Documentation Yet Another Accessible Keyboard (YAAK)

David Thaller, Sebastian Huber

May 28, 2012

1 Preface

Due to the fact that we only had 24 hours to code and to create the documentation, we didn't manage to document all features, but this documentation should give quite a good overview of what we have done!

2 Task

We chose the task, "Scannable Matrix for Keyboard and Commanding on Android Platform". The goal was to create an app which can display a grid of buttons (keyboard) which have a scanning interface. The layout of the buttons should be flexible and should be stored in a layout file, we used a XML structure for that.

3 Design

Figure 1 shows the components we developed during our 24 hour coding session. All outer green components got implemented. The main component is the keyboard, which stores the buttons, and its corresponding properties like the action string that should be sent to external applications when the button gets pressed. The layout of the keyboard and the functions of the buttons are loaded from a xml file. The structure of this file is explained in more detail in section 4.1 The Painter interface is used to create different Painter classes to support different drawing patterns which are described in the following list in detail:

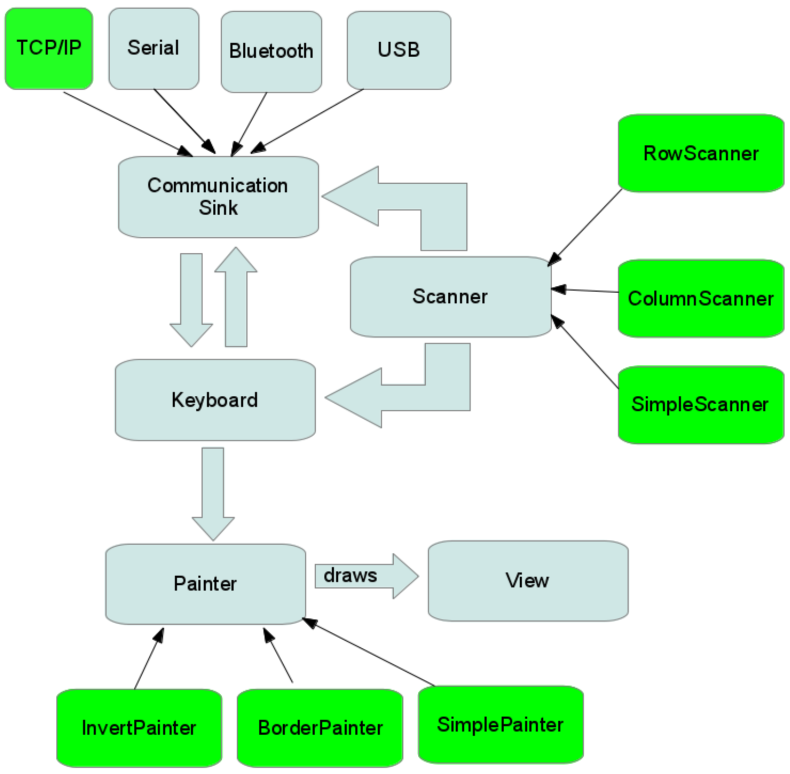

Figure 1: Block diagram of the design

- **SimplePainter:** Draws the buttons like it is encoded in the xml file –> see xml documentation for details
- **InvertPainter:** Draws the buttons foreground color, icon and background color and inverts them when the button gets selected by the scanner
- **BorderPainter:** Draws the buttons foreground color, icon and background color and draws a border with the specified color of the xml file around a selected button

The abstract Scanner class represents the interface to different Scanning methods. In our current implementation there are three different scanning strategies available:

- **RowScanner:**

- **ColumnScanner:** The scanner selects one Column after the other, beginning with the first one (left). When a trigger gets activated (touch event or TCP/IP Event) the scanner starts to select each button after the other in the selected column from top to the bottom.
- **SingleScanner:** The SingleScanner starts at the top left position and steps to the left in every scanning step until it reaches the end of the line. From there it jumps down one row and starts from the first column again. This procedure continues until the bottom right button gets selected. After this one the scanner starts again at the upper left corner.

