JOHANNES KEPLER
UNIVERSITÄT LINZ | JKU

Eine generische Web-of-Things Plattform für Smart Vending unter besonderer Betrachtung von Multi-Tenancy und Templates

Masterarbeit

zur Erlangung des akademischen Grades

Master of Science (MSc)

im Masterstudium

Wirtschaftsinformatik

Eingereicht von:
Andreas Neuhauser, BSc

Angefertigt am:
Institut für Wirtschaftsinformatik – Data & Knowledge Engineering

Beurteiler:
o. Univ.-Prof. Dipl.-Ing. Dr. techn. Michael Schrefl

Mitbetreuung:
Mag. Dr. Bernd Neumayr

Mitwirkung:
Stefan Penner, BSc

Linz, Juli 2015

Vorwort

Diese Masterarbeit gibt einen Einblick in die Entwicklung einer Web-of-Things-Plattform für komplexe Dinge. Während diese Arbeit insbesondere Multi-Tenancy als auch Templates diskutiert, fokussiert eine begleitende Arbeit, verfasst durch Stefan Penner, die Representational State Transfer (REST)-basierte Servicearchitektur und die Geschäftsregeln der Plattform. Die grundlegenden Abschnitte dieser Arbeit (Kapitel 2 - 5) sowie die konkrete Implementierung wurden von beiden Autoren gemeinsam entwickelt und sind in beiden Arbeiten deckungsgleich, sodass beide Arbeiten voneinander unabhängig gelesen werden können.

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Masterarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe. Die vorliegende Masterarbeit ist mit dem elektronisch übermittelten Textdokument identisch.

Linz, am 2. Juli 2015

Andreas Neuhauser, BSc

Kurzfassung

Die Fortschritte bei der Miniaturisierung von physischen Geräten und Sensoren in den letzten Jahrzehnten führten zu einem Aufschwung des Pervasive Computing. Das *Web-of-Things*, bei dem eine Vielzahl an alltäglichen Gegenständen als intelligente Objekte mit ihrer Umwelt über das Internet interagieren, erlangte durch die Nutzung von weit verbreiteten und interoperablen Standards wie HTTP, XML oder JSON zunehmende Bedeutung. Im Vergleich zum *Internet-of-Things* liegt der Fokus nicht in der Schaffung von Kommunikationsprotokollen, sondern in der Orchestrierung und Nutzung der Daten für Anwendungen. Die Autoren wurden durch die zunehmende Popularität des Verkaufs von Waren und Dienstleistungen mittels Automaten (gen. Vending) dazu veranlasst ein Smart Vending Szenario zu entwickeln, auf Basis dessen analysiert werden soll, ob und wie eine Integration von Automaten in das Web-of-Things erfolgen kann.

Das Ziel dieser Arbeit ist zu untersuchen wie komplexe Dinge, am Beispiel eines Getränkeautomaten, ins Web-of-Things integriert bzw. abgebildet werden können. Zunächst wird von den Autoren, unter Mitwirkung von Marktteilnehmern, ein Smart Vending Szenario erarbeitet und dessen Umsetzung in bestehenden Web-of-Things-Plattformen untersucht. Die durchgeführte Evaluierung zeigte, dass die verfügbaren Web-of-Things-Plattformen intentional nicht für komplexe Anwendungsszenarien entwickelt wurden. Vor allem aufgrund der fehlenden Unterstützung zur Abbildung und Integration komplexer Dinge wird in Folge von den Autoren eine eigene Plattform, namens *WoTCloud*, in Form eines Prototypen vorgeschlagen bzw. entwickelt.

Neben einer allgemeinen Erläuterung der Architektur und Implementierung werden im zweiten Teil der vorliegenden Arbeit spezifische Aspekte der Plattform WoTCloud näher erläutert. Dabei wird geklärt, wie *Multi-Tenancy* auf Datenbankebene umgesetzt wurde, um IT-Ressourcen kosteneffizient einer Vielzahl an Nutzern zur Verfügung zu stellen. Darüber hinaus wird untersucht, inwiefern der Data Layer von WoTCloud mittels *Sharding* bzw. dem Add-on *Elastic Scale* in Microsoft Azure SQL Database skaliert werden kann und so Engpässe vertikaler Skalierung vermieden werden können. Zuletzt wird ein entwickelter Wiederverwendungsmechanismus für WoTCloud auf Basis von *Templates* vorgestellt und im Kontext moderner Softwareentwicklung diskutiert.

Abstract

Recently there has been a paradigm shift taking place in the field of computing towards miniaturized pervasive objects which are becoming more and more popular in our daily lives. Smart things are digitally enhanced physical objects with communication capabilities. *Web-of-Things* emerged from a lack of standardized communication protocols by reusing well-proven and widely accepted protocols like HTTP, XML or JSON in order to connect smart things to the Web. At the same time Web-of-Things platforms have risen as central data hubs for things offering light-weight integration via REST services. Moreover, vending machines are becoming more attractive and are of great interest in the context of Web-of-Things due to obvious reasons.

The objective of the thesis is to investigate how complex things like vending machines could be enabled to interact with the Web-of-Things. Thus, the authors developed a smart vending scenario in order to evaluate state-of-the-art Web-of-Things platforms. Subsequently, market players have been interviewed about current shortcomings and challenges resulting in notional requirements concerning a vending operator's business. After evaluating four platforms with derived criteria the authors found that all of them are not capable of covering the smart vending scenario sufficiently. Especially the feature to represent and integrate complex things was missing. As a consequence, the authors went further and agreed on building a custom Web-of-Things-platform as part of the thesis.

Apart from a profound introduction of the platform called *WoTCloud*, the thesis highlights two important issues. First, *Multi-Tenancy* and its approaches are introduced where multiple customers share the same application and database instance leading to an improved hardware utilization. Due to the intention of deploying WoTCloud globally, the shared table approach has been chosen and implemented. In addition we reveal how *Sharding* can be used to scale out the data layer of WoTCloud in Microsoft Azure SQL Databases by *Elastic Scale* in order to overcome shortcomings of scale up strategies. Lastly, a reusability approach on the basis of *Templates* is implemented and discussed in the context of software engineering. Vending machine operators thus benefit from automating perpetual tasks.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

Abkürzungsverzeichnis

6LowPan	IPv6 over Low power Wireless Personal Area Network
API	Application Programming Interface
CSV	Comma-separated Values
DBMS	Datenbankmanagementsystem
EDIFACT	Electronic Data Interchange For Administration, Commerce and Transport
EPC	Electronic Product Code
ERP	Enterprise Resource Planning
EXI	Efficient XML Interchange
FTP	File Transfer Protocol
GPS	Global Positioning System
GSM	Global System for Mobile Communications
HATEOS	Hypermedia as the Engine of Application State
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure-as-a-Service
IIS	Internet Information Services
IoT	Internet-of-Things
JSON	JavaScript Object Notation
NFC	Near Field Communication
NIST	National Institute of Standards and Technology

PaaS Platform-as-a-Service

REST Representational State Transfer

RFID Radio Frequency Identification

ROA Resource Oriented Architecture

SaaS Software-as-a-Service

SAC Social Access Controller

SLA Service-Level-Agreement

SOAP Simple Object Access Protocol

SQL Structured Query Language

UDDI Universal Description, Discovery and Integration

UMTS Universal Mobile Telecommunications System

URI Uniform Ressource Identifier

VMO Vending Machine Operator

WCF Windows Communication Foundation

WoT Web-of-Things

WSDL Web Service Description Language

XML Extensible Markup Language

Kapitel 1

Einleitung

Dieses Kapitel dient der Hinführung zum Thema und der Motivation der Relevanz des Internet der Dinge. Es wird der Kernaspekt und die Zielsetzung der Arbeit vorgestellt sowie ein Ausblick auf die einzelnen Kapitel und die Struktur der Arbeit gegeben.

1.1 Motivation

Das Internet hat sowohl das berufliche als auch das private Leben vieler Menschen auf diesem Planeten umfassend verändert und wird dies auch noch zukünftig tun. Web 2.0, soziale Netzwerke und der Internetzugriff auf mobilen Geräten, wie Smartphones und Tablets, sind nur einige Schlagworte der letzten Jahre. Die Anzahl der Nachrichten und Informationen, die tagtäglich über die unterschiedlichsten Geräte und Kanäle ausgetauscht werden, nimmt rasant zu. Die Entwicklung des *Internet-of-Things*[1] *(IoT)*, einer der Top 10 strategischen Technologietrends 2015 [22], ist eine logische Konsequenz der laufenden Weiterentwicklung von Informations- und Kommunikationstechnologie, Anforderungen aus der Logistik und den Geschäftsmodellen der heutigen Zeit.

