Examensarbete

Design and implementation of a generic communication protocol for use in an IPTV environment

av

Mikael Nilsson

LIU-IDA/LITH-EX-G—11/023—SE

2011-08-08
Examensarbete

Design and implementation of a generic communication protocol used in an IPTV environment

av

Mikael Nilsson

LIU-IDA/LITH-EX-G—11/023—SE

2011-08-08

Handledare:
Examinator: Petru Eles
Abstract

With the presence of smartphones and tablets people are getting used to control their technical devices at home with custom applications. This thesis explores the possibility of controlling an IP-Set-Top-Box from an external device.

The focus of this report and thesis is finding a way for external devices to control IP-Set-Top-Boxes running the KreaTV platform. To open up for this possibility a service running in the KreaTV platform was designed and implemented to handle communication from external devices. A protocol for communication was structured and defined to allow devices to communicate with the platform.

The protocol includes a base structure defined for general use when communicating with applications and services in the platform. The base structure was extended and defined for communication with the Set-Top-Box application running the graphical user interface.

An Android application was used to demonstrate that the design solution presented in this report opens up for the possibility of external devices to control the IP-Set-Top-Box beyond the limitations of a normal infrared remote control. The Android applications main focus is controlling the Spotify service of the IP-Set-Top-Box.
Acknowledgements

First I would like to thank my supervisor, Andreas Wettergren for giving me the opportunity to do this thesis. I would also like to thank him for believing in me and for all the help, support and patience during the project. Without him everything would have taken a lot more time.

Big thanks to Fredrik Vinkvist for taking the time to implement the necessary parts in the Set-Top-Box GUI, he was also a big part in the creation of the protocol structure. Without him there would not be any demonstrations.

Thanks to Ali Pirouzram for taking the time to help with the implementation of the service, especially with the interface definition.

Thank to my examiner, Petru Eles, for the trust and freedom during the thesis. Big thanks for the very specific and detailed correction of the report.
# Table of Contents

1 Introduction ........................................................................................................................................ 1
   1.1 Background .................................................................................................................................. 1
      1.1.1 Overview of the previous thesis .............................................................................................. 1
   1.2 Purpose ........................................................................................................................................ 2
   1.3 Objectives ..................................................................................................................................... 3
   1.4 Limitations ................................................................................................................................... 4
      1.4.1 Report limitations .................................................................................................................... 4
   1.5 Document overview ....................................................................................................................... 4
   1.6 Glossary ....................................................................................................................................... 5

2 KreaTV basic overview ....................................................................................................................... 7
   2.1 Application platform ..................................................................................................................... 8
      2.1.1 TOI ......................................................................................................................................... 8
      2.1.2 IPC ......................................................................................................................................... 9

3 Design .................................................................................................................................................. 11
   3.1 Requirements ............................................................................................................................... 11
   3.2 Limitations ................................................................................................................................... 12
   3.3 Design proposals ........................................................................................................................... 13
      3.3.1 Design proposal 1 .................................................................................................................. 13
      3.3.2 Design proposal 2 ................................................................................................................ 16
      3.3.3 Design proposal 3 ................................................................................................................ 18
      3.3.4 Design proposal 4 ................................................................................................................ 21
   3.4 Conclusion .................................................................................................................................... 23

4 Design and implementation of the Android application ........................................................................ 25
   4.1 Overview ...................................................................................................................................... 25
   4.2 Main activity .................................................................................................................................. 26
      4.2.1 Main responsibilities .............................................................................................................. 26
      4.2.2 Client responsibilities ........................................................................................................... 27
   4.3 Spotify activity .............................................................................................................................. 28
      4.3.1 Spotify GUI .......................................................................................................................... 29
      4.3.2 User input ............................................................................................................................. 29
      4.3.3 Spotify information retrieval ............................................................................................... 31
      4.3.4 Spotify synchronisation of play queue with the STB ........................................................... 32

5 Design and implementation of the STB service .................................................................................. 33
Illustration Index
Illustration 1: Overview of the layers of the KreaTV platform.............................................................. 7
Illustration 2: Design proposal 1, sequence diagram. ............................................................................... 15
Illustration 3: Design proposal, 2 sequence diagram. ............................................................................... 17
Illustration 4: Design proposal 3, sequence diagram. ............................................................................... 20
Illustration 5: Design proposal, 4 sequence diagram. ............................................................................... 22
Illustration 6: Sequence diagram of communication between activities and STB. ................................. 28
Illustration 7: Spotify activity GUI. ........................................................................................................ 29

Index of Tables
Table 1: Service interface functions; this table contains the functions accessible in the extension service from other application or services in the platform................................................................. 35
Table 2: Observer interface functions; this table contains the functions implemented by observers to the extension service........................................................................................................................................... 36
1 Introduction

The IP-Set-Top-Box (IP-STB) is basically a simple computer connected to the TV instead of a normal computer monitor. It is providing the same basic functionality as a normal STB but the big difference is that the IP-STB is connected to an IP network from which it receives the TV steams. This difference opens up for a lot of possibilities. In this bachelor thesis one of these possibilities is explored.

