Domain Specific Software Engineering: Internet of Things

Seminar Domain Specific Software Engineering

Version 0.1

Zürcher Hochschule für Angewandte Wissenschaften

Daniel Brun

xx. 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) muss 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 eben.

1.3 Abgrenzung

Abgrenzung

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“
- Case Study „iotivity“

Struktur abschliessend verifizieren

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 nachher die in der Domäne eingesetzten Standards, Protokolle und Frameworks beschrieben. Im Kapitel 5 wird anschliessend eine kleine Fallstudio, beziehungsweise ein Praxistest mit dem Framework “iotivity“ durchgeführt. Im letzten Kapitel wird ein Fazit gezogen, eine Empfehlung an den abgegeben 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 eröffnen sich immer wieder neue Bereiche in der Informatik. Eines dieser neuen Gebiete wird als „Internet of Things“ bezeichnet.

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. Es kann auch als nächste Etappe in der Machine-2-Machine (M2M) Kommunikation gesehen werden.

Der IoT-Markt wird grosses Entwicklungspotenzial zugeschrieben, ein grösses sogar als vor einigen Jahren dem Smartphone Markt.

Prognose / Zahlen –> Aus PDF von LIEBH

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, können automatisch die Sonnenstoren eingefahren, die Fenster geschlossen und die Fensterläden heruntergelassen werden. Ein anders Beispiel wäre, dass wenn man nach Hause kommt automatisch beim Aufschliessen der Türe das Licht angeht und sich der Radio einschaltet. Die Heimautomation kann auch als Einbruchsschutz während längeren Abwesenheiten genutzt wurden. Durch den Einsatz von Machine-Learning kann das System die Gewohnheiten der Bewohner erlernen und das Verhalten bei Abwesenheit zu simulieren. Kommt es dennoch zu einem Einbruch oder einem Einbruchsversuch kann das System selbstständig reagieren und zum Beispiel die Fenster und Türen schliessen, die Fensterläden

herunterlassen, die Polizei und die Bewohner alarmieren und allenfalls Videos und Fotos anfertigen.

Im Industriebereich kann das IoT zum Beispiel zur Überwachung, Kontrolle und Steuerung von Produktionsanalgen oder Fertigungsprozessen eingesetzt werden.

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 Manipulationan abgesichert und die transportierten Daten vor fremden Augen geschützt werden. Kann ein Heimautomationssystem oder ein Überwachungssystem von aussen manipuliert werden, kann dies schwerwiegende folgen haben. Unter Umständen kann der Angreifer die Wohnungstüre öffnen und anschliessend so umprogrammieren, dass sie der Bewohner nicht mehr öffnen kann.

2.2 Domain Specific Software Engineering

Domain Specific Software Engineering, beziehungsweise das domänenspezifische entwickeln von Software, beschreibt den Entwicklungsprozess in einem bestimmten Kontext, beziehungsweise Problembereich. Der Entwicklungsprozess für medizinische Geräten und Software unterscheidet sich an vielen Stellen vom Entwicklungsprozess einer Unternehmung, welche Web-Seiten designt. Auch unterscheidet sich die Entwicklung von Web-Seiten von der Entwicklung von Mobile-Apps oder Fat-Client-Anwendungen.

2.3 Domain Specific Languages

Eine Domain Specific Language (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. Auch sollte die Semantik und Syntax so gewählt werden, das diese im Kontext Sinn ergibt und einfach lesbar und verständlich ist.

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 Umsetzung 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 Eigen- und Besonderheiten im Bereich Software Engineering aufgezeigt.