Eine Vielzahl von Objekten wird zukünftig eine Identität im Internet haben. Alltägliche Gegenstände werden zu *intelligenten Objekten*, die eigenständig mit ihrer Umwelt interagieren und somit den Menschen in den unterschiedlichsten Lebensbereichen und Arbeitsumgebungen unterstützen. So plant beispielsweise Samsung[2] bis 2020 alle Haushaltsgeräte (Kühlschränke, Waschmaschinen, etc.) nur mehr als intelligente, vernetzte Produkte anzubieten. Die Anwendungsmöglichkeiten des Internets der Dinge sind vielfältig:

[1] Die Begriffe Internet-of-Things und Internet der Dinge werden im Laufe der Arbeit synonym verwendet.

[2] http://futurezone.at/produkte/samsung-will-nur-mehr-vernetzte-haushaltsgeraete-anbieten/111.778.305

vom Smart Home[3] über die Analyse und Steuerung von Industrieanlagen[4] bis hin zu smarten Warenautomaten[5].

Die technische Vision des Internet der Dinge ist seit einigen Jahren Ausgangspunkt für (Jung-)Unternehmen zur Entwicklung von intelligenten Produkten bzw. ein treibender Faktor für viele Projekte in Forschungseinrichtungen. Bereits 2015 soll es laut Gartner [21] 4.9 Milliarden vernetzte Dinge geben. 2020 sollen es gar 20 Milliarden - 2/3 davon im Konsumentenbereich (B2C) - sein, bei einem geschätzten Marktvolumen von 7.1 Billionen Dollar[6]. Ausgangspunkt des Internet-of-Things war vor allem die Etablierung von Near Field Communication (NFC) und Radio Frequency Identification (RFID) Technologien im Konsumentenmarkt [48], kombiniert mit der ständig steigenden Verfügbarkeit als auch Verbreitung von mobilem Internet (vor allem durch Smartphones und Tablets). Durch diese Entwicklungen und den skalierbaren Austausch von Informationen (z. B. Facebook, Twitter), der durch soziale Medien ermöglicht wird, wurde ein enormer Platz für Innovationen im Bereich der Endbenutzer bzw. der Entwicklung von benutzerzentrierten Produkten geschaffen. Die Annäherung von Menschen und Dingen nimmt stetig zu.

Nichtsdestotrotz gibt es in diesem Bereich noch offene Forschungsthemen [3]. So ist neben der fehlenden Standardisierung vor allem auch Sicherheit ein großes Thema, da das Bedrohungspotenzial von 20 Milliarden Geräten enorm ist. Ein logischer Schritt hin zu einer möglichen Standardisierung der Kommunikation von und mit smarten Dingen ist das Web-of-Things (WoT): Die Nutzung des World Wide Web und aller assoziierten Technologien als Basis für die Kommunikation und den Datenaustausch.

1.2 Problemstellung

Das Web-of-Things besteht aus einer Vielzahl von smarten Dingen, die über Webschnittstellen erreichbar sind. Durch die Nutzung von weit verbreiteten und interoperablen Standards wie dem Hypertext Transfer Protocol (HTTP), Extensible Markup Language (XML) oder JavaScript Object Notation (JSON) erlangte das Web-of-Things schnell große Bedeutung, da der Fokus im Vergleich zum Internet-of-Things nicht in der Schaffung von Kommunikationsprotokollen zur Vernetzung, sondern in der Orchestrierung und Nutzung der Daten für weiterführende Anwendungsgebiete liegt. Neben der Vernetzung einzelner physischer Geräte gibt es auch zahlreiche (wissenschaftliche) Projekte (z. B. [9]), die eine Plattform als zentralen Datenhub

[3] http://www.wired.com/2015/03/internet-anything-open-source-smart-home-control/

[4] http://www.linemetrics.com

[5] http://blogs.sap.com/innovation/innovation/smart-vending-machine-the-future-of-technology-01253903.com

[6] http://www.forbes.com/sites/gilpress/2014/08/22/internet-of-things-by-the-numbers-market-estimates-and-forecasts/

für Dinge entwickelten. Hauptfokus der meisten Plattformen ist eine einfache Integration von Dingen und deren Daten über REST-Schnittstellen (Representational State Transfer), sowie die Erstellung von Mashups und das Teilen von Informationen auf sozialen Plattformen. Meist können auf diesen Plattformen simple Sensoren und teilweise auch Aktuatoren mit einigen Datenfeldern angelegt werden. Diese "einfachen" Sensoren sind aber oft nicht ausreichend für komplexere Anwendungsfälle der realen Welt.

Der zentrale Anwendungsfall dieser Arbeit - Smart Vending - stellt solch ein komplexes Szenario dar. Die Bedeutung von Web-of-Things für Märkte wie das Vending ist zweifellos groß[7]. Branchengrößen wie SAP präsentierten jüngst Prototypen von Automaten, die durchgängig neue, intelligente und vernetzte Konzepte umsetzen[8]. Bestehende WoT-Plattformen sind aber nicht dafür konzipiert, beispielsweise einen Getränkeautomaten mit all seinen Sensoren (z. B. Füllstandsensoren pro Schacht, generelle Sensoren) und Aktuatoren (z. B. Preisänderung, Getränk ausgeben) abzubilden um den Anforderungen von Betreibern solcher Automaten gerecht zu werden.

1.3 Zielsetzung und Abgrenzung

Das Ziel dieser Arbeit ist Abbildung und Integration von komplexen Dingen in einer Web-of-Things-Plattform am Beispiel eines Getränkeautomaten. Spezifisch werden Anforderungen für solch ein Szenario erhoben, eine Auswahl an bestehenden Plattformen hinsichtlich ihrer Eignung zur Abbildung eines solchen Szenarios evaluiert und schließlich eine Architektur für eine eigene Plattform vorgeschlagen und implementiert. In diesem Zusammenhang beschäftigen wir uns mit folgenden Forschungsfragen:

Komplexe Dinge: *Was benötigt eine Web-of-Things Plattform, um verschiedenste Getränkeautomaten abbilden zu können?* Die Autoren adressieren diese Frage in Bezug auf komplexe Dinge (z. B. einen Getränkeautomat), im Vergleich zu "simplen" Dingen (wie beispielsweise einem Lichtschalter oder einem Temperatursensor), und deren Anforderungen. Ziel ist die Erhebung, welche Features eine Plattform benötigt, um komplexe Dinge am Beispiel eines Smart Vending Szenarios verwalten zu können. Basierend darauf soll eine Architektur und eine prototypische Implementierung durchgeführt werden.

Evaluierung von Plattformen: *Unterstützen bestehende WoT-Plattformen das entwickelte Smart Vending Szenario?* Es sollen bestehende Plattformen hinsichtlich ihrer Eignung für ein Smart Vending Szenario evaluiert werden. In diesem Zusammenhang sollen auch weiterführende Szenarien, die

[7] http://webofthings.org/2012/10/15/i-saw-the-future-of-m2m-in-budapest-smart-vending-machines/

[8] http://blogs.sap.com/innovation/innovation/smart-vending-machine-the-future-of-technology-01253903.com

eine Ende-zu-Ende Integration der Geschäftsprozesse von Automatenbetreibern erlauben, basierend auf den Daten der Plattform ermöglicht werden.

Spezifische Aspekte: *Wie können spezifische Aspekte, die zur Integration von komplexen Dingen benötigt werden, umgesetzt werden?* Aufgrund der Tatsache, dass bestehende Plattformen nicht für Smart Vending geeignet sind, wird im Zuge dieser Masterarbeit eine eigene Plattform vorgestellt. Hauptaspekt ist die Berücksichtigung besonderer Aspekte, die für die Integration und Abbildung von komplexen Dingen notwendig ist. Die Generizität dieser Plattform soll insofern gegeben sein, sodass unterschiedlichste Arten und Typen von Getränkeautomaten berücksichtigt werden sollen. Ein besonderer Fokus soll auf die Architektur und die technische Umsetzung gelegt werden. In dieser Arbeit soll inbesondere geklärt werden, wie eine Web-of-Things-Plattform unter Berücksichtigung von *Multi-Tenancy* realisiert werden kann. Darüber hinaus gilt es zu klären, wie durch *Templates* eine Wiederverwendung von (komplexen) Things erreicht werden kann. In der begleitenden Arbeit von Stefan Penner werden die Representational State Transfer-basierte Architektur und Business Rules behandelt.

