Domain Specific Software Engineering: Internet of Things

Seminar Domain Specific Software Engineering

Version 0.9

Zürcher Hochschule für Angewandte Wissenschaften

Daniel Brun

18. Juni 2015

Eigenständigkeitserklärung

Hiermit bestätigt ich, dass vorliegende Seminararbeit zum Thema „Domain Specific Software Engineering: Internet of Things“ gemäss freigegebener Aufgabenstellung ohne jede fremde Hilfe und unter Benutzung der angegebenen Quellen im Rahmen der gültigen Reglemente selbständig verfasst wurde.

Thalwil, 18. Juni 2015 Daniel Brun

Inhaltsverzeichnis

KAPITEL 1

Einleitung

1.1 Hintergrund

Im Rahmen meines Bachelor-Studiums in Informatik an der Zürcher Hochschule für Angewandte Wissenschaften (ZHAW) wird im 6. Semester eine Seminararbeit zu einem vorgegebenen Themenbereich erarbeitet werden. Ich habe mich für den Themenbereich „Domain Specific Software Engineering“ entschieden.

Aus einem Themenkatalog konnte ein spezifisches Thema im Bereich „Domain Specific Software Engineering“ ausgewählt werden. Ich habe mich für das Thema „Internet of Things“ entschieden.

Für die Arbeit sollen circa 50 Arbeitsstunden aufgewendet werden. Dies entspricht etwa einem Umfang von 15 bis 20 Seiten. Zusätzlich gelten die Rahmenbedingungen gemäss dem Reglement zur Verfassung einer Seminararbeit ([Ste12])

1.2 Aufgabenstellung

Es soll ein Dokument zum Thema Domain Specific Software Engineering im Bereich Internet of Things erstellt werden. Das Papier soll die Schwierigkeiten der Software-Entwicklung in diesem Bereich aufzeigen und einen groben Überblick über das Thema geben.

1.3 Abgrenzung

Aufgrund des grossen Umfanges und Vielschichtigkeit dieser Domäne werde ich mich auf die wichtigsten Punkte konzentrieren.

1.4 Motivation

In den letzten Monaten bin ich immer wieder auf das Thema „Internet of Things“ aufmerksam geworden. Seit meiner Teilnahme an der Microsoft Konferenz in Barcelona Ende 2014 (TechEd Europe) war meine Neugier definitiv geweckt und ich hatte mir bereits einige Projekte überlegt, welche ich allenfalls als Semesterarbeit, Seminararbeit oder in einem anderen Schulprojekt realisieren könnte. Mit dem Seminar „Domain Specific Software Engineering“ hat sich mir nun eine erste Gelegenheit geboten, um mich in dieses Thema zu vertiefen. An dem Thema fasziniert mich vor allem das breite Spektrum an Innovations- und Kombinationsmöglichkeiten.

1.5 Struktur

Diese Arbeit gliedert sich in folgende Hauptteile:

- Ausgangslage
- Die Domäne „Internet of Things“
- Software Engineering in der Domäne „Internet of Tings“
- Schlusswort & Fazit

Im ersten Kapitel werden die Details zur Ausgangslage und die Hintergründe der Arbeit aufgezeigt. Im zweiten Kapitel wird die Ausgangslage in Bezug auf die betrachteten Themen „Internet of Things“, „Domain Specific Software Engineering“ und „Domain Specific Languages“ kurz erläutert. im darauffolgenden Kapitel wird die Domäne „Internet of Things“ im Detail beschrieben und aufgezeigt, was für Anforderungen an das Software Engineering bestehen. Im Kapitel 4 werden anschliessend die in der Domäne eingesetzten Standards, Protokolle und Frameworks beschrieben. Im letzten Kapitel wird ein Fazit gezogen und über die gesamte Arbeit reflektiert.

KAPITEL 2

Ausgangslage

Mit dem laufenden Fortschritt in der Computer- und Kommunikationstechnik und der damit einhergehenden Miniaturisierung und Mobilisierung entwickeln sich auch laufend neue Themenbereiche und Sparten in der Informatik. Das „Internet der Dinge“ („Internet of Things“) ist eines dieser Gebiete und hat in den letzten Jahren und Monaten für viele Diskussionen und viel Innovation gesorgt.

2.1 Internet of Things

Nach dem Aufkommen der Personal Computer, der Etablierung des Internets und dem Mobile-Trend der letzten Jahre, stellt das Internet of Things (IoT) oder „Internet der Dinge“ die nächste Etappe in der Entwicklung des Internets dar. Diese Etappe wird wiederum die Art und Weise wie Leute und vor allem Dinge miteinander kommunizieren und interagieren grundlegend verändern. Das IoT ist vorallem auch die nächste Etappe in der Machine-2-Machine (M2M) Kommunikation.

Dem IoT-Markt wird grosses Entwicklungspotenzial zugeschrieben, ein grösseres sogar als vor einigen Jahren dem Smartphone und Tablet Markt. Gab es im Jahr 2012 weltweit bereits 20 Milliarden „Connectable Things“ (Dinge die mit dem Internet verbunden sind, beziehungsweise verbunden werden könnten), werden es im Jahr 2012 bereits 32 Milliarden sein. [Eem, S. 5]

Mit der Etablierung des IoT wird eine Transformation von isolierten Geräten hin zu vernetzten Geräten statt finden. In Zukunft wird vermehrt die Kommunikation von Geräten untereinander und mit dem Internet im Vordergrund stehen. Im Privatbereich wird sich vieles, wenn nicht sogar alles, um die Heimautomation drehen. Wird von der Wetterstation ein Sturm gemeldet, werden automatisch die Sonnenstoren eingefahren und die Fenster und Rollläden geschlossen. Ein anderes Beispiel wäre, dass wenn man nach Hause kommt automatisch beim Aufschliessen der Türe das Licht angeht und sich der Radio einschaltet. Die Heimautomation dient auch als Einbruchsschutz während längeren Abwesenheiten. Durch den Einsatz von Machine-Learning-Algorithmen lernt das System die Gewohnheiten

der Bewohner und simuliert dann das Verhalten bei längerer Abwesenheit. Kommt es dennoch zu einem Einbruch oder einem Einbruchsversuch kann das System selbstständig reagieren und zum Beispiel die Fenster und Türen schliessen und die Polizei und die Bewohner alarmieren und allenfalls Videoaufnahmen und Fotos anfertigen.

In der Industrie wird das IoT zum Beispiel zur Überwachung, Kontrolle und Steuerung von Produktionsanlagen oder Fertigungsprozessen eingesetzt.

Durch das IoT ergeben sich viele Möglichkeiten und Chancen. Es ist jedoch nicht ausser Acht zu lassen, dass auch neue Risiken und Gefahren entstehen. So müssen die entstehenden Netzwerke gegen Angriffe und Manipulationen abgesichert und die transportierten Daten vor fremden Augen geschützt werden. Kann ein Heimautomationssystem oder ein medizinisches Überwachungssystem in einem Spital von aussen manipuliert werden, kann dies schwerwiegende Folgen haben.

Im Kapitel 3 Die Domäne „Internet of Things“ wird die Domäne des IoT im Detail beschrieben.

2.2 Domain Specific Software Engineering

Domain Specific Software Engineering, beziehungsweise das domänenspezifische entwickeln von Software, beschreibt den Entwicklungsprozess in einem bestimmten Kontext oder Problembereich. Der Entwicklungsprozess für Software von medizinischen Geräten unterscheidet sich an vielen Stellen vom Entwicklungsprozess einer Unternehmung, welche Web-Seiten designt und entwickelt. Auch unterscheidet sich die Entwicklung von Web-Seiten von der Entwicklung von Mobile-Apps oder Fat-Client-Anwendungen. Jede Domäne hat ihre eigenen Anforderungen und Besonderheiten. Diesem Umstand wird durch den Einsatz von Domain Specific Language (DSL)’s und spezifischen Entwicklungswerkzeugen und -Methodiken Rechnung getragen.

2.3 Domain Specific Languages

Eine DSL bezeichnet eine Programmier-, Modellierungs- oder Metasprache, welche für eine spezifische Problemdomäne entworfen und entwickelt wurde. Eine DSL addressiert dabei spezifische Schwächen und Problemstellungen der angesprochenen Domäne.

Eine gute DSL zeichnet sich durch ihre Einfachheit aus. Die Sprache sollte sich so nah als möglich am Problembereich befinden und auch die entsprechenden Ausdrücke und Begriffe verwenden. Dadurch wird die Eintrittshürde zur Verwendung der DSL heruntergesetzt und der Einarbeitungsaufwand reduziert. Die Semantik und Syntax sollte so gewählt werden, das diese im Kontext Sinn ergibt und einfach lesbar und verständlich ist. DSL’s werden oft auch von nicht Programmierern verwendet, da sich die Sprache sehr nahe an der fachlichen Sprache der Domäne befindet.

Es können zwei Arten von DSL’s unterschieden werden. Eine externe, oder auch unabhängige, DSL ist so ausgelegt, dass diese nicht von einer bestimmten Sprache abhängig ist. Die

Festlegung der Syntax und Grammatik liegt komplett in der Verantwortung und Entscheidungsfreiheit des Autors. Für die Implementation kann eine beliebige Programmiersprache verwendet werden.

Im Gegensatz dazu basiert die interne oder eingebettete DSL auf einer spezifischen Sprache (Wirtssprache). Diese Sprache gibt dabei die Einschränkungen für die zu implementierende DSL vor. Im Gegenzug muss sich der Autor nicht mehr um die Grammatik, Parser und Tools kümmern, da dies bereits von der Sprache zur Verfügung gestellt wird.

Externe DSL's sind flexibler, erfordern aber einen viel höheren Implementierungsaufwands als eine interne DSL.

Beispiele

- Externe DSL
  - SQL
  - Reguläre Ausdrücke
  - CSS
  - Sass
- Interne DSL
  - Rake (Ruby)

KAPITEL 3

Die Domäne „Internet of Things“

In diesem Kapitel wird die Domäne „Internet of Things“ näher beschrieben und die Auswirkungen auf die Softwareentwicklung aufgezeigt.

