Hochschule Luzern
Technik & Architektur

PAWI

myCoffee - Bericht

Mobile-App für Kaffeemaschinen

---

18. Dezember 2014

Studierender:
Philipp Rupp

Dozent:
René Meier

Auftraggeber:
W.B.S.S GmbH
Uraniastrasse 26
8001 Zürich

Abstract

Die Entwicklungen rund um das Internet der Dinge haben in den letzten Jahren stark zugenommen. Dieser Anstieg kann auf die starke Verbreitung des Internets sowie auf den stetigen Fortschritt in der Hardwareentwicklung zurückgeführt werden.

Die vorliegende Arbeit befasst sich mit genau diesem Thema. Im Rahmen des Informatikprojektes PAWI der Hochschule Luzern Technik & Architektur wurde ein Prototyp für das Internet der Dinge entwickelt. Konkret werden Getränkedaten einer Kaffeemaschine ausgelesen und auf einem Android Smartphone angezeigt.

Der umgesetzte Prototyp basiert auf einer Client/Server-Architektur. Die Daten der Kaffeemaschine werden von einem Raspberry Pi ausgelesen und als Client an einen zentralen Server weitergeleitet. Vom Server werden sie anschliessend lokal persistiert und an den Android-Client weitergegeben, wo sie der Nutzer betrachten kann. Nebst der Einsicht von Live-Daten, hat der Nutzer die Möglichkeit statistische Getränkedaten des aktuellen Monats abzurufen.

Die Kommunikation zwischen Clients und Server wurde mittels MQTT-Protokoll realisiert. Dabei handelt es sich um ein leichtgewichtiges Protokoll, welches per Publish/Subscribe-Verfahren funktioniert. Aufgrund dieser Eigenschaften kommt MQTT vor allem bei Machine-to-Machine Kommunikation zum Einsatz.

Abschliessend kann gesagt werden, dass das in diesem Projekt umgesetzte Konzept gut skaliert und einfach auf Geräte angewendet werden kann, welche ins Internet der Dinge eingebracht werden sollen.

Inhaltsverzeichnis

Abkürzungen und Definitionen

Abkürzung	Beschreibung
AES	Advanced Encrypted Standard Blockverschlüsselung
API	Application Programming Interface Vom Entwickler zur Verfügung gestellte Programmierschnittstelle
AWS	Amazon Web Services Verschiedene Cloud Webservices angeboten von Amazon.com
CoAP	Constrained Application Protocol Machine to Machine Nachrichtenprotokoll basierend auf REST
DB	Datenbank System zur elektronischen Datenverwaltung
DDCMP	Digital Data Communications Message Protocol Netzwerkprotokoll für Punkt-zu-Punkt Kommunikation
EC2	Amazon Elastic Compute Cloud Virtualisierte Computer von Amazon.com
EVA-DTS	European Vending Association Data Transfer Standard Standard Format für Daten, welche aus Automaten ausgelesen werden
IoT	Internet of Things (Deutsch Internet der Dinge)
MQTT	Message Queue Telemetry Transport Leichtgewichtiges Open Source Protokoll für Machine to Machine Datenübertragung
REST	Representational State Transfer Programmierparadigma für Webanwendungen
RSA	Rivest, Shamir und Adleman Asymetrisches kryptographisches Verfahren zur Verschlüsselung
SSL	Secure Socket Layer Vorgängerbezeichnung von TLS
TLS	Transport Layer Security Verschlüsselungsprotokoll zur sicheren Datenübertragung im Internet

1. Einleitung

Die heutige Welt wird immer globaler und vernetzter. Es gibt kaum noch Orte, an welchen kein Internetzugang zur Verfügung steht. Diese allgegenwärtige Möglichkeit, sich mit dem Welt weiten Netz zu verbinden, hat ein neues Phänomen hervorgebracht – Das Internet der Dinge, nachfolgend IoT (Internet of Things).

Das Ziel des IoT ist es, das nebst Computer und Smartphone auch normale Alltagsgegenstände *smart* werden. Dies wird erreicht, indem beispielsweise der Kühlschrank oder die Waschmaschine mit dem Internet verbunden sind, und so zu interaktiven Gegenständen werden. Durch diese Internetverbindung können Gegenstände miteinander, aber auch mit dem Besitzer kommunizieren. Bemerkt der Kühlschrank durch entsprechende Sensoren, dass ein bestimmtes Produkt ausgegangen ist, kann er dies dank Onlinezugang direkt an seinen Besitzer melden oder aber selbständig nachbestellen.

Die Entwicklungen rund um das IoT haben stark zugenommen. Dies hat auch die W.B.S.S. GmbH aus Zürich festgestellt und sich dazu entschieden, als Wirtschaftspartner der Hochschule Luzern Technik & Architektur einen IoT-Prototypen einer Kaffeemaschine entwickeln zu lassen.

1.1. Ausgangslage

Die zur Verfügung gestellte Maschine besitzt eine serielle Schnittstelle, über welche maschinenspezifischen Informationen ausgelesen werden können. Es ist jedoch nicht möglich, über die Schnittstelle Befehle zu senden, mit welchen die Maschine gesteuert werden kann.

1.2. Aufgabenstellung

Im Rahmen dieser Arbeit soll ein Prototyp erstellt werden, welcher es erlaubt maschinenspezifische Daten über die serielle RS232 Schnittstelle auszulesen, abzuspeichern und für andere Geräte freizugeben. Um an die entsprechenden Daten zu kommen und diese zur Verfügung zu stellen, soll ein Raspberry Pi zum Einsatz kommen. Das Raspberry Pi besitzt neben Anschlüssen zur RS232 Kommunikation auch einen Ethernet Anschluss, wecher die Anbindung ans Internet ermöglicht. Nachdem die Informationen der Kaffeemaschine gewonnen und in einer Datenbank abgelegt wurden, müssen diese an Clientgeräte weitergegeben werden. Diese Clients bereiten die gewonnenen Informationen für den Nutzer visuell auf.

Ein möglicher Client ist ein Smartphone, welches die empfangenen Informationen über eine entsprechende App anzeigt. Da nebst der Internetverbindung auch andere Technologien, wie beispielsweise Bluetooth oder ZigBee zum Einsatz kommen können, muss die Architektur der Datenübertragung vor Entwicklung der App festgelegt werden.
Nebst den maschinenspezifischen Informationen sollen auch mögliche Ansätze geprüft werden, um statistische Informationen von mehreren Kaffeemaschinen in einer App anzuzeigen.

Um konkrete Ideen für die Umsetzung des Prototypen zu erlangen, dient eine Konkurrenzanalyse zu Beginn der Arbeit. Konkret soll ermittelt werden, ob bereits ähnliche Ansätze existieren, um diese anschliessend auf ihre Vor- und Nachteile zu überprüfen. Die Erkenntnisse dieser Konkurrenzanalyse sollen in das Systemdesign einfliessen.

Zusätzlich soll auch das Thema Sicherheit bei der Umsetzung der Prototypen berücksichtigt werden. Konkret bedeutet dies, das Unbefugte keinen Zugriff auf die entsprechenden Maschinendaten erhalten sollen.

Die originale Aufgabenstellung ist dem Anhang A zu entnehmen.

2. Projektvorgehen

Der Einstieg in dieses Projekt bildete eine Konkurrenz- und Technologieanalyse. Konkret wurde untersucht, wie der Stand der Technik in der Kaffeemaschinenbranche im Zusammenhang mit internetfähigen Maschinen ist. Weiter zeigte die Analyse auf, was technisch Machbar ist und welche Technologien und Architekturen im IoT eingesetzt werden. Durch die Erkenntnisse dieser Analyse konnten die Ziele sowie das weitere Vorgehen für dieses Projekt konkretisiert werden. Sie bildete das Fundament dieser Arbeit, was sich in den Ergebnissen wiederspiegelt.

Nachdem eine grobe Vorstellung vom Endprodukt vorhanden war, wurde diese in mehreren Schritten verfeinert. Der erste Schritt bestand darin, die einzusetzenden Technologien für dieses Projekt zu evaluieren. Als Grundlage dafür dienten zum Einen die Kundenanforderungen, welche bestimmte Vorgaben bezüglich Technologiewahl trafen und zum Anderen Variantenanalysen, durch welche verschiedene Lösungsansätze miteinander verglichen werden konnten. Nach dieser Phase waren alle Technologien für die einzelnen Problembereiche inklusive Zusammenspiel dieser definiert. (siehe Kapitel 4)

Auf Basis der Technologieanalyse konnte eine Systemarchitektur festgelegt werden, welche später in einen Prototypen umgesetzt wurde. Die Architektur legt die Softwarekomponenten inklusive Verteilung und Kommunikation zwischen diesen fest und setzt den Grundstein für die Implementierung. (siehe Kapitel 3)

In der Implementierungsphase galt es, die erstellte Architektur umzusetzen. Als Vorgehensmodell für die Softwareentwicklung wurde Scrum gewählt. Scrum unterteilt die Entwicklungsphase in einzelne Sprints, in welchen die Software inkrementell entwickelt wird. Zu Beginn dieser Phase wurde die Anforderung an die Software in einzelne Stories heruntergebrochen, welche mit einer Aufwandschätzung versehen und im Backlog priorisiert wurden. Vor jedem Sprint wurde das konkrete Sprintziel definiert, anhand welchem die Stories für den bevorstehenden Sprint festgelegt werden konnten. Am Ende jedes Sprints fand ein Review statt, welches die Sprintergebnisse analysierte und Verbesserungen für das weitere Vorgehen aufdeckte. Waren einzelne Stories am Ende eines Sprints nicht komplett abgeschlossen, wurden diese in den nächsten Sprint übernommen. (siehe Projektmanagementplan in Anhang B)

Am Ende dieser Implementierungsphase lagen einzelne Komponenten vor, welche anschliessend durch Systemtests auf ihr korrektes Zusammenspiel geprüft wurden.

In einem letzten Schritt galt es, die gefundenen Fehler auszubessern und die Software robuster zu machen. Um die Weiterverwendung sowie Entwicklung zu gewährleisten, wurde als Abschluss eine Installationsanleitung erstellt, sowie mögliche Erweiterungen und Probleme festgehalten.

2.1. Projektaufwand

	Soll[h]	Ist[h]	Abweichung[%]
Initialisierung			
Kick-Off	2	1	-50.0
Schnittstelleineinführung	4	1.5	-62.5
Konkurrenzanalyse	10	6	-40.0
PMP	6	5	-16.7
Einrichtung	2	1.5	-25.0
Recherche	16	10	-37.5
Software			
Architektur	10	6	-40.0
Raspberry Pi Software	20	17	-15.0
Server Software	30	25	-16.7
Android App	32	55	71.9
Testing	10	12	20.0
Projektabschluss			
Bericht	30	44	46.7
Installationanleitung (readme auf CD)	8	2	-75.0
	180	186	3.3

Abbildung 1: Soll/Ist Vergleich

Der gesamte Projektaufwand von 180 Stunden wurde mit 186 effektiv geleisteten Stunden um 3.3% Überschritten. Diese geringe Überschreitung zeigt, dass die Aufwandschätzung zu Beginn des Projektes grundsätzlich gut war.

Werden die einzelnen Schätzpunkte betrachtet, wird jedoch ersichtlich, dass teilweise doch grosse Abweichungen zwischen Soll- und Ist-Wert vorhanden sind. Jedoch wurde die Soll-Zeit meistens Überschätzt. Lediglich in drei Punkten war der effektive Aufwand grösser als der geschätzte. Dies sind die Android App Entwicklung, das Testing und der Bericht. Vor allem die App Entwicklung weist mit 71.9% eine markante Abweichung zwischen Soll und Ist auf. Dies ist darauf zurückzuführen, dass zu Beginn des Projektes keine Kenntnisse in der Android Programmierung vorhanden waren. Weiter wurde der Aufwand zur Erstellung der grafischen Benutzeroberfläche massiv unterschätzt.

Trotz zum Teil massiven Abweichungen zwischen Soll- und Ist-Aufwand einzelner Schätzungen kann abschliessend gesagt werden, dass der Gesamtaufwand angemessen geschätzt und auch weitgehend eingehalten wurde.

3. Softwarearchitektur

In diesem Kapitel ist die Architektur des myCoffee-Systems beschrieben. Dazu wird die Systemübersicht aufgezeigt und mit Hilfe von Komponenten- sowie Klassendiagrammen detailliert beschrieben. Detaillierte Abläufe sowie die Interaktion der einzelnen Teilsysteme werden durch Sequenzdiagramme visualisiert und durch die Spezifikation von Schnittstellen beschrieben.

3.1. Systemübersicht

Abbildung 2 zeigt die einzelnen Akteure sowie den Datenfluss des myCoffee-Systems. Das System basiert auf einer Client/Server-Architektur, welche in drei Subsysteme aufgeteilt ist. Das Raspberry Pi dient als Datenlieferant und bildet somit den Datenlayer dieses Systems. Der Server als Businesslayer realisiert die Geschäftslogik. Das Android Smartphone visualisiert die Daten und schliesst somit das System als Präsentationslayer ab.

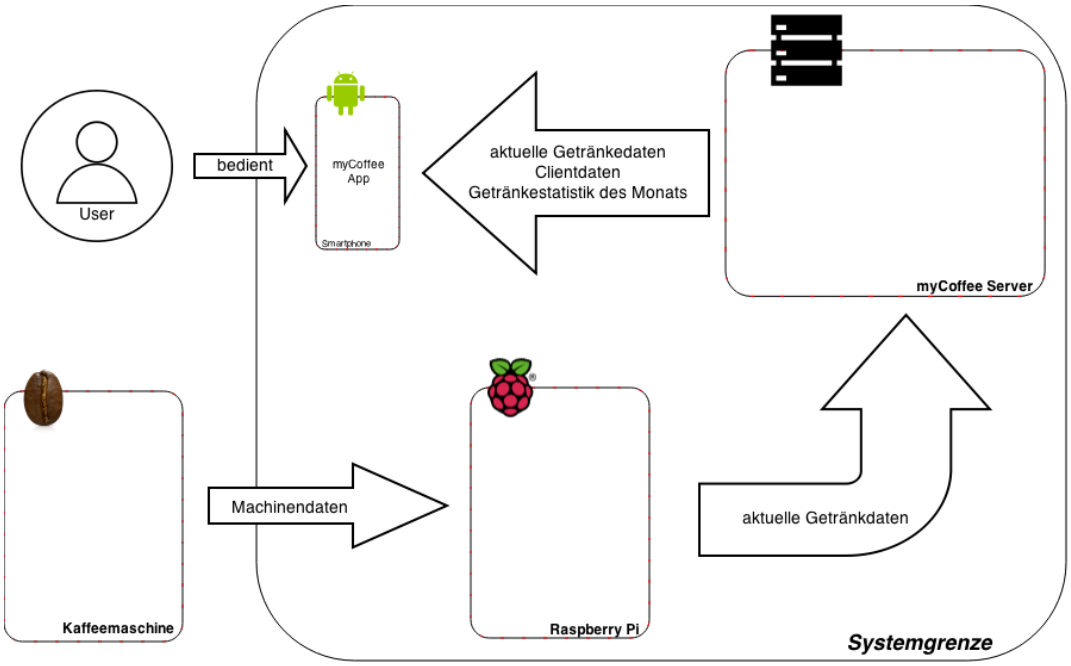

Abbildung 2: Systemübersicht

Rasperry Pi Das Raspberry Pi realisiert die Datenauslesung der Kaffeemaschine. Die empfangenen Maschinendaten werden gefiltert und per Publish-Befehl an den zentralen Server weitergeleitet. Da die Kaffeemaschine keine Events aussenden kann, werden die Daten alle 30s im Polling-Verfahren abgefragt. So kann sichergestellt werden, dass Datenänderungen der Kaffeemaschine mit einer geringen Verzögerung beim Endkunden angezeigt werden.

myCoffee Server Der Server ist das Herzstück dieses Systems. Er regelt die Client-Authentifizierung, empfängt Nachrichten und leitet diese an die entsprechenden Clients weiter. Zusätzlich speichert er die empfangenen Nachrichten sowie den Status aller verbundenen Clients in eine lokale Datenbank. Über entsprechende Befehle können Clients Daten dieser lokalen DB abfragen (siehe Kapitel 3.4.2).