1.4 Aufbau der Arbeit

Diese vorliegende Masterarbeit ist wie folgt strukturiert:

- *Kapitel 2* beschreibt die grundlegenden Begriffe und Zusammenhänge und führt in die Thematik von Internet- bzw. Web-of-Things ein.
- *Kapitel 3* dient der Beschreibung des Rahmenbeispiels, welches für die Umsetzung der WoT-Plattform relevant ist. Insbesondere werden Unzulänglichkeiten von bestehenden Smart Vending Lösungen diskutiert.
- *Kapitel 4* vergleicht bestehende WoT-Plattformen hinsichtlich ihrer Einsatztauglichkeit für die Abbildung komplexerer Objekte. Basierend auf einem Kriterienkatalog wird eine Auswahl an Plattformen evaluiert und verglichen.
- *Kapitel 5* fokussiert die Anforderungen und die Systemarchitektur der implementierten Web-of-Things-Plattform. Es werden die einzelnen Schichten erklärt und mögliche Interaktionsszenarien vorgestellt.
- *Kapitel 6* beleuchtet die Umsetzung von Multi-Tenancy anhand des Shared Table Ansatzes in WoTCloud. Weiters wird gezeigt wie eine skalierbare Datenschicht in WoTCloud mithilfe von Elastic Scale in Azure SQL Database realisiert werden kann.
- *Kapitel 7* widmet sich der Umsetzung des template-basierten Wiederverwendungsmechanismus in WoTCloud. Neben einer Einleitung wird die Umsetzung in Data und Presentation Layer beschrieben.
- *Kapitel 8* komplettiert diese Arbeit mit einem Resümee und Ausblick.

Teil I

Grundlagen

Kapitel 2

Vom Internet-of-Things zum Web-of-Things

Our vision is to create a "Smart World," that is, an intelligent infrastructure linking objects, information and people through the computer network. This new infrastructure will allow universal coordination of physical resources through remote monitoring and control by humans and machines. Our objective is to create open standards, protocols and languages to facilitate worldwide adoption of this network – forming the basis for a new "Internet of Things" [10].

2.1 Internet-of-Things

2.1.1 Historie

Der Begriff *Internet-of-Things (IoT)* bekam 2003 vermehrte Aufmerksamkeit, als das Auto-ID Center des Massachusetts Institute of Technology (MIT) ihre Vision eines Electronic Product Code (EPC) Netzwerkes zur automatischen Identifikation und Verfolgung der Warenflüsse von Gütern in Supply-Chains vorstellte [48]. Der Begriff Auto-ID subsumiert eine breite Klasse an Identifizierungstechnologien, die in der Industrie zur Automatisierung, zur Reduktion von Fehlern und zur Erhöhung der Effizienz benutzt werden. Diese Technologien umfassen unter anderem Barcodes, Smart Cards, Spracherkennung und biometrische Verfahren. Seit 2003 ist aber vor allem RFID einer der größten Treiber gewesen. David Brock [10] verwendete den Begriff Internet-of-Things jedoch schon 2001 in einem Whitepaper des Auto-ID Centers über ein Namensschema für physische Objekte. In Folge dessen wurde der Begriff in zahlreichen Publikationen, Konferenzbeiträgen und Büchern verwendet. Mattern [38] verweist auch auf das populärwissenschaftliches Buch *„Wenn Dinge denken lernen"* von Neil Gershenfeld, der

darin in sinngemäßer Weise den Begriff verwendet [24]:

> Es kommt mir so vor, als sei das rasante Wachstum des WWW nur der Zündfunke einer viel gewaltigeren Explosion gewesen. Sie wird losbrechen, sobald die Dinge das Internet nutzen.

2.1.2 Vision

Das Internet der Dinge ist ein neuartiges Paradigma, welches im Kontext moderner Telekommunikation über Drahtlosverbindungen einen stetigen Zuwachs verbucht. Die grundsätzliche Idee [3] hinter *Internet-of-Things* ist die Präsenz zahlloser meist ortsunabhängiger Geräte oder Objekte, welche meist als Dinge (Things) bezeichnet werden, wie beispielsweise RFID-Tags, Sensoren, Aktuatoren, Mobiltelefone, etc. Diese Dinge werden über ein eindeutiges Adressschema [3] adressiert und somit in die Lage versetzt, mit sich selbst und anderen Geräten/Menschen in ihrer Umgebung zu interagieren, um definierte Ziele zu erreichen. Mit diesem Konzept verschwimmt die Grenze zwischen virtueller und realer Welt zunehmend, da virtuelle Informationen nahtlos mit physischen Objekten verknüpft werden und daraus neuartige Möglichkeiten bzw. Anwendungsfelder geschaffen werden [48]. Uckelmann et al. führen weiter an, dass der Zugriff auf möglichst umfassende Informationen durch Informations- und Kommunikationstechnologien in Echtzeit in einer neuen Art und Weise erfolge [48]: unabhängig von Ort („anywhere“) und Zeit („anytime“). Dies erfordere aber eine offene, skalierbare, sichere und standardisierte Infrastruktur. Ein Umstand, der derzeit (noch) nicht vollständig existiere.

Laut Santucci verweist der Begriff Internet-of-Things - das *Internet der Dinge* - auf eine Vision der Maschinen der Zukunft [45]:

> *in the nineteenth century, machines learned to do; in the twentieth century, they learned to think; and in the twenty-first century, they are learning to perceive – they actually sense and respond.*

Die zentrale Rolle in dieser Vision nehmen *smarte Dinge* ein, die mit Informations- und Kommunikationstechnologie ausgestattet sind und über das Internet sich sowohl miteinander vernetzen, mit Menschen interagieren, als auch durch Sensoren ihren eigenen Kontext wahrnehmen können. Diese digitale Aufrüstung von alltäglichen Gegenständen eröffne laut Mattern [38] viele neue Möglichkeiten und erlaube sowohl die Optimierung bestehender Geschäftsprozesse als auch die Bereitstellung neuer Dienste, die durch „zeitnahe Interpretation von Daten aus der physischen Welt ökonomischen und gesellschaftlichen Nutzen stiften“ [38]. Durch die Anbindung an das Internet lasse sich zudem der Zustand von Dingen in bisher unerreichter Granularität und Zeit zu „fast verschwindenden Kosten [messen]“ und somit könne man in vielen Aspekten der realen Welt einen Nutzen stiften.

2.1.3 Evolution

Bereits in der Vergangenheit gab es viele Anwendungen von Auto-ID Technologien in Produktion und Logistik um sowohl unternehmensintern als auch unternehmensextern Informationen auszutauschen. Die meisten RFID-Installationen könnten in diesem Sinne als Intranet oder Extranet der Dinge bezeichnet werden. Während mit Intranet der Informationsfluss im Unternehmen gemeint ist, stehen beim Extranet der Dinge [48] traditionelle Kommunikationsmechanismen im Fokus, wie beispielsweise Electronic Data Interchange For Administration, Commerce and Transport (EDIFACT), welches vor allem dazu verwendet wird, um mit einigen wenigen definierten Partnern kommunizieren zu können. Die Autoren beschreiben das Internet der Dinge als logische Evolution des Extranets der Dinge. Basierend auf den bestehenden Ansätzen wurde die Skalierbarkeit durch das Internet und Web 2.0 entscheidend erhöht. Mit dem Internet der Dinge wird ermöglicht, dass global Informationen ausgetauscht und verwendet werden. Durch zunehmend neue Applikationen und eine breitere Akzeptanz wird auch die Pervasivität erhöht. Abbildung 2.1 veranschaulicht die Phasen vom Intranet der Dinge zu einem Internet der Dinge und versucht, auch eine zukünftige Integration von Dingen und Menschen zu verdeutlichen. Laut Uckelmann et al. [48] müssen vor allem aber die Kosten in Relation zum Nutzen für den jeweiligen Geschäftsfall stimmen sowie die Benutzerfreundlichkeit erhöht werden, um breitere Akzeptanz zu schaffen, sodass eine einfache Integration und Kommunikation von Ding und Mensch ermöglicht wird.