3.1 Ziel

Das primäre Ziel des IoT ist die Vernetzung von Dingen aus der realen Welt mit dem Internet und anderen Dingen. Die Dinge sollen intelligent und autonom mit anderen, ihnen unbekannten, Geräten und Anwendungen kommunizieren können um so einen Mehrwert für den Anwender zu schaffen und die eigenen Fähigkeiten zu erweitern.

3.2 Anwendungsbereich

Das IoT könnte grundsätzlich überall eingesetzt werden. Sei es in der Landwirtschaft, in der Pflege, in der Medizin oder in der Industrie. Das IoT ist mehr ein Konzept, beziehungsweise eine Architektur, als ein konkretes Produkt. Dadurch lässt es sich auf die jeweiligen Bedürfnisse der verschiedenen Problembereiche individuell anpassen. Neben der Anwendung im geschäftlichen Umfeld, findet das IoT auch im privaten Umfeld Anwendung. Das primäre Ziel ist dabei die Heimautomation, das heisst die intelligente Vernetzung verschiedener Geräte und Gegenstände im Haushalt. Nachfolgend werden einige weitere Beispiele für die konkrete Anwendungen des IoT aufgelistet.

- Überwachung von Pflegepatienten (Notruf, Unfälle, Stürze)
- Hausautomation (Schutz vor Einbrüchen, Energieoptimierung, Komfort)
- Überwachung von Herstellungs- und Fertigungsprozessen
- Überwachung von Gefahrensituationen in der Natur (zum Beispiel Waldbrände, Erdbeben, Felsstürze)

Auch im Bereich Big Data stellt IoT ein weiterer Meilenstein dar. Durch die Vernetzung von Millionen von Geräten können riesige Datenmengen gesammelt, analysiert und ausgewertet werden.

3.3 Subdomänen und Verwandte Domänen

Das IoT ist eine riesige Domäne mit vielen verschiedenen Anwendungsmöglichkeiten und somit auch vielen Subdomänen. Nachfolgend werden einige verwandte Domänen und Subdomänen aufgelistet und zwei spezifisch erläutert.

- Intelligente Städte („Smart Cities“)
- Intelligentes Umweltmanagement („Smart Environment“)
- Intelligentes Wassermanagement („Smart Water“)
- Intelligente Messeinrichtungen („Smart Measuring“)
- Sicherheit und Notfall („Security and Emergency“)
- Logistik im Einzelhandel („Retail Logistics“)
- Überwachung in der Industrie („Industrial Control“)
- Intelligente Landwirtschaft („Smart Agriculture“)
- Intelligente Massentierhaltung („Smart Animal Farming“)
- Haushaltsautomatisierung („Domotics and Homeautomation“)
- e-Health

3.3.1 Wireless Sensor Network

Wireless Sensor Network (WSN) sind Netzwerke von vielen verteilten Sensoren, welche gewisse physikalische Zustände oder Zustände in ihrer Umgebung und Umwelt überwachen. Die Forschung im Bezug auf WSN's begann bereits in den 1980er und begann sich dann nach der Jahrtausendwende in der Industrie zu etablieren. WSN's entwickelten sich lange unabhängig vom IoT. Der Begriff IoT wurde zum ersten Mal um das Jahr 1999 erwähnt und tauchte in den darauffolgenden Jahren immer wieder auf. Heute kann gesagt werden, dass die beiden Bereiche immer mehr und mehr verschmelzen und die WSN's als Teilbereich des IoT angesehen werden können. WSN kommen häufig in den Bereichen Industrie, Smart Cities, Smart Environment und Smart Measuring zum Einsatz.

Ein WSN-Node ist ein billig produzierbares Gerät, welches nur sehr wenig Strom benötigt. Idealerweise wird dieses über eine Batterie oder eine autonome Energiequelle (zum Beispiel ein Solar-Panel) betrieben. WSN-Nodes besitzen meistens nur eine einzelne Funktion und sind in der Regel mit einem WSN-Edge-Node verbunden.

Ein WSN-Edge-Nodes verbindet mehrere WSN mit einem Netzwerk oder dem Internet. Dieser Edge-Node nimmt die Funktion eines Gateways wahr und kommuniziert in der Regel mit einem Back-End System. Der Benutzer greift entweder über das Back-End System oder das Gateway auf die gesammelten Daten zu.

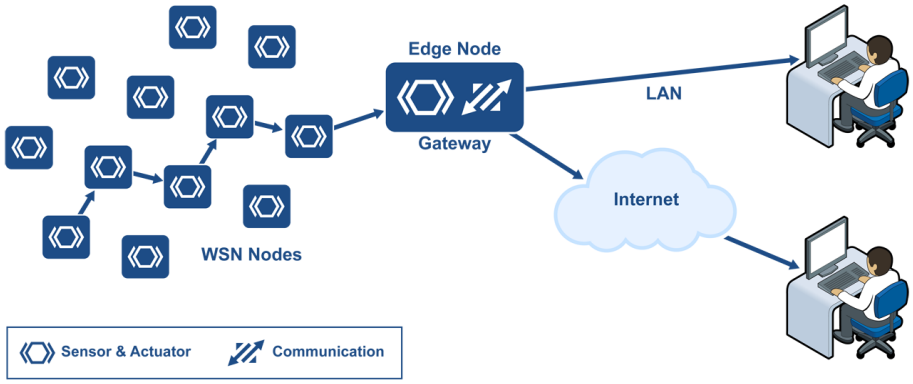

Abbildung 3.1: Schematische Darstellung eines WSN
Quelle: http://micrium.com/iot/devices/

Als Kommunikationstechnologie kommt entweder Wi-Fi oder eine Low-Power-Lösung zum Einsatz. Der Vorteil von Wi-Fi besteht in seinem hohen Verbreitungsgrad. Dem Gegenüber steht der hohe Energieverbrauch von Wi-Fi. Inzwischen gibt es eine breite Palette an Low-Power-Lösungen welche für den Einsatz auf solchen Geräten optimiert sind. Sie sind Energieeffizient, für lange Laufzeiten ausgelegt und verfügen meistens über die Fähigkeit ein Mesh-Network zu bilden. In einem Mesh-Network müssen nicht alle Geräte eine direkte Verbindung mit dem Gateway aufweisen. Es ist ausreichend, wenn es in der Nähe eines Nodes einen anderen Node gibt, der entweder direkt oder über einen anderen Node mit dem Gateway verbunden ist.

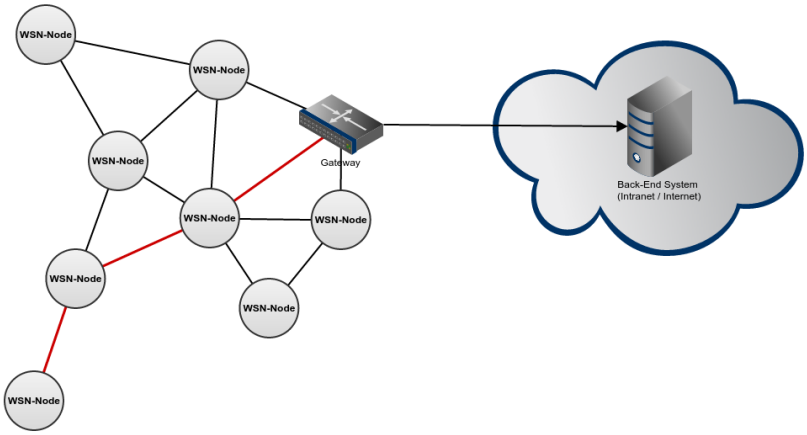

Abbildung 3.2: Schematische Darstellung eines Mesh-Networks

Der IEE 802.15.4 Standard wurde zum Beispiel speziell für den Einsatz in Low-Power-Systemen konzipiert. Die Standards, Protokolle und Frameworks, welche im IoT, Verwendung finden werden im Kapitel 4.2 Software Design & Software Construction erläutert.

3.3.2 Heimautomation

Bei der Heimautomation steht die Vernetzung und intelligente Kommunikation der Endgeräte untereinander im Vordergrund. Dabei kommt ein bunter Mix an unterschiedlichen Geräten und Technologien zum Einsatz, was eine enorme Herausforderung für die direkte Kommunikation darstelle. Mit dem Einsatz eines Homeautomation- oder Smart-Gateways können diese Herausforderungen reduziert werden. Diese Gateways unterstützen eine breite Palette an Übertragungs- und Kommunikationsprotokolle und sind so in der Lage diese miteinander, beziehungsweise mit dem Back-End System, zu verbinden.

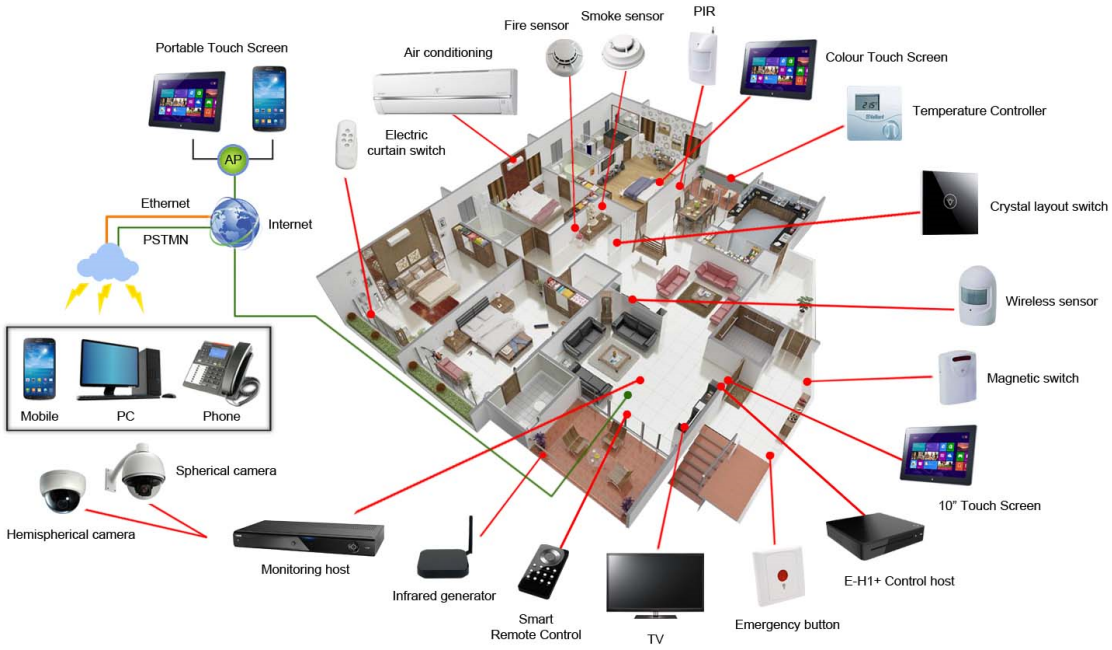

Abbildung 3.3: Szenario eines Heimautomatisierungs-Netzwerkes
Quelle: http://www.witura.com/wifi-smart-home-management-system.html

3.4 Architektur & Aufbau

Das IoT besteht vereinfacht dargestellt aus „Things“, „Gateways“ und „Back-End Systems“, welche miteinander kommunizieren. In diesem Kapitel werden diese drei Elemente und deren Verbindung im Detail beschrieben.

