Titel

Seminararbeit

Version 0.01

Funktionale Programmierung
Zürcher Hochschule für Angewandte Wissenschaften

Simon Lang, Daniel Brun

Date

Versionshistorie

Version	Datum	Autor(en)	Änderungen
0.01	13.04.2015	DBRU	Initiale Version

Daniel Brun (DBRU)

Eigenständigkeitserklärung

Hiermit bestätigt ich, dass vorliegende Semesterarbeit zum Thema „BigData mit RaspberryPi und F#“ gemäss freigegebener Aufgabenstellung ohne jede fremde Hilfe und unter Benutzung der angegebenen Quellen im Rahmen der gültigen Reglemente selbständig verfasst wurde.

Zürich, 20.06.2016 Simon Lang, Daniel Brun

Inhaltsverzeichnis

KAPITEL 1

Einleitung

Diese Arbeit wurde als Seminararbeit zur Vorlesung von funktionalen Programmiersprachen verfasst. In diesem Kapitel wird die Aufgabenstellungen und Rahmenbedingungen der Arbeit erläutert.

1.1 Hintergrund

Einer immer grösseren Beliebtheit erfreuen sich kleine Alltagsgegenstände welche mit dem Internet verbunden sind. Dieser Bereich wird IoT genannt. Diese Gegenstände sind in der Lage Daten zu erheben und weiterzuleiten. Da es zukünftig voraussichtlich immer mehr IoT Gegenstände geben wird fallen immer mehr Daten an. Diese Daten werden wegen ihrer Masse auch BigData genannt.

In dieser Arbeit wird evaluiert wie sich funktionale Programmiersprachen im Bezug auf IoT eignen, um BigData auszuwerten.

1.2 Ziel

Mit einem IoT Gerät sollen Daten aufgezeichnet werden. Diese werden als BigData gesammelt und sollen mit einer funktionalen Programmiersprache ausgewertet und ansprechend ausgegeben werden.

Das Hauptziel der Arbeit besteht darin zu überprüfen wie geeignet funktionale Programmiersprachen für die Auswertung von BigData sind. Als Nebenziel soll evaluiert werden, ob eine funktionale Programmiersprache zum erfassen von Daten auf einem IoT Gerät verwendet werden kann.

1.3 Aufgabenstellung

Die freigegebene Aufgabenstellung lautet wie folgt:

- Projektname: Seminar BigData mit RaspberryPi und F#
- Ausgangslage: Durch die rasante Entwicklung im Bereich IoT ergeben sich viele neue Anwendungsmöglichkeiten. Da der RaspberryPI immer leistungsfähiger geworden ist, soll evaluiert werden ob er sich für den Einsatz von funktionalen Sprachen im Bereich BigData eignet.
- Ziel der Arbeit: Es soll gezeigt werden wie F# Sharp auf einem RaspberryPi im Bereich BigData und IoT eingesetzt werden kann.
- Aufgabenstellung: Es soll gezeigt werden, wie eine funktionale Programmiersprache (F#) im Kontext von BigData und IoT eingesetzt und verwendet werden kann. Es soll eine Anwendung zur Sammlung von Sensordaten auf einem Raspberry PI und eine Anwendung zur Analyse / Auswertung der gesammelten Daten implementiert werden.

1.4 Erwartete Resultate

Gemäss freigegebener Aufgabenstellung werden folgende Resultate erwartet:

- Dokumentation
- Implementation / Prototyp

1.5 Abgrenzung

Aufgrund des Umfanges der Arbeit und der begrenzten Zeitdauer werden folgende Punkte von der Arbeit abgegrenzt:

- **Schnittstellendokumentation**
  In dieser Arbeit werden nicht die Schnittstellendokumentationen und -spezifikationen rekonstruiert. Es werden jeweils die relevanten Aspekte betrachtet und hervorgehoben.

1.6 Motivation

1.7 Struktur

1.8 Planung

KAPITEL 2

Recherche

In diesem Kapitel werden die Grundlagen recherchiert wie die beiden Teilprojekte „Sensordaten sammeln“ und „Sensordaten auswerten (BigData)“ angegangen werden könnten.