2.1.4 Technische Grundlagen

Das Internet der Dinge basiert laut Mattern [38] nicht auf einer einzelnen konkreten Technologie bzw. Funktionalität, sondern vielmehr auf einer Kombination und Ergänzung von teilweise konvergierenden Technikentwicklungen, die in ihrer Gesamtheit neuartig sind. Zu diesen Funktionen gehören [38]:

- *Kommunikation und Kooperation:* Damit ist die Vernetzung der Objekte zum Zweck des Datenaustausches bzw. zur Nutzung von Diensten in lokalen Netzen oder über das Internet gemeint. Voraussetzung dafür sind funkbasierte Technologien (z. B. Global System for Mobile Communications (GSM), Universal Mobile Telecommunications System (UMTS), Wifi, Bluetooth) und spezielle Weiterentwicklungen in diesem Bereich (z. B. Zigbee, IPv6 over Low power Wireless Personal Area Network (6LowPan)), die auf ressourcenarme Geräte/Dinge abzielen.
- *Adressierbarkeit:* Dinge sind nur dann relevant, wenn sie auch gefunden und deren Daten aus der Ferne konsumiert werden können. Folglich ergibt sich ein Bedarf nach einem (einheitlichen) Discovery-, Lookup-

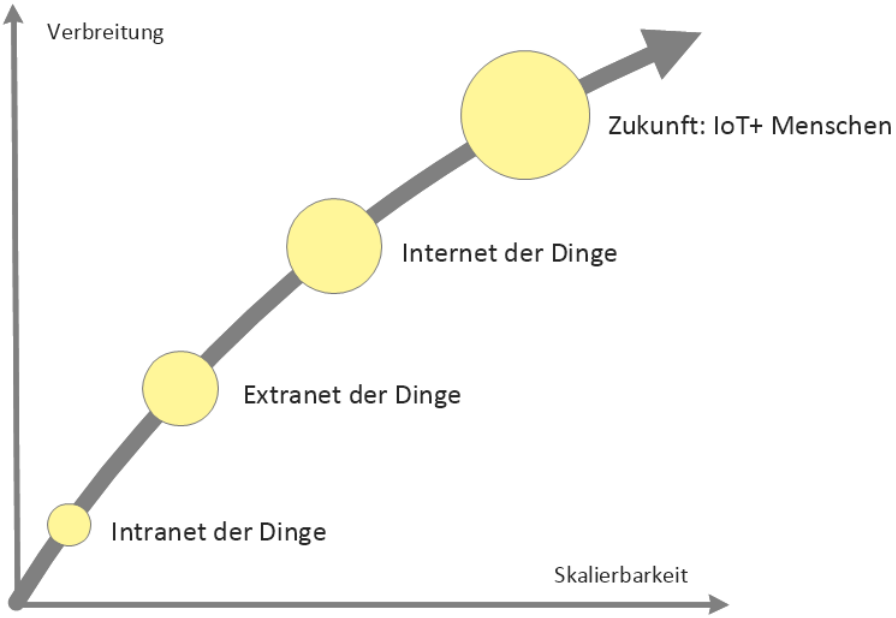


Abbildung 2.1: Die Evolution von Intranet der Dinge zu einer zukünftigen Vision des Internet der Dinge und Menschen (angelehnt an [48]).

oder Namensdienst.

- *Identifikation:* Ein wesentliches Kriterium ist die eindeutige Identifizierbarkeit von Dingen. Alleine aufgrund des historischen Kontexts von Internet-of-Things stellen RFID, NFC oder optische Barcodes Beispiele für relevante Technologien dar. Oft ist ein Vermittler (RFID-Leser), der auch passive Dinge identifiziert, beteiligt. Durch die Identifikation ist eine Verknüpfung von Informationen mit Objekten möglich, auch wenn diese wie im Fall von passiven RFID-Tags nur über den Vermittler mit einem Server verbunden sind.
- *Sensorik:* Objekte sammeln Daten über Ereignisse ihrer Umgebung durch Sensoren. Anschließend werden diese Daten aufgezeichnet oder weiterverarbeitet.
- *Effektorik*: Durch Aktuatoren können Objekte auf ihre Umwelt einwirken und somit sind auch Szenarien denkbar, die eine Beeinflussung und Steuerung von Dingen aus der Ferne (Remote) zulassen.
- *Eingebettete Informationsverarbeitung:* Durch Prozessoren, Mikrocontroller und Speicher werden Objekte zu Smart Things. Diese Bauteile ermöglichen eine Verarbeitung und Speicherung der Sensordaten. Smart Things können also auch ein Gedächtnis besitzen.
- *Lokalisierung:* Durch Global Positioning System (GPS), Mobilfunknetze oder ähnliche Technologien (Ultraschallmessungen, optische Tech-

nologien, ...) können Dinge ihre physische Lokation bestimmen und sind auch für andere auffindbar.

- *Benutzungsschnittstelle:* Smarte Objekte ermöglichen die direkte oder indirekte (beispielsweise über Smartphones) Kommunikation mit Menschen. Wichtig ist dabei, dass die Interaktion in möglichst einfacher und intuitiver Art und Weise erfolgt. Denkbar sind auch innovative Interaktionsparadimgen wie beispielsweise „tangible user interfaces“ oder aber auch Methoden aus den Bereichen Sprach-, Bild- und Gestenerkennung.

2.1.5 Standardisierung und Interoperabilität

Einer der größten Kritikpunkte der vielen Internet-of-Things Entwicklungen der letzten Jahre war und ist die fehlende Standardisierung der Kommunikation, die vor allem durch vergleichsweise geringe vorhandene Ressourcen und Leistungen der Things erschwert wird. Obwohl es einige Bemühungen und Initiativen in diese Richtung gibt, basieren die meisten Projekte auf den unterschiedlichsten Ausprägungen von Hard- und Software, die untereinander inkompatibel sind. Folglich koste laut Guinard & Trifa [27] auch die Entwicklung von simplen Anwendungen aufgrund der heterogenen Systeme noch sehr viel Zeit und benötigt tiefgehende Kenntnisse in spezialisierten Bereichen. Für jedes Deployment müsse man weiters sowohl Basisfunktionen als auch applikationsspezifische Interfaces anpassen, anstatt sich zur Gänze der Applikationslogik widmen zu können.

Aufgrund dessen, dass es noch keine klar spezifizierten, standardisierten, verbreiteten und interoperablen Kommunikationsprotokolle gab, entwickelte sich 2008 ein neues Paradigma, das vollständig auf Webstandards setzt, um Things untereinander und mit ihrer Umwelt zu vernetzen: das *Web-of-Things*.

2.2 Web-of-Things

Die zunehmende Vernetzung von Geräten über das Internet führte dazu, dass als logische Konsequenz - dem Web-of-Things - das World Wide Web und assoziierte Technologien als technische Basis für Smart Things dienen [28]. Das Web ist ein überzeugendes Beispiel für ein skalierbares weltweites Computernetz, in dem heterogene Hardware- und Softwareplattformen ohne Integrationsprobleme zusammenarbeiten. Bereits 2002 wurden physische Objekte über Barcodes mit Webseiten verlinkt, um zusätzliche Informationen und Services abzurufen [36]. Um diesen Ansatz zu erweitern, kann man Smart Things in standardisierte Webservice-Architekturen, durch Nutzung von Standards wie Simple Object Access Protocol (SOAP), Web Service Description Language (WSDL) & Universal Description, Discovery and Inte-

gration (UDDI) einbetten [28]. Dieser Ansatz ist in den meisten Fällen aber sehr schwierig umzusetzen, da die genannten Protokolle für die Anwendung mit simplen, ressourcenarmen Geräten viel zu schwergewichtig und komplex sind [29] und in puncto Bandbreite, CPU und Speicherplatz die Ressourcen der Geräte nicht ausreichen. Aus diesem Grund hat sich vor allem der REST-basierte Architekturstil [19] mit lose gekoppelten Webservices und somit die virtuelle Abbildung von physischen Dingen als ressourcenorientierte Architektur (Resource Oriented Architecture (ROA)) angeboten [27, 28].

2.2.1 Web-of-Things Schichtenmodell

Um die Integration von smarten Dingen mit bestehenden Services im Web zu erleichtern, entwickelte Dominique Guinard in seiner Dissertation [26] eine vierschichtige Applikationsintegrationsarchitektur, die die Entwicklung von Anwendungen mit smarten Dingen vereinfacht (siehe Abbildung 2.2). Diese Architektur soll vor allem die Eintrittsbarriere für Entwickler und technisch versierte Anwender verringern, einen direkten Zugriff über einen Webbrowser oder einen HTTP-Client ermöglichen und einen leichtgewichtigen Zugang zu Daten anbieten.

Obwohl Applikationen auf allen 4 Schichten aufgesetzt werden können, bietet jede höhere Schicht breitere Zugangsmöglichkeiten als ihre untergeordneten. So ist die Verfügbarkeitsschicht als Basisschicht zu verstehen und die Kompositionsschicht logisch gesehen als höchste Schicht, die von Anwendungen benutzt werden kann. In der Folge werden die einzelnen Schichten und ihre Grundgedanken kurz beschrieben.