In der nachfolgenden Grafik wird der Kommunikationsweg innerhalb eines WSN aufgezeigt. Das „Thing“ empfängt die Daten von einem Sensor, verarbeitet diese und benachrichtigt anschliessend über ein definiertes Protokoll das Gateway über die Daten. Das Gateway

empfängt die Daten von einem oder mehreren Sensoren, verarbeitet diese und benachrichtigt anschliessend das Back-End System. Die Benachrichtigung des Back-End Systemes erfolgt häufig über ein Standard Protokoll aus der Web-Domäne. Das Back-End System empfängt die Daten, verarbeitet diese und entscheidet, ob allenfalls weitere Schritte unternommen werden müssen. Zum Beispiel könnte er bei Überschreiten eines Grenzwertes den Benutzer via SMS informieren.

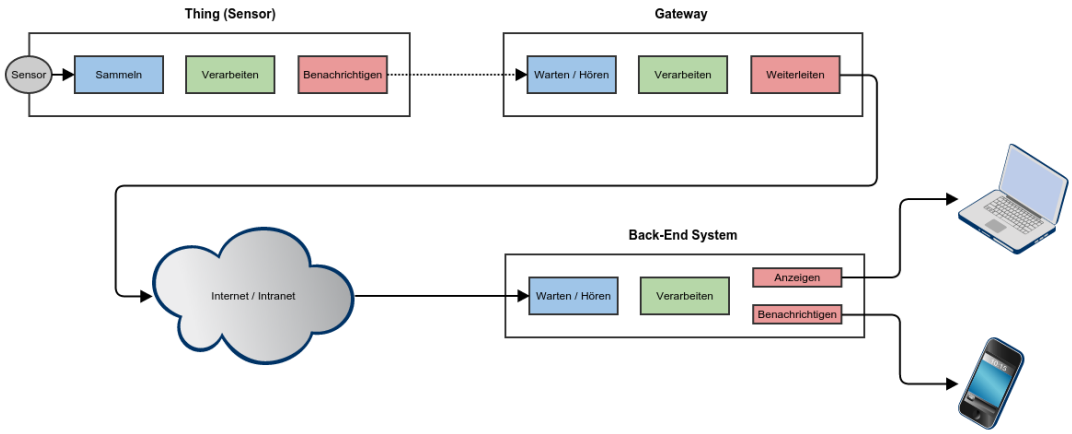

Abbildung 3.4: Beispielhafter Aufbau eines Kommunikationsweges innerhalb eines WSN.

3.4.1 Das „Thing“

Die Definition eines „Thing“ (oder auch „Smart Thing“), beziehungsweise eines „Dinges“, ist nicht ganz einfach. Grundsätzlich handelt es sich um einen physischen Gegenstand, ein „Embedded Device“ (auch Embedded System genannt) welches gewisse Funktionalitäten anbietet. Dies können Sensoren oder auch Kontroll- oder Steuerungselemente sein. Handelt es sich bei diesem „Ding“ um einen Alltagsgegenstand, wird dieser auch als „Smart Object“ bezeichnet. Ein „Embedded Device“ basiert auf einem Mikrocontroller, verfügt relativ gesehen nur über geringe Ressourcen (Prozessor, Arbeitsspeicher, Energie). Das „Embedded Device“ verfügt über einen definierten Stack an Kommunikationsprotokollen, welche entweder direkt oder über ein Gateway mit einem Back-End System kommuniziert.

Diese „Dinge“ und „Smart Objects“ sind nicht neu. Sie haben jedoch erst mit Aufkommen des IoT gelernt untereinander und mit dem Internet zu kommunizieren und autonom zu interagieren. Embedded Systems sind bereits heute überall gegenwärtig. Sei es in Autos, Wearables, Kühlschränken, Kaffeemschinen oder Smartphones.

Gemäss [Mor13] haben die „Things“ und die M2M-Kommunikation folgende Phasen durchlaufen:

- **Reactive information**
  Polling der Geräte für Informationen

- **Poactive information**
  Geräte Kommunizieren Informationen, wenn notwendig
- **Remotely controllable**
  Geräte sind aus der Ferne steuerbar
- **Remotely serviceable**
  Geräte sind aus der Ferne wartbar
- **Intelligent processes**
  Geräte sind Teil von intelligenten Prozessen
- **Optimised propositions**
  Informationen der Geräte können für die Entwicklung von neuen Produkten verwendet werden
- **New business models**
  Neue Möglichkeiten für Geschäftsmodelle
- **The Internet of Things**
  Teilung von Informationen mit Geräten von Dritt-Herstellern, um im gesamten einen Mehrwert zu generieren

3.4.2 Gateway / Bridge

Die Aufgabe eines Gateways, ab und zu auch als Bridge bezeichnet, ist es, „Things“ mit einem Netzwerk zu verbinden. Oft sind „Things“ nicht in der Lage sich direkt mit einem Netzwerk oder dem Internet zu verbinden. 85 % aller „Things“ sind heute nicht in der Lage direkt mit dem Internet zu kommunizieren [Cor14, S. 2]. In solchen Situationen wird eine Gateway eingesetzt, welches diese Lücke schliesst. Eine weitere Aufgabe eines Gateways kann die intelligente Filterung und Zusammenfassung von Daten sein, bevor diese an das Back-End System weitergeleitet werden. Dadurch wird verhindert, dass durch die „Things“ quasi ein Distributed Denial of Service (DDOS)-Angriff auf das Back-End System erfolgt.

3.4.3 Back-End Systeme

Die Back-End Systeme stehen oft in einem Netzwerk oder in der Public-Cloud, wo diese entsprechend den Bedürfnissen skaliert werden können. Zum Einsatz kommen Systeme zur Analyse und Auswertung der gesammelten Informationen. Bei grossen Datenmengen werden typischerweise Big-Data-Systeme eingesetzt. Je nach Anwendungsbereich können die gesammelten und ausgewerteten Daten über einen Computer oder ein mobiles Gerät abgerufen werden. Handelt es sich um ein Automatisierungssystem ist zusätzlich noch eine Steuer-Komponente integriert, welche auf Basis von Regeln und Benutzereingaben die Steuerung der Geräte übernimmt.

3.4.4 Das „Web of Things“

Das „Web of Things“ hat sich aus dem IoT entwickelt und ist ein Architekturansatz, welcher vorsieht alle IoT-Geräte in das bestehende Internet / Web einzubinden. Dazu sollen bereits für das Web etablierte Standards wie zum Beispiel HTTP, HTTPS, RSS oder REST zum Einsatz kommen. Auf jedem Gerät müsste ein minimaler Web-Server vorhanden sein, durch welchen die Integration in das Web erfolgt. Der Vorteil dieser Architektur liegt darin, dass alle bestehenden Möglichkeiten des Webs voll ausgenutzt werden können. Der grösste Nachteil ist, dass die heutigen Web-Protokoll nicht sehr sparsam mit Ressourcen (Hardware, Strom) umgehen.

3.4.5 Referenz Modelle und Referenz Architekturen

Zum heutigen Zeitpunkt gibt es mehrere Ansätze ein Referenz Model und eine Referenz Architektur für das IoT zu schaffen.

Nachfolgend werden einige dieser Referenz Modelle und Architekturen aufgelistet, jedoch aufgrund von deren Umfang und Komplexität nicht im Detail erläutert.

- **IoT-A**
  http://www.iot-a.eu/
- **WSO2 - A Reference Architecture for the Internet of Things**
  http://wso2.com/whitepapers/a-reference-architecture-for-the-internet-of-things/
- **IoT@Work - Final Framework Architecture Speciffication**
  https://www.iot-at-work.eu/downloads.html
- **Internet of Things Architecture (u.a. Cisco, IBM, Intel)**
  https://www.iotwf.com/iotwf2014/breakout

3.5 Unterschied zum Konzept des klassischen Internet

In der klassischen Auslegung des Internets stellen zentrale Server die Daten zur Verfügung, welche dann von Anwendern oder Bezügern abgerufen werden können. Die Datenhoheit liegt dabei bei diesen zentralen Servern. Kommuniziert wird entweder von Server zu Server oder von Anwender / Bezüger zu Server. Es war immer ein Server als intermediär notwendig. Das IoT bringt nun einen Paradigmenwechsel mit sich. Die Geräte in einem IoT sind oder werden in der Lage sein autonom miteinander zu kommunizieren und zu interagieren. Im klassischen Internet werden im Vergleich grosse Datenmenge pro Packet transportiert. Im IoT ist dies nicht mehr notwendig, dort werden in der Regel nur kleine Datenmengen, jedoch in sehr grossen Mengen, übermittelt. Auch ist es nicht essentiell, dass alle Pakete auch beim Ziel ankommen, da der Wert eines einzelnen Datenpacketes sehr gering ist.

Internet Protocol (IP) und Transmission Control Protocol over IP (TCP/IP) sind auf stabile, beziehungsweise stabile Netzwerke ausgelegt, in welchem grosse Datenmengen zuverlässig transportiert werden müssen. Die Anforderungen des IoT nach sich verändernden und

dynamischen Netzwerken, sehr kleinen Datenmengen, vielen Requests und eingeschränkten Geräten werden durch IP nicht optimal abgedeckt. Um den maximalen Nutzen zu erreichen werden neue, spezifische Protokolle benötigt

3.6 Effekte & Auswirkungen

Durch den Einsatz eines IoT kann Mehrwert in verschiedensten Bereichen und Ebenen geschaffen werden. Das IoT wird zahlreiche neue Geschäftsmodelle und Produktpaletten hervorbringen, welche sowohl einen Mehrwert für den Hersteller, als auch für den Konsumenten generieren. Neben den positiven Effekten sind auch die negativen Effekte nicht zu vernachlässigen. Im Kapitel 3.7 Herausforderungen werden die verschiedenen Herausforderungen erläutert, welche das IoT mit sich bringt.