Smartphone Die installierte myCoffee App auf dem Android Smartphone stellt diverse Maschinen- sowie Clientdaten dar. Nach erfolgreichem Login hat der User zugriff auf aktuelle Getränkedaten, sowie auf statistische Getränkedaten des aktuellen Monats. Der User kann die Darstellung nach belieben auf bestimmte Regionen, Maschinen und Produkte beschränken. Besitzt der angemeldete Benutzer Adminrechte, kann er Verbindungsinformationen zu angemeldeten Clients abrufen.

Kommunikation Die Kommunikation zwischen den einzelnen Subsystemen wird durch das MQTT-Protokoll realisiert.

3.2. Komponentenübersicht

Um die Komplexität zu minimieren und die Austausch- und Wartbarkeit zu erhöhen, wurde die Software der einzelnen Subsysteme in einzelne Komponenten heruntergebrochen. Folgende Diagramme visualisieren diese Softwarekomponenten und liefern ein detaillierteres Verständnis der Funktionsweise der einzelnen Subsysteme.

3.2.1. Raspberry Pi

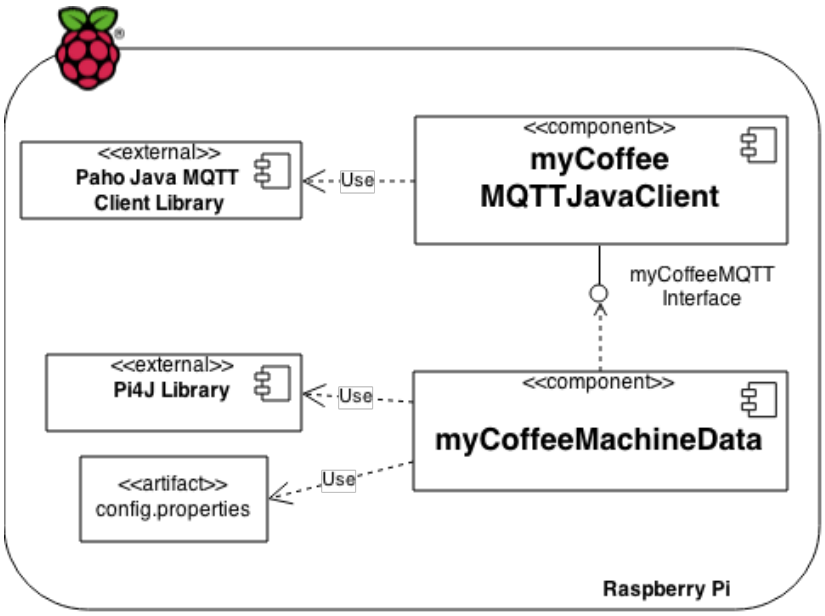

Abbildung 3: Softwarekomponenten Rasperry Pi

myCoffeeMachineData Die *MachineData* Komponente realisiert die serielle Kommunikation mit der Kaffeemaschine. Dazu wird die externe Library Pi4J benötigt, welche eine API für das Rasperry Pi zur seriellen Kommunikation zur Verfügung stellt (Pi4J, 2014). Weiter übernimmt *MachineData* das parsen der gelesenen Maschinendaten.

myCoffeeMQTTJavaClient Die *MQTTJavaClient* Komponente implementiert das *myCoffeeMQTT* Interface und stellt somit alle MQTT Funktionalitäten zu Verfügung (siehe Kapitel 3.4.1). Um dies zu realisieren wird die *Paho Java MQTT Client Library* benötigt, welche ein MQTT API zur Verfügung stellt (JavaClient, 2014). Da die *MQTTJavaClient* Komponenten ansonsten keine Abhängigkeiten hat, kann sie beliebig wiederverwendet werden.

Property File In diesem Property File können MQTT- sowie Maschinendaten angepasst werden, ohne den SourceCode zu verändern.

3.2.2. myCoffee Server

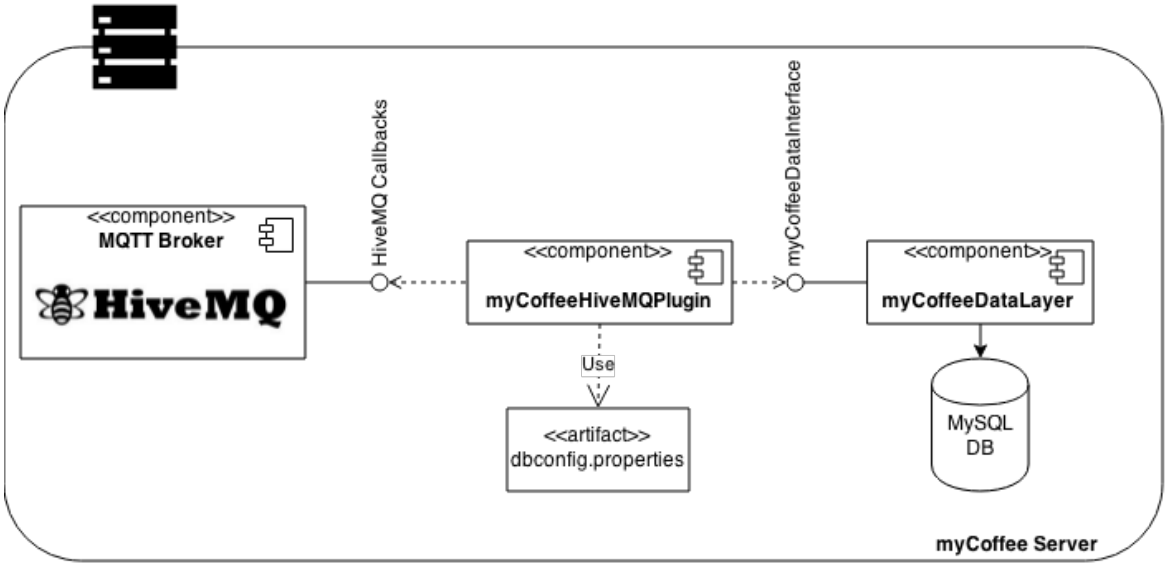

Abbildung 4: Komponentenübersicht Server

HiveMQ HiveMQ ist eine auf Java basierende Implementation eines MQTT Broker von dcSquare (dcSquare, 2014). Empfängt Nachrichten von Clients und leitet diese an die entsprechenden Abonnenten weiter. Neben allen Broker Funktionalitäten bietet HiveMQ auch die Möglichkeit, erweiterte Funktionen durch Plugins zu implementieren.

myCoffeeHiveMQPlugin Verwendet die vom MQTT Broker zur Verfügung gestellten Callbacks. Erweitert den Broker um folgende Funktionen:

- Clientauthentifizierung
- Persistierung von Empfangenen Nachrichten
- Persistierung von Clientverbindungen
- Auslesen von persistierten Daten durch Clients

myCoffeeDataLayer Stellt Funktionalitäten zum Schreiben sowie Lesen von Datenbankeinträgen zur Verfügung.

Property File In diesem Property File können die Verbindungsdaten der Datenbank angepasst werden, ohne den SourceCode zu verändern.

MySQL Datenbank In der Datenbank werden Empfangene Nachrichten, User-Credentials sowie Clientinformationen gespeichert.

3.2.3. myCoffee App

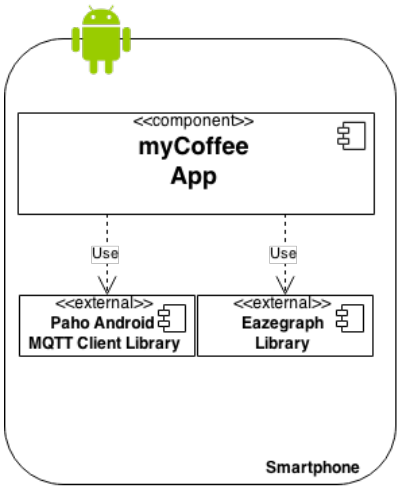

Abbildung 5: Komponentendiagramm myCoffee App

myCoffee App

Die myCoffee App realisiert die komplette Android Applikation. Damit die Applikation MQTT Client Funktionen implementieren und nutzen kann, wird die externe Paho Android MQTT Client Library verwendet (AndroidClient, 2014). Um Graphen und Statistiken visuell aufzuzeigen, wird die Eazegraph Library von blackfizz verwendet (Cech, 2014).

3.3. Klassenübersicht

Folgende Klassendiagramme liefern eine detaillierte Übersicht, wie die Software der einzelnen Komponenten und Subsysteme umgesetzt wurde. Übersichtshalber sind nur die wesentlichen Klassen mit den wichtigsten Attributen und Methoden aufgeführt. Weitere Informationen sind der Javadoc sowie dem Source Code zu entnehmen (Siehe Anhang D).

3.3.1. Raspberry Pi

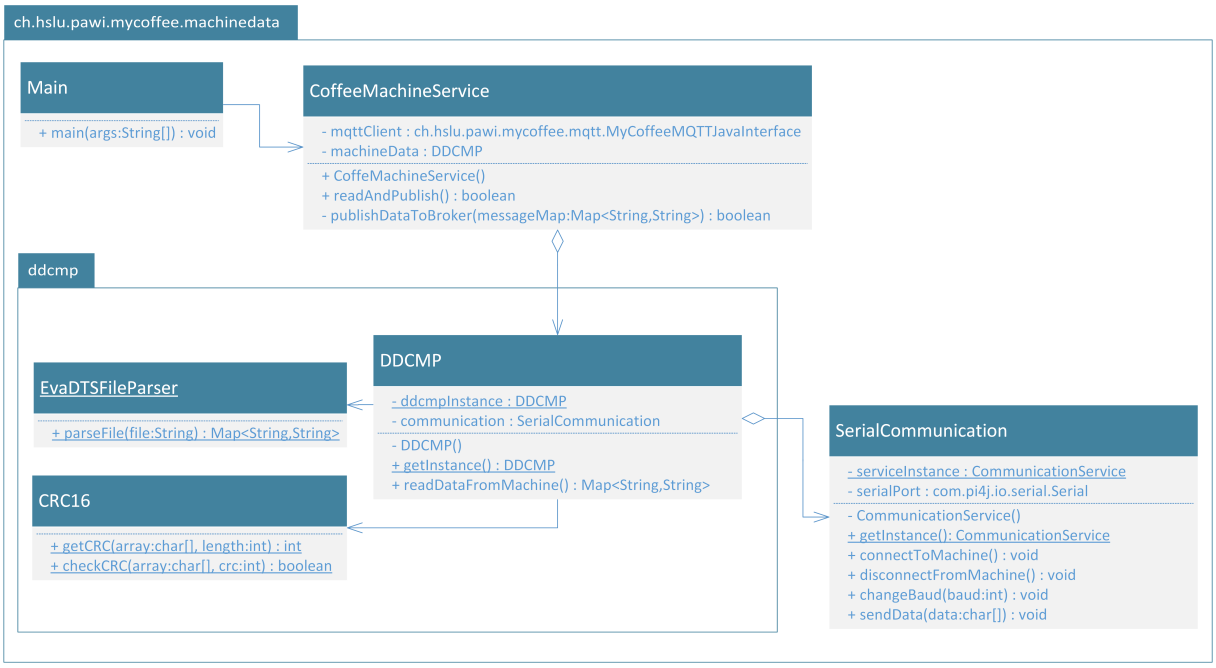

Abbildung 6: Klassendiagramm MachineData-Komponente

CoffeeMachineService Hauptklasse dieser Komponente. Sie dient primär der Abstraktion der zugrunde liegenden DDCMP sowie MQTT Implementation.
Mit Hilfe der *DDCMP* Klasse werden die Maschinendaten ausgelesen und anschliessend über die *MQTTJavaClient*-Komponente an den Broker weitergeleitet. Dies wird durch den Aufruf der *readAndPublish()* Methode erreicht.

DDCMP Stellt die Implementation des DDCMP-Protokolls dar. Über die Methode *readDataFromMachine()* wird die Kommunikation mit der Maschine initialisiert. War diese Erfolgreich, wird das EVA DTS File gelesen und geparst. Weitere Informationen zum DDCMP Protokoll sind dem Anhang E zu entnehmen.

EvaDTSFileParser Extrahiert die für dieses Projekt gebrauchten Informationen aus einem EVA DTS File und speichert diese in einer Map. Key ist der entsprechende Getränkename und Value repräsentiert die Anzahl entnommenen Getränke des entsprechenden Produkts. Weitere Informationen zum EVA DTS Format sind dem Kapitel 4.1 zu entnehmen.

CRC16 Berechnet und Überprüft die CRC16 Checksumme eines Bytepackets. Wird für das DDCMP Protokoll benötigt.

SerialCommunication Stellt Funktionen für die serielle Kommunikation zur Verfügung.