3.1 Ziel

Das primäre Ziel des IoT ist die Vernetzung von Dingen mit der realen Welt und mit anderen Dingen. Die Dinge sollen intelligent und autonom mit Party Geräten und Services kommunizieren können um so einen Mehrwert 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 Unternehmensbereiche 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 einige Beispiele für konkrete Anwendungen eines IoT

- Smart City
- Medizin und Altenpflege
- Optimierung Energienutzung und Energieerzeugung
- Verteidigung und Militär
- Kommunikationstechnologie
- Industrie (Produktion & Fertigung)

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 Architektur & Aufbau

3.3.1 Das „Thing“

Die Definition eines „Thing“, beziehungsweise eines „Dinges“, ist nicht ganz einfach. Grundsätzlich handelt es sich um einen physischen Gegenstand mit einem „Embedded System“ (auch Embedded Device genannt) welches gewisse Funktionalitäten anbietet. 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 und Arbeitsspeicher) und ist in der Regel auf Energieeffizienz ausgelegt. Das „Embedded Device“ verfügt in der Regel über einen beliebigen Kommunikationsstack mit 1 bis n Kommunikationsprotokollen welcher entweder direkt oder über ein Gateway mit einem Netzwerk kommuniziert. Die gesammelten Daten werden entweder auf dem Gerät oder auf einem Gateway gefiltert.

Diese „Dinge“ und „Smart Objects“ sind keine Erfindung des IoT. Es gibt sie schon eine längere Zeit. Mit dem IoT haben diese jedoch gelernt miteinander oder mit anderen Systemen (autonom) zu interagieren. Solche „Embedded Systems“ sind bereits heute überall gegenwärtig. Sei es in Autos, Wearables, Kühlschränken, Kaffeemschinen oder Smartphones.

Nachfolgend werden die verschiedenen Phasen aufgelistet, welche die Entwicklung dieser „Dinge“ bis heute durchlaufen haben.

Grafik S. 4, DataAwareArchitecutre

- Polling (Reaktive Auslieferung) der Informationen auf dem Gerät
- Proaktive- (Push-) Auslieferung der Informationen vom Gerät.
- Fernsteuerung der Geräte
- Fernwartung der Geräte

3.3.2 Gateway / Bridge

Die Aufgabe eines Gateways, ab und zu auch als Bridge bezeichnet, ist es, IoT Geräte mit einem Netzwerk zu verbinden. Oft sind IoT-Geräte nicht in der Lage sich direkt mit einem Netzwerk oder dem Internet zu verbinden. 85 % aller IoT-Endgeräte sind heute nicht in der Lage direkt mit dem Internet zu kommunizieren [Cor14, S. 2]. Gemäss In solchen Situationen wird eine Gateway eingesetzt, welches diese Lücke schliesst. Eine weitere Aufgabe eines Gateways kann es sein, die von den verbundenen Geräten erhaltenen Daten zu filtern und anschliessend weiterzuleiten. Die Analyse der Daten erfolgt dann auf einem Server im Netzwerk oder in der Cloud.

3.3.3 Back-End Systeme

Die Backe-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 (Stichtwort Big Data) und Auswertung der gesammelten Informationen. 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 oder Benutzereingaben den Geräten Befehle übermittelt.

3.3.4 Das „Web of Things“

Das „Web of Things“ ist eine Architekturansatz, welcher vorsieht alle IoT-Geräte in das bestehende Internet / Web einzubinden. Dazu sollen bereits für das Web etablierte Standards wie zum Beispiel 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 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 wurde entweder von Server zu Server oder von Anwender / Bezüger zum 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.

3.5 Effekte & Auswirkungen

Durch den Einsatz eines IoT kann Mehrwert in verschiedensten Bereichen und Ebenen geschaffen werden. Zum Beispiel in Industriebetrieben können Produkte schneller und besser entwickelt und produziert werden, was sich am Ende auf die Kosten auswirken wird. Es werden sich Auswirkungen in allen Bereichen des Lebens bemerkbar machen. Sei es im Privat- oder im Geschäftsumfeld.

3.6 Subdomänen

Das IoT ist eine riesige Domäne mit zig verschiedenen Anwendungsmöglichkeiten. Nachfolgend werden zwei Ausprägungen von solchen Subdomänen kurz beschrieben.