3.7 Herausforderungen

Im IoT gibt es viele Herausforderungen zu bewältigen. Die wichtigsten werden nachfolgend aufgelistet:

- Verfügbarkeit eines Internet-Zuganges am Einsatz- / Verwendungsort
- Sicherheit und Datenschutz
- Tiefe Kosten für Hard- und Software
- Energieversorgung
- Energieverbrauch
- Skalierbarkeit
- Fehlertoleranz
- Akzeptanz
- Robustheit (physisch und logisch)
- Entdecken von Geräten und Services (Device Discovery)
- Fernwartung von Geräten und Anwendungen

Einige dieser Herausforderungen haben grössere Auswirkungen auf die Software-Entwicklung. Ist in der normalen Softwareentwicklung eine Trennung der spezifischen Themen (Separation of Concerns) ohne weiteres möglich, ist dies im IoT nicht ohne weiteres der Fall. Für die Entwicklung eines IoT-Endgerätes sind Fähigkeiten aus den Bereichen „Distributed Systems“, „Embedded Systems“, „Networking“, „Operating Systems“ und „Software Development“ erforderlich. Entweder muss ein Entwickler all diese Skills mitbringen oder die Architektur muss die Trennung / Abstraktion der verschiedenen Schwerpunkte erlauben, sodass diese vom jeweiligen Spezialisten implementiert werden können. Um diesem breiten Spektrum an notwendigen Fähigkeiten gerecht zu werden sind heutige allgemeine

DSL's nicht geeignet. Der Ansatz „one size fits all“ ist hier nicht ideal. Eine IoT DSL sollte entsprechende Mittel zur Abstraktion und Trennung vorsehen, um eine effiziente Entwicklung zu ermöglichen. Weitere Herausforderungen sind die Heterogenität der Geräte, die Skalierbarkeit und die Implikationen im Bezug auf die rechtlichen Aspekte. Zu klären ist hier, wem die gesammelten Daten gehören (dem Betreiber des Gerätes? dem Betreiber des Back-End-Systemes, welches für die Analyse benötigt wird?) und wer für Aktionen verantwortlich ist, welche das Gerät selbstständig ausgeführt hat.

Beispiel: Ein Kühlschrank bestellt bei einem Lebensmittelhändler eine grosse Menge an Milch nach, da gemäss den Berechnungen und Analysen des Systems der Milchvorrat zu niedrig ist. Der Besitzer benötigt diese Milch jedoch gar nicht. Da der Kühlschrank nicht haftbar ist, stellt sich nun die Frage nach einer haftbaren Person. Ist es der Besitzer? Die Herstellerfirma? Der Entwickler? Dieses Beispiel mag banal sein, aber in der Industrie oder in der Medizintechnik können kleine Berechnungsfehler oder Fehleinschätzungen massive Auswirkungen haben, was auch entsprechend hohe Kosten nach sich ziehen kann.

3.8 Anforderungen an das Produkt und die Software

Aufgrund der Besonderheiten der Domäne IoT ergeben sich auch einige spezifische Anforderungen an das Produkt und die verwendete Software. Das Gerät und die Software sollten für den sie bestimmten Zweck über ausreichend leistungsfähige Ressourcen verfügen und entsprechend skaliert werden können. Zugleich muss mit den Ressourcen so schonend wie möglich umgegangen werden, um eine möglichst lange Laufzeit zu erreichen. „Things“ verfügen meistens über eine mobile Energiequelle, wie zum Beispiel eine Batterie oder einen Akku, oder eine erneuerbare Energiequelle. Daneben spielt auch die Sicherheit und der Datenschutz eine grosse Rolle, da die Geräte meistens nicht permanent überwacht werden(können). Hinzu kommen auch regulatorische und rechtliche Aspekte, Herausforderungen und Anforderungen, welche es zu berücksichtigen gibt.

Aufgrund des grossen Anwendungsgebietes ist auch das Spektrum an Anforderungen entsprechend gross. Bei einem Kühlschrank, welcher Teil eines IoT's oder Heimautomationsnetzwerkes ist, steht die energieeffizient oder Grösse des Gerätes nicht zwingend eine grosse Rolle. Grund dafür ist, dass der Kühlschrank im Vergleich sehr viel Platz aufweist und an eine permanente Stromquelle angeschlossen ist.

Im Kapitel 4 Software Engineering in der Domäne „Internet of Things“ werden die Unterschiede und speziellen Anforderungen an das Software Engineering in der Domäne „Internet of Things“ im Vergleich zur „Standard Domäne“ aufgezeigt.

KAPITEL 4

Software Engineering in der Domäne „Internet of Things“

In diesem Kapitel werden die Anforderungen, Eigenheiten und Unterschiede des Software Engineering in der Domäne „Internet of Things“ im Vergleich zur „Standard Domäne“ (Domäne der normalen Software Entwicklung) aufgezeigt.

Die Struktur dieses Kapitels orientiert sich grob an der Struktur des Buches „Software Engineering Body of Knowledge“ [Soc04].

Die Unterschiede zur „Standard Domäne“ sind je nach betrachtetem Element / Bauteil („Things“, Gateway, Back-End System) unterschiedlich. Die Entwicklung eines Back-End-Systemes unterscheidet sich grundlegend von der Entwicklung eines „Things“ und eines Gateways. Auch gibt es weitere Unterschiede je nach Subdomäne, für welche eine Entwicklung stattfindet. In dieser Arbeit wird der Fokus auf „Things“ und Gateways gelegt. Die Entwicklung für diese beiden Elemente wird stark durch die Domäne der „Embedded Systems“ geprägt.

4.1 Software Requirements

Im Requriements Engineering gibt es im Vergleich zur Standard Domäne nur wenige Unterschiede. Der Hauptunterschied liegt darin, dass gewisse Anforderungen implizit durch die Domäne vorgegeben sind. Betroffen sind dabei sowohl Anforderungen auf Ebene Hard- und Software, als auch auf Ebene Prozess. Nachfolgend werden die wichtigsten Anforderungen aufgelistet.

- Kostengünstig
- Energieeffizient
- Optimale Ressourcenausnutzung
- Betrieb ohne permanenten Stromanschluss
- Hohe Skalierbarkeit

- Lange Lebensdauer
- Geringer Wartungsbedarf
- Verlässlich
- Vertrauenswürdig
- Adäquate Kommunikationsprotokolle
- Möglichkeit für das Einspielen von Updates
- Erfüllung der rechtlichen und regulatorischen Anforderungen, Einschränkungen und Auflagen

Die Anforderungen auf Ebene Software haben einen direkten Einfluss auf die Hardware, die verwendet werden kann oder verwendet werden soll. Je nach dem steuern die Anforderungen an die Software die Hardware oder es gibt explizite Anforderungen an die Hardware, welche Anforderungen und Rahmenbedingungen an die Software vorgeben.

Im Bereich IoT ist es wichtig, dass die Anforderungen genau spezifiziert und korrekt umgesetzt werden. Eine spätere Korrektur kann sich als schwierig herausstellen. Wurden zum Beispiel für ein Sensor-Netzwerk mehrere Tausend Sensoren verteilt, welche nicht direkt mit dem Internet kommunizieren können, kann der Update-Prozess sehr aufwändig werden. Daher müssen entweder sämtliche Anforderungen bei Inbetriebnahme umgesetzt worden sein oder es muss ein entsprechender Update-Mechanismus implementiert werden.

Weitere Herausforderungen sind der Einsatz in einem unbekannten Umfeld (Umwelt & Technisch) und die schwierige Feststellbarkeit von Fehlern. Liefert ein Sensor konstant falsche Werte, kann dies aus der Ferne, beziehungsweise ohne Vergleichswert praktisch nicht festgestellt werden. Ein mögliches Szenario wäre ein Sensor, welche in einer Fertigungsanlage auf einem Fliessband die Stückzahlen erfasst. Sind diese nicht korrekt, beziehungsweise weichen von der effektiven Anzahl um ein paar Prozent ab, und ein Back-End-System bestellt auf Basis dieser Stückzahlen Nachschub für Einzelteile, kann dies massive Auswirkungen haben.

Neben den Anforderungen von Seiten des Kunden, beziehungsweise des Herstellers, gibt es auch noch regulatorische und gesetzliche Anforderungen zu erfüllen. Hier stellt sich zum Beispiel die Frage der Haftung, wenn ein System auf Basis der Informationen eines anderen Systemes autonom eine Aktion auslöst, wobei die gelieferten Informationen nicht korrekt waren.

Die Funktionalen Anforderungen an die Software an „Things“ weisen in der Regel eine geringe bis mittlere Komplexität auf. Grund dafür ist die Aufgabe des Gerätes. Diese besteht darin Sensorwerte auszulesen und an ein Gateway zu senden. „Things“ können aber nicht nur Sensorwerte bereitstellen, sondern auch Bedien- und Steuerelemente aufweisen. Das können zum Beispiel Motoren zur Steuerung eines Rollladens oder ein Display zur Anzeige der Temperatur sein.

Die Anforderungen an ein Gateway wiederum, können sehr komplex sein, da unter Umständen verschiedenste Protokolle unterstützt, Daten gefiltert und aggregiert und an das Back-End-System gesendet werden müssen. Auch obliegt es oft den Gateways die Back-End Systeme vor einer Überflutung mit Daten zu bewahren. Sie entscheiden wann ein Datenpacket an das Back-End System weitergeleitet wird. Die Anforderungen an Back-End Systeme entsprechen im grossen und ganzen denjenigen von Big-Data Systemen.