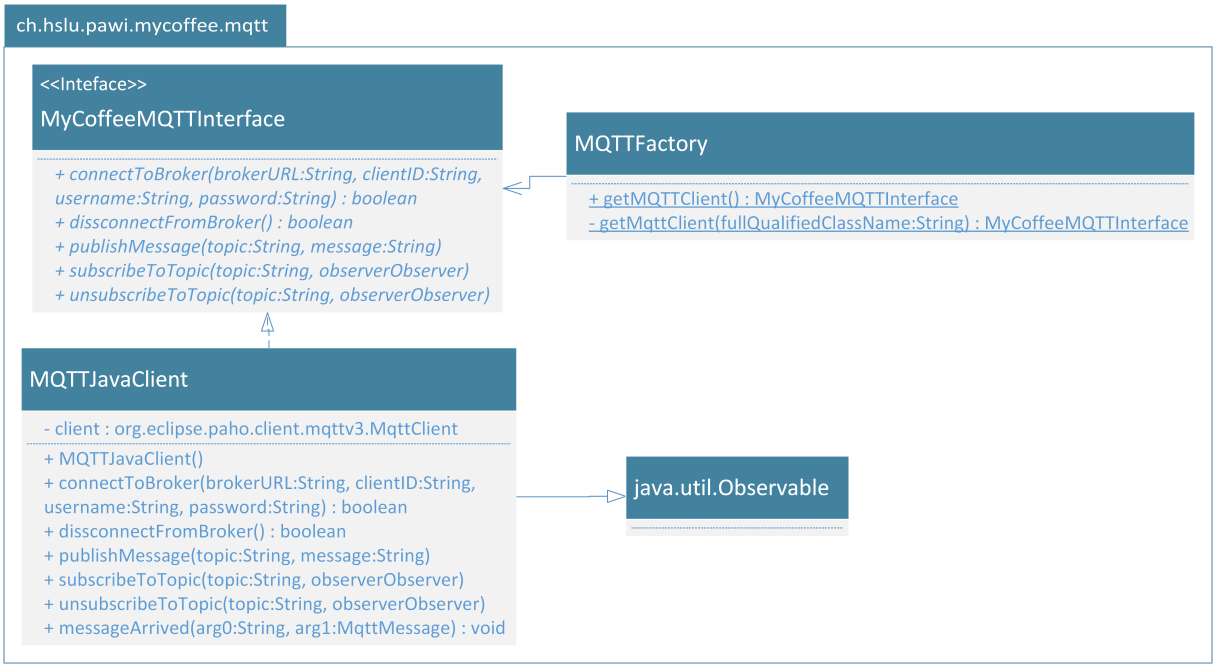

Abbildung 7: Klassendiagramm MQTTJavaClient-Komponente

MyCoffeeMQTTInterface Stellt dem Benutzer die API der *myCoffeeMQTTJavaClient*-Komponente zur Verfügung.

MQTTFactory ClassLoader für diese Komponente. Somit muss diese nicht direkt instanziert werden, sondern kann dynamisch zur Laufzeit geladen werden. Dies erhöht die Austauschbarkeit dieser Komponente.

MQTTJavaClient Implementiert das *MyCoffeeMQTT* Interface und stellt alle MQTT Funktionen zur Verfügung. Erbt von der Java Klasse *Observable*. Sobald sich ein Objekt mit *subscribeToTopic* für ein Thema registriert, wird dieses als Observer hinzugefügt. Jedes Mal wenn eine Nachricht mit dem entsprechenden Thema von *messageArrived* empfangen wird, wird der entsprechende Observer automatisch benachrichtigt und muss sich nicht selber um das abholen von neuen Nachrichten kümmern.

3.3.2. myCoffee Server

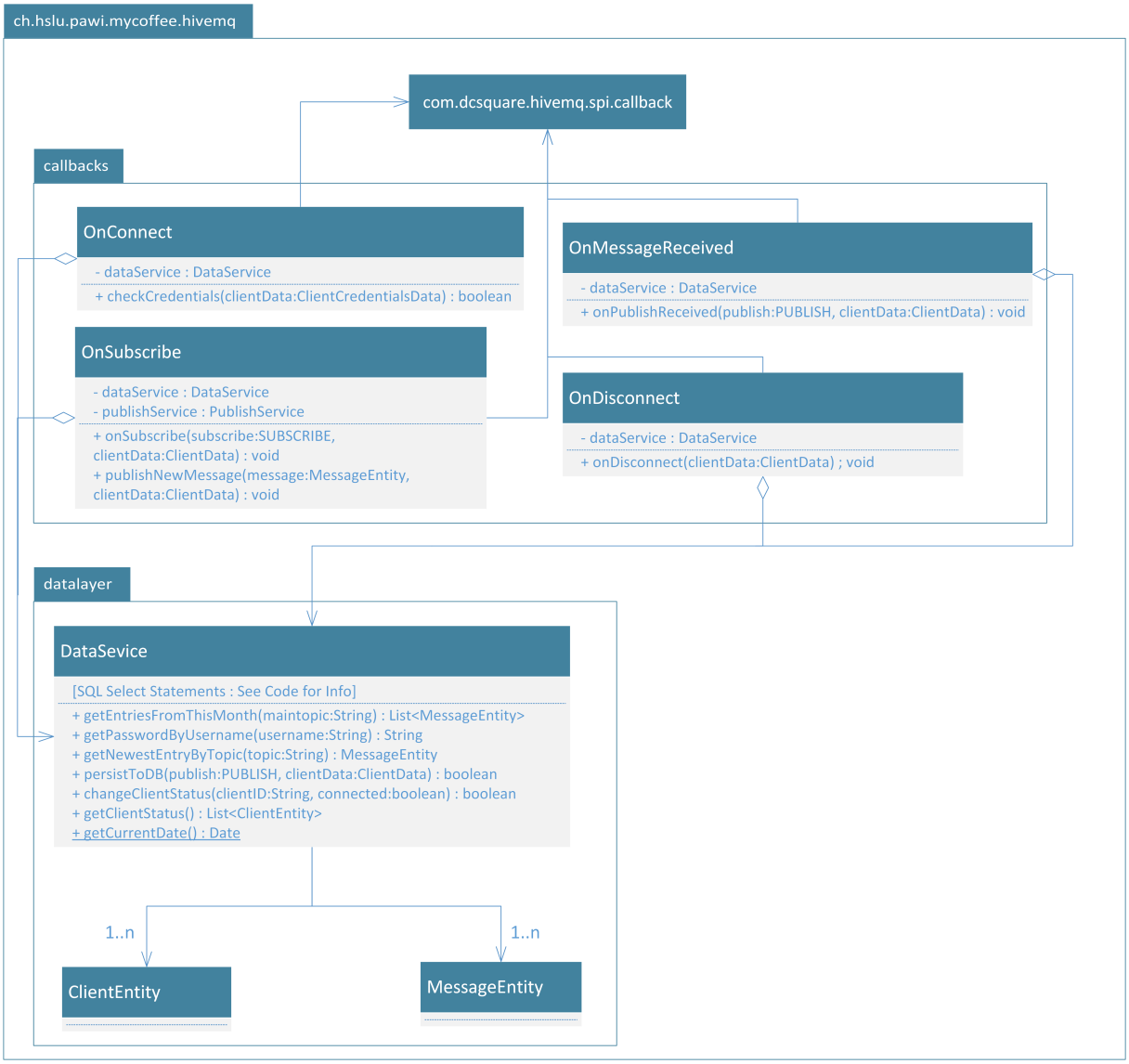

Abbildung 8: Klassendiagramm MyCoffeeHiveMQPlugin

OnConnect Wird automatisch aufgerufen, sobald sich ein User beim Server anmelden will. Überprüft die eingegeben Userdaten mit den Daten der DB. Falls diese korrekt sind, ist der User mit dem Server verbunden und der Verbindungsstatus des entsprechenden Clients wird in der Datenbank auf *connected* geändert.

OnDisconnect Wird automatisch aufgerufen, sobald sich ein User vom Server abmeldet. Der Verbindungsstatus des entsprechenden Clients wird in der Datenbank auf *dissconnected* geändert.

OnMessageReceived Wird automatisch aufgerufen, sobald eine Nachricht empfangen wird. Es wird überprüft, ob eine gleiche Nachricht vom selben Datum und Client bereits in der Datenbank ist, falls dies nicht der Fall ist, wird diese in der Datenbank persistiert. So wird sichergestellt, dass nur ein selber Eintrag pro Tag vorhanden ist.

OnSubscribe	Wird automatisch aufgerufen, sobald ein Client ein Thema abonniert. Falls es sich bei dem Thema um einen Befehl handelt, welcher der Broker ausführen soll, werden die entsprechenden Daten von der Datenbank geholt und mit *publishNewMessage* an den Client weitergeleitet. Die entsprechenden Befehle, welche der Broker verarbeitet, sind dem Kapitel 3.4.2.3 zu entnehmen.
DataService	Stellt der darüber liegenden Schicht Funktionen zum Lesen sowie Schreiben von Datenbankeinträgen zur Verfügung.
ClientEntity	Stellt ein Eintrag der Datenbanktabelle *Client* dar.
MessageEntity	Stellt ein Eintrag der Datenbanktabelle *Message* dar.

3.3.3. myCoffee App

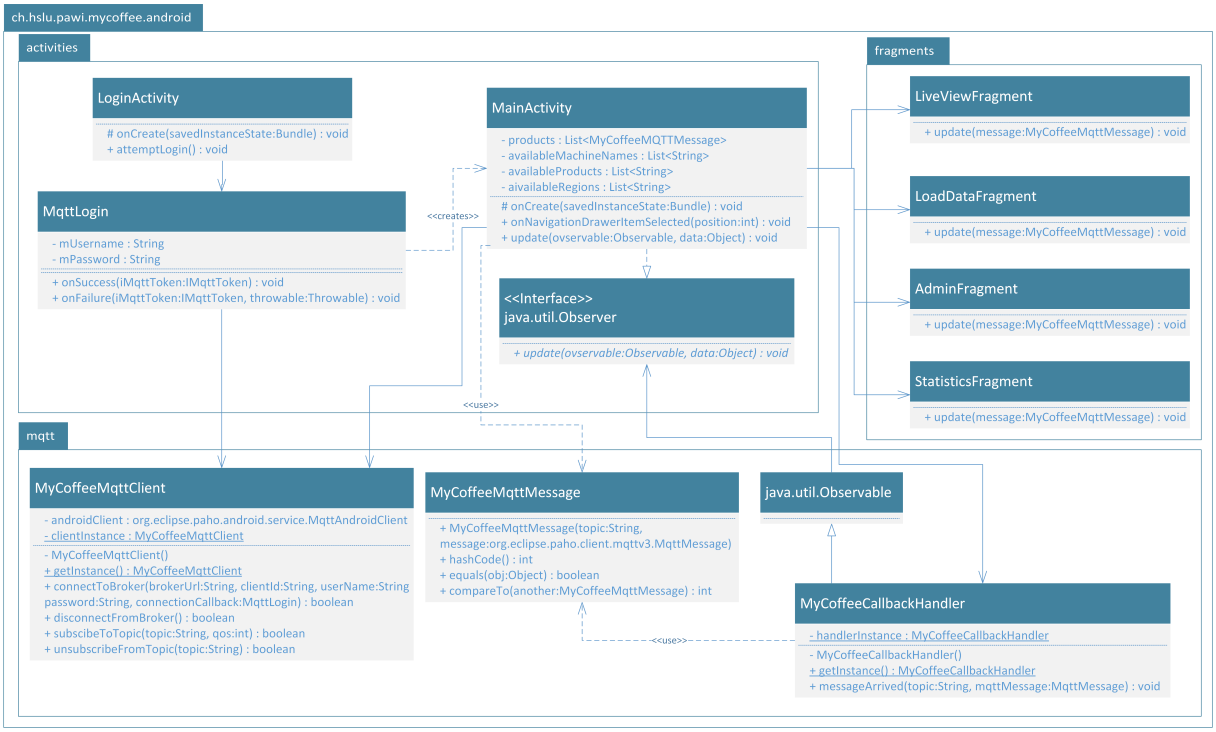

Abbildung 9: Klassendiagramm Android App

LoginActivity

Hauptaktivität, welche beim Start der Android App angezeigt wird. Präsentiert dem User zwei Textfelder, in welche er die fürs Login erforderlichen Daten eintragen kann. Entspricht dem Screen 1 in Kapitel 5.1.

MqttLogin

Wird aufgerufen, sobald der User das Login mit dem entsprechenden Button bestätigt. Läuft asynchron und stellt mit Hilfe des *MyCoffeeMqttClient* eine Verbindung zum Broker her und übermittelt die eingegebenen Anmeldedaten. Implementiert die *IMqttActionListener onSuccess* und *onFailure*, welche entsprechend der Broker Antwort aufgerufen werden. War der Loginversuch erfolgreich, sprich wird *onSuccess* aufgerufen, wird die *MainActivity* erstellt und angezeigt.
Wird *onFailure* aufgerufen, wird dies dem Benutzer mitgeteilt und er kann erneut einen Loginversuch starten.

MainActivity

Ist für die Verwaltung der einzelnen Fragmente zuständig. Bei den Fragmenten handelt es sich um entsprechende Screens, welche dem User abhängig vom aktuellen Menüpunkt, angezeigt werden. *MainActivity* registriert sich zu Beginn beim *MyCoffeeCallbackHandler* als Observer und empfängt so automatisch alle eingehenden MqttMessages, welche anschliessend an das aktive Fragment weitergeleitet werden. Verwaltet die von den Fragmenten gemeinsam genutzten Daten.

LoadDataFragment

Wird automatisch nach erfolgreichem Login aufgerufen. Ist für das Laden der Serverdaten vom aktuellen Monat verantwortlich. Kann vom User über keinen Menüpunkt aufgerufen werden. Entspricht dem Screen 2 in Kapitel 5.2.

LiveViewFragment	Wird automatisch nach erfolgreichem Laden der Daten aufgerufen. Ist für die Anzeige von den aktuellsten empfangenen Daten verantwortlich. Wird bei jedem Empfang von neuen Daten aktualisiert. Der User kann die Produktanzeige einschränken (siehe Screen 7 und 8 in Kapitel 5.7 und 5.8). Kann über den Menüpunkt *Live Data* aufgerufen werden (siehe Screen 6 in Kapitel 5.6). Entspricht dem Screen 3 in Kapitel 5.3.
StatisticsFragment	Ist für die Anzeige von statistischen Daten verantwortlich. Präsentiert dem User entsprechend den in Screen 7 und 8 in Kapitel 5.7 und 5.8 zu sehenden Einträge unterschiedliche Statistiken des aktuellen Monats. Wird über den Menüpunkt *Statistics* aufgerufen (siehe Screen 6 in Kapitel 5.6). Entspricht dem Screen 4 in Kapitel 5.4.
AdminFragment	Ist für die Anzeige von administrativen Daten verantwortlich. Dabei handelt es sich um Clientinformationen inklusive Verbindungsstatus. Wird über den Menüpunkt *Admin Settings* aufgerufen (siehe Screen 6 in Kapitel 5.6). Kann nur aufgerufen werden, falls der angemeldete User Admin Rechte besitzt. Entspricht dem Screen 5 in Kapitel 5.5.
MyCoffeeMqttClient	Stellt der Android Applikation alle MQTT Funktionen zur Verfügung. Verwendet dazu die Paho Android MQTT Library.
MyCoffeeMqttMessage	Repräsentiert eine empfangene MQTT Nachricht angepasst auf dieses Projekt.
MyCoffeeCallbackHandler	Implementiert die von der Paho MQTT Library zur Verfügung gestellten Callbacks. Wird insbesondere dann aufgerufen, wenn eine neue MQTT Nachricht empfangen wird. Erbt von Observable, damit sich Observer hier anmelden können, und so automatisch die empfangenen Nachrichten erhalten.

3.4. Schnittstellen

Damit die einzelnen Komponenten und Subsysteme miteinander agieren können, wurden folgende Schnittstellen definiert.

3.4.1. myCoffeeMQTT Interface

Dieses Interface legt fest, über welche Methoden die *myCoffeeMQTTJavaClient*-Komponente verwendet werden kann.

