Anhang: Software Dokumentation

Smarthome „Corona“ Pepper.

Erstellt von: Vasiliy Seibert, MOS 1, WiSe 20/21

Betreuer: Martin Kramer, Prof. Dr. Elmar Cochlovius

Erstellt am: 10.02.2020

Inhalt

Vorwort

Die Motivation sowie grundsätzliches Vorgehen bei diesem Projekt wird verstärkt in den anderen Dokumenten behandelt. Dieses Dokument setzt den Fokus auf die technische Umsetzung und soll dem Leser diese näher bringen.

Die Ergebnisse der Implementierung sind:

- Eine Stand-alone Applikation für den humanoiden Roboter „Pepper“ umgesetzt in Python und JavaScript mit Einbeziehung des Tablets
- Einrichtung und Verwendung einer VM zur plattformunabhängigen Arbeit.

Warum die VM, man kann ja alles auf seinem Rechner machen…

Kann man, ja. Aber das ist mit einer Reihe von Nachteilen verbunden die durch den Einsatz einer Virtual Machine mit entsprechender Netzwerkanbindung entfallen.

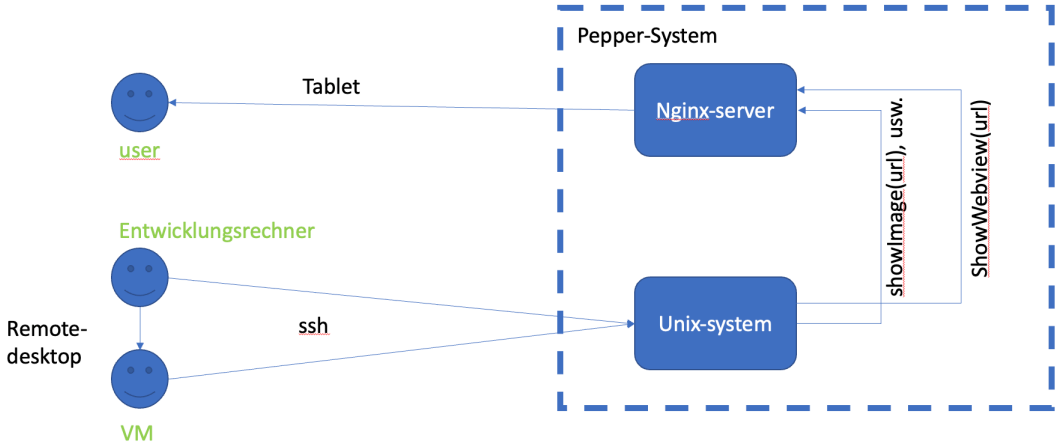
➔ Der Appell hier an der Stelle soll sein: konsequenter Einsatz der VM für die Arbeit an Semesterprojekten, Forschung und Thesis. Arbeit an dem eigenen PC sollte nur noch die Ausnahme sein.

Vorteile der VM:

- Plattformunabhängiges Arbeiten
- Eingerichtete Maschine mit allen benötigten Tools
- Kann von mehreren Studenten benutzt werden
- Keine einzelne Einrichtung nötig
- Arbeitsergebnisse bleiben an einem Ort (verschwinden nicht mit Rechner)
- Zentrales Aufbewahren, Verwalten von Dokus.

Es ist ein großer Vorteil, dass die VM einmal eingerichtet wird und dann für einen Großteil (wenn nicht alle) der anfallenden Arbeiten eingesetzt werden kann. Das wird gestärkt durch den Umstand, dass das Pepper System nicht laufend von Aldebaran aktualisiert wird. Um beispielsweise am Pepper zu arbeiten brauchen wir eine bestimmte (veraltete) Version von Choreograph. Aber da hört es nicht auf. Was stärker stört ist, dass man beispielsweise bei Python nicht mit der aktuellsten Version arbeiten kann, sondern auch hier auf eine veraltete Version zurückgreifen muss. Ein aktueller Trend ist bei transportablen Geräten wie Laptops, der Umstieg auf ARM- Prozessoren (siehe Apple M1 chip). Diese Unterstützen teilweise keine veralteten Python Versionen. Das mag jetzt noch ein reines Luxus Problem sein, wenn jedoch der Trend weiter voranschreitet könnte die VM eine optimale Lösung darstellen.