4.2 Software Design & Software Construction

Software Design und Software Construction unterscheiden sich bei den drei Hauptelementen des IoT („Things“, Gateways, Back-End Systeme) sehr stark voneinander. Über alles hinweg gesehen sollte ein möglichst hoher Abstraktionsgrad angestrebt werden, um die Interaktion mit unbekannten Geräten zu ermöglichen und zu vereinfachen. Durch den konsequenten Einsatz von Standards kann die Komplexität erheblich reduziert und die Wiederverwendbarkeit erhöht werden. Vor allem im Bereich Kommunikationsprotokolle gibt es einige IoT spezifische Standards, welche sich jedoch noch nicht flächendeckend durchgesetzt haben. Standards im Bereich Architektur, Device-Discovery und Service Beschreibung konnten sich noch nicht etablieren.

Zwei weitere zentrale Punkte sind Sicherheit und Vertraulichkeit. Sicherheit und Vertraulichkeit der Systeme und Daten sollte auf jeder Ebene, vom Endgerät bis zum Back-End System, berücksichtigt werden. Ein wichtiger Merkpunkt ist hier, je kleiner der implementierte Stack an Funktionalitäten, desto kleiner die Angriffsfläche des Systemes.

Das Design sollte zudem berücksichtigen, dass die „Things“ über lange Zeit und unter unbekannten Umständen und Einflüssen in Betrieb sind. Eventuell können die „Things“ auch nicht regelmässig mit Software-Updates ausgestattet werden, wohingegen sich die Back-End Systeme (und allenfalls Gateways) laufend weiterentwickeln. An den Schnittstellen zwischen den Hauptelementen sind hier entsprechende Vorkehrungen zu treffen, sodass auch eine Rückwärtskompatibilität gewährleistet ist.

Wie auch in normaler Software sollten alle Funktionalitäten mit Unit- und Integrations-Tests geprüft werden.

4.2.1 Things

Aufgrund der Ressourceneinschränkungen auf den „Things“ wird meistens eine hardwarenahe Programmiersprache wie C oder C++ verwendet. der Vorteil dieser Sprachen liegt darin, dass sie im Vergleich wenige Ressourcen benötigen jedoch doch einen gewissen Komfort und Abstraktionsgrad bieten. Auch der Einsatz von Assembler kann sinnvoll sein, gerade wenn es sich um ein „Thing“ handelt, welches über eine serielle oder parallele Schnittstelle direkt mit einem Gateway verbunden ist.

Für die Kommunikation mit dem Gateway ist der Einsatz ein einfachen und sparsamen Protokolls anzuraten. Die übermittelte Datenmenge ist grundsätzlich immer sehr klein und die Wichtigkeit eins einzelnen Datenpaket ist sehr gering. Daher sind für die Kommunikation

leichtgewichtige Protokolle zu bevorzugen. Meistens ist es auch nicht notwendig auf dem Gerät den vollen Protokoll-Stack zu implementieren. Der Client-Teil zur Übermittlung der Daten ist oft ausreichend (Read-Only).

Das Endgerät selbst bietet nur ein geringes Set an Funktionalitäten an. Die komplexen Funktionen sind entweder in den Back-End Systemen oder den Gateways realisiert. Nicht benötigte Funktionen sollten unbedingt weggelassen werden. Der Einsatz von Frameworks bietet sich insbesondere für die Abstraktion des Kommunikationsprotokolls und allfälliger Device-Management-Features an. Dabei ist jedoch der Ressourcenverbrauch der Frameworks bei der Auswahl zu berücksichtigen.

Das Endgerät stellt typischerweise keine direkte Präsentationsschicht und Datenhaltungsschicht zur Verfügung. Die Präsentation erfolgt entweder über ein Gateway oder über ein Back-End-System. Die Datenhaltung, beziehungsweise Historisierung, wird auch durch ein Back-End System oder ein Gateway implementiert.

4.2.2 Gateways

Ein Gateway verfügt über leistungsfähigere Hardware als ein „Thing“, da es einen grösseren Umfang an Funktionalitäten abdecken muss. Auf einem Gateway ist der Einsatz einer Hochsprache sinnvoll, da dadurch ein wesentlich höherer Abstraktionsgrad erreicht werden kann. Auch der Einsatz von Frameworks ist sinnvoll um Querschnittfunktionalitäten umzusetzen.

Da das Gateway über leistungsfähigere Ressourcen verfügt ist die Auswahl des Betriebssystemes und der Programmiersprache nicht mehr so essentiell, wie bei einem „Thing“. Wichtig ist, dass das Gateway eine breite Palette an Kommunikationsprotokollen unterstützt. Sowohl in Richtung der Endgeräte, als auch in Richtung der Back-End Systeme.

4.2.3 Back-End Systeme

Bei Back-End Systemen kommen klassische Enterprise- und Big-Data-Technologien zum Einsatz. Die Auswahl an verschiedenen Technologien und Produkten ist sehr umfangreich und wird deshalb hier nicht näher erläutert. Der verfügbare Stack an Protokollen spielt hier nicht mehr eine so grosse Rolle. Die Datenübertragung zwischen Gateway und Back-End System erfolgt über IP (v4 oder v6).

4.2.4 Betriebssysteme

Für IoT-Endgeräte und -Gateways werden in der Regel „Embedded Operating Systems“ verwendet. Nachfolgend werden einige „Embedded Operating Systems“ aufgelistet, welche heute verfügbar sind.

- TinyOS
- Contiki

- Mantis
- FreeRTOS
- LiteOS
- RIOT
- Saphire OS
- Google Brillo (Betriebssystem für „Things“ im Bereich Smart Homes)

4.2.5 Programmiersprachen

In den nachfolgenden Abschnitten werden einige Programmiersprachen, beziehungsweise Gruppen von Programmiersprachen, im Kontext von IoT erläutert. Diese dienen nur als Illustration. Es gibt noch zahlreiche weitere Programmiersprachen, welche für die Entwicklung von Software in dieser Domäne geeignet sind.

IoT DSL's gibt es in vielerlei Ausprägungen. Diese sind in der Regel Teil eines spezifischen Frameworks, einer Library oder einer Middleware. Im vierten Quartal von diesem Jahr wird voraussichtlich Google Weave erscheinen. Google Weave ist eine API und eine Sammlung von Tools für die Kommunikation von IoT-Geräte welche das Betriebssystem Google Brillo verwenden. Google Weave wäre eine DSL, welche im Kontext von Google Brillo eingesetzt wird. DSL's von spezifischen Produkten werden aus Gründen des Umfanges an dieser Stelle nicht näher betrachtet.

Assembler

Eine Implementation in Assembler kann unter bestimmten Voraussetzungen die beste Lösung für ein bestimmtes Problem sein. In der Regel wird der Assembler-Ansatz gewählt, wenn das Programm so effizient und sparsam wie möglich ablaufen und mit so wenig Ressourcen als möglich auskommen soll. Im Gegenzug erfordert die Entwicklung und Wartung mehr Aufwand und der Code ist erheblich komplexer, als wenn dieser in einer Hochsprache geschrieben wurde.

Allenfalls kann es sinnvoll sein, nur gewisse kritische Teile der Anwendung in Assembler zu implementieren.

C / C++

C und C++ kommen zum Einsatz, wenn eine hardwarenahe Programmierung erforderlich, aber trotzdem ein gewisses Niveau an Komfort und Abstraktion bei der Programmierung angestrebt wird. C und C++ werden im grossen und ganzen auf allen gängigen Betriebssystemen unterstützt. Der Code ist zwar nicht ohne Anpassung auf jedem Betriebssystem lauffähig. Der Aufwand dafür sollte sich in der Regel jedoch in Grenzen halten.

Java / .NET

Insbesondere bei Gateways und Back-End-Systemen ist die Implementation in einer gängigen oder spezifischen Hochsprache wie Java oder einer Sprache aus dem .NET-Framework sinnvoll.

Eine Hochsprache benötigt mehr Ressourcen, was auf „Things“ und Gateways unter Umständen zu Problemen führen kann. Für einige Hochsprachen gibt es Embedded-Varianten (Zum Beispiel Java ME Embedded), welche für den Einsatz auf solchen Systemen optimiert sind. Im Vergleich zu einer Implementation in C oder C++ werden relativ viele Hardwareressourcen benötigt und werden daher meistens nur auf leistungsfähigeren, beziehungsweise teureren, Systemen eingesetzt.

4.2.6 Standards

Existieren in einer Domäne Standards für die häufigsten Problemstellungen hat dies für den Endkonsumenten einen positiven Effekt. Durch die Standardisierung muss er sich nicht an einen spezifischen Hersteller binden und kann Produkte von verschiedenen Herstellern miteinander kombinieren. Er hat bei der Auswahl seiner Geräte beinahe völlige Wahlfreiheit. Im Gegensatz dazu sind die grossen Hersteller bestrebt, die Kunden so fest als möglich an sich zu binden, damit diese weitere Produkte aus ihrer Angebotspalette kaufen. Eine Standardisierung wird daher oft von grösseren Unternehmen erschwert. Oft werden Standards erst definiert, wenn sich ein Protokoll oder eine Methodik erst etabliert, wenn sich diese auf dem Markt durchgesetzt hat. Mit dem IoT besteht nun die Chance bereits früh einen gewissen Grad an Standardisierung zu erreichen. Ein hoher Grad an Standardisierung impliziert einen höheren Abstraktionsgrad, was wiederum einige positive Effekte auf die Verwendung und Entwicklung der Produkte hat. Da das IoT an sich nichts neues ist, sondern nur eine weiterer Schritt ein einer langen Kette von Entwicklungen, wurden bereits einige spezifische Standards entwickelt. Insbesondere für Übertragungs- und Kommunikationsprotokolle wurden mehrere Standards definiert. Wohingegen für andere Aspekte wie die Device Discovery oder die Service Beschreibung noch übergreifende Standards fehlen.

In der nachfolgenden Grafiken werden die wichtigsten IoT-Protokolle im OSI-Schichtenmodell dargestellt.