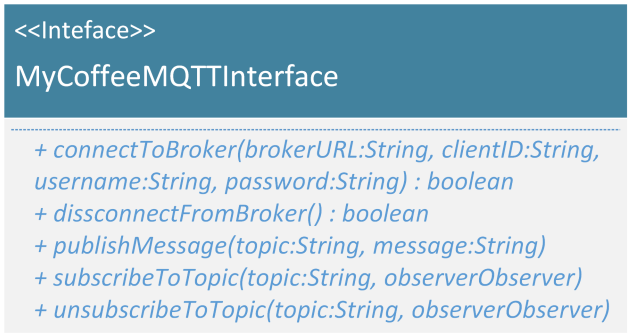

Abbildung 10: MyCoffeeMQTTInterface

Die entsprechenden Methoden sind der Abbildung 10 zu entnehmen.
Eine Beispielverwendung sieht wie folgt aus:

```
// Holt eine Instanz von der MQTTFactory
MyCoffeeMQTTJavaInterface mqttClient = MQTTFactory.getMqttClient();

// Stellt die Verbindung zum Broker her
mqttClient.connectToBroker("tcp://test.org:8000", "Test Client", "user", "1234");

// Publiziert eine Nachricht mit Topic "hello/world" und Inhalt "Hello World"
mqttClient.publishMessage("hello/world", "Hello World");

// Schliesst die Verbindung zum Broker
mqttClient.disconnectFromBroker();
```

3.4.2. MQTT Kommunikation

Zur Kommunikation zwischen Client und Server wird das MQTT Protokoll verwendet. MQTT basiert auf dem Publish/Subscribe-Verfahren und hat einen geringen Overhead. Ein Client kann beim Server ein bestimmtes Thema per Subscribe abonnieren. Publiziert ein anderer Client eine Nachricht unter diesem Thema, Empfängt dies der entsprechende Client automatisch. Weitere Informationen zum MQTT Protokoll sind dem Kapitel 4.3.1.2 zu entnehmen.

Damit die Clients die entsprechenden Themen abonnieren können, müssen diese bekannt sein. Daher werden sie wie folgt festgelegt.

3.4.2.1. Publisher Nachrichten (Raspberry Pi)

Die vom Publisher versendeten Nachrichten sehen wie folg aus:

Topic

```
myCoffee/region/city/machineName/productName
```

Payload:

```
Konsumierte Produkte als String
```

Beispiel:

```
Topic:   myCoffee/Luzern/Horw/Primus Prime 012345/Coffee
Payload: „10“
```

3.4.2.2. Subscriber Nachrichten

Der Subscriber abonniert durch Angabe des Topics die entsprechenden Nachrichten.
Dabei hat er mehrere Möglichkeiten, ein bestimmtes Thema zu abonnieren.

Spezifische Abonnierung:

```
myCoffee/region/city/machineName/productName
```

Durch diese Art kann genau angegeben werden, welches Topic abonniert werden möchte. Es werden nur Nachrichten von genau diesem Topic erhalten.

Wildcard Abonnierung mit +:

```
myCoffee/+/+/+/productName
```

Durch ein + wird symbolisiert, dass vom entsprechenden Teil des Topics alle Nachrichten ohne Einschränkung erhalten werden. Dies wird lediglich durch auf ein + folgende, spezifische Topicangaben eingeschränkt.

Wildcard Abonnierung mit #:

```
myCoffee/#
```

Durch Angabe einer # werden alle möglichen Topics abonniert, mit Begrenzung auf die Angaben vor der #.

3.4.2.3. MQTT Server Nachrichten

Durch das myCoffeeMQTTPlugin kennt der Server folgende Topics auf welche er mit entsprechenden MQTT Nachrichten antwortet:

1. control/myCoffee/ThisMonth
2. control/Clients

1. Liefert die Datenbankeinträge aller Produkte mit dem Hauptthema *myCoffee* vom aktuellen Monat zurück. Der Inhalt einer auf diese Anfrage versendete Nachricht sieht wie folgt aus:

dispenseValue/date/mainTopic/machineRegion/machineCity/machineName/productName

Beispiel:

10/2014u 11u 05/myCoffee/Luzern/Horw/Primus Prime 012345/Coffee

Dies bedeutet, dass am *05. November 2014* seit in Betriebnahme der *Primus Prime* mit Seriennummer *01234* in *Horw* im Kanton *Luzern* genau *10* Getränke des Typs *Coffee* entnommen wurden.

2. Liefert die Datenbankeinträge aller jemals an diesen Server angemeldeten Clients mit Verbindungsstatus zurück. Der Inhalt einer auf diese Anfrage versendete Nachricht sieht wie folgt aus:

clientName/timeStamp/connectionState

Beispiel:

1. Primus Prime 012345/2014u 11u 05 14:05:22/0
2. Galaxy Nexus/2014-11-22 09:35:57/1

1. Bedeuten, dass sich die Primus Prime 012345 zuletzt am 05. November 2014 um 14:05:22 angemeldet hat und momentan offline ist.
2. Bedeutet, dass sich das Galaxy Nexus zuletzt am 22. November 2014 um 09:35:57 angemeldet hat und momentan online ist.

3.5. Abläufe

Folgende Sequenzdiagramme visualisieren die Kommunikation sowie die Abläufe zwischen den einzelnen Komponenten und Subsysteme.

3.5.1. Login und Lesen von Datenbankeinträgen

Folgendes Sequenzdiagramm zeigt den detaillierten Ablauf vom Login eines myCoffee-App Users inkl. Lesen von Datenbankeinträgen:

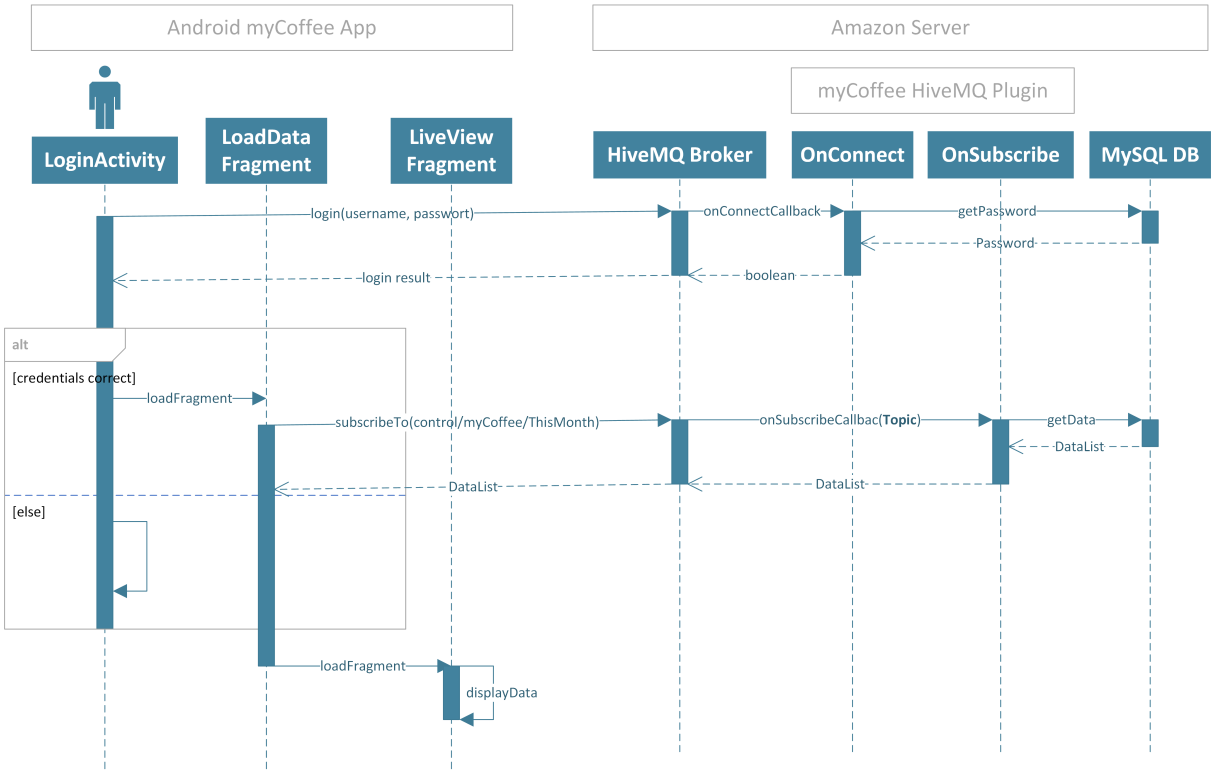

Abbildung 11: Sequenzdiagramm Login inkl. Lesen von DB Einträgen

Um das myCoffee-App nutzen zu können, muss sich der User mit den entsprechenden Daten einloggen. Diese werden anschliessend dem Server übermittelt. Die Implementierung des *OnConnect*-Callbacks überprüft anschliessend die eingegebenen Daten mit den in der Datenbank gespeicherten. Das Resultat wird der App zurückgeliefert und dem User präsentiert. Sind die Daten nicht korrekt, wird dies durch eine Warnung angezeigt und der User kann erneut einen Loginversuch starten. Sind sie korrekt, wird automatisch das *LoadData-Fragment* gestartet. Dieses abonniert beim Server das Thema *control/myCoffee/ThisMonth*. Die Implementierung des OnSubscribe-Callbacks holt alle Einträge des aktuellen Monats aus der Datenbank und sendet diese zurück an den Client. Dieser wechselt automatisch zum *LiveView-Fragment*, welches die geladenen Daten anzeigt.
Dieser Ablauf ist bei jedem Start der myCoffee Smartphone Applikation identisch. Das einmalige Laden und speichern aller Einträge des aktuellen Monats verbessert die Geschwindigkeit sowie Userfreundlichkeit der App. So müssen diese nicht bei jedem Screen-Wechsel nachgeladen werden.

3.5.2. Abonnieren und Empfang von Live Daten

Wenn sich ein Smartphone Client beim Broker für ein Thema registriert, empfängt er automatisch alle Nachrichten, welche von einem anderen Client unter dem selben Thema publiziert werden. Den genauen Ablauf visualisiert folgendes Sequenzdiagramm:

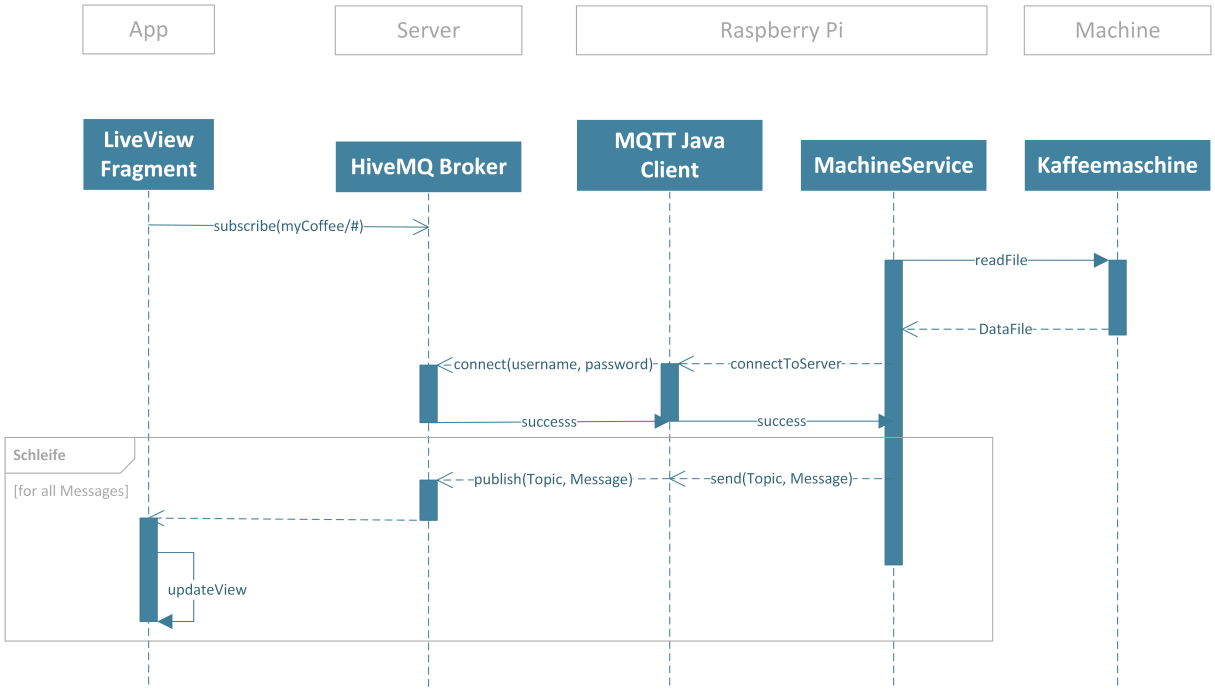

Abbildung 12: Sequenzdiagramm Abonnierung und Empfang von Live Daten

Nachdem sich der User erfolgreich angemeldet hat und alle Daten vom Server empfangen wurden, wechselt die Applikation automatisch in die *LiveView*. Dort werden zu Beginn die aktuellsten Daten der vorhin geladenen Monatsstatistik angezeigt. Dabei handelt es sich also nicht um Live Daten der Maschine, sondern um die aktuellsten Einträge in der Datenbank. Anschliessend abonniert die myCoffee-App automatisch das Thema *myCoffee/#* beim Broker. Somit werden alle Nachrichten, welche mit dem Thema *myCoffee* beginnen, automatisch vom Broker zu diesem Client weitergeleitet. Ist die Kaffeemaschine eingeschaltet, werden die Daten alle 30s vom Raspberry Pi ausgelesen. Nach erfolgreichem Login des Raspberry Pis, publiziert dieses die Daten unter dem entsprechenden Topic, welches mit *myCoffee* beginnt. Diese werden automatisch zum Smartphone weitergeleitet, welches die Daten in der LiveView aktualisiert. Von diesem Zeitpunkt an handelt es sich wirklich um Live Daten der Kaffeemaschine, welche alle 30s aktualisiert werden.