3.6.1 Wireless Sensor Network

image: http://micrium.com/iot/devices/

Wireless Sensor Network (WSN) sind Netzwerke von verteilten Sensoren, welche gewisse physikalische Zustände oder Zustände in ihrer Umgebung und Umwelt überwachen. 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. Ein WSN-Node besitzt nur eine einzelne Funktion und ist 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.

Solche Sensor-Netwerke können vielseitig eingesetzt werden. Verwendung finden diese zum Beispiel in der Industrie oder in Smart Cities.

Als Kommunikationstechnologie kommt entweder Wi-Fi oder eine Low-Power-Lösung zum Einsatz. Der Vorteil von Wi-Fi besteht im hohen Verbreitungsgrad. Jedoch ist der Energieverbrauch für Wi-Fi sehr hoch, was für die WSN-Nodes nicht unbedingt ideal ist. Es gibt inzwischen jedoch Low-Power-Lösungen welche für den Einsatz auf solchen Geräten optimiert sind. Sie sind Energieeffizient, sind für lange Laufzeiten ausgelegt und sind zum Teil schon in der Lage 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 auch über einen anderen Node mit dem Gateway verbunden ist.

Der IEE 802.15.4 Standard wurde zum Beispiel speziell auf den Einsatz in Low-Power-Systemen zugeschnitten. Ein weiterer Standard ist der 6LoWPAN (IPv6 over Low Power Wireless Personal Area Network) welcher auf Kapselung und Kompressionsmechanismen für kürzere Übermittlungszeiten basiert.

3.6.2 Heimautomation

image Heim-automation

Bei der Heimautomation steht die Vernetzung und intelligente Kommunikation der Endgeräte untereinander im Vordergrund. Dabei kommt ein bunter Mix an unterschiedlicher Geräte und Technologien zum Einsatz, wodurch eine grössere Herausforderung für die direkte Kommunikation entsteht. Mit dem Einsatz eines Homeautomation- oder Smart-Gateways können diese Herausforderungen reduziert werden. Diese Gateways sind in der Lage unterschiedlichste Geräte über unterschiedliche Kommunikationskanäle und Protokolle zu verbinden.

3.7 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,

beziehungsweise sparsam, wie möglich umgegangen werden, um eine möglichst lange Laufzeit zu erreichen. IoT Geräte verfügen meistens über eine mobile Energiequelle, wie zum Beispiel eine Batterie, 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 oder 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“ aufgezeigt.

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

Allgemein: Aufteilung nach IOT-Endgerät, Gateway, Back-End, Client, Querschnittsdomänen

Elem: Embedded Systems

4.1 Software Requirements

Im Requriements Engineering der Software Requriements 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 Produkt, als auch auf Ebene Prozess. Nachfolgend werden einige Beispiele möglicher impliziter Anforderungen aufgelistet.

- Kostengünstig
- Energieeffizient
- Betrieb ohne Stromanschluss
- Massiv skalierbar
- Lange Lebensdauer
- Verlässlich
- Vertrauenswürdig
- Möglichkeit für das Einspielen von Updates

Die Anforderungen auf Ebene Produkt haben einen direkten und starken Einfluss auf die Hardware, die verwendet werden kann oder verwendet werden soll. Je nach dem steuern die Anforderungen an das Produkt die Hardware oder es gibt explizite Anforderungen an die Hardware, welche die Anforderungen an die Software beeinflussen und gegebenenfalls einschränken.

Im Bereich IoT ist es wichtig, dass die Anforderungen genau spezifiziert und korrekt umgesetzt werden. Eine spätere Korrektur kann sich unter Umständen 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 sein. Daher müssen auch sämtliche Anforderungen umgesetzt werden, welche zwingend benötigt werden.