Architektur



Pepper Applikation in Python

Um eine Applikation auf dem Pepper auszuführen gibt es grundsätzlich mehrere Möglichkeiten. Unterscheiden kann man hinsichtlich der Umgebung, von der aus die Applikation initial erstellt und aufgerufen wird. Nachfolgende Grafik von Aldebaran gibt einen Überblick:

SDKs

Overview - *Supported programming languages*

Programming Languages	Bindings running on		Choregraphe support	
	Computer	Robot	Build Apps	Edit code
Python	✔	✔	✔	✔
C++	✔	✔	⊘	⊘
Java	✔	⊘	⊘	⊘
JavaScript	✔	✔	✔	⊘
ROS	✔	⊘	⊘	⊘

✔	OK
⊘	Not available

In diesem Fall konzentrieren wir uns auf eine mittels Python erstellte Applikation, die direkt auf dem Roboter ausgeführt wird.

Die grundsätzliche Herangehensweise ist:

- ALProxy zu importieren
- Einen ALProxy zu erstellen in dem Modul welches man verwenden möchte
- Eine Methode aufrufen

Wenn man wie in unserem Beispiel das Skript direkt auf dem Roboter ausführen möchte, dann reicht es den benötigten Service direkt in der Methode/ in der Klasse aufzurufen

Ein Codebeispiel für ein einfaches „Hallo Welt“ Programm (auf dem Roboter) kann so aussehen:

```
1  #! /usr/bin/env python
2  # -*- encoding: UTF-8 -*-
3
4  """Example: Use showWebview Method"""
5
6  import qi
7  import argparse
8  import sys
9  import time
10 from PIL import Image
11 import naoqi
```

Hier importieren wir einige benötigte Komponenten. Line 10 kann übersprungen werden weil diese erst später relevant wird.

```
93 class Pepper():
94
95   def __init__(self):
96     pass
97
98   def defaultPosture(self):
99     try:
100         goToPosture = session.service("ALRobotPosture")
101
102         goToPosture.goToPosture("StandInit", 0.5)
103         print "die goToPosture Methode wurde aufgerufen"
104
105    except Exception, e:
106         print "Error was: ", e
107
```

Hier erstellen wir eine Klasse Pepper, die in der Main später noch instanziiert wird. Die __init__ Methode ist zwingend notwendig auch wenn in dieser nichts geschehen soll. Des weiteren ist hier eine Methode <defaultPosture> definiert die den Roboter in eine Standartposition versetzen soll. Der Inhalt der Methode wurde deshalb nicht in die <__init__> gepackt weil <defaultPosture> im Verlauf des Programms noch weitere Male aufgerufen wird.

```
def halloWelt(self) {
  try:
    self.tts.say('Hallo Welt')
  except Exception, e:
    print "Error was: ", e
}
```

Die Methode <halloWelt> versucht <.tts.say(String)> (tts = text to speech) aufzurufen. Falls das nicht geht wird eine Exception geworfen. Jetzt muss man noch in der <main> die Klasse Pepper instanziieren und die Methode <halloWelt> aufrufen.

Also beispielsweise:

```
myPepper=Pepper()
myPepper.halloWelt()
```

Verwendung von Peppers Tablet

Die Darstellung von Inhalten auf Peppers Tablet gestaltete sich zu Beginn schwierig und undurchschaubar. Wie so oft ist es jedoch so, dass wenn man die Lösung erstmal gesehen hat, es gar nicht mehr so kompliziert ist.