3.5.3. Persistierung von Clientinformationen

Folgendes Sequenzdiagramm zeigt den detaillierten Ablauf zur Persistierung von Clientinformationen:

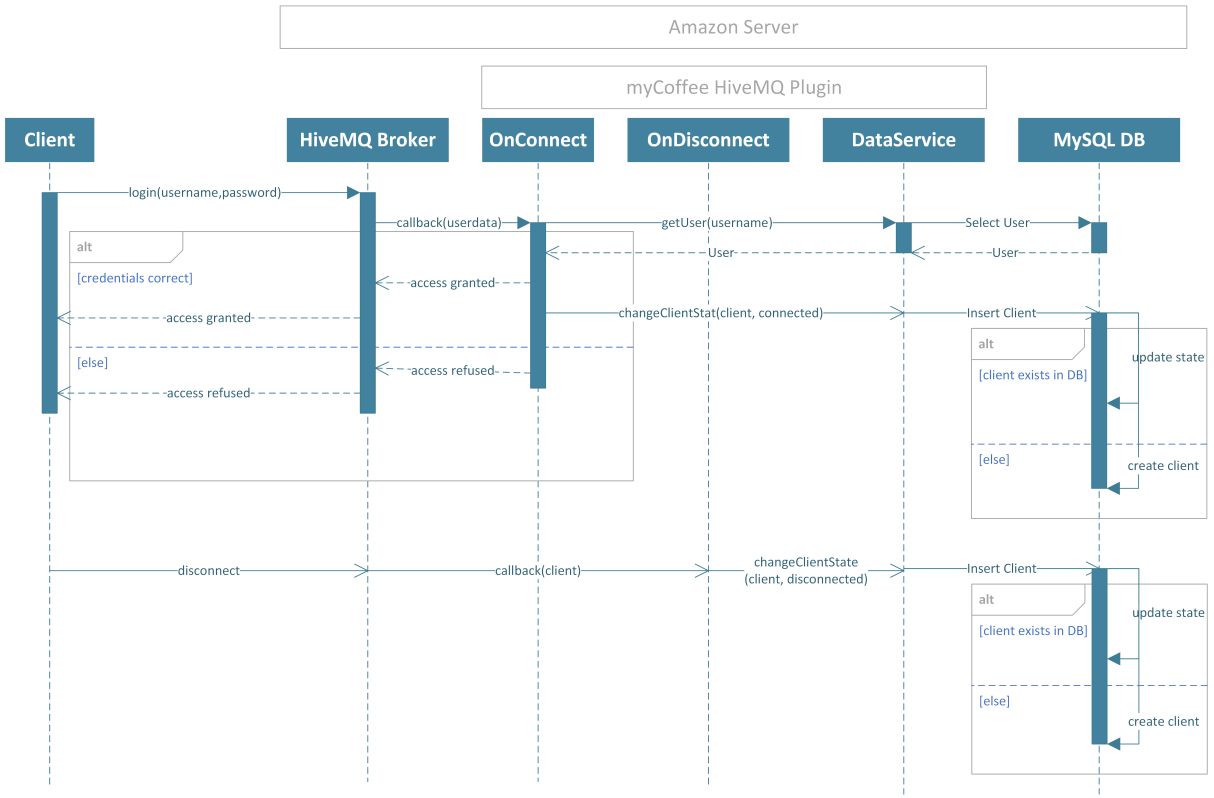

Abbildung 13: Sequenzdiagramm Persistierung von Clientinformationen

Meldet sich ein User beim *HiveMQ Broker* an, wird automatisch das *OnConnect-Callback* aufgerufen. Sind die Userdaten korrekt, wird der Client in der entsprechenden Datenbank-Tabelle erfasst bzw. der Status aktualisiert. Meldet sich dieser Client wieder ab, wird der Verbindungsstatus durch die Implementierung des *OnDisconnect-Callbacks* in der Datenbank aktualisiert.
Besitzt ein myCoffee-App User Administrator Rechte, kann er diese gespeicherten Clientinformationen inkl. Verbindungsstatus über das Thema *control/Clients* auslesen.

3.6. Datenstrukturen

Zur persistenten Speicherung von Nachrichten sowie Clientinformationen wurde auf dem Server eine MySQL Datenbank eingerichtet. Abbildung 14 zeigt das bewusst einfach gehaltene Datenmodell.

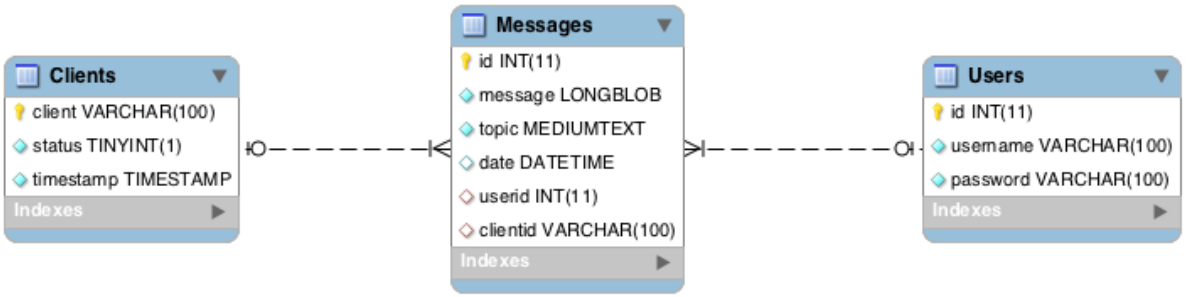

Abbildung 14: Physisches Datenbankmodell

Clients Jeder Client, welcher sich mindesten ein Mal beim Server angemeldet hat, wird in dieser Tabelle gespeichert. Der Primärschlüssel *client* beinhaltet einen eindeutigen Clientnamen. Es können nicht mehrere Clients gleichzeitig mit dem selben Namen beim Server angemeldet sein. Das Feld *status* signalisiert den aktuellen Verbindungsstatus des jeweiligen Clients. Der Status *connected* wird mit einer 1 signalisiert, *disconnected* mit einer 0. Das Feld *timestamp* wird bei jedem Statuswechsel aktualisiert. Somit kann festgestellt werden, zu welchem Zeitpunkt die letzte Interaktion zwischen Client und Server stattgefunden hat.

Messages In dieser Tabelle werden die empfangenen Nachrichten abgespeichert. Jede Nachricht besitzt einen Inhalt und ein Topic, welche durch *message* bzw. *topic* repräsentiert wird. Das Feld *date* beinhaltet das Empfangsdatum der entsprechenden Nachricht. Jede Nachricht kann einem eindeutigen Client und User zugeordnet werden.

Users Damit nicht jedem Client ein eigener Username sowie Passwort zugewiesen werden muss, werden diese in einer separaten Tabelle verwaltet. Gleichzeitig können mehrere Clients mit dem selben User beim Server angemeldet sein.

4. Design Entscheide

Dieses Kapitel beinhaltet die wichtigsten Entscheide, die im Verlaufe des Projekts gefällt wurden inklusive Begründung.

4.1. Datenlieferant

Als Datenlieferant dient eine Kaffeemaschine. Die Maschine besitzt einen RS232 Anschluss, über welchen Maschinendaten ausgelesen werden können. Die konkreten Daten sind der Tabelle 1 zu entnehmen.[1]

Art der Daten	Beschreibung	Beispiele
Getränkestatistik	Von der Maschine unterstützte Getränke inkl. Preis. Entnommene Getränke seit Initialisierung und letztem Reset.	- Espresso - Cappuccino - Macchiato - Milk
Zutatenstatistik[2]	Verbrauchte Zutatenmenge seit Initialisierung und letztem Reset.	- Verbrauchte Wassermenge - Verbrauchte Kaffeemenge
Maschinenstatistik[2]	Maschinenstatistik seit Initialisierung und letztem Reset.	- Einsatzdauer Mühle - Einsatzdauer Pumpe - Anzahl Reinigungen
Verkaufsstatistik	Anzahl verkaufte Getränke seit Initialisierung und letztem Reset.	- Espresso - Cappuccino - Macchiato - Milk
Fehlerstatistik[3]	Die Fehlerstatistik beinhaltet die 20 zuletzt aufgetretenen Fehler. Sind während der letzten Auslesung mehr als 20 Fehler aufgetreten, sind die ältesten nicht ersichtlich.	- Mühle blockiert - Übertemperatur - Element Defekt
Events[3]	Events, welche seit der letzten Auslesung aufgetreten sind inklusive Datum.	- Bohnenbehälter leer - Wasserbehälter leer

Tabelle 1: Maschinendaten

Um an diese Daten zu kommen, muss jeweils ein komplettes File ausgelesen werden. Das entsprechende File liegt im EVA DTS Format vor. Hierbei handelt es sich um einen Standard der European Vending Association, welche für die Spezifizierung von Standards und Protokollen für Vending Maschinen verantwortlich ist.[4]

[1] Detaillierte Informationen zu den Daten sind dem Dokument *Specification_M2M_CoffeeLink_v1_6* auf der CD zu entnehmen.
[2] Die komplette Liste ist dem Dokument *CountersAndStatistics_v0_1* auf der CD zu entnehmen.
[3] Die komplette Liste ist dem Dokument *ErrorsAndEventsList_v1_2* auf der CD zu entnehmen.
[4] Die komplette Spezifikation ist dem *Dokument EVADTS_6-1 _ 04 June 2009* auf der CD zu entnehmen.

Um an das entsprechende File zu gelangen, kommt das DDCMP Protokoll zum Einsatz. Dies ist ein Netzwerkprotokoll, welches eine Punkt zu Punkt Kommunikation ermöglicht. Bei DDCMP handelt es sich um ein umfangreiches Protokoll. Die Implementierung des kompletten Protokolls hätte den Rahmen dieses Projektes gesprengt. Daher wurden lediglich die für die Datenauslesung relevanten Befehle implementiert (Siehe Anhang E).

4.1.1. Beispiel EVA DTS File

Ein EVA DTS File sieht wie folgt aus:

```
DXS*SAR0000001*VA*V0/6*1
ST*001*0001
ID1*0000000000*Prime*2.12**CT1S_2M_2P
ID4*2
ID5*20140925*145741**EU
AM1*01024585*1.2*1 .1*WinCE 6.0 R3
CB1**Power Section*211
VA2*0*0*0*0
VA3*0*10*0*10
PA1*0**Coffee
PA4*4*0*4*0
PA6**Coffee****
PA1*1**Espresso
PA4*2*0*2*0
PA6**Espresso****
PA1*2**Cappuccino
PA4*0*0*0*0
PA6**Cappuccino****
PA1*8**Cappuccino
PA4*2*0*2*0
PA6**Cappuccino****
PA1*10**Milk
PA4*2*0*2*0
PA6**Milk****
SA2*GRINDER_1*0*35
SA2*GRINDER_2*0*36
SA2*POWDER_1*0*0
SA2*POWDER_2*0*0
SA2*TOTAL_WATER*0*4849
EA3*0*20140925*145741**20140905*105341***2*0
EA9*3*10
MA5*OE_240*20140925*145708*FRESH_WATER_TANK_EMPTY
MA5*OE_402*20140925*145709*FRONT_DOOR_OPEN
[...]
G85*3da1
SE*67*0001
DXE*1*1
```

Die für dieses Projekt interessanten Daten sind rot eingefärbt. Sie können wie folgt interpretiert werden:

Produkt Information
PA1*<PRODUCT_IDENTIFIER>*<PRODUCT_PRICE>*<PRODUCT_NAME>
PA4*<FREE_VENDS_SINCE_INITIALIZATION>*0*< FREE_VENDS_SINCE_LAST_RESET>*0
PA6*<BEVERAGE_IDENTIFIER>*<BEVERAGE_NAME>****<BEVERAGE_IDENTIFIER_2>

Die restlichen Maschinendaten werden in diesem Projekt nicht verwendet.

4.2. Datenauslesung und Weiterleitung

Die Kundenanforderung legt fest, dass ein Raspberry Pi mit installiertem Linux zum Einsatz kommen soll.

Damit Maschinendaten ausgelesen werden können, muss das Raspberry Pi per serielle RS232 Schnittstelle mit der Kaffeemaschine verbunden werden. Dies kann auf zwei Arten realisiert werden. Entweder werden die herausgeführten Rx/Tx Pins verwendet, oder aber ein USB zu Serial Converter.
Bei der ersten Variante müsste ein Spannungswandler eingesetzt werden, da das Raspberry Pi nur Pegel von 3,3V verarbeiten kann und die Maschine mit >5V arbeitet. Daher kommt die zweite Variante zum Einsatz. Der USB-Serial Converter kann an einen der vier verfügbaren USB Ports des Raspberry Pi angeschlossen werden.

Zur Weiterleitung der Daten bietet sich der Ethernet Port des Raspberry Pi an. Mit diesem kann eine Internetverbindung hergestellt werden. Um möglichst flexibel zu sein wird zusätzlich ein WiFi Dongle eingesetzt, damit der Internetzugang auch an Orten ohne Ethernet Anschluss möglich ist.

Die Software zur Datenauslesung, Verarbeitung und Weiterleitung welche auf dem Raspberry Pi zum Einsatz kommt, wird in Java geschrieben.

4.3. Daten Empfang, Weiterleitung und Persistenz

Eine Kundenanforderung ist es, die Daten an ein Smartphone zu senden sowie für statistische Zwecke persistent zu speichern.
Die Konkurrenzanalyse hat gezeigt, dass es sich bei einer Architektur mit einem zentralen Server um ein dynamisches und skalierendes Modell zur Erfüllung von genau diesen Anforderungen handelt (siehe Kapitel 5). Zusätzlich müssen Sicherheitsaspekte nur einmal beachtet und implementiert werden. Aus diesen Gründen kommt dieses Architekturmodell zum Einsatz.

Konkret soll der Server die Clientauthentifizierung, die Entgegennahme und Weiterleitung von Daten, sowie die persistente Speicherung dieser übernehmen. Um die konkreten Technologien zu ermitteln, welche diese Aufgaben optimal erfüllen, wurde eine um fassende Variantenanalyse vorgenommen.

4.3.1. Variantenanalyse Kommunikationsprotokoll

Damit die Daten von der Kaffeemaschine zum Kunden gelangen, muss ein Protokoll implementiert werden, welches von allen Teilnehmern verstanden wird. In einer groben Voranalyse wurden bereits mehrere Protokolle analysiert und auf Einsatzmöglichkeit in diesem Projekt geprüft. Als Ergebnis wurden drei Protokolle ausgewählt, welche nachfolgend in einer vertieften Analyse untersucht werden.

4.3.1.1. CoAP

Constrained Application Protocol (CoAP) ist ein Web Protokoll, welches auf der REST Architektur für speicherarme *embedded* Systeme basiert. Das Haupteinsatzgebiet von CoAP ist das Internet of Things sowie die *machine-to-machine* (M2M) Kommunikation. (Tarkoma, 2012, S. 140-141)

Eigenschaften:

- Datenübertragung über UDP
- GET, POST, PUT, DELETE Methoden
- URI Support zur Ressourcenabfrage
- Asynchrone Datenübertragung
- *Datagram Transport Layer Security* (DTLS) Verschlüsselung

Nachrichtentypen:

- **confirmable:** Müssen vom Empfänger mit einem *ACK* bestätigt werden,
- **nonconfirmable:** Basieren auf dem *fire and forget* Verfahren und bieten keine Garantie der erfolgreichen Übermittlung.