1.1 Background

In a previous master thesis performed by two students at Motorola the possibility of connecting a device over WiFi to the STB was explored. The thesis was a proof of concept implementation of a phone running Android that was successively connected to the STB. It is recommended to read their report “Design and Implementation of an Android-device Based Remote Control System”[1] for better understanding of this report.

1.1.1 Overview of the previous thesis

The result of the previous thesis was an android application running on a Motorola Milestone and an application implemented in the STB. The purpose of the application in the STB was to act as a service for the phone to connect to. The Transmission Control Protocol (TCP) was chosen for the communication between the STB and the phone. The STB application was implemented as the TCP server and the Android application as the TCP client. JSON was chosen as the text based data exchange protocol.

The Android application was split up into four tabs: gestures, Electronic Program Guide (EPG), Spotify and remote control.

- The gesture tab was implemented to receive input based on gestures. Up, down, right, left, single tap and double tap. Single tap represents “OK” and double tap represents “BACK”.
• The EPG tab was implemented to display EPG data as a list filtered by time, show or channel. Long tapping a show in the list shows the description of the show. Tapping a show changed channel to that show.

• The Spotify tab was a place holder for future work.

• The remote control tab was implemented to receive input based on button presses.

All input from the gestures, remote control tabs and tapping a show from the EPG tab was sent to the STB application as key codes. All three tabs used the same key codes; the EPG tab sent the key code representing the channel broadcasting the tapped show.

The STB applications main purpose was to translate the key codes sent form the phone into key codes matching input from the Infrared (IR) remote control and then write it to a file. This file is used for automated testing emulating the IR remote control. It generates events for all key codes written to it. These events propagate through the layers of the KreaTV architecture up to the application platform layer which handles them.[1]

1.2 Purpose

The main purpose for this thesis is to find an alternative solution for the KreaTV platform to receive commands from the Android phone. The main reasons for an alternative solution are:

• Applications in the STB cannot distinguish commands sent form the Android phone from the regular IR remote control commands.

• Custom commands, for example EPG data and Spotify play lists, are not possible to implement. All commands are limited to those of a normal IR remote control.
• The STB application is directly using functionality below its own layer in the architecture.

1.3 Objectives

The objectives of this thesis are to find an alternative solution that solves the above mentioned problems and to realise an implementation that proves that it works. This implies the following:

• Present a design solution for the STB application to interact with the KreaTV platform in a way so that it can handle custom commands and distinguish them from IR remote control commands.

• Define and structure a protocol to use for communication between external devices and the KreaTV platform.

• Implement the above mentioned solution in the KreaTV platform and the Android application.

• Implement the above mentioned communication protocol.

Implementing the protocol means that all communication between the Android phone and STB shall use this protocol format. Implementing the solution for interaction between the KreaTV platform and Android application means to handle all commands being sent from the Android phone using this solution. To prove that the design can handle custom commands, a simple version of the Spotify tab shall be implemented. All data used in the Spotify tab should be provided dynamically from the STB.
1.4 Limitations
This is a bachelor degree thesis, so the work is limited to 10 weeks. The objectives may be altered or extended to fit the time plan.

1.4.1 Report limitations
This is the public, limited version of the report. The full version of the report is only available for Motorola employees. The entire chapter describing the protocol is left out and the majority of the evaluation chapter. Some specific information is left out from the chapter describing the design and implementation of the STB service.

1.5 Document overview
- Chapter 2 aims to give a basic but limited overview of the KreaTV platform for better understanding of the rest of the report.

- Chapter 3 describes the design proposals presented at Motorola as the theoretical part of this thesis.

- Chapter 4 describes the design and implementation in the Android application.

- Chapter 5 describes the design and implementation of an STB service.

- Chapter 6 describes the defined protocol.