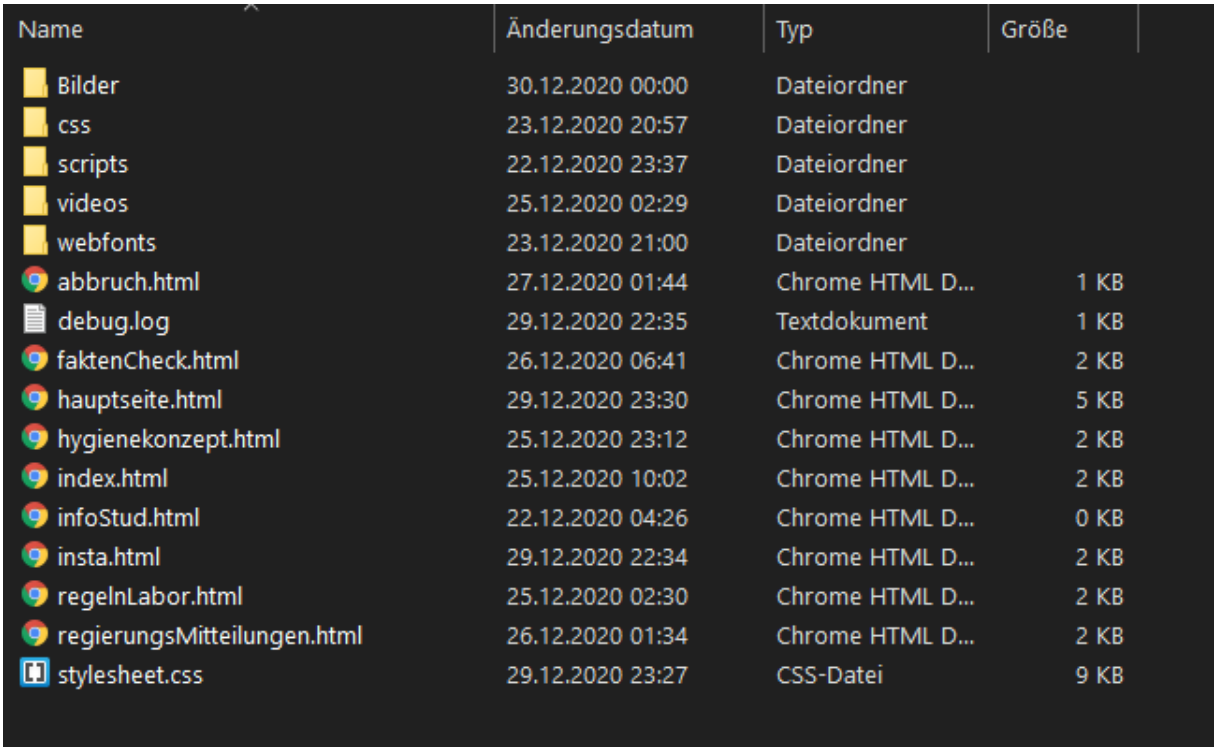
Es ist hilfreich sich vor Augen zu halten, dass das ‚eigentliche Peppersystem' und das Tablet zwei verschiedene Systeme sind, die über eine Schnittstelle miteinander verbunden sind.

Dementsprechend

müssen die auf dem Tablet angezeigten Inhalte (Bilder, Videos, Webinhalte) nicht auf dem unix Filesystem abgelegt sein, sondern auf dem Webserver. Das Python Script muss trotzdem auf dem unix Verzeichnis liegen. Im Falle unserer Applikation stelle ich diesen Sachverhalt mit Webinhalten (showWebview) dar. Dasselbe Vorgehen kann auch für Bilder (showImage) und Videos (playVideo) adaptiert werden. Wobei zu beachten ist, dass Bilder zuvor in den Cache geladen werden müssen (preLoadImage):

ALTabletService::showWebview(url):