Beispiel:

- **Request:** GET coap://server/home/temperature
- **Response:** ACK mit Payload: „22.3C“

4.3.1.2. MQTT

Message Queue Telemetry Transport (MQTT) ist ein nachrichtenorientiertes Transport Protokoll, welches auf dem *publish/subscribe* Prinzip basiert. MQTT wird ähnlich wie CoAP ebenfalls für leichtgewichtige M2M Kommunikation eingesetzt und wird als das Internet of Things Protokoll der Zukunft gehandelt. (Jaffey, 2014)

Eigenschaften:

- Zentraler Server (*Broker*)
- Publisher/Subscriber bauen eine TCP Verbindung zum Broker auf
- Clients können sowohl Publisher, als auch Subscriber sein
- Asynchrone Datenübertragung
- Client Authentifizierung durch Broker möglich (optional)
- SSL/TLS Verschlüsselung

Nachrichtentypen:

- **QoS 0:** fire and forget (Vergleichbar mit CoAP *nonconfirmable*)
- **QoS 1:** delivered at least once (Vergleichbar mit CoAP *confirmable*)
- **QoS 2:** delivered exactly once

Beispiel:

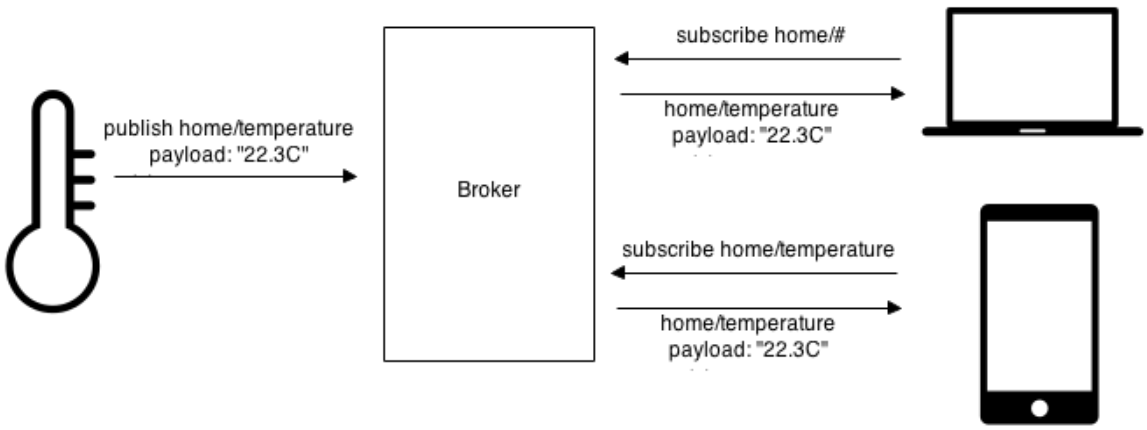

Abbildung 15: MQTT Beispiel

4.3.1.3. XMPP

Extensible Messaging and Presence Protocol (XMPP) ist ein *instant messaging* Protokoll welches weit verbreitet ist. Firmen wie Google, Twitter und Facebook verwenden XMPP. Zum Nachrichtenaustausch wird XML verwendet, was dieses Protokoll nicht so leichtgewichtig macht wie CoAP oder MQTT. (Tarkoma, 2012, S. 193-140)

Eigenschaften:

- Datenübertragung über TCP
- Client-Server oder Peer-to-Peer Kommunikation
- XML Payload
- Authentifizierung über *Simple Authentication and Security Layer* (SASL)
- SSL/TLS Verschlüsselung

4.3.1.4. Protokollvergleich

Die detaillierte Analyse der Protokolle hat gezeigt, dass alle drei aufgrund ihrer Funktionen in diesem Projekt eingesetzt werden könnten. Daher gilt es abzuschätzen, welches die Kundenanforderungen am optimalsten erfüllt.

	CoAP	**MQTT**	**XMPP**
1. Transportprotokoll	UDP	TCP	TCP
2. Architektur	peer-to-peer	Client/Server	peer-to-peer Client/Server
3. Kommunikationsmodell	Request/Response	Publish/Subscribe	Publish/Subscribe Status updates Service discovery Alerts
4. Payload	Userdefiniert	Userdefiniert	XML
5. Leichtgewichtig	Sehr	Sehr	Mittel
6. Sicherheit	Mittel	Hoch	Sehr hoch
7. Library Implementationen	iOS, Android, Java, Javascript, Python, C, C#	iOS, Android, Actionscript, C, C++, Clojure, Dart, Delphi, Erlan, Go, Haskell, Java, Javascript, C#, Python...	iOs, Android, C#, Java, C, C++, Python, Ruby, Erlang, Lisp, Javascript...

Tabelle 2: IoT Protokollvergleich

4.3.1.5. Ergebnis Variantenanalyse Kommunikation

Damit der Entscheid auf ein konkretes Protokoll gefällt werden kann, werden die einzelnen Eigenschaften Gewichtet. Die Nummern der Kriterien sind der Tabelle 2 zu entnehmen.

Kriterien Nr.	Gewichtung	CoAP			MQTT			XMPP		
		Eigenschaft	**Note**	**N*G**	**Eigenschaft**	**Note**	**N*G**	**Eigenschaft**	**Note**	**N*G**
1	**1**	UDP	1	1	TCP	10	10	TCP	10	10
2	**10**	p-t-p	1	10	C-S	10	100	p-t-p/C-S	10	100
3	**10**	R/R	1	10	P/S	10	100	P/S	10	100
4	**1**	Userdefined	10	10	Userdefined	10	10	XML	1	1
5	**10**	Sehr	10	100	Sehr	10	100	Mittel	1	10
6	**1**	Mittel	1	1	Mittel	1	1	Hoch	10	10
7	**1**	Java	10	10	Java	10	10	Java	10	10
Total		142			331			241		

Tabelle 3: Gewichtung Kommunikationsprotokolle

Die Gewichtung hat gezeigt, dass das MQTT Protokoll am besten für dieses Projekt geeignet ist. Daher wurde dieses zur Kommunikation zwischen Servern und Clients verwendet.

4.3.2. MQTT Broker

Da zur Kommunikation das MQTT Protokoll eingesetzt wurde, musste ein MQTT Broker evaluiert werden. Bei diesem Broker handelt es sich um eine zentrale Serverinstanz, welche alle MQTT spezifischen Aktionen zwischen Client und Server übernimmt. Konkret melden sich die Clients beim Broker an um anschliessend Nachrichten zu publizieren und/oder Themen zu abonnieren. Der Broker verwaltet diese Nachrichten und leitet sie an die entsprechenden Abonnenten weiter.

Da es bereits unzählige Broker Implementationen in verschiedenen Programmiersprachen gibt, wird in diesem Projekt keine neue implementiert. Die Wahl ist auf den HiveMQ Broker gefallen (dcSquare, 2014). Dieser komplett in Java implementierte Broker bringt den Vorteil mit sich, dass er durch Plugins erweitert werden kann. Da für dieses Projekt die Broker Funktionalitäten erweitert werden mussten, konnte dies bequem über Plugins realisiert werden. Einziger Nachteil von HiveMQ ist, dass er nur bis 25 gleichzeitig verbundenen Clients kostenlos ist. Sind mehr Clientverbindungen gewünscht, muss eine einmalige Gebühr bezahlt werden.

Um den HiveMQ Broker in diesem Projekt zu betreiben, musste ein passender Server gefunden werden. Naheliegend wäre gewesen, einen schulinternen Server zu verwenden. Einziger Nachteil wäre jedoch gewesen, dass ein externer Zugriff nur per VPN möglich gewesen wäre. Daher ist die Wahl auf eine Amazon EC2 Serverinstanz mit installiertem Ubuntu gefallen. Dies aus dem Grund, da sie für dieses Projekt genügend Rechenleistung und Speicherplatz zur Verfügung stellt und für ein Jahr kostenlos ist.

4.4. Daten Visualisierung

Zur Visualisierung der Daten kommt ein Smartphone zum Einsatz. Die Zielplattform für die Applikation wurde vom Kunden nicht festgelegt. Um die optimale Plattform für dieses Projekt zu ermitteln, wurde eine Variantenanalyse zur Plattformwahl durchgeführt.

4.4.1. Variantenanalyse Smartphone Plattform

Grundsätzlich gibt es drei Plattformen die in Frage kommen, nämlich Android, iOS und Windows Phone. Android untersützt Java, iOS benutzt Objective C oder Swift und für Windows Phone kann unter anderem in C++ oder C# entwickelt werden. Eine weitere Möglichkeit wäre Xamarin. Dabei handelt es sich um ein Framework, mit welchem in C# geschriebener Code für alle drei Plattformen wiederverwendet werden kann. Lediglich das User Interface muss für jede Plattform separat implementiert werden.

	Android	**iOS**	**Windows Phone**
1. Programmiersprache	Java	Objectiv C, Swift	C++, C#
2. Entwickler	Google	Apple	Microsoft
3. Verfügbare Bibliotheken	Viele	Viele	Mittel
4. Verbreitung	Hoch	Hoch	Niedrig
5. Programmiererfahrung	Keine	Keine	Keine

Tabelle 4: Vergleich Smartphone Plattformen

4.4.1.1. Entscheid Smartphone Plattform

Damit der Entscheid auf eine konkrete Plattform getroffen werden kann, wurden die Eigenschaften mit Gewichtung belegt und miteinander verglichen.

Kriterien Nr.	**Gewichtung**	**Android**			**iOS**			**Windows Phone**		
		Eigenschaft	**Note**	**N*G**	**Eigenschaft**	**Note**	**N*G**	**Eigenschaft**	**Note**	**N*G**
1	**10**	Java	10	100	Objective-C Swift	1	10	C++, C#	10	100
3	**1**	Viele	10	10	Viele	10	10	Mittel	1	1
4	**1**	Hoch	10	100	Hoch	10	100	Niedrig	1	10
5	**10**	Keine	1	10	Keine	1	10	Keine	1	10
Total		220			130			121		

Tabelle 5: Gewichtung Smartphone Platform

Die Gewichtung hat gezeigt, dass die Android Plattform am besten für dieses Projekt geeignet ist. Daher wurde das myCoffee App für Android geschrieben.

4.5. Sicherheit

Die Kundenanforderung legt fest, dass das Thema Sicherheit bei der Datenübertragung angedacht werden muss.

Als minimale Sicherheit wurde eine User Authentifizierung implementiert. Diese vergleicht den anzumeldenden User mit den Daten in der Datenbank. So kann sichergestellt werden, dass nicht jeder der in Besitz der myCoffee-App ist, auch unbeschränkten Zugriff auf die Daten hat. Im aktuellen Stand des Projektes werden diese Daten jedoch unverschlüsselt abgespeichert und übermittelt. Um die Sicherheit zu steigern, sollten diese in einer produktiven Umgebung mit einem Hash versehen und abgespeichert werden.

Um die Sicherheit noch weiter zu erhöhen, könnten alle Daten verschlüsselt übertragen werden. Da das MQTT-Protokoll auf einer TCP Übertragung basiert, würde sich eine SSH bzw. TLS Verschlüsselung anbieten.

Aus Zeitgründen wurde in diesem Projekt nur die minimale Sicherheit mittels User Authentifizierung implementiert.

5. myCoffee Android Smartphone Applikation

In diesem Kapitel ist das Endergebnis der myCoffee Android Applikation zu sehen. Anhand von Screenshots werden die Funktionalitäten der App erläutert.

Abbildung 16: App Screen 1 Login

5.1. Screen 1: Login

Nach Starten der Applikation wird die Login Page präsentiert.

Bestandteile:

- Feld zur Eingabe des Benutzernamens
- Feld zur Eingabe des Passworts
- Login Button
- myCoffee Bild

Funktionsweise:
Nach Eingabe der Userangaben kann eine Verbindung durch betätigen des Login Buttons hergestellt werden.
Falls der User eines der Felder leergelassen hat, wird er darauf aufmerksam gemacht.
Falls das Passwort nicht korrekt ist, wird dies dem User mitgeteilt und er hat die Möglichkeit, seine angaben zu korrigieren.
Bei erfolgreicher Anmeldung wird er zu Screen 2 weitergeleitet.

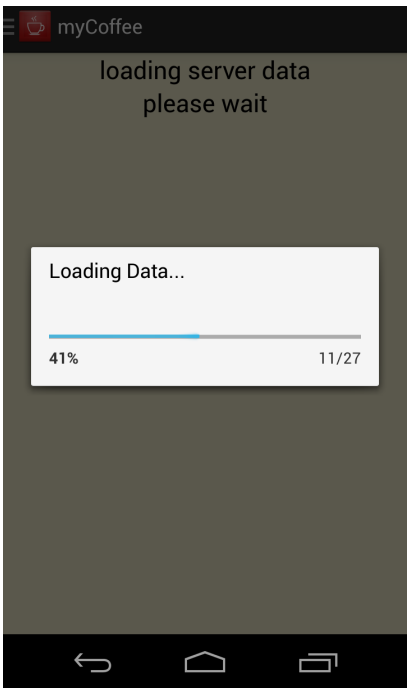

Abbildung 17: App Screen 2 Loading Data

5.2. Screen 2: Loading Data

Dieser Screen ist ein Übergangs-Screen, welcher keine Userinteraktion benötigt bzw. erlaubt.

Bestandteile:

- Loading Data Progressbar

Funktionsweise:
Bei jeder neuen Verbindung zum Server werden die gespeicherten Maschinendaten vom aktuellen Monat heruntergeladen. Dieser Vorgang wird dem User durch einen Ladebalken veranschaulicht.
Die Anzeigedauer dieses Screens ist abhängig von der zu ladenden Datenmenge. Diese basiert vor allem auf Konsumationsmenge und Monatsfortschritt.
Sind alle Daten geladen, wird zu Screen 3 gewechselt.

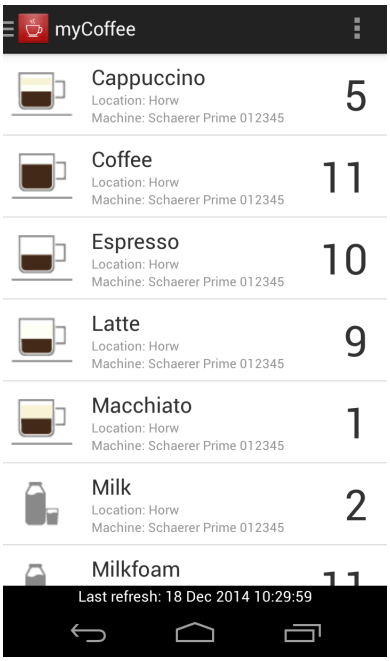

Abbildung 18: Screen 3 Live View

5.3. Screen 3: Live View

Hauptscreen der Applikation, welche dem User die aktuellen Maschinendaten präsentiert.

Bestandteile:

- Liste mit Produktinformationen
- Liste ist nach Region und Maschine unterteilt
- Produktname (Bsp. Cappucino)
- Region (Bsp. Horw)
- Maschinenname (Bsp. Primus 12345)
- Anzahl Konsumation
- Anzeige der letzten Datenaktualisierung
- Einstellungen (Oben rechts)
- Navigation (Wischen von links nach rechts)

Funktionsweise:
Grundsätzlich werden hier die Daten aller verfügbaren Maschinen angezeigt. Der Benutzer kann die Produktanzeige nach Region, Maschinenamen und Produkt über die Einstellungen anpassen. Nach dem Start der Applikation werden die aktuellsten Daten der Maschinen aus der Datenbank angezeigt. Falls die Maschinen online sind, senden sie alle 30s ein update ihrer Daten, welches von der Applikation empfangen und angezeigt wird. Die Aktualität der Daten wird durch *Last refresh* angezeigt. Über die Navigation kann der User zu anderen Screens navigieren (siehe Screen 6: Navigation).