2.1 Ausgangslage

Die Vorlesung zu diesem Semniar befasst sich mit den Konzepten der Funktionalen Programmierung. Zur Veranschaulichung dieser Konzepte wurde die Programmiersprache F# des .NET-Frameworks verwendet. Aufgrund dessen haben wir uns entschieden auch dieses Seminar mit der uns nun bekannten Sprache F# umzusetzen. Als IoT Gerät wird ein Raspberry Pi[1] verwendet. Der Raspberry Pi ist ein Einplatinencomputer welcher von der britischen Raspberry Pi Foundation entwickelt wurde. Der Raspberry Pi bietet den Vorteil, dass er sehr weit verbreitet ist[2], er kostengünstig ist und es inzwischen eine sehr grosse Anzahl an Sensoren auf dem Markt gibt mit welchem man Daten sammeln kann[3].

2.2 Der Raspberry Pi

Wie bereits im vorangehenden Kapitel beschrieben, handelt es sich beim Raspberry Pi um einen Einplatinencomputer. Dieser Einplatinencomputer bietet verschiedene zentrale Hardware-Schnittstellen um externe Geräte für Input und Output anzuschliessen.

Vom Raspberry Pi gibt es folgende Modelle:

- Raspberry Pi Compute Module
- Raspberry Pi Zero
- Raspberry Pi Model A

1 **Raspberry_Pi_2016-04-24**.
2 **Raspberry_Pi_Erfolgsgeschichte_2016-04-24**.
3 **Raspberry_Pi_Sensor_2016-04-24**.

- Raspberry Pi Model A+
- Raspberry Pi Model B
- Raspberry Pi Model B+
- Raspberry Pi 2 Model B
- Raspberry Pi 3 Model B

Evtl. ein Bild / List mit Daten dazu?

Am 29 Februar 2016 ist die neuste Version, der Raspberry Pi 3 (Model B), auf dem Markt erschienen[1]. Einige Zahlen zu dem Gerät:

- 1.2GHz 64-bit quad-core ARM Cortex-A53 CPU (10x die Leistung eines Raspberry Pi 1 und 50-60% die Leistung eines Raspberry Pi 2)
- Integriertes 802.11n wireless LAN und Bluetooth 4.1
- Komplette Kompatibilität zu Raspberry Pi 1 und 2 (Model B)

2.3 F# mit dem Raspberry Pi

Um F# auf dem dem Raspberry PI auszuführen, gibt es grundsätzlich zwei Möglichkeiten, welche nachfolgend erläutert werden. Da es sich bei F# um eine Sprache des Microsoft .NET-Frameworks handelt, wird für die Ausführung zwingend eine Implementierung des .NET-Frameworks benötigt.

2.3.1 Linux (Raspbian)

Ein weit verbreitetes Betriebsystem für den Raspberry Pi ist das Raspbian[2] OS. Bei dem Namen handelt es sich um eine Zusammenfassung von Raspberry und Debian. Demnach handelt es sich auch um eine Debian Distribution, welche spezifisch für den Raspberry Pi entwickelt wurde.

Eine Möglichkeit um unter Linux, beziehungsweise Raspbian, F# auszuführen ist das Mono-Framework[3]. Dabei handelt es sich um eine Open Source Implementierung von Microsoft's .NET Framework.

1 **Raspberry_Pi_3_2016-04-24**.
2 **FrontPage_-_Raspbian_2016-04-24**.
3 **Mono_2016-04-24**.

2.3.2 Windows 10 IoT

Microsoft hat mit Windows 10 IoT eine Version ihres Betriebssystem herausgebracht, welches speziell für leistungsschwächere Geräte entwickelt wurde[1]. Bei der IoT Version von Windows 10 ist das .NET Framework bereits standardmässig an Bord. Demnach sollte es keine Probleme geben um F# auf dieser Plattform zu betreiben.

2.4 Verwendete Hardware für die Umsetzung

Wir haben uns entschieden für die Umsetzung die nachfolgend aufgelisteten Hardware-Komponenten zu verwenden:

- Raspberry Pi 2 Model B
- Raspberry Pi 3 Model B
- GrovePi Sensoren
  - Temperature & Humidity Sensor
  - Sound Sensor
  - Light Sensor
  - Blue LED

GrovePi Sensoren

Für den Raspberry Pi gibt es viele Sensoren auf dem Markt. Herauskristallisiert hat sich jedoch das Starter Kid GrovePi+[2]. Dieses beinhaltet unter anderem Ton-, Temperatur-, Feuchtigkeit- und Lichtsensoren. Der Vorteil an den GrovePi Sensoren besteht an den geringen Kosten, dem einfachen Anschluss an den Raspberry Pi und die vielen verfügbaren Beispiele in unterschiedlichsten Programmiersprachen.

