index.html` will be sent to SSF server using the SSF Communication Library (II.). As the SSF server knows the corresponding receiver of the message, namely the coupled TV, the message is directly forwarded to this TV (III.). On TV client, the message will be forwarded to a consumer based on message type (IV.). For a message of type `PATH` the action of the consumer will be to open the page of the value included in the message, in this example it is `index.html` (V.).

Actions done on consuming messages of type `SELECT` are to select an item within the current opened page. In this case the value of the message is the `ID` of the item to be selected or unselected. For messages of type `INPUT`, the value is a text entered by the user in an input text field item in addition to the `ID` of that item. The action done on consuming messages of type `INPUT` is to update the item text value with the text received using the `ID` included in the message.

5. OVERALL CONCLUSION

This document described and explained the conception and the architecture of the mobile client as a second-screen-service for the GeTVivid platform. In this role, the mobile client was meant to be an additional remote control device, as well as a standalone application for the usage of the GeTVivid platform.

Several end-user tests during the project time showed that the conception of the system worked out very well for both tasks. But the test, as well as the individual experiences during this time, showed also, that among the potential end-users the acceptance of modern mobile devices like tablet computers in general is not that low as assumed in the very beginning of the project. This fact makes the TV as a “key channel” and the simultaneous usage of both devices in a synchronized way, much less crucial points for the end-user acceptance as expected. As a conclusion of these findings, the usage of the mobile client as a second independent device of the system gets more on focus for further developments. Consequently the next steps of development could be:

- Development of a GeTVivid application for mobile devices, without the restrictions of the HbbTV 1.1 standard, even if this forecloses the mirror-like synchronization between TV and mobile device. Such a mobile client, as a native app or still as a re-designed hybrid app, could fully exploit the powerful hardware resources of modern mobile devices, which will lead to better user experience and enhanced expectance of the whole GeTVivid platform.
- Porting the current Android version of the mobile client to other mobile platforms like iOS or Windows Mobile, in order to make the system reachable for a broader range of hardware devices and end-user.