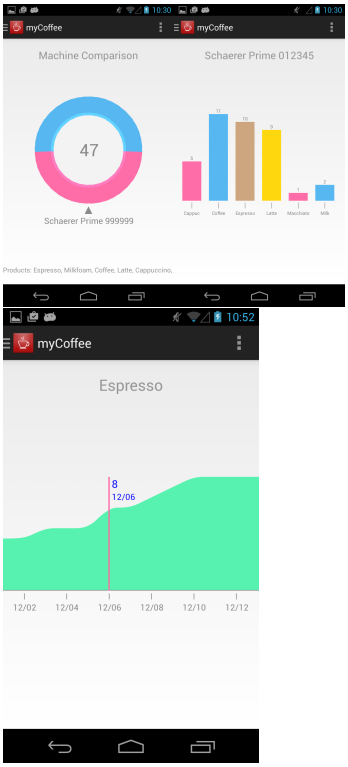

Abbildung 19: Screen 4 Statistik

5.4. Screen 4: Statistik

Zeigt verschiedene Statistiken des aktuellen Monats an.
Bestandteile:

- Kuchen-, Balken- oder Liniendiagramm
- Einstellungen (Oben rechts)
- Navigation (Wischen von links nach rechts)

Funktionsweise:
Die Anzeige ist Abhängig von folgenden Benutzereinstellungen

Maschine = 1, Produkte = 1:
Liniendiagramm des entsprechenden Produktes über die Tage des aktuellen Monats.

Maschine = 1, Produkte >1:
Balkendiagramm mit Produktevergleich der gewählten Maschine. Die aktuellsten Daten aus der Datenbank werden angezeigt.

Maschine > 1:
Kuchendiagramm mit einem Stück pro Maschine. Das Stück repräsentiert die Summe aller Produktkonsumationen der entsprechenden Maschine und ausgewählten Produkte.

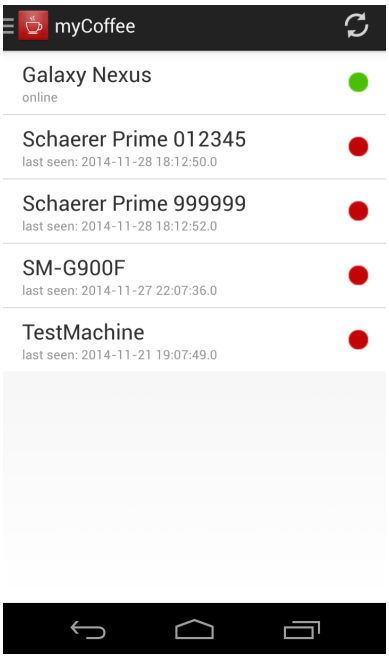

Abbildung 20: Screen 5 Admin Panel

5.5. Screen 5: Admin Panel

Zeigt Admin Informationen an. Kann nur mit entsprechenden Admin rechten geöffnet werden.

Bestandteile:

- Liste mit Clients
- Verbindungsstatus
- Refresh Button (Oben rechts)
- Navigation (Wischen von links nach rechts)

Funktionsweise:
Holt alle Clientinformationen aus der Serverdatenbank und zeigt diese an. Bei diesen Clients handelt es sich um solche, welche sich mindestens ein Mal mit dem Server verbunden haben.
Der Verbindungsstatus wird durch einen grünen bzw. roten Punkt signalisiert. Falls der Client aktuell Verbunden ist, besitzt er den Status *online* und einen grünen Punkt.
Ist er offline, wird Datum und Zeit der letzten Clientconnection angezeigt und der Punkt ist rot.

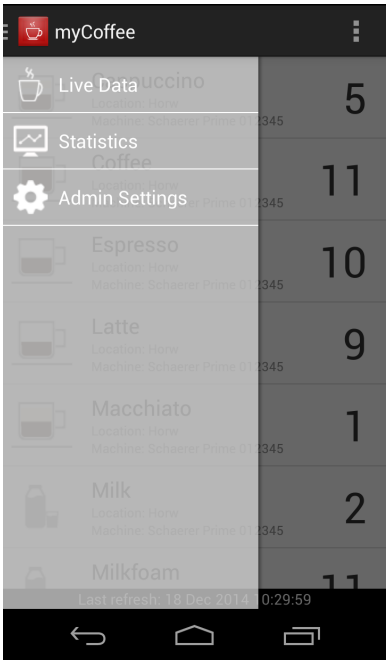

Abbildung 21: Screen 6 Navigation

5.6. Screen 6: Navigation

Ermöglicht den Wechsel zwischen Screen 3, 4 und 5.

Bestandteile:

- Link zu Live View (Screen 3)
- Link zu Statistik (Screen 4)
- Link zu Administration (Screen 5)

Funktionsweise:
Wird durch wischen vom linken Rand nach rechts angezeigt. Durch antippen der einzelnen Menüoptionen wird zwischen den Screens gewechselt.
Falls ein User ohne Adminrechte auf den Admin Screen zugreifen will, wird ihm der Zugang verwehrt und er wird durch eine entsprechende Meldung informiert.

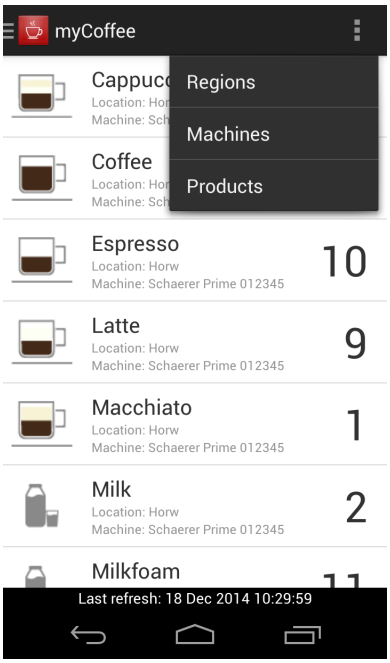

Abbildung 22: Screen 7 Settings

5.7. Screen 7 Settings

Erlauben dem User, individuelle Einstellungen vorzunehmen. Haben Einfluss auf Screen 3 und 4.

Bestandteile:

- Menüoptionen für die Untermenüs

Funktionsweise:
Durch antippen eines Menüpunktes wird ein entsprechendes Untermenü geöffnet. Die Einträge des Untermenüs sind abhängig von den in Screen 2 geladenen Daten vom Server. Die Einträge der Untermenüs werden dynamisch von den empfangenen Daten extrahiert. Ein mögliches Untermenü ist unter Screen 8 zu sehen.

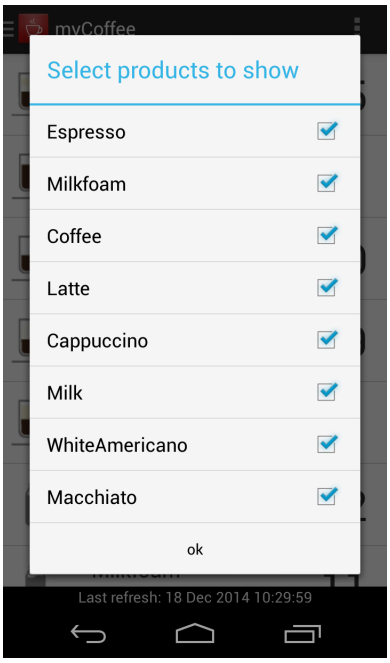

Abbildung 23: Screen 8 Product Settings

5.8. Screen 8 Produkt Settings

Untermenü zur Auswahl der anzuzeigenden Produkte.

Bestandteile:

- Liste mit Auswahlmöglichkeiten

Funktionsweise:
Durch selektieren der Checkbox kann ausgewählt werden, welche Produkte angezeigt werden sollen. Diese Einstellungen haben unabhängig vom aktuellen Screen direkten Einfluss auf Screen 3 und 4.
Entsprechend der empfangenen Daten werden in dieser Liste mehr oder weniger Einträge angezeigt. Die Einträge sind nicht fest einprogrammiert. Sobald eine Nachricht mit einem Produkt eintrifft, welches noch nicht in der Liste vorhanden ist, wird die Liste erweitert.

Durch betätigen der zurück Taste wird der User unabhängig vom aktuellen Screen automatisch abgemeldet und Screen 1 wird angezeigt.
Falls der User die Applikation schliesst, ohne sich abzumelden, wird die Verbindung zum Server automatisch sauber geschlossen.
Falls der User die Applikation pausiert, werden die in dieser Zeit gesendeten Nachrichten nicht empfangen. Es werden erst bei fortsetzen der Applikation wieder Daten empfangen.

6. Konkurrenz- und Technologieanalyse

Bei dieser Analyse ging es im wesentlichen darum, den aktuellen Stand in Bezug auf IoT in der Kaffeemaschinenbranche zu ermitteln. Zusätzlich wurden bestehende IoT Geräte und Konzepte analysiert. Die gewonnenen Erkenntnisse bildeten die Grundlagen für das weitere Vorgehen in diesem Projekt.

6.1. Jura Impressa Web Pilot

Bei der Firma Jura handelt es sich um einen Kaffeemaschinenherstellt mit Sitz in der Schweiz, welcher Geräte für den persönlichen sowie professionellen Gebrauch herstellt. Im Jahr 2004 brachte Jura den „IMPRESSA Web Pilot“ auf den Markt. Mit diesem Zubehör ist es möglich, die Maschine via Computer zu konfigurieren. Weiter können mit dem Web Piloten Gerätedaten an den Computer übertragen und dank einer Web-Schnittstelle mit dem Jura Service Portal ausgetauscht werden. (CNO-Research, 2005)

6.1.1. Funktionsweise Impressa Web Pilot

Der Web Pilot besteht im Wesentlichen aus der MemoryBox und der Jura Weboberfläche. Bei der MemoryBox handelt es sich um einen USB-Stick, mit welchem Nutzungsdaten und Einstellungswerte von der Kaffeemaschine an den Computer übertragen werden können. In die entgegengesetzte Richtung lassen sich Rezepte sowie Konfigurationen übertragen. Die eigentliche Verbindung zwischen Kaffeemaschine und Jura Weboberfläche übernimmt der Computer. (CNO-Research, 2005)

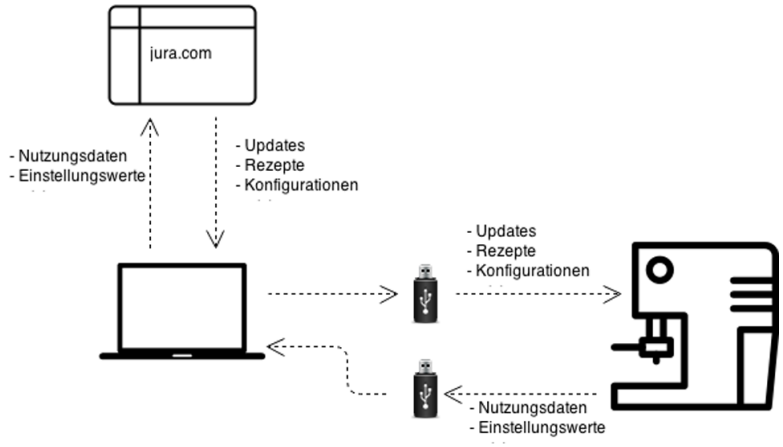

Abbildung 24: Web Pilot Informationsflüsse

Die Weboberfläche liefert dem Nutzer aktuelle Software Updates sowie voreingestellte Rezepte und Konfigurationen. Der eigentliche Zweck der Weboberfläche ist es jedoch, dem Nutzer einen Kundendienst anzubieten. Im Fehlerfall einer Maschine kann der Kunde die ausgelesenen Daten der Weboberfläche übermitteln. Mit diesen Daten wird anschliessend eine Online Analyse durchgeführt. Der konkrete Ablauf dieser Analyse ist in Abbildung 25 ersichtlich. (CNO-Research, 2005)

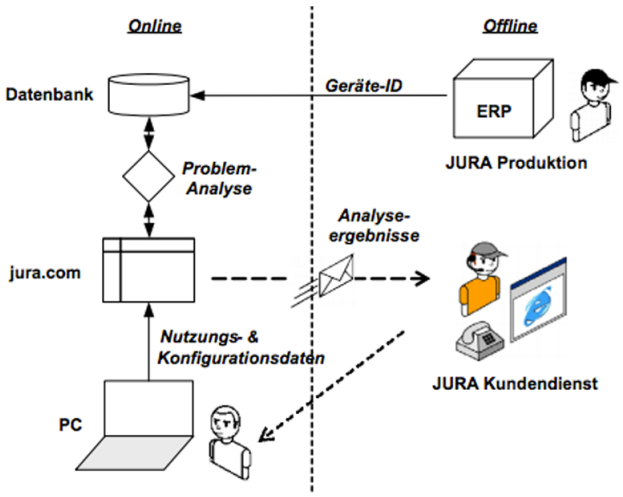

Abbildung 25: Ablauf einer Online Analyse (CNO-Research, 2005)

Übermittelt der Kunde die entsprechenden Daten, startet das System eine Problemanalyse. Dazu benutzt es neben den empfangenen Daten auch Produktionsdaten wie Baujahr, Typ und andere Parameter. Ist eine Lösung gefunden, wird diese dem Benutzer präsentiert. Ist dieser damit nicht einverstanden, kann er die Analyseergebnisse an den Kundendienst weiterleiten, welcher anschliessend mit dem Kunden kontakt aufnimmt. (CNO-Research, 2005)

6.2. Allgemeine Architektur von Internet of Things

Zühlke entwickelt unter anderem Lösungen für Kunden, welche ihren Produkten einen Internetanschluss spendieren wollen. Dabei stellen sie eine aus ihrer Sicht optimale Lösung für dieses Vorhaben vor. Die Architektur bezieht sich nicht auf ein bestimmtes Produkt, sondern kann universell für Geräte verwendet werden, welche in ein IoT eingebunden werden sollen. Bei der Zühlke Architektur steht vor allem der Sicherheitsaspekt im Vordergrund. (Zühlke, 2014)

6.2.1. Ausgangsszenario

Häufig sollen Geräte mit entsprechenden Sensoren und Aktoren an ein lokales LAN angeschlossen werden. Über ein lokales IP Gateway sollen diese Informationen schliesslich ins Internet gelangen.
Naheliegend für dieses Vorhaben ist es, einen Web Server auf dem IP Gateway einzurichten. Über diesen werden die gewonnenen Informationen verteilt und Nutzer können mit den Geräten interagieren.
Damit auch ausserhalb des lokalen LAN mit den Geräten kommuniziert werden kann, kommt ein Dynamischer DNS Service und Port Weiterleitung zum Einsatz. Ohne entsprechende Absicherung öffnet sich so ein gefährliches Sicherheitsloch. Doch dieser Ansatz ist nicht nur hinsichtlich Sicherheit nicht optimal. Ein eigenständiger Server für eine Hand voll Geräte ist mit nicht verhältnismässigem Aufwand verbunden und skaliert nicht.