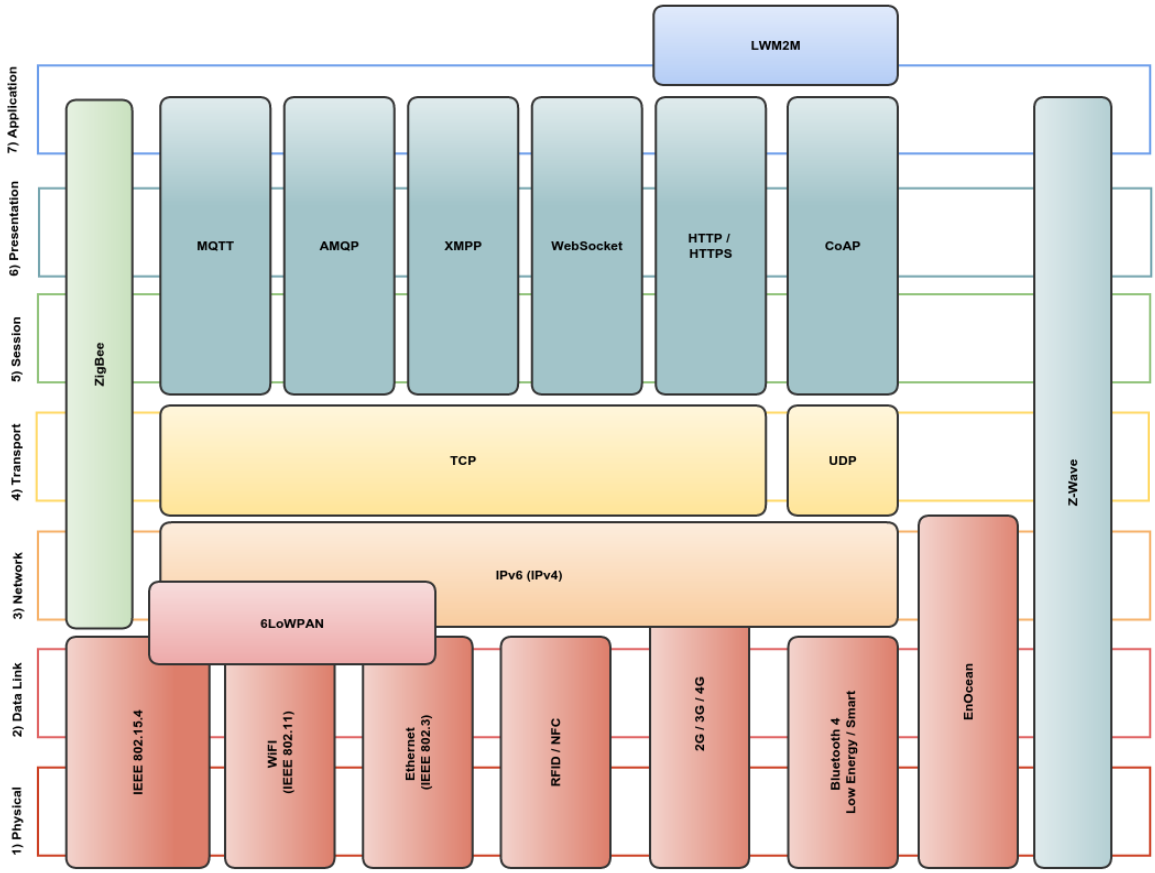

Abbildung 4.1: IoT-Protokollstack

4.2.7 Standards: Kommunikationsprotokolle

Die ersten Standards für die IoT-Welt haben sich im Bereich der Kommunikationsprotokolle entwickelt. Einige dieser Kommunikationsprotokolle wurden spezifisch für das IoT entwickelt, andere gibt es schon länger und wurden ursprünglich für einen anderen Zweck entwickelt. Nachfolgend werden die wichtigsten Kommunikationsprotokolle für den Bereich IoT beschrieben.

MQTT

Das Message Queue Telemetry Transport (MQTT) ist ein leichtgewichtiges „Publish / Subscribe“ Protokoll welches für die Messungen aus der Ferne, Überwachung und Datensammlung eingesetzt wird. Es wird vorwiegend in grossen Netzwerken mit vielen kleinen Geräten eingesetzt, welche überwacht und kontrolliert werden müssen. Die Hauptaufgabe von MQTT liegt beim Sammeln von Daten von vielen verschiedenen Endgeräten. MQTT hat eine einfache Paket-Struktur und basiert auf TCP/IP, wodurch sichergestellt wird,

dass keine Daten verloren gehen. MQTT ist dafür ausgelegt die Daten zu Sammeln und anschliessend an Enterprise Technologien, wie zum Beispiel einen Enterprise Service Bus (ESB), weiterzugeben.

Das Protokoll ist jedoch nicht für die direkte Device-2-Device (D2D)-Kommunikation geeignet und arbeitet nicht in Real Time. Die Daten werden mit einer Verzögerung von bis zu mehreren Sekunden übermittelt und verteilt.

Für MQTT wird ein zentrale Server, ein sogenannter MQTT- oder Message-Broker benötigt. Dieser ist dafür zuständig die eingehenden Nachrichten den interessierten Clients zu verteilen. Die Clients senden dem Message-Broker entweder Daten für ein bestimmtes Thema („Topic“) oder registrieren sich für ein bestimmtes Thema, um die übermittelten Daten zu erhalten.

XMPP

Das Extensible Messaging and Presence Protocol (XMPP) wurde ursprünglich als Protokoll für den Instant Messaging Dienst Jabber entwickelt. XMPP ermöglicht eine Text-Kommunikation zwischen „Punkten“. Die Kommuunikation erfolgt nicht direkt von Punkt zu Punkt sondern über einen dezentralisierten Server. XMPP ist ein offener Standard und somit kann jeder seinen eigenen Server betreiben. Die Stärken von XMPP liegen bei der Adressierung, der Sicherheit und der Skalierbarkeit, wobei aktuell noch keine End-To-End Verschlüsselung unterstützt wird. Die Adressierung erfolgt über das Schema *username@domain.com/resource*. Das Protokoll ist ideal für Consumer-Orientierte Anwendungen.

Auch XMPP basiert auf TCP/IP, ist aber auch in der Lage über Hypertext Transfer Protocol (HTTP) oder WebSocket's zu kommunizieren. Der Datenaustausch erfolgt in „near-real-time“, also beinahe Echtzeit.

AMQP

Das Advanced Message Queuing Protocol (AMQP) ist ein binäres transaktionsbasiertes Nachrichten Protokoll, welches für den Einsatz in Nachrichten-orientierter Middleware konzipiert wurde. Es ist in der Lage tausende von zwischengespeicherten Transaktionen ohne Datenverlust zu verarbeiten. Das Protokoll ist darauf ausgelegt keine Daten zu verlieren. Als Basis dient ein zuverlässiges, beziehungsweise im Hinblick auf Daten verlustfreies, Transport-Protokoll wie zum Beispiel TCP/IP und verschieden definierbare Nachrichtenübermittlungsgarantien. AMQP bietet verschiedene Kommunikationsarten wie zum Beispiel „Publish / Subscribe“, „Request / Response“ und „Store and Forward“.

Im Gegensatz zu anderen Standardisierungsversuchen im Middleware-Bereich setzt AMQP nicht auf dem API-Level, sondern auf dem Wire-Level. Somit wird eine Kommunikation zwischen verschiedenen Middleware-Plattformen ermöglicht.

HTTP

Das HTTP ist ein auf TCP/IP zustandsloses Protokoll zur Übermittlung von Daten auf Anwendungsebene. HTTP wird im Web praktisch überall eingesetzt und kann sich auch für den Einsatz im IoT eigenen.

WebSocket

WebSocket ermöglicht eine Voll-Duplex Kommunikation über eine einzelne Transmission Control Protocol (TCP)-Verbindung. Das WebSocket Protokoll ermöglicht eine bessere Kommunikation zwischen Browsern und Webseiten / Servern. Jeder der Verbindungsteilnehmer kann jederzeit Daten an den anderen Teilnehmer senden.

CoAP

Das Constrained Application Protocol (CoAP) ist ein leichtgewichtes, binäres Protokoll, welches speziell für kleine, beziehungsweise ressourcenarme Geräte konzipiert wurde. Semantisch ist CoAP and HTTP ausgerichtet und die Übersetzung von CoAP nach HTTP ist möglich. Wie HTTP basiert auch CoAP auf dem REST-Modell. Es können somit Ressourcen unter bestimmten URL's angeboten werden, welche über GET, POST, PUT und DELETE angesprochen werden können. CoAP zeichnet sich durch seine Einfachheit, Multicasting-Fähigkeit, den geringen Overhead, die Unterstützung von Geräte-Erkennug (Device-Discovery), einen einfachen „Publish / Subscribe"-Mechanismus und ein einfaches Caching aus. CoAP verwendet User Datagram Protocol (UDP) als Transportprotokoll.

LWM2M

Das Lightweight M2M Protokoll ist ein Device-Management-Standard der Open Mobile Alliance. Das Protokoll selbst basiert auf CoAP und ist als Client-Server-Protokoll ausgelegt.

4.2.8 Standards: Übertragungsprotokolle

Die Anforderungen der Domäne IoT erfordern auch Neuerungen im Bereich der Übertragungsprotokollen und -standards. Nachfolgend werden die wichtigsten Übertragungsprotokolle und -standards für den Bereich IoT kurz beschrieben.

IEEE 802.15.4

Der Standard IEEE 802.15.4 wurde speziell für den Einsatz auf eingeschränkten Geräten, beziehungsweise eingeschränkten Netzwerken entwickelt und gilt als Schlüsselfaktor für die Etablierung des IoT. Dieser Standard bildet die Basis für weitere Protokolle und Standards wie ZigBee, WirelessHART oder 6LoWPAN. Der Standard definiert für die Übertragung die Frequenzbänder 868/915 MHz und 2.4 GHz. Es werden Stern und Punkt-zu-Punkt Netzwerke unterstützt.

ZigBee

ZigBee ist ein offener Standard für drahtlose Netzwerke, welcher von der ZigBee Alliance entwickelt wird. Die Basis von ZigBee bildet der Standard IEEE 802.15.4. ZigBee ist speziell für IoT-Endgeräte ausgelegt und arbeitet daher besonders energieeffizient und ressourcenschonend. Aktuell werden Punkt-zu-Punkt, Stern und Mesh Netzwerk-Topologien unterstützt. Zusätzlich sind im Funktionsumfang verschlüsselte Verbindungen, Collision-Avoidance und Acknowledgements enthalten. Innerhalb von ZigBee gibt es weitere Standards auf Applikations-Ebene für bestimmte Anwendungsbereiche wie zum Beispiel Home-Automation oder Smart Energy. Die Reichweite des normalen ZigBee Standards im Freien beträgt zwischen 10 und 100 Metern.