The scanning algorithms RowScanner and ColumnScanner also support TimeOutScanning. This means that if a row or column got selected by the user and the single buttons are stepped through more than n times (specified in the xml file) the scanning algorithm goes back to the rowscanning or columnscanning. This is useful if a user chose the wrong row/column to go back without triggering an unwanted action.

The communication module is used to create a connection to allow other applications to use the keyboard. One scenario would be that the tablet PC is mounted on a wheelchair and the scanning keyboard to control the television is running on it. The user also uses the AsTeRICS (www.asterics.eu) mobile platform in his/her daily life. These two systems communicate with each other over the communication layer (USB cable, WLAN, LAN, Serial, Bluetooth etc.) to trigger button presses at the keyboard with sensors plugged into the AsTeRICS system and to control actuators with AsTeRICS. In the first protopty we only implemented a TCP Server which is explained in more detail in section 4.2.

One component which is missing on figure 1 (due to time problems ;)) is the Trigger component. This component is used to create triggerevents from the sensors of the phone. At the moment there is only one Trigger implemented which reacts if a touchevent on the display occurs. But it would be very easy to implement additional ones which behave totally different like

- react on the event when the user ends a touch
- use the accelerometer to generate triggers
- change the time how long a user must press the touchscreen to generate an event (this would be very useful for people with spasm who often touch the screen unintentional for a short time).

The modular design of the whole application should be very useful for further implementation, because nearly every part can be easily substituted just by implementing the predefined interfaces and add a new keyword to the xml file. We chose this development model, because we think it there are lots of different scenarios where the keyboard can be used and therefore it should be very easy to enhance the functionalities and adapt the interface to the needs of all kind of different users.

4 Implementation

4.1 XML Structure

The xml structure defines all properties of the keyboard, like the scanner, painter, grid size that should be used. It also stores the information of all buttons like if it has an icon or a text or both and the action that should be sent to external applications when a button gets pressed. In the Appendix you can find an example xml layout for a 3x2 keyboard to control a television.

When the application starts it looks in the folder /sdcard/yaak/ for a xml file called "default.xml". If it does not exist the application won't start.

In the next subsections the xml items will be described.

4.1.1 keyboard element

This is the root element of the xml file.

Attributes:

1. **rows:** Datatype: int; Defines how many rows the keyboard has
2. **cols:** Datatype: int; Defines how many columns the keyboard has
3. **bgcolor:** Datatype: color string (#RRGGBB);; Defines the background color of the keyboard.

4.1.2 painter element

Specifies the painter which should be used with this keyboard.

Attributes:

1. **method:** Datatype: string; Defines which Painter should be used to draw the buttons. Possible entries are: simple, invert, border.
2. **bordercolor:** Datatype: color string (#RRGGBB); Defines which color should be used to draw the border of a button. If this property is not set, no border will be drawn.

3. **fontcolor:** Datatype: color string (#RRGGBB); Defines the fontcolor of the text of a button. Gets overridden if a fontcolor attribute is set at the button element itself.

4. **bgcolor:** Datatype: color string (#RRGGBB); Defines the backgroundcolor of a button. Gets overridden if a bgcolor attribute is set at the button element itself.

4.1.3 scanner element

Specifies the scanner which should be used with this keyboard.
Attributes:

1. **method:** Datatype: string; Defines which Scanner should be used. Possible entries are: single, row, column.

2. **scantime:** Datatype: int; Defines the scantime in ms. This is the time intervall the scanner stays at one button and a user can actually trigger it.

3. **repeattime:** Datatype: int; Defines the repeattime in ms. This is the time intervall the scanner stays at a pressed button to give the user the chance to activate it a few times after each other (example given to increase the volume of the tv multiple times). If the user does not trigger an event during this period the scanning process continues its work.

4. **timeoutrounds:** Datatype: int; Defines how many rounds the scanners rowscanner and columnscanner should cycle through the columns/rows before the go back to row or columnsscanning. If timeoutscanning should be disabled the value should be -1 or the attribute should not be set.