6.2.2. Mögliche Lösung – Eine Cloud

Die Architektur von Zühlke greift genau diese Nachteile auf und versucht diese zu beheben.
Die zentrale Stelle ist eine Cloud. Diese kann als private Cloud auf einem zentralen Server oder als Public Cloud bei einem externen Dienstleister realisiert sein.

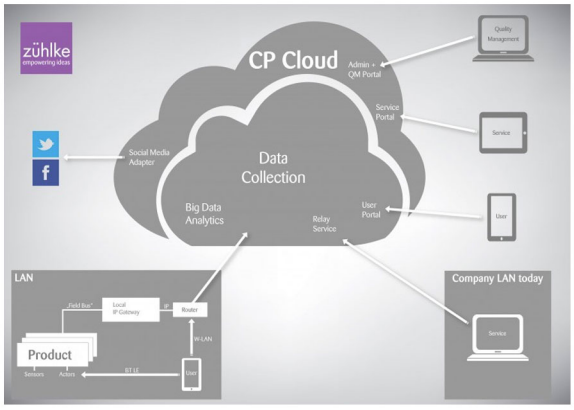

Abbildung 26: Zühlke IoT Cloud Architektur (ZühlkeImage, 2014)

Vorteil dieser Lösung ist es, dass sich das IoT Gerät ebenfalls als Client in die Cloud verbinden kann und so die Server Aufgaben ausgelagert werden können. Somit können alle Sicherheitsaspekte auf dem Server erledigt werden und müssen nicht vom IoT Gerät selber übernommen werden.

6.3. Spark open source IoT Toolkit

Bei Spark Lab Inc aus San Francisco handelt es sich um ein interessantes Start Up Unternehmen, welches das erste mal im Jahr 2013 auf der Croudfunding Plattform Kickstarter in Erscheinung trat und dort erfolgreich die ersten Investoren an Land zog. (Kickstarter, 2013)
Spark verspricht seinen Kunden ein einfach einzusetzendes IoT Toolkit, mit welchem im handumdrehen Geräte ans Internet angebunden werden können.
Das Toolkit beinhaltet den Spark Core, welcher die Hardware bildet. Neben Hardware bietet Spark ihr eigens entwickeltes OS an, welches in der Cloud operiert und die Kommunikation zwischen Cores und Clients regelt. (Spark, 2014)

6.3.1. Spark OS

Das Spark OS ist ein IoT Betriebssystem, welches in der Cloud zu Hause ist. Diese Architektur basiert auf dem bereits vorgestellten Modell von Zühlke (siehe Kapitel 6.2). Die Spark Cores, welche an die Endprodukte angeschlossen sind, sowie die Anzeigegeräte für den Endkunden verbinden sich als Clients zu der Cloud.

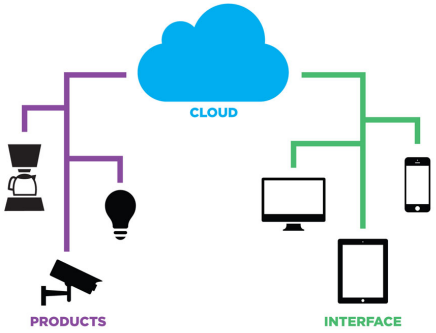

Abbildung 27: Spark Cloud (SparkImage2, 2014)

Die Cloud selber agiert somit als Server für all die verschiedenen Clients. Zur Kommunikation wird eine REST API angeboten, über welches die Clients mit der Cloud kommunizieren können. Weiter unterstützt die Cloud das Publish/Subscrib-Modell. Dabei kann sich ein Spark Core als Publisher eines beliebigen Dienstes registrieren. Anschliessend können beliebig viele Clients diesen Dienst als Subscriber abonnieren und erhalten so automatisch aktuelle Nachrichten des Publishers. (Spark, 2014)

Die ganze Kommunikation zwischen IoT Geräten, der Cloud und Anzeigegeräte ist mit Standards wie RSA, AES und SSL/TLS verschlüsselt und somit gegen Zugriff von Unbefugten gesichert.

6.4. Ergebnis Konkurrenz- und Technologieanalyse

Die Konkurrenzanalyse lieferte hinsichtlich Datennutzung sowie Systemarchitektur wertvolle Informationen.

6.4.1. Datennutzung

Am Beispiel des Jura Web Piloten hat sich gezeigt, dass die gewonnenen Daten als Grundlage für die Problemanalyse dienen können. Dank eines Internetanschlusses kann der Kunde von überall und zu jeder Uhrzeit den Online Kundendienst in Anspruch nehmen und erhält eine Problemlösung in Echtzeit.
Die Architektur des Jura Web Piloten lässt jedoch hinsichtlich Benutzerfreundlichkeit einige Wünsche offen. Der Nutzer muss die Daten eigenhändig mittels USB-Stick zwischen Computer und Kaffeemaschine übertragen. Es besteht keine Möglichkeit die Maschine direkt mit dem Computer oder dem Internet zu verbinden.

6.4.2. Architektur

Das Zühlke Architekturmodell zeigte vor allem die Stärken eines zentralen Servers auf. Würden die Clients die Daten direkt bei den IoT Geräten abrufen, müsste jedes Gerät die Rolle eines Servers übernehmen. Durch Auslagerung dieser Serverrolle auf eine zentrale Stelle, muss der Webserver nur einmal aufgesetzt werden und alle Sicherheitsaspekte können zentral geregelt werden.
Dass dieses Modell in der Praxis erfolgreich eingesetzt werden kann, zeigt das Toolkit von Spark.

6.4.3. Erkenntnisse für dieses Projekt

Die Konkurrenzanalyse hat vor allem die System Architektur dieses Projektes beeinflusst. Das Architekturmodell von Zühlke und das Publish/Subscribe-Prinzip wie es von Spark eingesetzt wird, konnte in diesem Projekt erfolgreich umgesetzt werden. Konkret dient die Kaffeemaschine als Publisher und das Smartphone kann das entsprechende Thema als Subscriber abonnieren. Das Publizieren sowie Abonnieren wird von einem zentralen Server verwaltet, welcher die Weiterleitung der Daten an die entsprechenden Clients übernimmt.
Vorteil dieses Modells ist es, dass neue Daten in Echtzeit beim Smartphone ankommen, ohne dass dieses in regelmässigen Abständen nach neuen Daten fragen muss.

Weiter hat sich gezeigt, dass die Datenübertragung übers Internet einwandfrei und mit entsprechenden Massnahmen auch sicher funktioniert. Aus diesem Grund wird das Internet als Übertragungsmedium verwendet, ohne auf andere Technologien wie Bluetooth oder ZigBee zurückzugreifen.

Hinsichtlich der Datennutzung hat die Konkurrenzanalyse den interessanten Aspekt eines Online Kundendienstes in Echtzeit aufgezeigt. Eine ähnliche Umsetzung wäre aber im Rahmen dieses Projektes nicht möglich gewesen. Daher werden die gewonnenen Daten der Kaffeemaschine dem Benutzer so zur Verfügung gestellt, dass er einen Überblick über die aktuellen Daten sowie über statistische Daten der vergangenen Tage erhält.

7. Schlussfolgerung

Hauptvorteil der umgesetzten Architektur ist es, dass sie beliebig skaliert und generisch ist. Die Kaffeemaschine als Datenlieferant kann beliebig durch ein anderes Produkt ausgetauscht werden und das System funktioniert mit geringen Anpassungen problemlos weiter.

Das MQTT Protokoll mit MQTT Broker und Publish/Subscribe-Verfahren hat sich als sehr einfach und extrem nützlich für dieses Projekt herausgestellt. Das Protokoll ist sehr leichtgewichtig, was den Einsatz in speicher- und leistungsarmen Geräten begünstigt. Zusätzlich gibt es unzählige Implementationen in verschiedenen Programmiersprachen von MQTT-Clients und Brokern, so dass praktisch jedes Gerät als Client eingebunden werden kann.

Es wurden alle geforderten Funktionen der Aufgabenstellung umgesetzt oder zumindest angedacht. Daher wird dieses Projekt als Erfolgreich betrachtet und liefert dem Auftraggeber sicherlich wertvolle Erkenntnisse und eine solide Ausgangslage, zur Entwicklung eines marktreifen Produktes, welches auf dem in diesem Projekt erstellten Konzept und Prototypen basiert.

8. Ausblick

Der in dieser Arbeit entwickelte Prototyp liefert den Beweis, dass das erarbeitete Konzept für diese Aufgabenstellung funktioniert. Damit dies aber in der Praxis eingesetzt werden kann, müssen folgende Anpassungen und Weiterentwicklungen vorgenommen werden.

Das Thema Sicherheit wurde in diesem Projekt nur minimal umgesetzt. Um die Sicherheit im produktiven Einsatz zu steigern, sollten Userdaten verschlüsselt abgespeichert werden. Zusätzlich sollte eine Verschlüsselung der Datenübertragung mittels SSL bzw. TLS vorgenommen werden.

Der HiveMQ Broker war der ideale MQTT Broker für dieses Projekt. Er konnte schnell in Betrieb genommen und die Funktionalität konnte durch eigene Java Plugins erweitert werden. Zusätzlich ist die Nutzung für bis zu 25 gleichzeitig angemeldeten Clients kostenlos. Im produktiven Einsatz wird diese Zahl schnell überschritten und es würden einmalige Lizenzkosten anfallen. Falls dies nicht gewünscht ist, müsste eine passende Alternative gesucht oder eine eigene Implementation realisiert werden.

Aus Zeitgründen wurde die Smartphone Applikation nur für Android erstellt. Um ein möglichst breites Nutzersegment anzusprechen, sollte mindestens noch ein App für iOS hinzukommen. Neben Smartphone Applikationen zur Visualisierung könnten auch Web Applikationen entwickelt werden.

Das Error-Handling wurde soweit implementiert, wie es der Rahmen dieses Projektes erlaubt und benötigt. Um das Endprodukt jedoch robuster zu machen, müssten gezielt noch zusätzliche Ausnahmebedingungen und Grenzfälle abgefangen werden. Als Beispiel dient der Verbindungsunterbruch zum Server. Dieser wird beim aktuellen Prototypen zwar detektiert, jedoch nicht behandelt. So kann es im schlimmsten Fall sein, dass der User erst bei erneutem Login wieder mit dem Server verbunden ist.

9. Lessons Learned

Eine der Haupterkenntnisse dieses Projektes ist, dass die Gestaltung grafischer Benutzeroberflächen viel Zeit in Anspruch nehmen. Vor allem dann, wenn man einen gewissen Wert auf die Optik legt. In diesem Projekt wurden teilweise mehrere Stunden damit verbracht, die Oberfläche der Android Applikation aufzuhübschen, ohne die Funktionalität zu erweitern. Daher sollten in einem nächsten Projekt entweder die Ansprüche an die grafische Benutzeroberfläche heruntergeschraubt oder aber mehr Zeit für die Entwicklung dieser eingeplant werden.

Weiter hat das Projekt wieder einmal gezeigt, dass an der Aussage, man solle genügend Zeit in die Initialisierungs- und Konzeptionsphase investieren, wirklich etwas dran ist. Denn eine gute Voranalyse und durchdachte Architektur verringert den Zeitaufwand der Implementierungsphase enorm und grundlegende Fehler werden früh erkannt. Dies wird in kommenden Projekten sicherlich weiter so umgesetzt.

Neben den hauptsächlich softwarespezifischen Erkenntnissen und Fortschritte, welche in diesem Projekt gesammelt werden konnten, wurden auch wertvolle Erfahrungen bezüglich Dokumentation erhalten. Der Aufbau dieser wurde nämlich im Laufe des Projektes mehrmals grundsätzlich umgestellt. Dies kann durch die gewonnene Erfahrung in kommenden Projekten vermieden und die Zeit für wichtigere Aspekte verwendet werden.

Neben den hier aufgeführten Erkenntnissen wurden im Laufe des Projektes viele kleinere Erfahrungen gesammelt. Die Summe aus all diesen Erkenntnissen und Erfahrungen sind insbesondere im Hinblick auf die bevorstehende Bachelor Diplomarbeit von grossem Nutzen. Diese kann nun mit mehr Projekterfahrung in Angriff genommen werden.

10. Literaturverzeichnis

AndroidClient. (2014). *Android Service*. Abgerufen am 14. 12 2014 von eclipse.org: https://eclipse.org/paho/clients/android/

Cech, P. (2014). *Eazegraph github repository*. Abgerufen am 14. 12 2014 von github.com: https://github.com/blackfizz/EazeGraph

CNO-Research. (2005). *unic.com.* Abgerufen am 01. 10 2014 von unic.com: https://www.unic.com/content/dam/downloads/unic_cno_case_jura_05.pdf

dcSquare. (2014). *HiveMQ Enterprise Grade MQTT Broker*. Abgerufen am 14. 12 2014 von hivemq.com/: http://www.hivemq.com/

Jaffey, T. (02 2014). *Eclipse Newsletter: MQTT and CoAP, IoT Protocols*. Abgerufen am 08. 10 2014 von eclipse.org: http://eclipse.org/community/eclipse_newsletter/2014/february/article2.php

JavaClient. (2014). *paho Java Client*. Abgerufen am 19. 11 2014 von eclipse.org: https://eclipse.org/paho/clients/java/

Kickstarter. (02. 06 2013). *Spark Core: Wi-Fi for Everything (Arduino Compatible)*. Abgerufen am 01. 10 2014 von kickstarter.com: https://www.kickstarter.com/projects/sparkdevices/spark-core-wi-fi-for-everything-arduino-compatible

Pi4J. (2014). *The Pi4J Project*. Abgerufen am 14. 12 2014 von pi4j.com/: http://pi4j.com/

Spark. (2014). *Develop*. Abgerufen am 01. 10 2014 von spark.io: https://www.spark.io/features

SparkImage1. (2014). *Spark Image Hand.* Abgerufen am 14. 12 2014 von amazonaws.com: https://s3.amazonaws.com/spark-website/hand.jpg

SparkImage2. (2014). *Spark Image Cloud.* Abgerufen am 14. 12 2014 von amazonaws.com: https://s3.amazonaws.com/spark-website/cloud-and-things.png

SparkImage2. (2014). *Spark Image Security.* Abgerufen am 14. 12 2014 von amazonaws.com: https://s3.amazonaws.com/spark-website/security.png

Tarkoma, S. (2012). *Publish/Subscribe Systems: Design and Principles.* West Sussex, United Kingdom: John Wiley & Sons.

Zühlke. (12. 06 2014). *How Connected Products merge into the Internet of Things.* Abgerufen am 01. 10 2014 von blog.zuehlke.com: http://blog.zuehlke.com/en/the-architecture-of-the-internet-of-things/

ZühlkeImage. (2014). *Zühlke Architektur Bild.* Abgerufen am 14. 12 2014 von blog.zuehkle.com: http://blog.zuehlke.com/wp-content/uploads/2014/05/Typical-target-architecture1-01-755x533.jpg

11. Tabellenverzeichnis

12. Abbildungsverzeichnis

Anhang