Die Recherchen haben ebenfalls gezeigt, dass es eine Library für .NET gibt mit welchem die Sensoren angesprochen werden können[3].

1 **Windows_IoT_2016-04-24**.
2 **GrovePi_2016-04-24**.
3 **NuGet_GrovePi_2016-04-24**.

2.5 Datenauswertung

Die von den Sensoren gespeicherten Daten werden in einem noch zu definierenden Format abgespeichert und danach für die Datenauswertung ausgelesen. Dafür wurde vom Dozenten die Library F# Data empfohlen[1]. Diese kann Daten in den Formaten CSV, HTML, JSON und XML entgegennehmen und verarbeiten.

Dokumentie wir den SW-Setup der Hard-ware? Installa-tionsan-leitung?

1 **Fsharp_Data_2016-04-24**.

KAPITEL 3

Sensordaten sammeln

Dieses Kapitel beschäftigt sich mit dem Teilprojekt der Sammlung von Sensordaten. Die Aufbereitung der gesammelten Daten ist Teil eines weiteren Kapitels.

3.1 F# auf dem Raspberry Pi

Im Abschnitt 2.3 F# mit dem Raspberry Pi wurden zwei Möglichkeiten aufgezeigt, welche es ermöglichen F# auf einem Raspberry Pi laufen zu lassen.

Es ist zu erwarten, dass der Weg über Window 10 IoT der einfachere sein wird, da dort das benötigte .NET Framework Teil des Systems ist. In dieser Seminararbeit wurden bewusst beide Varianten (Linux (Raspbian) und Windows 10 IoT) ausprobiert und getestet. Die folgenden Abschnitte erläutern den Setup und die Erkenntnisse aus dem Test der beiden Varianten.

3.1.1 Variante 1: Raspbian Jessie

1: Installation & Konfiguration von Raspbian Jessie

Zuerst wurde Raspbian Jessie auf dem Raspberry Pi installiert. Für die Installation des Betriebssystems wurde NOOBS (New Out Of Box Software) verwendet. NOOBS ist ein Installationsmanager für Betriebssysteme, der es ermöglicht per Klick ein gewünschtes Betriebssystem zu installieren. Es wurde dabei die Anleitung verwendet.

Je nach Netzwerk und Setup sind einige kleinere weitere Konfigurationen notwendig (z.B. Konfiguration des WLAN, Deaktivierung von IPv6 in einem IPv4 Netzwerk).

2: Installation des Mono-Frameworks & F#

Im Anschluss wurde die neuste Version des Mono-Frameworks (4.2.3) und der F#-Interactive Shell (F# Version 4.0) über die in den Software-Repositories vorhandenen Pakete installiert.

```
1 sudo apt-get install mono-complete fsharp
```

3: Installation & Konfiguration GrovePi

Die Installation und Konfiguration wurde anhand der vom Hersteller zur Verfügung gestellten Anleitung durchgeführt. Der Setup wurde anschliessend mit einem bereitgestellten Python-Script getestet.

Schwierigkeiten
Bei der Installation musste ein Git-Repository des Herstellers geklont werde. Dieses beinhaltet die Firmeware und verschiedenste Code-Beispiele und Beispiel Scripts. Das Repository scheint jedoch von gewissen Netzwerkknoten nicht erreichbar zu sein. Das Repository konnte nur im Netzwerk der Zürcher Hochschule für Angewandte Wissenschaften (ZHAW) geklont werden.

Testsetup
Die Installation und Konfiguration des GrovePi wurde über ein Testsetup mit Hilfe einer einfachen LED und eines mitgelieferten Python-Scripts getestet.

4: Anbindung des GrovePi via GrovePi-NuGet-Library

Nun galt es den GrovePi über F# anzusteuern und entsprechende Sensordaten auszulesen. Während der Recherchen wurden wir auf eine NuGet-Library[1] aufmerksam, welche die entsprechende Funktionalität bereitstellt. In der Entwicklungsumgebung (Monodevelop) wurde eine neues F# Projekt angelegt und das entsprechende NuGet-Packet installiert. Der erste testweise Build mit dem NuGet-Paket schlug fehl. Gemäss der Fehlermeldung wird für dieses Paket das Microsoft .NET Framework 5 benötigt. Beim Microsoft .NET Framework handelt es sich um eine Neuauflage des klassischen .NET Frameworks. Im nachfolgenden Abschnitt werden die wichtigsten Aspekte des .NET Frameworks 5 (beziehungsweise .NET Core) erläutert.