Als erstes erstellen wir unsere Webinhalte mit allen was wir darin enthalten haben möchten (HTML, CSS, JS, etc)



Name	Änderungsdatum	Typ	Größe
Bilder	30.12.2020 00:00	Dateiordner	
css	23.12.2020 20:57	Dateiordner	
scripts	22.12.2020 23:37	Dateiordner	
videos	25.12.2020 02:29	Dateiordner	
webfonts	23.12.2020 21:00	Dateiordner	
abbruch.html	27.12.2020 01:44	Chrome HTML D...	1 KB
debug.log	29.12.2020 22:35	Textdokument	1 KB
faktenCheck.html	26.12.2020 06:41	Chrome HTML D...	2 KB
hauptseite.html	29.12.2020 23:30	Chrome HTML D...	5 KB
hygienekonzept.html	25.12.2020 23:12	Chrome HTML D...	2 KB
index.html	25.12.2020 10:02	Chrome HTML D...	2 KB
infoStud.html	22.12.2020 04:26	Chrome HTML D...	0 KB
insta.html	29.12.2020 22:34	Chrome HTML D...	2 KB
regelnLabor.html	25.12.2020 02:30	Chrome HTML D...	2 KB
regierungsMitteilungen.html	26.12.2020 01:34	Chrome HTML D...	2 KB
stylesheet.css	29.12.2020 23:27	CSS-Datei	9 KB

Das darüberliegende Verzeichnis muss die Bezeichnung ‚html' haben. Das Verwenden einer index.html ist sinnvoll ist aber für die Methode nicht zwingend notwendig -> es kann auf jede beliebige html datei zugegriffen werden.

Anschließend müssen wir die HTML Dateien auf das Pepper System übertragen. Über den Entwicklungsrechner können wir per SSH auf Pepper zugreifen. Beide Systeme müssen sich dafür im selben Netzwerk befinden und auf dem Entwicklungsrechner muss ein SSH Client eingerichtet werden. Pepper verwendet als OS eine Linux Distribution die extra für die Aldebaran Roboter erstellt wurde. Dieses OS ist FTP fähig. Aldebaran empfiehlt die Verwendung von einem Hilfsprogramm zur Dateiübertragung mit dem Namen ‚Filezilla' (Download: https://filezilla-project.org/). Für das Zugreifen auf das Pepper OS mittels SSH ist ein Passwort notwendig.

Nachdem wir über SSH auf Pepper zugegriffen haben starten wir Filezilla und verbinden uns mit Pepper. (sftp://nao@192.168.0.(41/40) -> Port 22).

Die Webinhalte hinterlegen wir auf:

[/data/home/nao/.local/share/PackageManager/apps/deineApplikation/html]

Das dazugehörige Pythonscript hinterlegen wir auf dem Unix Verzeichnis:

[/data/home/nao/deineApplikaiton]

In dem Python script müssen wir noch die Methode showWebview definieren und in der main ausführen:

```
157    def webviewAufrufen(self):
158        """
159        This example uses the showWebview method.
160        To Test ALTabletService, you need to run the script ON the robot.
161        """
162        # Get the service ALTabletService.
163
164        try:
165            tabletService = session.service("ALTabletService")
166
167            # Ensure that the tablet wifi is enable
168            #tabletService.enableWifi()
169
170            # Display a web page on the tablet
171            #tabletService.showWebview("http://www.google.com")
172
173            #time.sleep(5)
174
175            # Display a local web page located in boot-config/html folder
176            # The ip of the robot from the tablet is 198.18.0.1
177            tabletService.showWebview("http://198.18.0.1/apps/vasTest/index.html")
178            #tabletService.hideWebview()
179            # Hide the web view
180
181        except Exception, e:
182            print "Error was: ", e
183
```

