Implementierung von Kamera, Lautsprecher und Mikrofon

In der ursprünglichen Planung sollten diese drei Aspekte von einer IP-Kamera mit eingebauten Mikrofon und Lautsprecher realisiert werden. Auf dieser Grundlage haben wir angefangen einen Ansatz zur Realisierung zu planen. Im folgenden Text wird zuerst die Videoübertragung angeschaut und danach die Audioübertragung. Bei beiden Teilen setzen sich dabei aus den ersten Umsetzungen und der finalen Lösung zusammen.

**Videoübertragung**

Das Kamerabild sollte dabei als Livestream eingebunden werden. Folgender Code zeigt das Einbinden des Streams in die Sitemap:

Video url="http://96.10.1.168/mjpg/1/video.mjpg" encoding="mjpeg" label="Kamera" visibility=[camSwitch==ON]

Das Video-Tag gibt an das es sich um ein Video, bzw. in unserem Fall einen Stream, handelt. Der Parameter „url“ den Pfad zur Quelle, in diesem Fall einen öffentliche IP-Kamera. Der Parameter „endcoding“ gibt an in welchem Format der Stream vorliegt, mjpeg ist dabei empfohlen von OpenHab. „label“ stellt den angezeigten Text in der Sitemap da. „visibility“ sagt aus ob der Stream angezeigt werden soll oder nicht. In diesem Fall wird dies von dem Switch „camSwitch“ abhängig gemacht. Ist der Switch eingeschaltet, so wird der Stream gezeigt.

Im Browser funktioniert dies sehr gut und der Stream wird flüssig dargestellt. Auf dem Smartphone hingegen kann es zu Performance-Problemen kommen.

Ein weiterer gravierenderer Nachteil ist, dass wenn der Stream über den Switch ausgeblendet und wieder eingeblendet wird, nicht der Stream ab- und wieder aufgebaut wird, sondern komplett neu angefordert wird, ohne abgebaut zu werden. Dies führt dazu, dass das Raspberry Pi schnell überlastet werden kann.

Als alternative Lösung haben wir einen USB-Kamera verwendet, die alle fünf Sekunden ein Bild erstellt. Dazu wurde ein Java-Programm erstellt, das prüft ob eine Datei vorhanden ist.

private boolean isCamSwitchOn**()** **{**

File file **=** **new** File**(**"/opt/openhab/webapps/cameras/kameraAn"**);**

**if** **(**file**.**exists**())**

**{**

**return** **true;**

**}**

**else** **return** **false;**

**}**

Wenn diese Datei vorhanden ist wird ein Bild erstellt und gespeichert.

Runtime rt **=** Runtime**.**getRuntime**();**

**try**

**{**

Process pr **=** rt**.**exec**(**"fswebcam -r 1280x720 /opt/openhab/webapps/cameras/image"**);**

pr**.**waitFor**();**

**}**

**catch** **(**Exception e**)**

**{**

e**.**printStackTrace**();**

**}**

Die zu prüfende Datei wird von OpenHab mittels executeCommandLine in den Rules erzeugt und kann so auch wieder entfernt werden.

rule KameraAN

when

Item camSwitch received command ON

then

executeCommandLine**(**"touch /opt/openhab/webapps/cameras/kameraAn"**)**

end

rule KameraAUS

when

Item camSwitch received command OFF

then

executeCommandLine**(**"rm /opt/openhab/webapps/cameras/kameraAn"**)**

end

Es sind theoretisch mehr Bilder pro Sekunde möglich, jedoch hat sich in der Praxis gezeigt, dass es dann dazu kommen kann das es schwarze Bilder gibt. Wir gehen davon aus, dass OpenHab die Bilder nicht schnell genug verarbeiten kann. Ein Nachteil dieser Methode ist, dass der nun die Bilder nicht mehr ausgeblendet werden können. Die Vorteile sind jedoch enorm. Durch die wenigen Bilder die gemacht werden müssen, ist die Performance sowohl im Browser als auch in der App wesentlich besser und es kommt zu keiner Überlastung mehr. Auch wird bei dem Streaming auf die App weniger Daten verbraucht, was durchaus wichtig sein kann, wenn man sich im mobilen Netz befindet. Die Personen vor der Tür sind auch weiterhin gut erkennbar.

**Audioübertragung**

Die ursprüngliche Idee der Audioübertragung sollte über die Lautsprecher und das Mikrofon der IP-Kamera laufen. Dabei sollte die Person vor der Tür mit der Person an dem Smartphone reden können und umgedreht.