Device Accessibility (Verfügbarkeit)

Die Verfügbarkeitsschicht (Device Accessibility) dient als Basisschicht und adressiert alle Themen rund um einen konsistenten Zugriff auf smarte, vernetzte Objekte. Guinard bedient sich dazu einer ressourcenorientierten Architektur [26] und behandelt Smart Things somit als „first-class citizen“ analog zu Webseiten. Physische Dinge werden als logische Ressourcen hierarchisch modelliert und sind über eine Uniform Ressource Identifier (URI)-Struktur eindeutig adressierbar [27, 28]. Darüber hinaus dienen diese URIs aber auch der Vernetzung von Dingen. Durch die Benutzung von HTTP als Applikationsprotokoll und nicht nur als Transportprotokoll, wie beispielsweise in WS*-Szenarien, erfolgt die Kommunikation zustandslos und verfügt neben der HTTP-Verben GET, POST, PUT und DELETE auch über ein einheitliches Interface [26, 51]. Die konkrete Repräsentation der Ressourcen wird üblicherweise von Client und Server ausgehandelt (Content Negotation) und erfolgt meist in XML oder JSON.

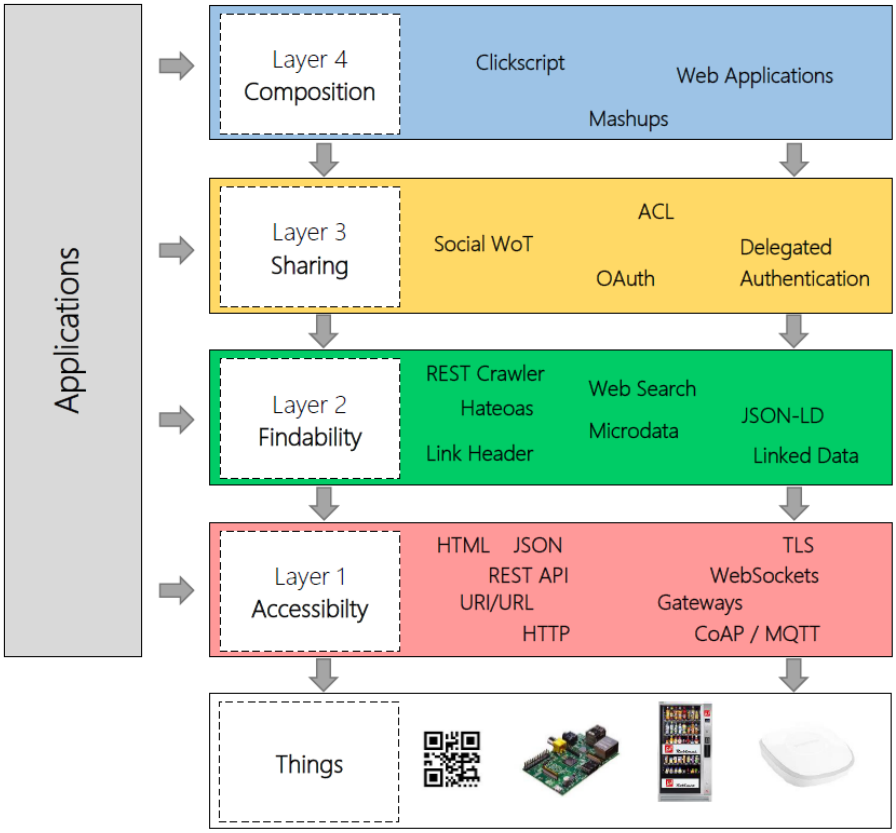


Abbildung 2.2: Die 4 Schichten einer Web of Things Architektur (angelehnt an [26, 30]).

Findability (Auffindbarkeit)

Ausgehend von der Verfügbarkeit einer REST-Schnittstelle von Smart Things gilt es natürlich, deren Services auch zu finden und zu integrieren. Guinard [26] propagiert in diesem Zusammenhang ein Beschreibungsmodell für Smart Things und deren Services basierend auf Metadaten. Nutzvolle Metadaten könnten laut ihm beispielsweise aus der ressourcenorientierten Abbildung der Dinge durch die REST-Architektur extrahiert werden. Anhand einer Implementierung mit Microformats zeigt er auf, dass auch eine Integration in bestehende Suchmaschinen ermöglicht werden kann. Ein großer Nachteil von mobilen Geräten sei jedoch, dass sich der Standort ständig ändern könne. Aus diesem Grund beschreibt Guinard auch eine weborientierte Lookup-Infrastruktur für das Auffinden von Dingen, die auf mehreren lokalen Lookup-Infrastrukturen basiert [26]. Das Ziel der Auffindbarkeit ist also sowohl Menschen als auch Applikationen zu ermöglichen, Services von je-

nen Smart Things über das Web zu konsumieren, die sie in ihrem aktuellen Kontext benötigen.

Sharing (Teilbarkeit)

Basierend auf den ersten beiden Schichten werden die smarten Geräte theoretisch ohne Restriktionen öffentlich über das Web zugänglich. Dies führt dazu, dass auch das Teilen von Informationen (z. B. Sensordaten) erleichtert wird. So könnte beispielsweise der Energieverbrauch durch Sensoren im Haus mit einer Community geteilt werden. Diese Umstände führen jedoch zu gravierenden Eingriffen in die Privatsphären und erfordern deshalb gewisse Restriktionen. Guinard [26] beschreibt deshalb einen selektiven Mechanismus zum Teilen von gewünschten Informationen über einen Social Access Controller (SAC). Dieser ermöglicht auch die Integration von sozialen Netzwerken (z. B. für Werbezwecke) und führt somit zu einem *Social Web-of-Things*. Ein weiteres Anwendungsfeld ist die Aggregation von Daten bzw. Ereignissen zu Feeds. In diesem Zusammenhang kann der SAC auch zur Syndizierung von Daten mehrerer Dinge dienen.

Composition (Komposition)

In der Kompositionsschicht stellt Guinard das Konzept von physischen Mashups vor. Er definiert dazu 3 verschiedene Entwicklungsansätze [26]:

1. *Manuelle Entwicklung von Mashups:* Der erste Ansatz, das manuelle Entwickeln von Mashups, erlaubt es Entwicklern basierend auf Webtechnologien und den Application Programming Interfaces (APIs) & Daten der Smart Things in einfacher Art und Weise Applikationen zu erstellen, die die ersten 3 Ebenen des Web-of-Things-Architekturmodells ausnutzen.
2. *Widget-basierte Entwicklung von Mashups:* Bei der widget-basierten Entwicklung von Mashups erfolgt die Kommunikation mit den smarten Dingen zentral über ein eigenes Software-Framework, welche als Black Box angesehen werden kann, die die Kommunikation abstrahiert und die Daten transparent liefert. Durch Widgets (meist in Form einer Kombination aus Hypertext Markup Language (HTML) und Javascript-Code) können Domänen-Experten nun Anwendungen erstellen. Bei diesem Ansatz sind keine tiefgreifenden Software-Entwicklungskenntnisse notwendig.
3. *Entwicklung durch Endbenutzer über Mashup-Editoren*: Der dritte Ansatz stellt bereits die Endbenutzer in den Mittelpunkt und erlaubt, dass sich Benutzer über Mashup-Editoren eigene kleine Applikationen zusammenstellen. Diese Mashups werden über visuelle Metaphern und Regeln erstellt und nutzen Webseiten bzw. Smart Things als Datenquellen.

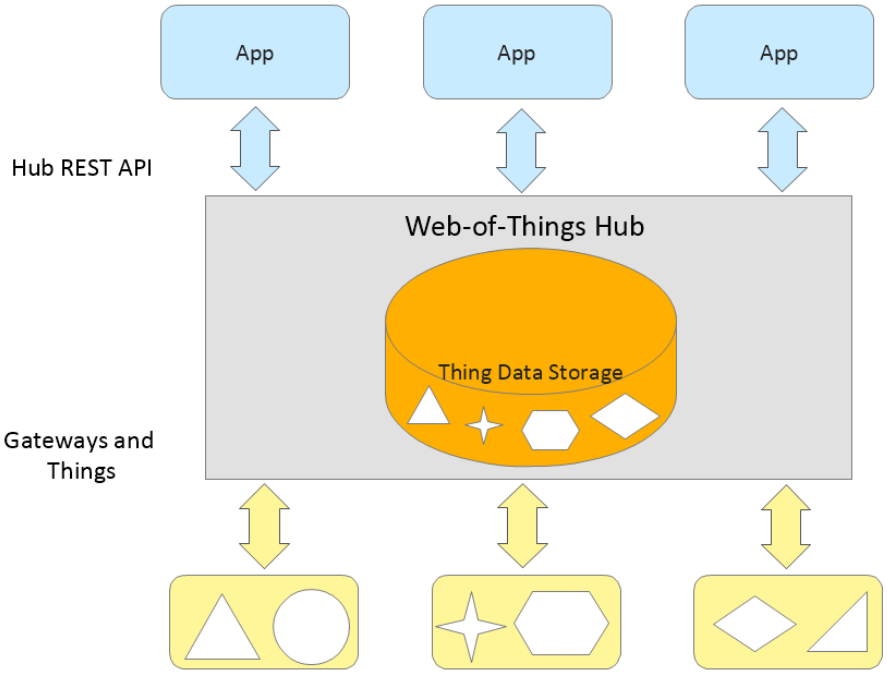


Abbildung 2.3: Konsistente Schnittstellen durch Web-of-Things Hubs (angelehnt an [7]).