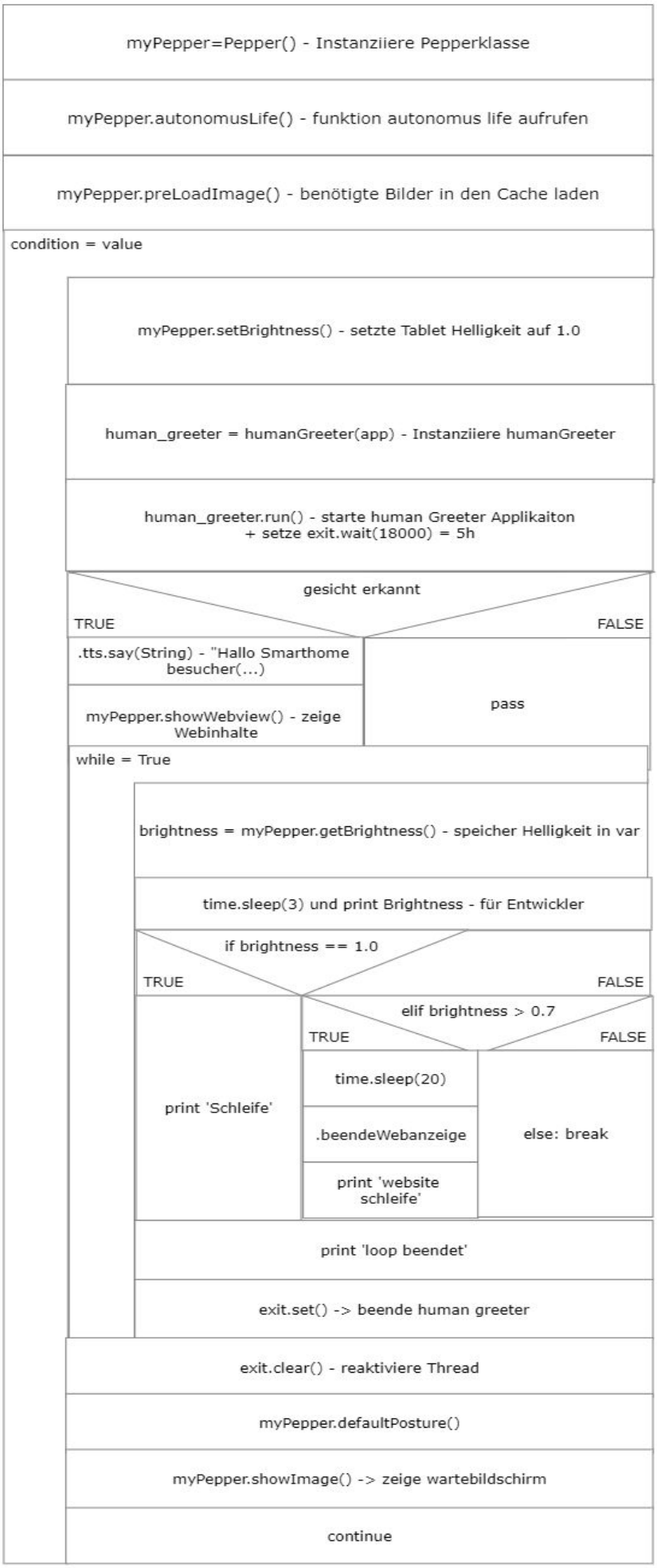
Man beachte hier noch die IP Adresse. <192.18.0.1> ist die Adresse des Peppers über die USB Schnittstelle. Der Pfad der Webinhalte wird aus Perspektive des Tablets beschrieben -> ab PackageManager. Die Angabe des Html Ordners muss weggelassen werden. -> hinzufügen führt zum Fehler. (?). Das Darstellen von Bildern und Videos erfolgt nach demselben Prinzip.

Programmdokumentation

Das Programm besteht aus einem Python Script und diversen Webinhalten, darunter auch einer Javascript Datei über welche auf die Qi Services zugegriffen wird.