.NET Core
Ursprünglich war es geplant .NET Core als „.NET Framework Version 5“ auszurollen. Inzwischen wurde von Microsoft entschieden, .NET Core unter einer eigenständigen Versionsnummer zu führen (Ausgehend davon war die im vorangehenden Abschnitt beschriebene Fehlermeldung nicht mehr korrekt.).

Nachfolgend eine Auflistung der wichtigsten Aspekte und Ziele von .NET Core:

1 **NuGet_GrovePi_2016-04-24**.

- Hoher Grad an Portabilität
  - Plattformunabhängigkeit
  - Architekturunabhängigkeit (32-Bit / 64-Bit)
- Open Source Implementation
- Kleine und optimierte Runtime
  - Modularer Aufbau
  - Runtime wird als NuGet Package ausgeliefert
- Default Compiler für x64: „Roslyn“ just-in-time Compiler
- Verfügbare Runtime Varianten: .NET Native (Windows Only), Core CLR (Common Language Runtime, Multiplattform), Weitere ...

Die nachfolgende Grafik zeigt eine grobe Übersicht über das .NET Framework und .NET Core:

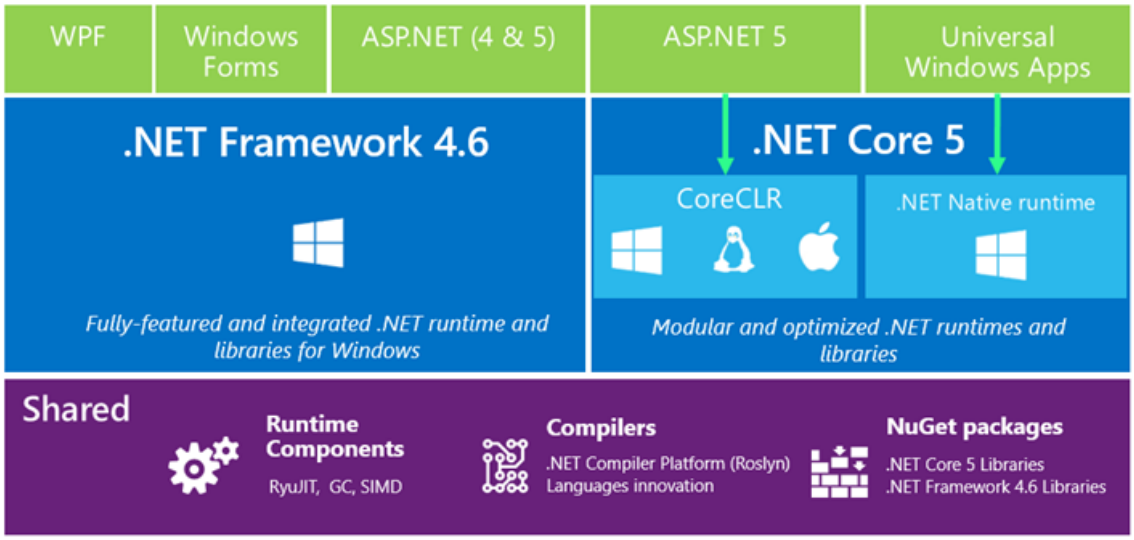

Abbildung 3.1: Übersicht .NET Framework & .NET Core
Quelle: https://blogs.msdn.microsoft.com/bethmassi/2015/02/25/understanding-net-2015/

5: Installation von .NET Core

Aufgrund der Fehlermeldung haben wir nun versucht .NET Core auf dem Raspberry Pi zu installieren. Zur Zeit zu dem dieser Setup durchgeführt wurden, existierten keine offiziellen Microsoft-Quellen zur Installation von .NET Core unter Linux.

Um .NET Core unter Raspbian zu installieren, wird das bereits installierte Mono Framework benötigt. Anschliessend kann das .NET / DotNet Execution Environement (DNX) installiert

werden. Diese Execution Environement beinhaltet alle notwendigen Libraries, um .NET Core Applikation auszuführen. Um DNX zu installieren ist es am einfachsten den .NET / DotNet VersionManager (DNVM) zu verwenden. Mit dem Version Manager können einfach verschiedene Version von .NET parallel installiert werden.