Weitere Herausforderungen sind der Einsatz in einem unbekannten Umfeld (Umwelt & Technisch) und die schwierige Feststellbarkeit von Fehlern. Liefert zum Beispiel 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 nun nicht korrekt und ein Backe-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 eine Aktion auslöst und die gelieferten Informationen nicht korrekt sind.

4.2 Software Design

Aufgrund der Ausgangslage dieser Domäne gibt es einige Restriktionen im Bereich Software Design. Aufgrund der Einschränkungen auf Hardware-Seite können unter Umständen keine Hochsprachen oder grössere / komplexere Frameworks eingesetzt werden um die benötigte Software zu implementieren. Dies macht sich im Software Design bemerkbar, wo nach dem Prinzip „As much as needed, as less as possible“ gearbeitet werden sollte. Nicht benötigte Funktionalitäten und Gimmicks verbrauchen auf IoT-Endgeräten wertvolle Ressourcen.

Handelt es sich bei der zu designenden Software um eine Software für ein IoT-Gateway oder ein IoT-Endgerät mit integriertem Gateway sieht es meistens etwas anders aus und der Einsatz einer Hochsprache und von spezialisierten Frameworks ist grundsätzlich sinnvoll. Aber auch hier wird es Hardware-Seitige Einschränkungen geben, welche berücksichtigt werden müssen.

Im Gegensatz zu einem Gateway stellt ein Endgerät in der Regel nur ein kleines Set an Funktionalitäten zur Verfügung. Zum Beispiel die Bereitstellung der Daten eines oder mehreren Sensoren. Der komplexeste Teil der Software stellt die Bereitstellung eines Kommunikationsstacks dar. Hier sollte wenn immer möglich ein geeigneter Standard oder etablierter proprietäres Protokoll genutzt werden. In der Regel stellt ein IoT-Endgerät 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 durch ein Back-End-System sichergestellt.

Um eine gute und einfache Maschinen zu Maschinen Kommunikation zu ermöglichen, sollte ein möglichst hoher Abstraktionsgrad angestrebt werden. Durch den Einsatz von geeigneten Standard-Protokollen und Frameworks kann dies gut realisiert werden. Aktuell gibt es jedoch noch keine, beziehungsweise nur wenige, grossflächig etablierte Protokolle und Standards, welche speziell auf die Domäne IoT zugeschnitten sind.

4.2.1 Betriebssysteme

4.2.2 Programmiersprachen

C, C++, Zum Teil: Java –> OS benötigt, von daher C,C++ und Java Java benötigt nicht zu vernachlässigendd Ressourcen, z.B. Oracle Java ME Embedded: Memory: 130 KB - 700 KB, ROM: 350 KB - 2000 KB, + Netzwerkstack, Kernel, nicht mehr ein embedded system, Java nur auf fähigeren, teurerern Systemen

4.2.3 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.

4.2.4 C / C++

4.2.5 Java / .NET

Für die Implementation kann es durchaus sinnvoll sein eine Hochsprache, wie Java oder .NET C# einzusetzen.

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 werdne 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. IoT bereits begonnen gewisse Standards zu entwickeln.

Bsp AllJoyn vs. iotivity

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 definiert.

Nachfolgend werden die wichtigsten Kommunikationsprotokolle für den Bereich IoT beschrieben.

Grafik: http://electronicdesign.com/embedded/understanding-protocols-behind-internet-things

MQTT

Das Message Queue Telemetry Transport (MQTT) ist ein leichtgewichtiges „Publish / Subscribe“ Protokoll welches für die Fermessung, Remote Monitoring 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 ein einfache Paket-Struktur und basiert auf Transmission Control Protocol over IP (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, kann aber auch über Hypertext Transfer Protocol (HTTP) oder WebSocket's. 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. wird im Web praktisch überall eingesetzt und kann sich auch für den Einsatz im IoT eigenen.

WebSocket

WebSocket ist ein 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 auch HTTP basiert CoAP auch 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.