2.2.2 Web-of-Things-Plattformen

Während Guinard in seinem Web-of-Things-Architekturmodell (siehe 2.2.1) prinzipiell davon ausgeht, dass jedes Thing einen eigenen Webserver hostet und via REST-API als Datenzugriffspunkt dient, gibt es auch noch andere mögliche Umsetzungsvarianten. Auch die Tatsache, dass Smart Gateways als Mediatoren zwischen Thing und dem Konsumenten dienen, ist in gewissen Anwendungsfällen nicht ausreichend. Ein Kritikpunkt dieser Architektur ist, dass sämtliche Zugriffe direkt auf das ressourcenarme Gerät erfolgen und auch gewisse Security- und Privacy-Mechanismen auf jedem Gerät berücksichtigt werden müssten. Eine weitere Schwachstelle ist der Umstand, dass, obwohl sämtliche Kommunikation über Webstandards erfolgt, es noch keine standardisierte Vorgehensweise gibt, wie man physische Objekte konkret in das Web integriert [7]. Damit ist gemeint, dass Applikationen, die auf den Daten vieler Dinge basieren, auch eine konsistente und interoperable Schnittstelle auf allen Things erfordern würde.

Diese Überlegungen führen zu einem anderen Architekturmodell: Anstatt Smart Things direkt anzusprechen, greift man auf das Konzept eines „Hubs“ zurück, der in Form einer zentralen Plattform die Daten vieler Dinge bündelt [7, 8]. In diesem Zusammenhang ist es auch möglich, dass Smart

Things Daten (z. B. bei einem Ereigniseintritt oder aber auch periodisch) an die Plattform liefern, anstatt über Polling oder Publish-/Subscribe Ansätze abgefragt zu werden. Diese zentralen Plattformen dienen als Basis für spätere Anwendungen und sind im Sinne des Web-of-Things selbst REST-basierte Webanwendungen [8, 9]. Ein Vorteil dieses Ansatzes ist die Reduzierung der Zugriffe auf Smart Things, die nur zentral durch die Plattform oder im umgekehrten Sinn vom Thing zur Plattform erfolgt. Weiters bietet die Plattform eine einheitliche Schnittstelle für jegliche Anwendungen, die auf den Daten vieler Things basiert. Darüber hinaus bieten diese Plattformen vielfältigere Möglichkeiten hinsichtlich Datenpersistierung, zeitbasierten Abfragen und Integration zwischen Anwendungen und Things.

2.3 Zusammenfassung

Die zunehmende Bedeutung der Integration von digitalen Artefakten mit der physischen Welt hat dazu geführt, dass Menschen und Applikationen immer öfter mit „smarten“ Dingen interagieren. Als Folge dieser Entwicklung wird im Rahmen des *Internet der Dinge* nach Möglichkeiten gesucht, um diese smarten Dinge miteinander zu vernetzen. Das Web-of-Things nutzt das World Wide Web und den REST-Architekturstil um eine Integration und Kommunikation von smarten Dinge in standardisierter Art und Weise zu ermöglichen. Neben der Einführung und Beschreibung von Internet-of-Things und Web-of-Things wurde in diesem Kapitel auch eine vierschichtige Applikationsintegrationsinfrastruktur beschrieben. Darüber hinaus wurde der Unterschied zwischen der Betrachtung von Smart Things als drahtlose Sensorknoten und der Integration dieser in Web-of-Things-Plattformen, die als zentraler Hub dienen, vorgestellt.

Kapitel 3

Smart Vending

Die physische Beschaffenheit vieler Dinge ist oft sehr komplex. Folglich sollten Web-of-Things-Plattformen auch in der Lage sein, solche komplexen Dinge abbilden zu können. Da der Verkauf von Waren und Dienstleistungen mittels Automaten immer populärer wird, wird im Rahmen dieser Arbeit beispielhaft die Integration von Verkaufsautomaten ins Web-of-Things diskutiert.

Der in Abbildung 3.1 gezeigte Getränkeautomat ist dabei ein ideales Beispiel für ein komplexes Thing. Aus der Produktbeschreibung [47] geht hervor, dass der Automat aus 6 Ebenen mit je 8 Verkaufsschächten besteht, also insgesamt 48 verschiedene Produkte anbieten kann.

Um ein besseres Verständnis vom Vending-Geschäft aufbauen zu können, haben die Autoren der vorliegenden Arbeit Gespräche mit Automatenbetreibern, sowie mit einem Hersteller von Telemetrielösungen gesucht. Dabei konnten Bedürfnisse bzw. aktuelle Problemfelder identifiziert werden.

3.1 Bedeutung

Der Bundesverband der deutschen Automatenwirtschaft definiert Vending als „Verkauf von Waren und Dienstleistungen durch Automaten“ [11]. Die Automaten werden von sogennanten „Operatoren“ aufgestellt und von diesen betrieben, befüllt, gereinigt und gewartet. Darüber hinaus existiert der Begriff des „Public Vending“, welcher den Verkauf von Waren und Dienstleistungen an öffentlichen Plätzen wie Schulen, Universitäten, Museen, Banken, Baumärkten, usw. umfasst [11].

Um die Bedeutung der Vending-Industrie zu unterstreichen, führt die European Vending Association an, dass 295 Millionen Kunden mindestens einmal pro Woche einen Verkaufsautomaten verwenden [18]. Insgesamt erzielt diese Branche einen Umsatz von 11,3 Milliarden € pro Jahr mit rund 3,77 Millionen Verkaufsautomaten in Europa [18].

Abbildung 3.1: Aktueller Kaltgetränkeautomat der Serie Robimat von Sielaff [47].

3.2 Befragung von Marktteilnehmern

Die nachfolgend angeführten Unternehmen sowie Ansprechpartner erklärten sich bereit, den Autoren Einblick in die Vending-Branche, sowie in aktuelle Probleme und Herausforderungen zu gewähren.

3.2.1 Rudolf Wagner KG

Weinstraße 31
A-4664 Oberweis
Ansprechpartner: GF Ruldolf Wagner

Die von den Autoren befragte Rudolf Wagner KG ist mit etwa 4000 eingesetzten Automaten im Heiß- sowie im Kaltgetränkebereich eine der größten Automatenbetreiber Österreichs. Darüber hinaus ist das Unternehmen auch als selbstständiger Kantinen- und Buffetbetreiber tätig. Die Firma Rudolf Wagner KG bietet seinen Kunden drei verschiedene Aufstellvarianten für Automaten an, die nachfolgend erläutert werden und stellvertretend als Ge-

schäftsmodelle für das Vending Geschäft stehen.

- *Operating:* Im Falle von Operating wird der Automat leihweise von der Firma Wagner zur Verfügung gestellt und auch ordnungsgemäß gewartet und regelmäßig befüllt. Die Automaten bleiben dabei im Eigentum des Betreibers. Die Abrechnung wird ebenfalls vom Automatenbetreiber übernommen. Die etwaige Meldung von Störungen obliegt dem Kunden.
- *Miete:* Beim Mietmodell wird gegen eine monatliche Miet- und Servicekostenpauschale der gewünschte Automatentyp zur Verfügung gestellt und vom Betreiber gewartet. Der Unterschied zum Operating besteht darin, dass die Betreuung, Reinigung und Abrechnung durch den Kunden zu erfolgen hat. Die Füllprodukte dürfen dabei meistens nur beim Automatenbetreiber bezogen werden.
- *Kauf:* Wird der Kauf eines Automaten bevorzugt, so hat der Kunde sämtliche Betreuungs-, Reinigungs-, Befüllungs-, und Abrechnungstätigkeiten selbst zu erledigen. Bei Kauf des Automaten besteht keine Bindung bzgl. der Füllprodukte vom Operator.