- Chapter 7 evaluates the results of this project.
### 1.6 Glossary

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STB</td>
<td>Set-Top-Box, a device to receive signals which are translated into a content for the TV screen.</td>
</tr>
<tr>
<td>IP-STB</td>
<td>A Set-Top-Box receiving its signals over an IP network</td>
</tr>
<tr>
<td>IP</td>
<td>Internet protocol, protocol used for routing packages of data.[9]</td>
</tr>
<tr>
<td>KreaTV</td>
<td>A software platform designed for IP-Set-Top-Boxes.</td>
</tr>
<tr>
<td>TOI</td>
<td>TV Open Interface, a collection of interfaces provided by service in the KreaTV platform.</td>
</tr>
<tr>
<td>TOI/JS</td>
<td>Refers to TOI interfaces defined in Javascript</td>
</tr>
<tr>
<td>TOI/C++</td>
<td>Refers to TOI interfaces defined in C++.</td>
</tr>
<tr>
<td>TEI</td>
<td>TV Extension Interface, a collection of interfaces provided from extension services in the KreaTV platform.</td>
</tr>
<tr>
<td>TEI/JS</td>
<td>Refers to TEI interfaces defined in Javascript.</td>
</tr>
<tr>
<td>TEI/C++</td>
<td>Refers to TEI interfaces defined in C++.</td>
</tr>
<tr>
<td>IPC</td>
<td>Inter Process Communication, used for communications between processes.[2]</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol, a protocol used for transferring packages of data between hosts.[11]</td>
</tr>
<tr>
<td>IR</td>
<td>InfraRed, used for transferring data over the air.</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface, an interface that lets the user interact with a device through graphics.</td>
</tr>
<tr>
<td>JSON</td>
<td>Lightweight text based protocol designed to be human readable.[12]</td>
</tr>
<tr>
<td><strong>EPG</strong></td>
<td>Electronic Program Guide, a way to display programs available on TV.</td>
</tr>
<tr>
<td><strong>URI</strong></td>
<td>Uniform Resource Identifier, used to identify a name or resource on the internet.[13]</td>
</tr>
</tbody>
</table>
2 KreaTV basic overview

The KreaTV platform is based on a lightweight architecture designed for a TV environment, it is designed to run on very limited hardware resources. The platform is built up by layers to make it easy to reuse for different Set-Top-Boxes and, at the same time, increase flexibility and fault tolerance. At the bottom there is a hardware abstraction layer, which abstracts the hardware from the software so that the software can be reused.[7]

Illustration 1: Overview of the layers of the KreaTV platform.
### 2.1 Application platform

The KreaTV application platform consists of a number of services. The most basic and important services manage the platform to efficiently make use of the scarce hardware resources. This means starting services and monitoring that every service is in a safe state. If a service should fail, recovery is performed. This means that all services and applications are restarted to make sure that the entire platform is in a safe state. Applications are also activated, managed and monitored by a service to make sure that they are running as intended and that all applications meet the requirements to run on the platform. The applications are running protected from each other and from the rest of the platform. Since the applications are running protected from each other they are only allowed to communicate with platform services but not with other applications.

The other services not dedicated to running and manage the platform provide functionality for the applications, for example media streams, information and settings. The functionality provided to applications by the platform services is limited and TV oriented. The platform functionality can be extended by adding optional services, also called extension services. This is used to increase the available functionality beyond the existing TV oriented services, for example a Spotify service.

#### 2.1.1 TOI

Applications use the TV Open Interface (TOI) to access services provided by the platform. Platform services provide interfaces through which the service functionality is accessed. TOI is a collection of these interfaces provided by the platform services. The current version of TOI supports C++ and Javascript. Depending on which language is used, TOI/C++ and TOI/JS is referred to as the interface used by an application. There is a second collection of interfaces called TV Extension Interface (TEI); this interface is provided by extension services.
The platform is running a browser which is used by applications that require a user interface. The browser supports Javascript, so portals in the browser can gain access to the platform services through TOI/JS, which is implemented as a plug-in to the browser.

2.1.2 IPC
Inter-Process Communication (IPC) is used when calling a function defined in an interface of a service. A set of interfaces are used in combination with IPC-server and IPC-client objects to provide everything needed by services and applications to communicate in the platform.
3 Design

The idea of the STB application is that it is not limited to communicate with just an Android device. Any external device able to connect to the STB using TCP should be able to control the STB in the same way as an Android phone. The external device just has to send a set of predefined JSON messages to the STB application, abstracting the external device from the rest of the STB.

3.1 Requirements

The first task to solve is to redesign how the STB application interacts with the rest of the applications and services of the KreaTV platform. One of the main problems to solve was that the STB Graphical User Interface (GUI) could not distinguish if commands were sent from an external device or the IR remote control. The following example illustrates why this was considered a problem.

Example: Someone is sitting at home watching his favourite TV show. Another family member, a bored little brother is sitting in school with an Android phone connected to the STB via 3G. The little brother switches the channel for his own amusement making it impossible to watch the TV show.