Installation DNVM

```
curl -sSL https://raw.githubusercontent.com/aspnet/Home/dev/dnvminstall.sh |
DNX_BRANCH=dev sh && source ~/.dnx/dnvm/dnvm.sh
```

Installation DNX .NET Core

```
sudo apt-get install libunwind8 gettext libssl-dev libcurl4-openssl-dev zlib1g
libicu-dev uuid-dev
mozroots --import --sync
dnvm upgrade -u
dnvm install latest -r coreclr -u
```

Die oben beschriebenen Schritte konnten soweit problemlos durchgeführt werden. Anschliessend galt es das Test-Projekt auf das neue Build-System von DNX umzustellen, welches mit .NET Core verwendet werden muss. Während dieser Umstellung hat sich gezeigt, dass der DNVM und die DNX bereits wieder verworfen wurde und direkt durch die .NET Core CLI (Command Line Interface) abgedekct werden[1]. Zum Zeitpunkt dieser Arbeit bestand jedoch neben dem DNVM keine andere Möglichkeit .NET Core unter Linux / Mono zu installieren. Glücklicherweise konnte das .NET Core CLI zusammen mit der .NET Core Runtime dennoch via DNVM installiert werden.

Die nächste Herausforderung bestand anschliessend mit dem neuen Build-System das korrekte Target-Framework zu finden. Aufgrund dessen, dass es noch keinen ersten finalen Release von .NET Core CLI gibt, kursierten im Web verschiedenste Definitionen für das Target-Framework. Darunter zum Beispiel „dotnet“ und „dnxcore50“.

Längere Recherchen und Analysen haben anschliessend zwei Fakten zu Tage gebracht, mit welchen wir nicht gerechnet hatten.

- **.NET Core in 32-Bit**
  Aktuell gibt es vom .NET Core keine 64-Bit Implementation. Sämtliche Raspberry Pi Modelle basieren auf einer ARM 32-Bit Architektur. Ausgenommen davon ist das neuste Model, der Raspberry Pi 3 Model B. Ebenfalls gibt es Raspbian aktuell nur in einer 32-Bit Version.
- **GrovePi NuGet Paket verwendet Windows-Only Funktionalitäten**
  Das GrovePi NuGet Paket hat Abhängigkeiten zur Universal Windows Platform (UAP). Die Universal Windows Platform gibt es nur für Windows, nicht für Linux

1 http://dotnet.github.io/docs/core-concepts/dnx-migration.html

oder Mac OS. Diese Abhängigkeit rührt wahrscheinlich daher, dass die Library ursprünglich spezifisch für Windows 10 IoT entworfen wurde.

6: Finaler Setup

Aufgrund dieser letzten Erkenntnisse haben wir uns entschieden die Umsetzung auf Raspbian Jessie (32-Bit) mit dem regulären Mono-Framework zu realisieren. Aufgrund dessen, dass das GrovePi NuGet Paket nicht verwendet werden kann, muss der Zugriff, beziehungsweise die Kommunikation zwischen Raspberry Pi und GrovePi auf eine andere Art gelöst werden.

3.1.2 Variante 2: Windows 10 IoT

Windows 10 IoT ist eine Version des Betriebssystems von Microsoft, welches speziell für kleinere Geräte mit weniger Rechenleistung konzipiert wurde.

Die Installation gemäss der Anleitung auf dem Github Account from Microsoft[1] war nicht erfolgreich. Der Raspberry Pi startete nicht und blieb beim Rainbow Screen[2] hängen. Mit dem NOOBS[3] Installer, welcher von der Raspberry Pi Foundation zur Verfügung gestellt wird, war die installation von Windows 10 Io

3.2 Speicherung der Sensordaten

Zum Abspeichern der Sensordaten haben sich verschiedene Datenformate angeboten. Um die Komplexität möglichst gering zu halten und maximale Flexibilität zu erreichen, haben wir uns entschieden die Sensordaten File-basiert abzuspeichern.

Neben CSV (Comma Separated Values) standen auch JSON (JavaScript Object Notation) oder XML (Extensible Markup Language) zur Auswahl. Da die Daten in einem kontinuierlichen Stream geschrieben werden müssen haben wir uns schlussendlich für ein klassisches CSV-File entschieden. Dies bietet den Vorteil, dass neue Datensätze ohne Probleme laufend ans Ende der Datei angehängt werden können.