Hier traten schnell Probleme auf, da OpenHab nicht von Haus aus dazu ausgelegt ist mehrere Audioeingänge und Audioausgänge zu verwalten. Als Lösung dieses Problems sind wir auf PulseAudio gestoßen. PulseAudio ist ein netzwerkfähiger Audioserver, der bei der Verwaltung von Audiogeräten hilfreich ist. PulseAudio hat auch ein Binding für OpenHab und war somit die scheinbar Ideal Lösung des Problems. Der Plan war somit das Mikrofon der IP-Kamer mit dem Lautsprechern des Smartphones zu verknüpfen und das Mikrofon des Smartphones mit den Lautsprechern der IP-Kamera. Nach dem erfolgreichen Aufsetzen eines PulseAudio-Servers auf unserem Pi mussten wir jedoch feststellen, dass PulseAudio nur die Audiogeräte erkenne kann, die entweder direkt am Pi angeschlossen sind oder die die mit einer IP angesprochen werden können. Somit hätten wir zwar die Komponenten an der IP-Kamer ansprechen können, jedoch nicht die auf dem Smartphone. Auch haben wir keine andere Lösung gefunden auf die Android-Ebene zuzugreifen mittels OpenHab.

Der zweite Lösungsansatz ist VoIP

**//Padre**

Die finale Umsetzung haben wir durch Text to speech (TTS) realisiert. Dabei war der erste Versuch mit MaryTTS und der zweite, erfolgreiche, Versuch mit GoogleTTS. Für beide gibt es ein Binding in OpenHab, jedoch muss für MaryTTS ein lokaler Server aufgesetzt werden, der die Verarbeitung vornimmt. Leider ist es uns nicht gelungen diesen in Betrieb zu nehmen. Daher haben wir uns GoogleTTS zugewandt. Hier jedoch ist es möglich ohne Binding auszukommen und mittels Shell-Skript eine unkomplizierte Lösung zu bauen. Um das Skript verwenden zu können muss zusätzlich MPlayer installiert werden um eine entsprechende Audioverarbeitung vornehmen zu können. Des Weiteren wird ALSA benötigt, was in der Regel jedoch schon auf Raspbian vorinstalliert ist. Das Skript muss mit einem String aufgerufen werden, welcher gesagt werden soll. Innerhalb des Skriptes wird der MPlayer aufgerufen mit einigen Parametern. Einer der Parameter gibt die Quelle der Audiodatei an. In diesem Fall ist es die Sprachausgabe von Google Translate. Das Skript sieht wie folgt aus:

#!/bin/bash

say**()** **{** **local** IFS**=+;/**usr**/**bin**/**mplayer **-**ao alsa **-**really-quiet **-**noconsolecontrols "http://translate.google.com/translate\_tts?ie=UTF-8&client=tw-ob&q=$\*&tl=DE-de"**;** **}**

say **$\***

Die auszugebenden Sätze werden dabei von uns vorgegeben. Mittels einer Auswahlliste werden diese in unsere Sitemap eingebunden.

Selection item=sagenAuswahl mappings=[ 0="Bin gleich da", 1="Bin nicht zu Hause", 2="Bin in einer Stunde da", 3="Stellen Sie es in der Garage ab" ]

Das Selection-Tag erzeugt die Auswahlliste. Der „item“ Parameter gibt das entsprechende Item in der items-Datei an, dazu später mehr. Der Parameter „mappings“ stellt das Kernstück des Selection-Tags da. Dadurch werden den Zahlen, die später benötigt werden, ein String zugewiesen. Der String wird auch auf der Benutzeroberfläche angezeigt.

In der items-Datei muss ein entsprechender Eintrag gemacht werden, damit eine Regel ausgelöst wird, wenn ein zusagender Satz angewählt wird:

Number sagenAuswahl "" <micro>

Dieses Item muss von Typ „Number“ sein, da in der Regel ein switch-case ist. Die Anführungszeichen sind leer, da es bei der App sonst zu einer Überschneidung kommt, zwischen den Text der dort angegeben ist und dem Text der gesprochen werden soll.

Die Regel sieht wie folgt aus:

rule auswahlBox

when

Item sagenAuswahl received command

then

logInfo("auswahlBox", "Wir sind vor switch case ")

switch(receivedCommand)

{

case 0: executeCommandLine( " /opt/openhab/configurations/scripts/speech.sh Bin gleich da")

case 1: executeCommandLine( " /opt/openhab/configurations/scripts/speech.sh Bin nicht zu Hause")

case 2: executeCommandLine( " /opt/openhab/configurations/scripts/speech.sh Bin in einer Stunde da")

case 3: executeCommandLine( " /opt/openhab/configurations/scripts/speech.sh Stellen sie es in der Garage ab")

}

end

Die Regel wird ausgelöst, sobald das Item „sagenAuswahl“ einen neuen Wert erhält. Die Variable „receivedCommand“ ist dabei die Zahl die in der Sitemap dem entsprechenden String zugeordnet ist. Je nachdem welcher Satz gesagt werden soll, wird mittels „executeCommandLine“ das Skript aufgerufen, mit dem entsprechenden Satz.

Die vorgegebenen Sätze können beliebig erweitert werden. Somit ist auch gut denkbar das Personen die nicht sprechen können trotzdem mit den Personen vor der Tür kommunizieren können.