Herr Wagner sieht im Vending noch großes Potenzial, aber auch Herausforderungen, die es zu lösen gilt. Die meisten Kunden der Rudolf Wagner KG sind Firmenkunden, die ihren Mitarbeitern Heiß- und Kaltgetränkeautomaten zur Verfügung stellen wollen. Laut Wagner gibt es viele proprietäre Telemetrielösungen, die aber ein unzureichendes Kosten-Nutzen-Verhältnis aufweisen und sich nicht in bestehende Warenwirtschaftssysteme bzw. vorhandene Enterprise Resource Planning (ERP)-Lösungen integrieren lassen. Weiters liefern die Telemetrielösungen noch keine Echtzeitinformationen. Public Vending ist für Wagner nicht relevant bzw. noch sehr unterentwickelt.
Darüber hinaus nennt Wagner folgende Herausforderungen:

- *Einbruch und Vandalismus*
- *Miteinbeziehung des Endkonsumenten:* Durch neue smarte Telemetrie bzw. Vending-Lösungen könnten die Endkonsumenten besser integriert werden, indem diese Füllstände bzw. Standorte von Automaten am Smartphone ermitteln können.
- *Bezahlung mittels NFC:* Derzeit sind die Betriebskosten im Zusammenhang mit NFC noch zu hoch für den flächendeckenden Einsatz. Aufgrund der Tatsache, dass sich die Bezahlung mit Quick nicht durchgesetzt hat (Akzeptanz von rund 2 Prozent), lässt NFC als Hoffnungsträger der bargeldlosen Bezahlung aufleben.

3.2.2 Brunnhofer Verpflegungsautomaten

Bahnhofstraße 9
A-8792 St. Peter-Freienstein

Ansprechpartner: Ing. Manfred Brunnhofer

Das Unternehmen Brunnhofer Verpflegungsautomaten zählt mit 3 Mitarbeitern zur Kategorie der kleinen Automatenbetreiber. Der Automatenbetreiber setzt für Eierautomaten, die sich weit vom Standort entfernt befinden, die Lösung Vendon[1] ein. Vendon ist eine Telemetrielösung bestehend aus Hard- und Software. Die Lösung ist laut Brunnhofer noch sehr teuer (190€/Automat in der Anschaffung und 9€/Monat für die Nutzung der Plattform) und daher noch nicht für den flächendeckenden Einsatz tauglich.

Allerdings sind gerade die kleinen Automatenbetreiber auf Telemetrielösungen angewiesen, da die Automaten weniger oft zyklisch angefahren werden. Große Betreiber sind bis zu zwei mal wöchentlich zur Befüllung und Reinigung beim Automaten vor Ort und daher nicht auf eine solche Lösung angewiesen. Die Automatenhersteller Dallmayr und café+co haben eigene Telemetrielösungen entwickelt, welche jedoch wiederum proprietär sind und nicht kompatibel mit anderen Herstellern sind. Automatenbetreiber setzen jedoch eine Vielzahl an Automaten unterschiedlicher Hersteller ein, was folglich zum Einsatz mehrerer heterogener Systeme und Plattformen führt.

Zuletzt ist es nicht möglich, über die Softwaresysteme schreibend auf die Automaten zuzugreifen. Eine dynamische Preisauszeichnung ist somit nicht möglich. Die Erwartungshaltung von Brunnhofer an das zukünftige Vending ist ebenfalls groß: Zukünftig soll es möglich sein, bezahlte Werbung während der Zubereitung schalten zu können, sowie die Bezahlung per Near Field Communication durchzuführen.

3.2.3 Vendidata Software Entwicklung GmbH

Swarovskistraße 15
A-6130 Schwaz
Ansprechpartner: GF Manfred Steiner

Vendidata entwickelt Verwaltungssoftware für Automaten jeglicher Art. Die entwickelte Lösung zielt dabei darauf ab, den Automaten mit einer speziellen Komponente auszustatten, welche in Verbindung mit einem Handheld via Infrarot drahtlos Information austauscht. Wird ein Automat von einem Mitarbeiter angefahren, können mittels Handheld sämtliche Daten (Anzahl und Art der gefüllten Produkte, eingesammelte Erlöse, usw.) vor Ort ausgelesen werden. Die Daten werden dann mittels Bluetooth auf das Mobiltelefon des Mitarbeiters übertragen und via GSM zu der Backoffice-Software übertragen. Mittels dieser Lösung wird die lückenlose Aufzeichnung des Tagesgeschäftes ermöglicht.

Laut Steiner ist das Hauptproblem im Vending Geschäft die Tatsache,

[1] http://www.vendon.net/

dass die Automatenhersteller die verabschiedeten Standards nicht einhalten. Deshalb beinhaltet der Handheld von Vendidata über 1000 Korrekturdateien, um die Daten verschiedenster Automaten korrekt auslesen und standardisiert übertragen zu können. Natürlich versuchen die Hersteller eigene proprietäre Lösungen anzupreisen und zur Erhöhung der Kundenbindung Telemetrie & Automat aus einer Hand anzubieten. Telemetrie sollte immer herstellerunabhängig und flächendeckend erfolgen. Steiner ist ebenfalls der Meinung, dass der Einsatz von Remote-Telemetrie für bestimmte Automatentypen wie Snack-, Eier-, Heiß- und Kaltegetränkeautomaten viele Vorteile bietet. Ausgenommen davon sind jedoch Kaffeeautomaten, aufgrund der Tatsache, dass zweimal wöchentlich eine Reinigung mit gleichzeitiger Befüllung stattfindet.

3.2.4 Sielaff Austria GmbH

Kaiser-Max-Straße 51
6060 Hall in Tirol
Ansprechpartner: GF Olf Thiele

Sielaff Austria ist ein Tochterunternehmen der deutschen Sielaff GmbH & Co. KG mit Sitz in Herrieden, Deutschland. Laut Geschäftsführer Thiele haben sich in Österreich Remote-Telemetrielösungen aktuell nicht durchgesetzt, da der Preis solcher Lösungen die zu erzielenden Kosteneinsparungen nicht rechtfertigt. Gerade bei Kaffeeautomaten ergibt sich die Situation, dass der Automat alle paar Tage angefahren werden muss, da eine Reinigung und eine Bohnenauffüllung notwendig ist.

Die Tatsache, dass die Automatenhersteller standardisierte Protokolle unterschiedlich implementieren, erschwert die Entwicklung einer einheitlichen Telemetrielösung. Ein Operator betreut jedoch meistens mehrere verschiedene Automaten von unterschiedlichen Herstellern. Der Einsatz von nur einem Remote-Telemetriesystem ist somit fast nicht möglich.

3.3 Ergebnisse der Befragung

Durch die Befragung der Marktteilnehmer konnten die Autoren wertvolle Informationen über die derzeitige Situation im Vending gewinnen. Diese Erkenntnisse werden nachfolgend zusammengefasst festgehalten.

- **Proprietäre Telemetrielösungen**
  Die Tatsache, dass Hersteller nach wie vor verabschiedete Standardprotokolle wie Multi Drop Bus[2] oder EVA-DTS[3] unterschiedlich implementieren, erschwert eine zentrale Plattform zur Verwaltung von

[2] http://www.vending-europe.eu/en/standards_protocols/mdb-icp.html
[3] http://www.vending-europe.eu/en/standards_protocols/eva-dts.html

allen Automaten eines Operators. Die Hersteller drängen daher eher auf eigene proprietäre Plattformen zur umfassenden Verwaltung. Die Betreiber müssen dadurch je nach Umfang der Automatenflotte eine Vielzahl an Systemen einsetzen.

- **Fehlende Echtzeitinformationen**
  Derzeit verfügbare Telemetriesysteme sind nicht in der Lage, Informationen wie bspw. Füllstand oder Fehlercodes in Echtzeit an die Betreiber zu übermitteln. Gerade aber im Vending ist die Verfügbarkeit von Informationen über den aktuellen Zustand erfolgskritisch, denn Stillstand resultiert im fehlenden Umsatz.
- **Unausgewogenes Kosten-Nutzen-Verhältnis**
  Auch die Kosten von verfügbaren Telemetriesystemen rechtfertigen aufgrund fehlender Features die Kosten von Hard- und Software noch nicht.
- **Unzureichende Integrationsmöglichkeiten**
  Weiters beklagen Automatenbetreiber die fehlende Integration solcher Systeme in eingesetzte Warenwirtschaftssysteme, welche Lagerstände kontrollieren bzw. Bestellanforderungen verschicken. Auch eine Einbindung des Endkonsumenten ist derzeit nicht vorhanden.
- **Regelbasierte Aktionen**
  Zuletzt wäre es auch wünschenswert aufgrund gesammelter Statusinformationen Aktionen am Automaten einzuleiten. Beispielsweise wäre es wünschenswert, bei Unterschreiten eines gewissen Füllstands den nächst gelegenen Refiller zu benachrichtigen/beauftragen. Diese regelbasierten Automatismen sind derzeit ebenfalls nicht verfügbar.