Bei JSON und XML ist dies nicht ohne weiteres möglich, da die Daten dort in Hierarchischer Form abgespeichert werden.

3.3 Umsetzung

In diesem Kapitel wird die konkrete Umsetzung der Problemstellung „Sammeln von Sensordaten“ beschrieben.

1 **install_win10iot_2016-04-25**.
2 **RPi_Rainbowscreen_2016-04-25**.
3 **NOOBS_2016-04-25**.

3.3.1 Verwendete Hardware

Für die Realisierung wurden nachfolgend aufgelisteten Hardware-Komponenten verwendet:

- Raspberry Pi 2 Model B
- Raspberry Pi 3 Model B
- GrovePi Sensoren
  - Temperature & Humidity Sensor (GrovePi Port D3)
  - Sound Sensor (GrovePi Port A0)
  - Light Sensor (GrovePi Port A2)
  - Blue LED (GrovePi Port D4)

3.3.2 Verwendete Software

Für die Realisierung werden folgende Softwarekomponenten verwendet:

- Raspbian Jessie
- GrovePi Firmware + Beispiele
- Mono 4.2

3.3.3 Dokumentation der Implementation

Die konkrete Implementation der Problemstellung „Sensordaten sammeln“ wird im Abschnitt 3.4 Dokumentation der Implementation dokumentiert.

3.3.4 Datentransfer zum Raspberry Pi

Die kompilierten Sourcen wurden via Linux-Befehl „scp“ auf den Raspberry Pi übertragen.

```
1 scp -rp ./Raspberry.FGrove pi@raspberrypi.local:/home/pi/Development/FUP/
```

3.3.5 Datentransfer vom Raspberry Pi

Der Datentransfer vom Raspberry Pi wurde durch einen einfachen HTTP-Server auf Python-Basis bewerkstelligt. Python wurde bei der Installation des GrovePi automatisch mitinstalliert. Dadurch konnten wir den „SimpleHTTPServer“ nutzen, welcher standardmässig mit Python ausgeliefert wird.

```
1 python -m SimpleHTTPServer 8000
```

3.3.6 Betrieb & Starten der Applikation

Damit der Raspberry Pi zum Sammeln von Sensordaten theoretisch auch unabhängig von einem Netzwerk betrieben werden könnte, wurde der Raspberry Pi als WiFi-Access-Point konfiguriert. Beim Start wird geprüft, ob ein bekanntes WiFi-Netzwerk verfügbar ist. Ist dies nicht der Fall, werden entsprechend die Konfigurationen für den WiFi-Access-Point geladen. Die Konfiguration dieses Access-Points, beziehungsweise Ad-Hoc-Netzwerkes, wurde anhand von Anleitungen von Lasse Christiansen und cpetty33 durchgeführt.

Mit diesem Setup kann nun jederzeit auf den Raspberry Pi zugegriffen und entweder die Anwendung gestartet / gestoppt oder die Daten heruntergeladen werden.

Für den Start der Anwendung musste ein zusätzliches Tool namens „screen“ installiert werden. Mit diesem Tool ist es einfach möglich mehrere Hintergrundverarbeitungen zu starten, welche nicht automatisch beendet werden, wenn die aktuelle Session (SSH-Verbindung) beendet wird.

Start der Anwendung

```
1 ssh pi@raspberrypi.local (SSH into Raspberry Pi))
2 screen -S rpfs (create new screen session "`rpfs"')
3 python -m SimpleHTTPServer 8000 (start Python HTTP-Server)
4 Ctrl + a + c (create new window in current screen session)
5 sudo fsharpi --lib:/home/pi/Development/FUP/Raspberry.FGrove/libs "/home/pi/
  Development/FUP/Raspberry.FGrove/Program.fsx" (start F\# application)
6 ctrl + a + d (detach from current screen session)
```

Beenden der Anwendung

```
1 ssh pi@raspberrypi.local (SSH into Raspberry Pi))
2 screen -r rpfs (attach to the existing screen session "`rpfs"')
3 ctrl + c (terminate F\# application)
4 exit (close current window in screen session)
5 ctrl + c (terminate Python HTTP-Server)
6 exit (closes current window and screen session)
```

3.4 Dokumentation der Implementation

Nachfolgend werden die wichtigsten Punkte der einzelnen Komponenten dokumentiert.