ZigBee

Z-Wave

6LoWPAN

ANT

Bluetooth

EnOcean

DASH7

ISA100

Wireless HART

Wireless M-Bus

EtherCAT

Modbus

Profinet

HomePlug

HomeGrid

4.2.8 Standards: Middleware

DDS

Das Data Distribution Service (DDS) Data Distribution Service, Geräte welche direkt Geräte-Daten verwenden, Verteilung zu anderen Geräten, Interaktion mit Infrastruktur unterstützt, Daten-zentrierter Middleware-Standard, Wurzeln: High-Performance Verteidigung, Industrie, Embeddedd Applications, effektiv: millionen von nachrichten pro sekunde zu mehreren gleichzeitigen empfängern

Publish, Subscribe Architecture

Unterschied: Daten an Infrastruktur oder an anderes Gerät, Geräte sind schnell, mit vielen Geräten kommunizieren, TCP Point to Point to restriktiv, DDS: Detailierte qualitiy of service conrol, multicast, konfigurierbare verfügbarkeit, Redundanz

Filter und Selektion von Daten, bzw. Bestimmung was wohin geht

Direct Device-to-device bus, relational data model, ähnlich Datenbank,

Z.B.: Militär, Windparks, Asset-Tracking, Fahrzeug Test und Sicherheit Distribute Device Data, fast bus for integrating intelligent machines (D2D)

Wann: Chaos / Desaster wenn für 5min offline?, Performance sehr wichtig (ms / us) / mehr als 100 Anwendungen / mehr als 10'000 Datenwerte, code mehr als 3 Jahre entwickelt?

4.2.9 Weitere Protokolle

Device-Discovery

4.2.10 Frameworks

In diesem Kapitel werden einige Frameworks vorgestellt, welche ....

Frameworks –> mobile / web –> Vereinfachung Entwiclung, Abstraktion Implementations-Details, apache ISIS? http://iot.eclipse.org/java/open-iot-stack-for-java.html

AllSeen IOTIVITY

4.2.11 Thread

Protokolle: Thread, Netzwerk-Protokoll, Fokus: Security, Low Energy, notwendiger Chip, schon in vielen Geräten vorhanden, gestützt auf 6LoWPAN, IPv6 over Low power Wireless Personal Area Network,

4.2.12 AllJoyn

Qualcomm entwickelt, anschliessend: Linux Foundation, AllSeen Alliance (Cisco, Microsoft, LG, HTC, ...) Verbindung, Wartung Geräte in WLAN-Netzwerk, Kontrolle, Benachrichtungs Service,

4.2.13 Komplett-Lösungen

4.3 Software Construction

Unit, Integration Testing, Standards verwenden –> Sicherstellung Wiederverwendbarkeit, Kompatibilität, Update-Möglichkeit einbauen, evtl. sehr langer Einsatz ohne Update, Hardware berücksichtigen, Configuration Language, Toolkis Languages, Programming languages, Optimierung, Tuning, Reuse

Je nach Anwendungsbereich: Hardwarenahe, Optimierung Für die Entwicklung von Anwendungen für IoT-Geräte können grundsätzlich, Firmware für Spezifische Hardware, Adressierung Netzwerk- und Verbindungsprobleme, Security, oft nur eine spezifische Aufgabe, Integration Test wichtig HW, System Test (Interaktion mit Gateway, Netz, andere Geräte), Test update-Prozess, Test in Echtumgebung, keine Oberfläche

4.4 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-Geräte 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 Backend-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 verfügen kann, 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.

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 Extrembedingungen 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.

Ein intensives und umfassendes Testing resultiert in einer hohen Qualität. Da auf den Endgeräten nur ein geringes Set an Funktionalitäten verfügbar sind, gestaltet sich das Testing an dieser Stelle etwas einfacher, was aber durch die umfangreicheren Tests mit externen Einflüssen kompensiert wird.