The STB GUI needs to be able to distinguish commands sent from an external device so that options for blocking them can be provided to the user. Another problem with the way commands were forwarded by the STB application was that it wrote key codes directly to layers below itself, without using the KreaTV application platform. The new design had to solve this problem and meet a set of other requirements. Here is a full list of the requirements for the new design:

1. To keep the original idea of the STB application holding TCP connections to external devices and communicating through using the JSON text protocol.
2. The STB application should use the KreaTV application platform, not directly use functionality in a lower level of the architecture.

3. The STB GUI should know if the command was sent from the IR remote control or an external device.

4. Not limiting the design to just that the STB GUI being able to distinguish the commands sent from external application from those coming from the IR remote control; other applications should also be able to make this distinction.

5. The STB application should follow the design rules of the KreaTV application platform.

6. The STB application should have a limited impact on the rest of the platform.

7. The STB application should be able to access all data needed for the Android application.

8. The STB application should be able to send status updating events to external devices containing, for example, channel and volume switches.

### 3.2 Limitations

One aspect that limits the current options for the design was that the platform services are not holding all information. Larger data structures, for example EPG data and the play queue for Spotify, exist in the STB GUI application running in the browser. Parts of the data still exist in services but important parts are only available inside the STB GUI application. The requirement that “The STB application should be able to access all data needed for the Android application.” is affected by this limitation, since the Android application is using some of the data stored exclusively in the STB GUI application. A request was that the design ideas could also suggest designs for integrating the STB application disregarding some
requirements and the current limitations of the platform. This is because the platform is constantly evolving and sometime in the future today's limitations may not exist and another more elegant solution can be implemented.

3.3 Design proposals

With requirements and limitations in mind four proposals were considered. Only two of them are possible to implement in the current version of the platform. The solutions range from fully dependent of the STB GUI to only depend of the platform in the future.

3.3.1 Design proposal 1

The first solution is designed to give the STB GUI full control over communication with external devices. The motivation for this solution was that it makes it possible to give the user an option to let the IR remote control to be the highest priority for the STB. This also opens up for the possibility for the STB GUI to interpret the commands as something else than what they were assigned for. That means it is possible for the external application to control things it was not intended for, without actually implementing that particular functionality. For example, if a game is running, the STB GUI could decide that volume up button key presses from the external device are controlling something in the game instead of changing the volume.

To accomplish giving the STB GUI full control there has to be communication between the STB application and STB GUI. All data to and from external devices would go through the STB GUI, abstracting most of the platform services from the STB application. To follow the platform rules the STB application cannot run as an platform application since the STB GUI is running as an application in the browser. This means that the STB application has to be changed into a service. To limit the impact on the rest of the platform, the solution is to run it as an extension service. To adapt the design for all the requirements, the following is required by the STB extension service and STB GUI:
STB service:

- Running the STB application as an extension service, to limit impact on the rest of the platform and to follow rules of the platform design regarding communication between applications.

- The STB extension service should be implemented as an “observable” and let all observers receive commands sent from external devices. Thus not just the STB GUI is limited to viewing incoming commands.

- The STB extension service must forward all communication between the STB GUI and external devices.

GUI:

- The STB GUI must be implemented as observer to the STB extension service to get access to all commands sent from external devices.

- The STB GUI has to extend its responsibilities to include updating external devices with status events and information requests.

- The STB GUI must respond to the commands sent by external applications.

- The STB GUI must be able to format all data being sent to external devices to the JSON format, so that external devices can use it.

- The STB GUI must be able to parse all data sent from external devices.
The figure illustrates the communication between different applications and services involved in a Channel change.

The positive sides of this design alternative are:

+ The design meets all requirements.

+ The STB GUI has control over commands sent from external devices.

The negative sides of this design alternative are:

- External devices depend on the STB GUI to function. This is negative because the STB GUI is more likely to be switched out than a platform service.

- If information requested by an external application does not exist in the STB GUI application, internal communication in the platform increases when the STB GUI retrieves the information for the external device instead of the STB extension service receiving it directly.
3.3.2 Design proposal 2
The second solution is designed to give the STB GUI partial control over communication between the platform and external applications. The main idea of the design is to improve the weaknesses in the first solution but keep the strengths. The major difference compared to the first solution is that communication to external devices should not exclusively go through the STB GUI. Events and information available from the platform services can be retrieved directly from the corresponding service. Only information that is just stored in the STB GUI application is retrieved from the STB GUI. Commands sent from external devices are still forwarded to the STB GUI to keep the extra possibilities it opens up for. To adapt the design for all the requirements the following is required by the STB extension service and STB GUI:

STB service:

- Running the STB application as an extension service to limit the impact on the rest of the platform and to follow rules of the platform design regarding communication between applications.