3.4.1 Komponenten

Die Anwendung besteht aus folgenden Teilkomponenten:

- Raspberry.IO.InterIntegratedCircuit (C# Library)
- Raspberry.GrovePi (C# Library)
- Raspberry.FGrove (F# Applikation)

3.4.2 Source Code

Der Source Code wird in folgendem Git-Repository verwaltet: FUP_Sem_IOT

3.4.3 Raspberry.IO.InterIntegratedCircuit (C# Library)

Bei dieser Komponente handelt es sich um einen Klon / Fork der Teilkomponente „Raspberry.IO.InterIntegratedCircuit“ des Git-Repositorys raspberry-sharp-io von Raspberry#.

Diese Library ist aktuell nicht als NuGet-Packet verfügbar und wurde spezifisch für .NET unter Mono optimiert. Die Library verwendet die NuGet-Packete „Raspberry.IO.GeneralPurpose“ und „Raspberry.System“, welche von Raspberry# veröffentlicht und im genannten Git-Repository gepflegt werden.

Das Ziel dieser Library ist es, den Zugriff auf den Inter-Integrated Circuit (I2c)-Bus des Raspberry Pi zu abstrahieren / vereinfachen.

3.4.4 Raspberry.GrovePi (C# Library)

Diese Komponente stellt unter Verwendung der Library „Raspberry.IO.InterIntegratedCircuit“ spezifische Funktionalitäten für den Zugriff auf den GrovePi via I2c zur Verfügung.

Die Library besteht aus einer einzigen Wrapper-Klasse, welche die I2c und GrovePi spezifischen Elemente für die eingesetzten Sensoren abstrahiert. Über den I2c-Bus müssen spezifische Befehle gesendet werden, um die gewünschten Sensordaten auszulesen. Diese Befehle werden von einem Arduino auf dem GrovePi Board interpretiert und verarbeitet. Die zulässigen Befehlssequenzen wurden den Hersteller-Dokumentationen entnommen.

- Raspberry Pi GPIO Connector
- GrovePi Port Description
- GrovePi Protocol and Adding Custom Sensors
- GrovePi Firmware Documentation
- GrovePi Sound Sensor
- GrovePi Temperature & Humidity Sensor
- GrovePi Light Sensor

Nachfolgend werden die verwendeten Befehlsfolgen aufgezeigt:

I2C auf dem Raspberry

```
1 SDA-Pin: P1Pin03 (Nummer des SDA GPIO-Pins auf dem Raspberry Pi Board, notwendig
  für I2C, Data-Line)
2 SCL-Pin: P1Pin05 (Nummer des SCL GPIO-Pins auf dem Raspberry Pi Board, notwendig
  für I2C, Clock-Line)
3 I2C Device Address: 0x04
```

Allgemeines Befehlsschema

```
Byte 0: 1 (Dummy, statischer Wert)
Byte 1: Command (ID der auszuführenden Aktion)
Byte 2: Pin (Nummer des anzusprechenden Pin auf dem GrovePi)
Byte 3: Data (Optional, Input-Daten)
Byte 4: Data (Optional, Input-Daten)
```

Sound / Noise Sensor

```
Byte 0: 1 (Dummy)
Byte 1: 3 (Command: Analog-Read)
Byte 2: 0 (Pin: 0)
Byte 3: 0
Byte 4: 0
Result: 3 bytes (Sensordaten: Umgebungslautstärke)
```

Temperature & Humidity Sensor

```
Byte 0: 1 (Dummy)
Byte 1: 40 (Command: Read DHT-Sensor)
Byte 2: 3 (Pin: 3)
Byte 3: 0
Byte 4: 0
Result: 9 bytes (Sensordaten: Temperatur, Luftfeuchtigkeit)
```

Light Sensor

```
Byte 0: 1 (Dummy)
Byte 1: 3 (Command: Analog-Read)
Byte 2: 2 (Pin: 2)
Byte 3: 0
Byte 4: 0
Result: 3 bytes (Sensordaten: Helligkeit / Lichtintensität)
```

LED (On)

```
Byte 0: 1 (Dummy)
Byte 1: 2 (Command: Digital-Write)
Byte 2: 4 (Pin: 4)
Byte 3: 1 (Data: On)
Byte 4: 0
Result: -
```