4.5 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. Muss das Gerät ausgetauscht, die Energiequelle erneuert oder die Software manuell aktualisiert werden, können hohe Maintenance-Kosten entstehen.

4.6 Software Configuration Management

Das Software Configuration Management ist für den Rollout von Updates sehr wichtig, da unter Umständen verschiedene Versionen von Hardware und Software im Einsatz sein können. Stürzt ein gerät

Wichtig für ausrollen von Updates, Kompatibilität evtl. Firmware auf verschiedenen HW-Typen, hand in hand mit Configuration Mgmt, Software Configuration Control: wichtige Aufgabe, Freigabe: SW configration Control board,

Wichtig Prozess, intensive Retests

SW Release Management and Delivery

4.7 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.

4.8 Software Engineering Process

Defect Tracking, small scale projects, fast product turnaroudn,

4.9 Software Engineering Tools and Methods

SW-Construction: Specific für SW, SW-Testin: speziell, evtl. spezifisch programmiert,

Methoden: Entweder Strukturiert, Daten-Orientiert, Objekt-Orientiert oder Spezifisch (z.B. Real-Time)

4.10 Software Quality

Höhere Qualität notwendig, übergeordnetes ziel,

4.11 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.

Wetter, Physische Einflüsse, Durch das Gerät erzeugte Hitze, Lange Lebensdauer

SW Muss: Skalierbar für breite Palette an verschiedenen Gerätekategorien Modular sein –> nur Auswahl (RAM-Footprint) Verbunden (Daten rein/raus) Verlässlich:Zertifizierung für kritische Applikationen

http://ercim-news.ercim.eu/en98/special/internet-of-things-a-challenge-for-software-engineering http://link.springer.com/chapter/10.1007

Layer: Transport-Layer: TCP zum Teil overkill für IOT-Device –> UDP UDP: beser geeignet für relatime-data, TCPS Acknowledgment and retransmission unnützer overhad für solche Anwendungen (Stück sprache nicht rechtzeitig übermittelt -> Retransmission sinnlos),

Service-based Application: composition and orchestration of Services

Evtl. Struktur nach kategorie: Connectivity, Management, Security, API, ...

Real-Time schwierig, trade off datenverlust, oft TCP als Datenbasis -> messung mit Quality of service besser

http://postscapes.com/internet-of-things-software-guide https://www.silabs.com/Supporthttp://www.datam source/35-open-source-tools-for-the-internet-of-things-1.html https://blog.profitbricks.com/top-49-tools-internet-of-things/ http://www.businesscloudnews.com/2015/05/14/samsung-announces-open-internet-of-things-platform/ https://www.rti.com/company/news/iot-connectivity-webinar.html http://www.iotsworldcongress.com/documents/4643185/4c3cc80c-03b0-41be-baf0-6a1a0fa7db07

SWE: https://hal.inria.fr/hal-01064075/document http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumbe http://www.cio.com/article/2843814/developer/how-to-develop-applications-for-the-internet-of-things.html http://www.appdevelopersalliance.org/internet-of-things/

http://link.springer.com/chapter/10.1007

Protokolle / Standards für verschieden Phasen: Collect Data, Access, Process, Collect, Control / distribute

Middleware / Platforms / OS: http://postscapes.com/internet-of-things-software-guide

Weitere-Protokolle: http://postscapes.com/internet-of-things-protocols Tools: http://www.datamation.com/ source/35-open-source-tools-for-the-internet-of-things-1.html

4.12 middleware

Arten: Point-To-Point, Client/Server, Publish/Subscribe, Queuing, Data-Centric

KAPITEL 5

Case-Study „iotivity

-iotivity -parse

5.1 Aufsetzen der Entwicklungsumgebung

5.2 Entwicklung

5.3 Deployment und Betrieb

Quellenverzeichnis

[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. 8).

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

[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. 13).

[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.

Abbildungsverzeichnis

Tabellenverzeichnis

Liste der noch zu erledigenden Punkte