Z-Wave

Z-Wave ist ein Kommunikationsstandard für die drahtlose Übermittlung von Daten im Bereich der Heimautomation. Er wird von der Firma Sigma Designs in Zusammenarbeit mit der Z-Wave Alliance entwickelt und ist ebenfalls auf einen geringen Energieverbrauch ausgelegt. Der Standard verwendet Frequenzen im Bereich von 850 bis 950 MHz und ist als Mesh-Netzwerk ausgelegt, wobei maximal 4 Hops und maixmal 232 Geräte zugelassen sind. Die Verwaltung des Netzwerkes erfolgt durch einen Primärcontroller. Die Reichweite im freien Gelände beträgt ca. 150 Meter.

Zusätzlich zum Übertragungsstandard können Z-Wave-Geräte auf Anwendungsebene in verschiedene Klassen eingeteilt werden. Pro Geräteklasse müssen durch den Hersteller bestimmte Pflichtkommandos implementiert werden. neben den Pflichtkommandos kann der Hersteller weitere Funktionalitäten implementieren. Diese müssen jedoch mit dem Z-Wave-Standard konform sein.

6LoWPAN

6LoWPAN steht für „IPv6 over Low Power Wireless Personal Area Networks“ und ist ein auf dem Standard IEEE 802.15.4 basierendes Protokoll zur drahtlosen Datenübermittlung. Es ermöglicht die Übermittlung von IPv6-Packeten über den IEEE 802.15.4 Standard und wurde spezifisch für den Einsatz auf eingeschränkten Geräten entwickelt.

6LoWPAN kann mit jedem anderen IP-Netzwerk (Ethernet oder Wi-Fi), beziungsweise IP fähigen Gerät kommunizieren.

ANT

ANT ist ein proprietärer Funkstandard der Firma Dynastream und wurde ursprünglich für die Messung und Überwachung im Bereich Sport und Fitness geschaffen. Mit ANT können drahtlose Personal Area Network (PAN)'s gebildet werden. Ebenfalls wird Mesh-Networking unterstützt. Auch ANT verwendet das Frequenzband um 2.4 GHz.

Bluetooth Smart / Bluetooth Low Energy

Bluetooth Low Energy (oder auch Bluetooth Smart) ist eine optionale Erweiterung des Bluetooth Standards und ist auf den Einsatz auf eingeschränkten Geräten optimiert. wie auch der normale Bluetooth Standard verwendet Bluetooth Low Energy das 2.4 GHz Frequenzband.

EnOcean

EnOcean ist ein Funkstandard der gleichnamigen Firma EnOcean. Das Hauptmerkmal dieser Technologie ist der Fakt, dass auf der Sender-Seite nicht zwingend eine Energie-Quelle vorhanden sein muss. Die zur Übermittlung notwendige Energie wird durch eine physische Interaktion gewonnen (zum Beispiel Betätigung eines Lichtschalters). EnOcean verwendet die Frequenzen 902 MHz, 928.35 MHz und 315 MHz.

Weitere Standards

- DASH7 (Open Source RFID-Standard)
- ISA100.11a (Drahtloser Netzwerk-Standard der International Society of Automation (ISA), basiert auf IEEE 802.15.4)
- MiWi (Proprietäres drahtloses Netzwerkprotokoll basierend auf IEEE 802.15.4)
- WirelessHART (Protokoll für drahtlose Sensor-Netzwerke basierend auf IEEE 802.15.4 und dem Highway Addressable Remote Transducer Protocol (HART))
- EtherCAT (Ethernet for Control Automation Technology, Echtzeit-Ethernet Protokoll)
- NFC (Near Field Communication, Standard zum drahtlosen Austausch von Daten über kurze Distanz)
- RFID (Radio Frequency Identification, Standard zur berührungslosen Identifikation und Lokalisierung)
- WAMP (Web Application Messaging Protocol, Offener Standard, Subprotokoll des WebSockets Protokoll)

4.2.9 Frameworks / Erweiterte Protokoll-Stacks

In diesem Kapitel werden einige Frameworks und Erweiterungen von Protokoll-Stacks vorgestellt. Frameworks vereinfachen die Entwicklung und erlauben einen höheren Abstraktionsgrad in der Anwendung.

Thread

Thread ist eine drahtloses Netzwerk-Protokoll welches auf 6LoWPAN und somit auf dem Standard IEEE 802.15.4 und IPv6 basiert. Thread erweitert 6LoWPAN um zusätzliche Funktionalitäten im Bereich Sicherheit, Routing und Setup des Gerätes. Genauere Details zu Thread konnten nicht gefunden werden. Weitere Informationen sind nur über eine kostenpflichtige Mitgliedschaft in der ThreadGroup erhältlich. Thread steht in direkter Konkurrenz zu Iotivity und AllJoyn.

IoTivity

IoTivity ist ein Open Source Software Framework des Open Interconnect Consortium (OIC). Ziel von IoTivity ist eine Referenz-Implementation der vom OIC definierten IoT-Standards. Das Framework bietet Funktionen zur Device Discovery, Verbindungs- und Ressourcen-Management. Zusätzlich sind folgende Module enthalten:

- Virtual (Soft) Sensor Manager
  Aggregierung des Inputs von einem oder mehreren Sensoren und Darstellung als virtuelle Ressource. Die Sensor-Daten können über Queries abgefragt werden.
- Protocol Plugin Manager
  Integrationsmöglichkeit für nicht Open Interconnect Consortium (OIC)-Protokolle.
- Things Manager
  Verwaltung von Ressourcen-Gruppen und Ausführen von Gruppen-Aktionen.
- Ressource Offloading / Notification Manager
  Verwaltung von Ressourcen für eingeschränkte Geräte.
- Smart Home Protocol
  Framework mit Services zur Implementation eines Heimautomations-Controllers.

Bei OIC Mitglied sind unter anderem Cisco, Intel, Atmel, Honeywell, Dell, Siemens, Acer, Realtek und Hewlett Packard.

AllJoyn

AllJoyn ist ein Open Source Framework der Allseen Alliance. Das Framework bietet Kern-Services zur Erkennung (Discovery) von Geräten und Anwendungen und vereinfacht die Kommunikation und Interaktion mit unbekannten Geräten und Anwendungen. Die Allseen Alliance hat eine grosse Anzahl an Mitgliedern, darunter Canon, Microsoft, Qualcomm, Sony, Cisco, HTC und Lenovo.

4.2.10 Weitere

- ZeroMQ von iMatix
- mbed von ARM
- Internet of Things Platform von Intel

4.3 Software Testing

Das Testing, ist wie in allen Domänen, ein essentieller Bestandteil des Software Entwicklungs Prozesses. In dieser Domäne ist das Testing noch wichtiger, da die IoT-Endgeräte allenfalls niemals ein Update erhalten (können). IoT-Endgeräte werden in grossen Massen hergestellt und auch eingesetzt. Wurden zum Beispiel in einer Öl-Pipeline durch die Wüste mehrere Tausend Sensoren installiert, welche Messwerte und Überwachungsdaten liefern, kann sich die Korrektur eines Fehlers als sehr schwierig herausstellen. Entweder verfügt das Gerät über eine entsprechende Update-Funktionalität und hat eine Verbindung zu einem Back-End System, der Update erfolgt manuell durch einen technischen Mitarbeiter oder das Gerät wird komplett ausgetauscht. Trotzdem, dass ein Gerät über eine Update-Schnittstelle und eine Verbindung zu einem Netzwerk verfügt, kann es vorkommen, dass ein Update aufgrund mangelnder Hardware-Ressourcen oder zu geringer Bandbreite nicht übertragen oder eingespielt werden kann.

Auch haben Fehler üblicherweise grosse Auswirkungen und sind oft von der Ferne nur schwer festzustellen. Liefern die Sensoren der erwähnten Öl-Pipeline falsche Daten über den Zustand der Leitung, wird gegebenenfalls ein Leck oder ein entstehender Riss nicht rechtzeitig entdeckt, was schlussendlich in einem finanziellen Schaden resultiert.

Dies führt auch bereits zu einem weiteren wichtigen Punkt. IoT-Endgeräte und zum Teil auch Gateways können unter verschiedensten physikalischen Bedingungen eingesetzt werden. Bei Servern kann man in der Regel davon ausgehen, dass diese in einem mehr oder weniger gut klimatisierten Rechenzentrum oder einem Server-Raum stehen. Die Bandbreite an physikalischen Einflüssen ist relativ überschaubar. Bei IoT-Geräten sieht das etwas anders aus, dort sind die Einflüsse und Bedingungen nur schwer vorhersehbar. Daher haben die meisten Hersteller entsprechende Labor-Umgebungen, wo Hard- und Software unter Extrem- und Ausnahmebedingungen getestet werden.

Nicht nur die Umwelt kann einen Einfluss auf die Geräte haben, sondern auch das verwendete Netzwerk oder die Architektur des Netzwerkes. Auch dafür werden entsprechende Testumgebungen benötigt. Zusätzlich muss die Interaktion mit unbekannten Drittgeräten getestet werden.

Der Funktionsumfang der IoT-Endgeräte ist in der Regel sehr überschaubar und die Funktionalitäten weisen eine geringe Komplexität auf. Dadurch wird der Test der eigentlichen Funktionalität etwas vereinfacht, was aber durch die bereits oben erwähnten Punkte mehr als kompensiert wird.

Für Gateways und Back-End Systeme wird für das Software Testing analog der Standard-Domäne verfahren. Bei den Gateways ist zusätzlich sicherzustellen, dass diese über die korrekten Versionen der eingesetzten Kommunikationsstacks verfügen. Ist dies nicht der Fall, können die IoT-Endgeräte unter Umständen nicht mehr mit dem Gateway kommunizieren.