LED (Off)

```
Byte 0: 1 (Dummy)
Byte 1: 2 (Command: Digital-Write)
Byte 2: 4 (Pin: 4)
Byte 3: 0 (Data: Off)
Byte 4: 0
Result: -
```

Der GrovePi verarbeitet die eingehenden Befehle sequentiell. Damit sichergestellt ist, dass zum abgesetzten Befehl auch die korrekten Daten zurückgeliefert werden, wurde der Lock-Mechanismus von C# verwendet. Dadurch wird der Zugriff auf den GrovePi so eingeschränkt, dass kein simultaner Zugriff mehr möglich ist.

```
// Byte-Array for results
byte[] ret;

// Create lock, if locked: wait till lock is released
lock (_locker) {
  // Send command to GrovePi
  connection.Write (new[] { (byte)1, (byte)3, (byte)2, (byte)0, (byte)0 });

  // Wait a few ms (allows the arduino to collect the data)
  Thread.Sleep (10);

  // Read the response
  ret = connection.Read (3);

// release the lock
}
```

3.4.5 Raspberry.FGrove (F# Applikation)

Diese Komponente realisiert die Kern-Logik der Anwendung, die Sammlung und Speicherung der Sensordaten. Dazu wird die implementierte C# Library „Raspberry.GrovePi“ verwendet.

Pro Sensor wurde ein eigenständiger Timer implementiert, welche periodisch gestartet und anschliessend zurückgesetzt wird. Pro Timer wurde ein Event-Listener registriert, welcher die spezifischen Sensordaten ausliest und anschliessend in ein Comma Separated Values (CSV)-File schreibt.

Nachfolgend wird die Implementation exemplarisch für den „Sound Sensor“ aufgezeigt:

```
// Create timer for sound / noise sensor
let nTimer, nEventStream = startTimerAndCreateObservable 100

// Subscribe to event
nEventStream |> Observable.subscribe (fun _ -> processNoiseEvent())
```

startTimerAndCreateObservable

```
// Creates and starts a new timer and returns an observable event
let startTimerAndCreateObservable timerInterval =
    // Setup timer
    let timer = new System.Timers.Timer(float timerInterval)

    // Autoreset and enable
    timer.AutoReset <- true
```

```
8     timer.Enabled <- true
9
10    // Return observable event
11    (timer,timer.Elapsed)
```

writeData

```
1 // Writes the sensordata to a file. If the file doesn't exists it will be created
  . Otherwise the data will be appended to the end of the file.
2 let writeData suffix (data:String) (date:DateTime) (location:String) =
3     let filepath = "./data/SensorData-" + suffix + ".csv"
4
5     if not(File.Exists(filepath)) then
6         printfn "Data-File doesn't exists, creating file"
7         use streamWriter = new StreamWriter(filepath,false)
8         streamWriter.WriteLine "Date;Time;SensorData;Location"
9         streamWriter.Flush()
10        streamWriter.Close()
11
12    use streamWriter = new StreamWriter(filepath,true)
13    [ date.ToString("dd.MM.yyyy");
14      date.ToString("hh:mm:ss.fff");
15      data;
16      location]
17    |> List.fold (fun r s -> r + s + ";") ""
18    |> streamWriter.WriteLine
19
20    streamWriter.Flush()
21    streamWriter.Close()
```

readNoiseSensor

```
1 // Reads the data from the sound / noise sensor.
2 let readNoiseSensor() =
3   grovePiWrapper.readNoiseSensorData()
```

processNoiseEvent

```
1 // Processes the timer event for the sound / noise sensor.
2 let processNoiseEvent() =
3      let sensorValue = readNoiseSensor()
4      writeData "N" (sensorValue.ToString()) DateTime.Now currentLocation
```

KAPITEL 4

Sensordaten auswerten und aufbereiten

Dieses Kapitel beschäftigt sich mit dem Teilprojekt der Auswertung und Aufbereitung der gesammelten Sensordaten.

4.1 Verwendete Hardware

4.2 Verwendete Software

4.3 Dokumentation der Implementation

KAPITEL 5

Schlusswort

5.1 Fazit

5.1.1 Sensordaten sammeln

5.1.2 Sensordaten aufbereiten und auswerten

5.2 Dank

Abbildungsverzeichnis

ANHANG A

Anhang