- The STB extension service must be able to parse incoming messages from external devices. This is so that the service is able to decide if the information can be retrieved from a platform service or should be forwarded to the STB GUI.

- The STB extension service should be implemented as an “observable” and let all observers receive commands sent from external devices. This is so that not just the STB GUI is limited to viewing incoming commands.

- The STB extension service must be able to retrieve information from platform services and send the information in JSON format to external devices.

- The STB extension service must subscribe as an observer to events from platform services relevant to the external application. When events fire they have to be forwarded to the external application.
• The STB extension service must forward all commands sent from external devices to the STB GUI.

GUI:
• The STB GUI must be implemented as observer to the STB extension service to get access to all commands sent from external devices.

• The STB GUI must respond to the commands sent by external applications.

• The STB GUI must be able to parse data sent from external devices.

• The STB GUI must be able to format all data being sent to external devices to the JSON format so that external devices can use it.

Illustration 3: Design proposal, 2 sequence diagram.
The figure illustrates the communication between different applications and services involved in a Channel change.

The positive sides of this design alternative are:

+ The design meets all requirements.

+ The STB GUI has control over commands sent from external devices.
The negative sides of this design alternative are:

- External applications are dependent of the STB GUI for commands sent to the STB to work.

- Extra parsing in the STB extension service of data that is going to be forwarded to the STB GUI. When the data reaches the STB GUI it needs to be parsed again a second time.

### 3.3.3 Design proposal 3

The third solution is designed for the future. The idea of this solution is to remove the huge dependency of the STB GUI but still meet the requirements and keep the strengths of previous solutions. To meet the requirement “The STB GUI should know if the command was sent from the IR remote control or an external device.” communication to the STB GUI is required. This solution aims to limit this communication, so that if the STB GUI is changed in the future and the functionality needed in the STB GUI is not implemented, minimal changes are required in the STB extension service to keep it functioning.

The STB extension service would retrieve all information and event notifications from the platform services, similarly to the second solution. The main difference from previous solutions is that commands sent from external devices are parsed and forwarded by the STB extension service to the corresponding platform service.

To still keep the STB GUI involved, after a message from an external device is parsed and identified as a command, it is forwarded to the STB GUI. This lets the STB GUI process the command and sends back a response that lets the STB extension service know if the command is approved to be forwarded to the corresponding platform service. This leaves the STB GUI open to all the possibilities as in the previous solutions. If the command is going to be interpreted by the STB GUI as something that it was not intended for, the STB GUI just has to respond to the STB extension service that the command is disapproved to forward.
If the STB GUI is replaced and the functionality to respond to commands is not implemented, all that has to be changed in the STB extension service is to disable that commands are being forwarded for approval of the STB GUI before sent to the corresponding platform service. Another solution that does not require the STB extension service to change would be to implement a simple application that subscribes as an observer to the STB extension service responding with an approval to all commands received. To adapt the design for all the requirements, the following is required by the STB extension service and STB GUI:

**STB service:**

- Running the STB application as an extension service to limit the impact on the rest of the platform and to follow rules of the platform design regarding communication between applications.

- The STB extension service must be able to parse incoming messages from external devices.

- The STB extension service should be implemented as an “observable” and let all observers receive commands sent from external devices. This is so that not just the STB GUI is limited to viewing incoming commands.

- The STB extension service must be able to retrieve information from platform services and send the information in JSON format to external devices.

- The STB extension service must subscribe as an observer to events from platform services relevant to the external application. When events fire, they have to be forwarded to the external application.

- The STB extension service must forward all commands sent from external devices to the STB GUI for approval then forward them to the corresponding platform service.
GUI:

- The STB GUI must be implemented as observer to the STB extension service to get access to all commands sent from external devices.

- The STB GUI must respond to the commands sent by external applications.

- The STB GUI must be able to parse data sent from external devices.

Illustration 4: Design proposal 3, sequence diagram.

The figure illustrates the communication between different applications and services involved in a Channel change.

The positive sides of this design alternative are:

+ The STB GUI has control over commands sent from external devices.
The negative sides of this design alternative are:

- External applications are dependent on the STB GUI for commands sent to the STB to work. This just needs minimal changes to work around, if the STB GUI is replaced and functionality is left out in the new GUI.

- The design does not work around current limitations of the platform, causing it to fail to meet the requirement “The STB application should be able to access to all data needed for the Android application.”.

Because the design of this solution does not work around current limitations of the platform, it is not possible to implement it in the current version. For this solution to work in the future, all information has to be stored and managed in platform services; no information or parts of information can exclusively exist in the STB GUI application.