4.4 Software Maintenance

Die Maintenance von Software welche für IoT-Endgeräte entwickelt wurde ist schwierig, beziehungsweise aufwändig. Wie bereits im vorangehenden Kapitel aufgezeigt, kann die Software auf IoT-Endgeräte schwierig zum updaten sein. Ist ein entsprechender Update-Mechanismus vorhanden, ist ein besonders Augenmerk auf den Rollout zu legen, da schnell mehrere 10’000 Geräte betroffen sein können. Hier stellen sich Fragen wie: Wann wird das Update eingespielt? Welche Geräte erhalten das Update in welcher Reihenfolge? Was sind die Auswirkungen während / nach dem Update? Wie werden die Besitzer der Geräte informiert? Gibt es Abhängigkeiten zwischen verschiedenen Software-Versionen? Gibt es Abhängigkeiten zwischen Endgerät, Gateway und Back-End System?

Unter Umständen können gewisse Geräte nicht aktualisiert werden. Daher muss sichergestellt werden dass das Gateway und gegebenenfalls auch das Back-End System rückwärts kompatibel bleiben. Durch ein vorausschauendes und gutes Schnittstellendesign kann diese Situation umgangen oder entschärft werden.

4.5 Software Configuration Management

Um ein Software-Update erfolgreich auszurollen ist das Configuration Management sehr wichtig. Zuerst muss der Update-Prozess in einer Testumgebung für die sich im Betrieb befindlichen Geräte getestet werden. Dafür müssen Informationen bezüglich eingesetzten Versionen von Hard- und Software aus dem Configuration Management herangezogen werden. Sind sämtliche Tests gut verlaufen kann der entsprechende Update ausgerollt und nach und nach die dazugehörigen Configuration Items aktualisiert werden.

4.6 Software Engineering Management

Das Software Engineering Management entspricht grundsätzlich dem Vorgehen in der Standard-Domäne. Es müssen jedoch allen Beteiligten die Eigen- und Besonderheiten dieser Domäne bekannt sein. Besonderes Gewicht ist dem Risk und Quality Management beizumessen.

4.7 Software Engineering Process

Der Software Entwicklungsprozess von IoT-Anwendungen ist je nach Element unterschiedlich. Die Entwicklung für ein IoT-Endgerät ist relativ kurz, wobei für die Implementation viele verschiedene Fähigkeiten erforderlich sind. Im Gegensatz dazu ist der Entwicklungsprozess von Gateways und Back-End Systemen aufwändiger und entspricht eher dem Entwicklungsprozess in der Standard-Domäne.

4.8 Software Engineering Tools and Methods

Die eingesetzten Tools und Methoden sind wiederum vom betroffenen Element abhängig. Bei Endgeräten kommen Werkzeuge und Methoden zum Einsatz, welche sich besonders für die Entwicklung und das Testing von Embedded Systems eignet. Allenfalls müssen sogar eigene Tools entwickelt werden, um den Software Entwicklungsprozess und das Testing optimal zu unterstützen. Denkbar wäre hier zum Beispiel eine Simulationsanlage zum automatisierten Testen der Geräte unter verschiedenen physikalischen Bedingungen und Umwelteinflüssen. Als Programmiermethoden kommen eher strukturierte, objektorientierte oder spezifische (zum Beispiel bei Real-Time Anwendungen / Betriebssystemen) Methoden zur Anwendung.

Die Entwicklung der Gateways und Back-End Systeme bedient sich im grossen und ganzen der Werkzeuge und Methodiken der Standard- und Big-Data Domäne.

4.9 Software Quality

Die Software Qualität ist im Bereich IoT extrem wichtig, insbesondere im Bereich der Endgeräte. Da das Ziel der klassischen Softwareentwicklung ebenfalls ein hohe Qualität ist, gibt es auch hier grundsätzlich keinen Unterschied im Rahmen der Software Entwicklung.

4.10 Verwandte Disziplinen

Es gibt einige weitere Disziplinen, welche aufgrund der starken Innovationskraft einen grossen Einfluss auf die Domäne „Internet of Things“ und das Software Egineering haben. Dies sind unter anderem das Computer Engineering, Computer Science, Mathematik und System Engineering.

KAPITEL 5

Schlusswort & Fazit

5.1 Fazit

Bereits heute gibt es viele Standards, Protokolle, Frameworks und komplette IoT-Plattformen. Es haben sich jedoch noch keine übergreifenden Standards etabliert, beziehungsweise kein Standard konnte sich auf dem gesamten Markt durchsetzen. Im Bereich der drahtlosen Übertragungsprotokolle kämpfen Z-Wave und ZigBee und in naher Zukunft auch 6LoP-WAN im die Vorherrschaft. Z-Wave und ZigBee haben jeweils rund 300 Hersteller auf ihrer Seite, welche entsprechende Produkte herstellen und vertreiben. Die Angebotspalette an Produkten welche einer dieser Standards verwendet ist recht gross und beinhaltet unterschiedlichste Geräte.

Für die Endkonsumenten resultiert das Ganze in einem Wirrwar an verschiedenen Produkten und (In-)Kompatibilitäten. Der Konsument muss sich für eine Seite entscheiden, da die verschiedenen Standards und Produkte nicht miteinander kompatibel sind.

Das IoT hat ein enormes Potenziell, welches jedoch nur voll ausgeschöpft werden kann, wenn alle Geräte miteinander kompatibel sind.

5.2 Reflexion

Nach dem die anfängliche Euphorie eine Arbeit über ein innovatives Thema wie das Internet der Dinge zu schreiben verflogen war, habe ich rasch gemerkt, dass das Ganze nicht so einfach werden wird. Durch die Vielschichtigkeit und verschiedenen Ausprägungen der einzelnen Elemente waren viele Recherchen notwendig. Auch im Bereich der Standards, Protokolle und Frameworks benötigte ich viel Aufwand für die Recherchen um ein möglichst umfassendes Bild zu erhalten. Da das Thema IoT an Sich nicht neu ist, aber doch erst in den letzten Jahren richtig aufgekommen ist, fanden sich sehr unterschiedliche Ansichten, Meinungen und Fakten zum gesamten Themenkomplex.

Im Endeffekt habe ich viele neue und interessante Sachen gelernt und konnte viel von den getätigten Recherchen profitieren. ich denke, dass sich das gesammelte Wissen sicherlich bei einer späteren Gelegenheit als nützlich erweisen wird.

Ursprünglich hatte ich geplannt einen Praxistest, beziehungsweise eine Demo eines bestimmten Frameworks (iotivity) zu machen. Dies ist jedoch an technischen Schwierigkeiten und an der fehlenden Zeit zur Umsetzung gescheitert. Ich werde mich sicherlich auch in Zukunft ab und zu mit dem IoT beschäftigen um kleinere Privatprojekte zu verwirklichen.

Quellenverzeichnis

[Eth] *ANT*. URL: http://www.thisisant.com/.

[Ewia] *ANT*. EN. URL: https://en.wikipedia.org/wiki/ANT_(network).

[Cor14] Corporation, Intel: *Developing Solutions for the Internet of Things*. Intel Corporation. 2014. URL: http://www.intel.com/content/www/us/en/internet-of-things/white-papers/developing-solutions-for-iot.html (siehe S. 12).

[daC13] daCosta, Francis: *Rethinking the Internet of Things: A Scalable Approach to Connecting Everything*. ISBN: 978-1-4302-5740-0. Apress Media LLC, 2013.

[Emi] *Designing the Internet of Things*. EN. Micrium. URL: http://micrium.com/iot/overview/.

[IEC14] IEC: *Internet of Things: Wireless Sensor Networks*. EN. International Electrotechnical Commision (IEC). 2014. URL: http://www.iec.ch/whitepaper/pdf/iecWP-internetofthings-LR-en.pdf.

[Inn14] Innovations, Real-Time: *Unterstanding the Internet of Things Protocols*. EN. Real-Time Innovations (RTI). 2014. URL: http://de.slideshare.net/RealTimeInnovations/io-34485340.

[Epo] *Internet of Things Software*. EN. Postscapes. URL: http://postscapes.com/internet-of-things-software-guide.

[JIM15] JIMBO: *Exploring the Protocols of IoT*. EN. 2015. URL: https://www.sparkfun.com/news/1705.

[Mor13] Morish, Jim: *The Emergence of M2M/IoT Application Platforms*. EN. Machina Research. 2013. URL: https://machinaresearch.com/static/media/uploads/Machina%20Research%20White%20Paper%20-%20M2M_IoT%20Application%20Platforms.pdf (siehe S. 11).

[Pat12] Patel, Pankesh: *Enabling High-Level Application Development in Internet of Things*. EN. hal-00732094. 2012. URL: https://hal.inria.fr/hal-00732094/file/main.pdf.

[Sch13] Schneider, Stan: *Understanding The Protocols Behind The Internet Of Things*. English. 2013. URL: http://electronicdesign.com/embedded/understanding-protocols-behind-internet-things.

[Soc04] Society, IEEE Computer: *Software Engineering Body of Knowledge*. 2004 (siehe S. 17).

[Ste12] Stern, Olaf: *Reglement: Seminararbeit*. Deutsch. ZHAW. 2012. URL: https://ebs.zhaw.ch/files/documents/informatik/Reglemente/Bachelor/Seminararbeit/a_Reglement-Seminar_Studiengang-Informatik_V2.1.docx (siehe S. 1).

[Sub08] Subramaniam, Venkat: *Creating a DSL in Java*. EN. JavaWorld. 2008. URL: http://www.javaworld.com/article/2077865/core-java/core-java-creating-dsls-in-java-part-1-what-is-a-domain-specific-language.html.

[tbd] TBD: *tbd*. URL: http://www.networkworld.com/article/2456421/internet-of-things/a-guide-to-the-confusing-internet-of-things-standards-world.html.

[Eem] *The Digital Universe of Opportunities*. EN. Research by IDC. EMC2. 2014 (siehe S. 3).

[Yu15] Yu, James: *Connecting Hardware with the Cloud: Parse for IoT*. EN. 2015. URL: http://blog.parse.com/learn/connecting-hardware-with-the-cloud-parse-for-iot/.

[Ewib] *Z-Wave*. EN. URL: https://en.wikipedia.org/wiki/Z-Wave.

[Edi] *ZigBee*. EN. Digi. URL: http://www.digi.com/technology/rf-articles/wireless-zigbee.

[Ezi] *ZigBee*. EN. URL: http://www.zigbee.org/.

[Ezw] *ZWave*. EN. URL: http://www.z-wave.com/.

Abbildungsverzeichnis