Eine Herausforderung war es, die Applikation „autark“ zu gestalten. Damit wird in diesem Kontext gemeint, dass die Applikation selbstständig und ohne Eingriff des Entwicklers laufen soll. Um dieses Ziel zu realisieren war der Einsatz von Multithreading, Eventheandling sowie eine umfassende Exception Verwaltung.

Das Python Script läuft folgendermaßen ab:


myPepper=Pepper() - Instanziiere Pepperklasse
myPepper.autonomusLife() - funktion autonomus life aufrufen
myPepper.preLoadImage() - benötigte Bilder in den Cache laden
condition = value
myPepper.setBrightness() - setzte Tablet Helligkeit auf 1.0
human_greeter = humanGreeter(app) - Instanziiere humanGreeter
human_greeter.run() - starte human Greeter Applikaiton
+ setze exit.wait(18000) = 5h
gesicht erkannt
TRUE
FALSE
.tts.say(String) - "Hallo Smarthome besucher(...)
myPepper.showWebview() - zeige Webinhalte
pass
while = True
brightness = myPepper.getBrightness() - speicher Helligkeit in var
time.sleep(3) und print Brightness - für Entwickler
if brightness == 1.0
TRUE
FALSE
print 'Schleife'
elif brightness > 0.7
TRUE
FALSE
time.sleep(20)
.beendeWebanzeige
print 'website schleife'
else: break
print 'loop beendet'
exit.set() -> beende human greeter
exit.clear() - reaktiviere Thread
myPepper.defaultPosture()
myPepper.showImage() -> zeige wartebildschirm
continue

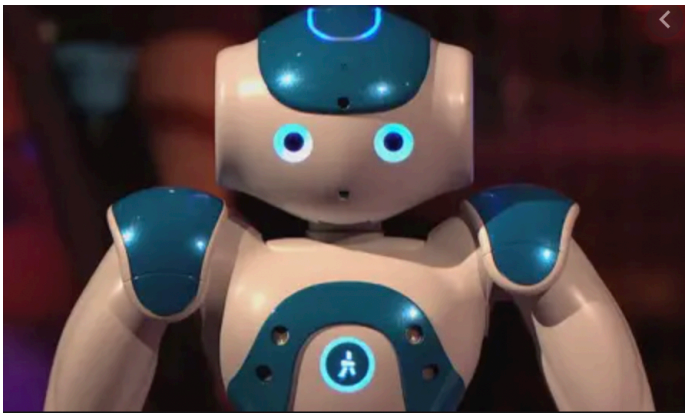
Fazit, nächste Schritte

Die Herausforderung bei dieser Entwicklung lag ganz klar auf der Umsetzung in Python mit der Einbindung des Tablets. Das Verwenden des Tablets hat Probleme bereitet aber diese gelten für weitere Projekte als gelöst, sofern diese Dokumentation zur Verfügung gestellt wird. Der Weg von der graphischen Entwicklungsumgebung weg ist wichtig, weil wir so einen erhöhten Gestaltungsfreiraum haben und andere Konzepte, Paradigmen leichter Einbinden können (beispielsweise Ansteuerung anderer Geräte über den Roboter).

Ich werde persönlich noch an diesem Projekt weiterarbeiten und versuchen weitere HW mit dem Pepper System zu verknüpfen. Mein Ziel ist es, den Pepper von einem externen Tablet anzusteuern. Wenn das gelingt könnte man noch über Augmented Reality nachdenken.

Die Verknüpfung mit einem externen Tablet wäre sehr wichtig, vor allem für den kleinen Bruder des Pepper: den NAO Roboter.

Diese Herangehensweise habe ich bei meiner Recherche noch bei keinem anderen gesehen.



Wir (die Hochschule) nehmen den NAO gerne zu Messen und anderen Veranstaltungen mit um Schüler von der Fakultät Informatik zu überzeugen. Ein eigenes, auf die Hochschule zugeschnittenes User Interface würde einen Mehrwert schaffen.