3.3.4 Design proposal 4

The fourth solution is, like the third one, designed for the future. The idea behind this solution is no dependency on other applications and minimal impact on the platform. With no communication to other applications, the STB application is not required to run as a service. The STB application would only communicate with platform services. Nothing is required from the STB GUI but the following is required by the STB application for this solution to work:

**STB application:**

- Running the STB application as a platform application.

- The STB application must be able to parse incoming messages from external devices.

- The STB extension service must be able to retrieve information from platform services and send the information in JSON format to external devices.
The STB extension service must subscribe as an observer to events from platform services relevant to the external application. When events fire, they have to be forwarded to the external application.

The figure illustrates the communication between different applications and services involved in a Channel change.

The positive sides of this design alternative are:

+ Minimal impact on the rest of the platform, because it is running as an application.

+ Only dependent of the platform to function; this leaves it unaffected from changes in other applications.

The negative sides of this design alternative are:

- The design fails to meet the requirements “The STB GUI should know if the command was sent from the IR remote control or an external device.” and “Not limiting the design to just that the STB GUI being able to of distinguish the commands sent from external application from those coming from the IR remote control; other applications should also be able to make this distinction.”
The design does not work around current limitations of the platform, causing it to fail to meet the requirement “The STB application should be able to access to all data needed for the Android application.”.

The idea is that, in the future, the STB GUI does not need to know if a command was sent from the IR remote control or from an external device. The state of all settings and data would be in the platform services instead of partly existing in the STB GUI application. The setting for disabling input from an external devices would be stored in a service making it accessible for all applications. When the state of all data exists in platform services it reduces the limitation for new applications running on top of the platform.

### 3.4 Conclusion

All solutions were presented in an internal meeting at Motorola. The internal conclusion was that the first suggestion for the design was the one that was going to be implemented. The STB application should run as an extension service fully dependent of the STB GUI. Exceptions could be made in the future for some specific data that could be retrieved directly from platform services by the STB extension service, but the current implementation should rely completely on the STB GUI.
4 Design and implementation of the Android application

4.1 Overview

The Android application is split up into four tabs, each one runs its own activity and then there is a main activity to manage the tabs. This report will focus on the Main and Spotify activity. More information about the other activities and their tabs is available in the report of the previous thesis.

[1] This is a list to give a quick and simple overview of the activities and tabs of the application:

- Main, is the tab activity initializing the tabs and managing them. This activity also initializes the TCP client which is a separate class holding the connection.

- Gestures, receives input based on gestures: Up, down, right, left, single tap and double tap. Single tap represents “OK” and double tap represents “BACK”.

- EPG, displays EPG data as a list, filtered by time, show or channel. Long tapping a show in the list shows the description of the show. Tapping a show change channel to that show.

- Spotify, displays a play queue and lets the user play, remove, pause, stop and switch tracks.

- Remote control, displays a remote control and receives input based on button presses.
4.2 Main activity

The Main activity is the one that manages all the other activities and handles communication both inside the application and the TCP connection to the STB. The code of the Main activity is split up into two modules, Main and Client. Main is managing the other activities and instantiates and holds a Client object which is responsible for the TCP connection and the parsing and formatting of data. Other activities access the functionality in Client from functions in Main. Main is responsible for observing the Client object and decides when it is time to reconnect when the connection is lost.

4.2.1 Main responsibilities

The following is the responsibilities of Main:

- Creating the tab layout and the activities running in each tab.
- Creating the options menu available everywhere in the application.
- Handling input from the physical keyboard of the phone. This makes the keyboard usable in the entire application.
- Providing functions for the other activities to access functionality in Client.
- Providing functions to show notifications. The notifications are shown independent of the current tab being used.
- Keeping track of the status of Client and deciding when to reconnect if the connection is lost.

Main is responsible for displaying notifications independent of the current tab being used. The notifications activate from events received from the STB GUI. Events are received no matter if the change came from the phone or IR remote control. There are five different notifications:
• **Volume changed**: activates when the volume is changed in the STB. The notification displays a volume bar with the current volume.

• **Channel changed**: activates when the channel is changed. The notification displays a logo for the new channel.

• **Spotify track changed**: activates when a new track starts to play. The notification displays a text with status, title of the track, and name of the artist.

• **Spotify track paused**: activates when a track is paused. The notification displays a text with status, title of the track, and name of the artist.

• **Spotify track stopped**: activates when a track is stopped. The notification displays a text with status, title of the track, and name of the artist.

The notifications can be turned off and on from settings in the options menu. The Spotify track notifications share the same setting; volume and channel notifications have their own setting.