4.1.4 tcp element

The tcp element stores information about the TCP Server. More information about the protocol used can be found in section 4.2.
Attributes:

1. **enable:** Datatype: int; Defines if the TCP/IP Server should be activated. Possible entries are: 1: active, 0: disabled

2. **port:** Datatype: int; Defines the port the Server should use.

4.1.5 rows element

The rows element stores <row> elements which contain the buttons.

4.1.6 row element

The row element contains the <button> elements of each row.

4.1.7 button element

The <button> element stores all information of one button.

Child Elements:

1. **icon:** Datatype: string; Relative path to the icon image file from. The position must be relative to the folder /sdcard/yaak/. The Icon tag can also be empty. Then no Icon is used for the button.
2. **text:** Datatype: string; The text of the label of a button. If empty no text will be displayed.
3. **action:** Datatype: string; The action string defines the command that will be executed when the button gets pressed. Possible values are:
   - @load:file.xml –> loads a new keyboard
   - @quit –> quits the application
   - @intent:appname –> start the application appname on the android device
   - some string –> sends the given string to all active communication nodes like TCP.

Attributes:

1. **fontcolor:** Datatype: color string (#RRGGBB); Defines the fontcolor of the text of a button.
2. **bgcolor:** Datatype: color string (#RRGGBB); Defines the backgroundcolor of a button.

4.2 TCP/IP Protocol

The TCP/IP Interface can be used to connect the keyboard with external applications like AsTeRICS (www.asterics.eu) to integrate a huge amount of different actuators to trigger an event for the scanning engine. It supports multiclient support, which allows that concurrent connection of multiple applications to the keyboard. The actual protocol is very simple at the moment, but can be extended very easily to support additional commands.

The server can either receive data or send data.

4.2.1 Receive Data

There are two commands at the moment that will do something.

1. "trigger". This will raise an event in the scanning engine.
2. "quit". This will close the TCP/IP connection with the connected client.

4.2.2 Send Data

Whenever a button of the keyboards gets pressed, and the action of this button is no internal one like quit the application, load a different keyboard or start another application, the action string specified in the xml file will be sent to all connected clients, to allow them to react to the event.

4.3 Screenshots

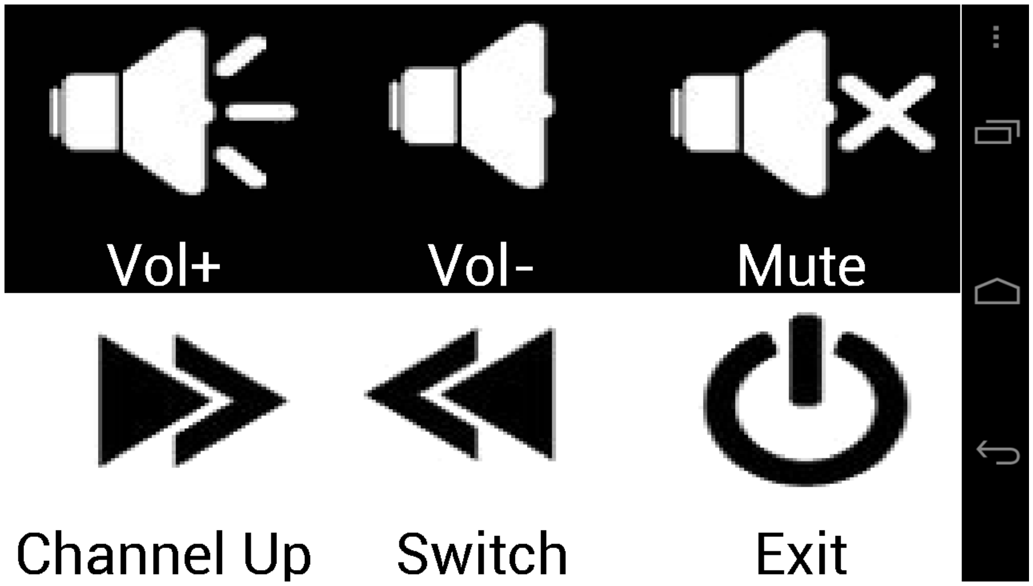

Figure 2: Screenshot of the RowScanner with the InvertedPainter

5 Suggestions for further development/ Further steps

- The already implemented painters could be enhanced by graphically skilled persons to make them look better, and new painting methods could be implemented
- Due to the fact that we had no tablet while the development, it should be tested also on devices with bigger screens
- Additional communication interfaces should be implemented like bluetooth or a serial cable over the ADK to talk to embedded systems and microcontrollers like the arduino which is already integrated into the AsTeRICS project
- the label of a button could be read out with a TextToSpeech system when it gets selected