### 4.2.2 Client responsibilities

The following is the responsibilities of Client:

• Setting up the TCP client socket.

• Formatting all data that is being sent to the STB according to the JSON format.

• Parsing all incoming data and forwarding it to the corresponding activity through functions in Main.
• Initializing and managing a separate thread to handle input from the TCP socket.

Inside the client class there is a private class running in a separate thread, which is managed by the Client object. This is to make it possible to communicate with the Android application in an asynchronous manner. Client is responsible for formatting all outgoing data into JSON and write it to the output buffer. The second thread is responsible for parsing all input data written to the input buffer and call functions in Main depending on the input data. In the Android operating system only the main thread of an application can change the graphic.[6] The second thread has to use a handler to communicate all data received to the main thread. It also has to call functions in the main thread for it to change the graphic.

Illustration 6: Sequence diagram of communication between activities and STB.
The figure illustrates the Spotify activity sending a request to play a track.

4.3 Spotify activity

The idea of the Spotify activity is to let the user control the Spotify service in the STB from the phone. The current version of the Spotify activity lets the user manage the play queue.
4.3.1 Spotify GUI
The GUI of the Spotify tab consists of the play queue represented by a list of tracks, 4 buttons and a notification field.

Illustration 7: Spotify activity GUI.
Shows the layout of the Spotify tab in the Android application.

4.3.2 User input
The play queue lets the user scroll through the list of tracks. For each track name, the duration, artist and album is displayed. The buttons are used to receive input based on the current track. The notification field is used to display status and information about the current track. There are seven ways for the Spotify activity to receive input from the user:

- Scrolling in the play queue: When a play queue is available, a request for 20 tracks of the play queue is sent to the STB. Having 20 tracks lets the user scroll down without the need for a new request right away because approximately 7 tracks are visible on
the screen. When the user starts to scroll down and only 10 tracks remain off screen
the following 20 tracks of the play queue are requested. This way tracks are added
dynamically to the play queue as the user scrolls down.

• Clicking a track in the play queue sends a play command for that track to the STB.

• Long pressing a track in the play queue gives the user a dialogue with the option to
remove that track from the play queue. If the user decides to remove the track, a
remove track command is sent to the STB.

• PLAY/PAUSE button: if there is a track currently paused or stopped, the button will
be labelled PLAY and clicking the button results in a play command for the current
track. If the track was paused it is resumed, if it was stopped it starts from the
beginning. If there is a track currently playing, the button will be labelled PAUSE and
clicking the button will result in a pause command for the current track.

• STOP button: if there is a track currently playing or paused, clicking the button results
in a stop command for the current track.

• PREV button: if there is a track currently playing, paused or stopped, clicking the
button results in a play command for the previous track in the play queue to the current
track.

• NEXT button: if there is a track currently playing, paused or stopped, clicking the
button results in a play command for the next track in the play queue to the current
track.
4.3.3 Spotify information retrieval

The Spotify activity receives information and events asynchronously from the STB when something changes in the STB GUI Spotify application concerning the play queue. The activity is updated with the following information and events:

- Track changed: notifies the activity that the focus has changed to a new track.

- Track playing: notifies the activity that the current track has started to play. The notification field displays a scrolling text with the new track and status “Now playing”.

- Track stopped: notifies the activity that the currently playing track is stopped. The notification field shows a scrolling text with the track and status “Stopped”.

- Track paused: notifies the activity that the currently playing track is paused. The notification field shows a scrolling text with the track and status “Stopped”.

- Track added: notifies the activity that a track has been added to the play queue. The new track is added to the play queue displayed in the activity.

- Track removed: notifies the activity that a track has been removed from the play queue. The track is then removed from the play queue in the activity.

- Play queue: is a response to the tracks requested when the user scrolls in the play queue. It contains the part of the play queue requested and is added to the play queue in the activity.

- New play queue: notifies the activity that there is a new play queue.
4.3.4 Spotify synchronisation of play queue with the STB

When the application starts, a status request is sent to the STB Spotify application. If the application is running, a request for the play queue and a request to register to events are sent. A request is also sent for the current playing track and its status. The STB Spotify application automatically sends a status message notifying that it is running on start up in case the Android application was started first. When the play queue is changed in the STB Spotify application the Spotify activity is notified, the play queue is cleared, and a request for the new play queue is sent. When tracks are removed or added in the STB Spotify application this is handled with events informing which track has been added or removed. This avoids a re retrieval of the entire play queue if just minor changes are made.
5 Design and implementation of the STB service

The purpose of the STB extension service is to provide functionality for an external device to connect to and communicate with the STB. The external device provides the GUI and the service is just providing the possibility of interaction with the STB.

5.1 Goals and requirements

The goals for the STB extension service was for it to meet all requirements of the design solution selected. It runs as an observable extension service that accepts TCP connections from external devices. All messages sent to the service from an external device are forwarded to its observers that handle the message. The design of the solution that was selected aimed to give the STB GUI full control over the communication to and from external devices. To accomplish this the following was required from the STB application:

• The STB application is implemented as an extension service to limit the impact on the rest of the platform and to follow rules of the platform design regarding communication between applications.

• The STB extension service is implemented as an “observable” and lets all observers receive commands sent from external devices. This is so that not just the STB GUI is limited to viewing incoming commands.

• The STB extension service forwards all communication between the STB GUI and external devices.

Considering that the service is only forwarding messages between applications its implemented so that it does not need any information about the applications or the text protocol used to
communicate between them. This makes the service usable for other applications than the Android application and STB GUI.

5.2 Implementation

The code for the service is split up into two modules originating from the previous thesis work.[1] One of them taking care of all the tasks of the service and the other one is just there to keep the functionality of parsing JSON data.

5.2.1 Main module

The main module is the event handler and it has the following responsibilities:

- Initializing as an extension service with all the required functions for a service to run.

- Opening a TCP socket and listening for connections from clients.

- Reading data from the TCP input buffer and forwarding it to the observers through IPC.

- Writing messages to the TCP output buffer.

- Managing observers.

- Handling disconnects of the Android phone.

A task for this module is to handle a disconnection of the Android phone. An exception had to be introduced with the goal of making the service completely separated from the applications communicating through it. When the service discovers that the Android phone has disconnected, it sends out messages to the observers notifying that the Android phone
unregisters from all events provided. This is so that the STB GUI does not keep sending out
events that have no receiver. This avoids waste of hardware resources.

5.2.2 Parser
The other module of the service is the parser. It is used by the service to format messages into
the JSON format. This is needed to handle a disconnect of the Android phone. Another reason
for this module to exist is that it was decided that in the future some specific information
could be directly retrieved from platform services instead of going through the STB GUI. For
the service to provide such functionality it need to be able to parse JSON data.

5.3 Interfaces
Two interfaces are defined to enable applications to use the functionality of the service. One
of them provided everything required for applications to forward messages to external
devices. The other interface defined provided everything required for the service to notify its
observers. The interfaces are defined for both Javascript and C++ and a plug-in for the
browser is created so that the STB GUI can access the functions.

5.3.1 Service interface
The service interfaces makes it possible for the extension service to provide access for
applications or services in the STB to the following functions:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RegisterObserver()</td>
<td>Used by observers to register for messages from external devices.</td>
</tr>
<tr>
<td>ReleaseObserver()</td>
<td>Used by an observer to unregister form messages.</td>
</tr>
<tr>
<td>SendMessage()</td>
<td>Used by applications or services to send a message to an external device.</td>
</tr>
</tbody>
</table>

*Table 1: Service interface functions; this table contains the functions accessible in the extension service from other application or services in the platform.*
5.3.2 Observer interface

The observer interface makes it possible for the extension service to forward messages sent from external devices to the observers. The following function is implemented by observers:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnMessage()</td>
<td>Used by the extension service to deliver a message from an external device to the observer.</td>
</tr>
</tbody>
</table>

*Table 2: Observer interface functions; this table contains the functions implemented by observers to the extension service.*
6 Evaluation

In the previous thesis[1] it was proven that the STB can be connected with an Android phone and that the phone can control the STB. The goal with this project was to prove that the Android phone or any other external device can do more than just controlling the STB in the same way as a normal IR remote control. The goal was also to prove that the Android phone can be updated dynamically with information from the STB.

6.1 Results

The result of this project is an Android application capable of controlling the STB in the same way as an IR remote control and with custom commands based on the applications running in the platform. To prove that this works, the Spotify tab was implemented to show the play queue of the Spotify application in the STB. The queue is updated dynamically as it changes and the Android application is able to perform custom commands not available from the IR remote control.

The solution was an extension service in the STB accepting connections from external devices. The service is just responsible for forwarding information from the connected device to applications running in the platform interested in the communication. This service can be used to communicate between any external device and internal application. It is not limited to an Android application and the STB GUI.

A protocol is defined for communication between an external device and internal platform applications. The base structure of the protocol is designed with the goal to work between any external application and internal application in STB.
Bibliography


[3] Motorola Wiki (internal documentation)

[4] Motorola course material (internal documentation)


[9] Internet Protocol – RFC791


På svenska

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.
Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

In English

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.
The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.
According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.
For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/

© [Mikael Nilsson]