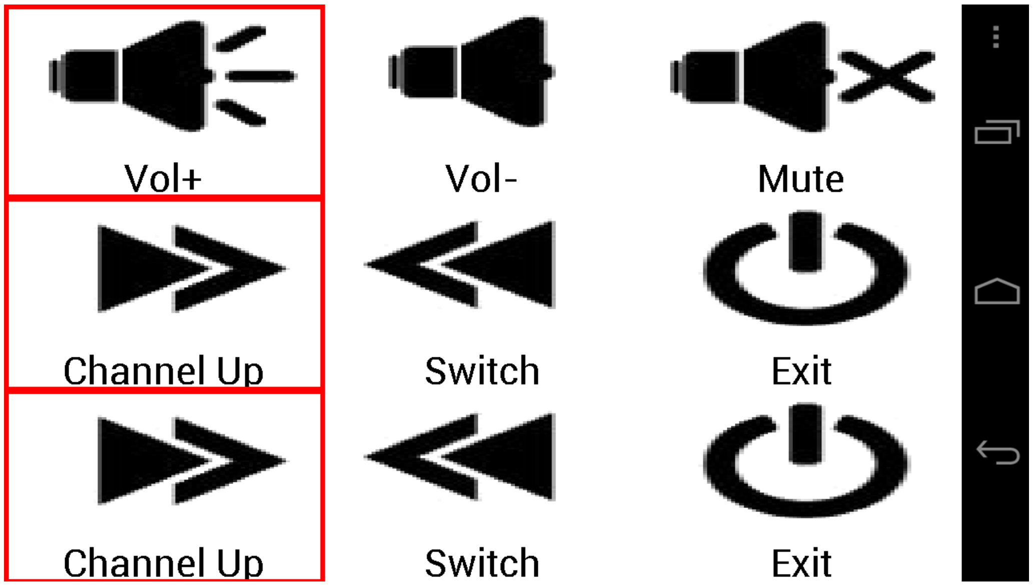

Figure 3: Screenshot of the ColumnScanner with the BorderPainter

6 Conclusion

We worked hard, had fun and slept nothing, cheers ;)

7 Appendix

7.1 Example xml keyboard

The following example shows a 3x2 keyboard used to control a tv.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<keyboard rows="2" cols="3" bgcolor="#FFFFFF">
        <painter method="invert" fontcolor="#000000" bgcolor="#FFFFFF"
            bordercolor="#FFFF00" bordersize="2.0f"/>
    <scanner method="row" timeoutrounds="3" scantime="1000" repeattime="1000"/>
    <tcp enable="1" port="4444"/>
        <rows>
                <row>
                        <button bgColor="#FF00FF">
                                <icon>volup.png</icon>
                                <text>Vol+</text>
                                <action>vol+</action>
                        </button>
                        <button>
                                <icon>voldown.png</icon>
                                <text>Vol-</text>
                                <action>vol-</action>
                        </button>
                        <button>
                                <icon>mute.png</icon>
                                <text>Mute</text>
                                <action>mute</action>
                        </button>
                </row>
                <row>
                        <button>
                                <icon>chup.png</icon>
                                <text>Channel Up</text>
                                <action>ch++</action>
                        </button>
                        <button>
                                <text>Switch</text>
                                <icon>chdown.png</icon>
                                <action>@load:keyboard1.xml</action>
                        </button>
                        <button>
                                <icon>power.png</icon>
                                <text>Exit</text>
                                <action>@quit</action>
                        </button>

                </row>
        </rows>
</keyboard>
```