3.4 Smart Vending Szenario

Auf Basis der von den Marktteilnehmern kommunizierten Unzulänglichkeiten haben die Autoren ein simples Smart Vending Szenario entworfen, die den Einsatz von Telemetrielösungen attraktiv machen würde. Nachfolgend werden die wichtigsten Anwendungsfälle skizziert:

- **Füllstandsbenachrichtigung**
  Durch die Vernetzung eines Automaten sollen aktuelle Füllstände automatisch an den Betreiber übermittelt werden. Weiters soll im Falle der Erreichung eines kritischen Füllstandes automatisch eine Nachlieferung bei einem Refiller angeordnet werden.
- **Analyse des Konsumverhalten**
  Durch die Nutzung des Automaten entstehen für den Betreiber wertvolle Informationen über das Konsumverhalten. Diese Informationen sollen ausgewertet werden können, um Trends im Konsumverhalten feststellen sowie Strategieanpassungen vornehmen zu können.

- **Suche nach bestimmten Automaten**
  Kunden sollen vorab die Verfügbarkeit von bestimmten Produkten im Automaten ermitteln können. Sollte ein bevorzugtes Produkt nicht mehr erhältlich sein, soll der Kunde zum nächst gelegenen Automaten verwiesen werden.
- **Einfaches Anlegen von Automaten unter Wiederverwendung**
  Zur einfacheren Verwaltung der gesamten Automatenflotte sollen die einzelnen Automatentypen (z. B. Kaltegetränkeautomat Sielaff Robimat 75 mit allen Schächten bzw. Sensoren) als Templates global definiert werden können. Die Neuanlage eines konkreten Automats kann dann auf Basis dieser definierten Vorlagen erfolgen. Die Vermeidung dieser repetitiven Tätigkeiten führt zur effizienteren Anlage von Automaten.
- **Geschäftsregeln**
  Zuletzt soll die Plattform die Definition von einfachen Geschäftsregeln auf Basis von Event-Condition-Action erlauben. Damit könnte der Automatenbetreiber beispielsweise bei Unterschreitung der Boilertemperatur eines Kaffeevollautomaten benachrichtigt werden. Auch das Setzen von automatischen Maßnahmen mittels Aktuatoren soll ermöglicht werden, sodass z. B. Preise auf Basis der Außentemperatur gesetzt werden können.

3.5 Zusammenfassung

Die zunehmende Bedeutung von Vending und Web-of-Things veranlasste die Autoren ein Smart Vending Szenario zu entwickeln, auf Basis dessen analysiert werden soll, ob und wie eine Integration von Automaten in das Web-of-Things erfolgen kann. Hierfür nahmen die Autoren Kontakt mit Betreibern und Herstellern auf, um Bedürfnisse und Probleme bisheriger Lösungen feststellen zu können. Auf Basis der durchgeführten Interviews konnten die Autoren feststellen, dass derzeit noch eine Vielzahl an Problemen existiert, welche den flächendeckenden Einsatz von Remotetelemetrielösungen verhindern. Einerseits bereitet die Heterogenität der Automaten durch die unterschiedliche Implementierung von Protokollen große Probleme und andererseits rechtfertigt der Nutzen nicht die Kosten von derzeit verfügbaren Telemetrielösungen. Daher soll im nachfolgenden Kapitel untersucht werden, ob aktuell verfügbare Web-of-Things Plattformen verwendet werden können, um Automaten ins Web integrieren zu können.

Das in diesem Kapitel entwickelte Szenario wird in Folge benötigt, um eine Vorteilhaftigkeit für Automatenbetreiber sicherstellen zu können. In Kapitel 4 wird nun anhand der entwickelten Anwendungsfälle untersucht, ob aktuelle Web-of-Things Plattformen dazu verwendet werden können, die genannten Anwendungsfälle abdecken zu können.

Kapitel 4

Evaluierung bestehender Plattformen

In diesem Kapitel werden nun aktuelle Web-of-Things Plattformen hinsichtlich ihrer Eignung in Bezug auf das entwickelte Smart Vending Szenario aus Abschnitt 3.4 evaluiert. Um eine Evaluierung durchführen zu können, wurden zuerst Evaluierungskriterien aus dem entworfenen Szenario abgeleitet. Anschließend wurden vier WoT-Plattformen ausgewählt und untersucht. Aufgrund der Vielzahl an existierenden WoT-Plattformen wurden vier Vertreter ausgewählt und evaluiert. Diese Evaluierung wurde bereits im Rahmen einer Seminararbeit am Institut für Data & Knowledge Engineering im Juli 2014 durchgeführt. Jedoch wurde zu dieser Zeit, aufgrund fehlendem Kenntnisstand im Vending-Umfeld, keine Evaluierung anhand komplexer Things, Business Rules, Templates und Erweiterbarkeit durchgeführt. Aus diesem Grund wurde die Evaluierung mit den unten stehenden Kriterien auszugsweise neu durchgeführt.

4.1 Kriterien

In Folge werden die für die Evaluierung benötigten Kriterien erläutert. Diese wurden von dem in Abschnitt 3.4 entwickelten Szenario abgeleitet.

4.1.1 Abbildung komplexer Things

Das Kriterium „Abbildung komplexer Things“ soll feststellen, ob es möglich ist ein komplexes Thing strukturiert und realitätsnah abbilden zu können. Struktuiert meint dabei die Fähigkeit Sensoren und Aktuatoren so zusammenzufassen bzw. zu gruppieren, damit diese der Realität bestmöglich entsprechen. Aktuatoren stellen dabei Zugriffspunkte auf das Thing dar, die gewisse Aktionen bzw. Veränderungen hervorrufen. Als Beispiel kann ein Getränkeautomat angeführt werden, welcher aus verschiedensten Schächten

mit untergeordneten Sensoren besteht.

4.1.2 Business Rules

Unter Business Rules meinen die Autoren eine Unterstützung von regelbasierten Automatismen. Konkret soll untersucht werden, inwiefern Regeln auf Basis von Sensorwerten definiert bzw. ausgeführt werden können.

4.1.3 Erweiterbarkeit

Das Kriterium Erweiterbarkeit soll die Option einer Plattformerweiterung, in welcher Form auch immer, untersuchen. Der Zugriff auf eine angebotene Web API stellt dabei keine Erweiterbarkeit in unserem Sinne dar.

4.1.4 Templates

Um den Operator wesentlich zu unterstützen, wird untersucht ob eine Plattform fähig ist Templates anzulegen bzw. zu verwalten. Eine Plattform erfüllt dieses Kriterium, wenn es die Anlage von Things auf übergeordneter Ebene ermöglicht, die bei einer Neuanlage zu einem späteren Zeitpunkt wiederverwendet werden können. Selbst die Verfügbarkeit eines einfachen Copy & Paste-Mechanismus auf Ebene eines gesamten Thing erfüllt dieses Kriterium.

4.1.5 Unterstützte Datenformate

Bei den unterstützten Formaten für Daten soll geklärt, welches Datenformat für Requests und Responses in HTTP verwendet werden kann. Zur Auswahl stehen dabei XML, JSON und Efficient XML Interchange (EXI). EXI[1] ist dabei eine effizientere und binäre Repräsentation von XML, die weniger Bandbreite beim Transfer in Anspruch nimmt.

4.1.6 Visualisierung

Unter diesem Punkt fallen mögliche Darstellungsformen der erfassten Sensorwerte. Es wird untersucht, ob die Werte anhand von Listen und/oder in Form von Diagrammen dargestellt werden können. Weiters soll untersucht werden, ob Standorte von Things in Kartenform dargestellt werden können, um sofort den Aufenthaltsort eines Things ermitteln zu können.

4.1.7 Web Services

Unter diesem Punkt soll geklärt werden, ob die zu untersuchende Plattform entweder klassische Webservices via SOAP oder schlankere REST-basierte

[1] http://www.w3.org/TR/exi

Webservices unterstützt.

4.2 Untersuchte Plattformen

Insgesamt wurden vier WoT-Plattformen zur Evaluierung herangezogen. Aufgrund der Vielzahl an existierenden WoT-Plattformen wurden zwei Vertreter (Evrythng und WoTKit) aufgrund des Bekanntheitsgrades und zwei Plattformen (ThingSpeak und Paraimpu) willkürlich ausgewählt.

4.2.1 Evrythng

EVRYTHNG[2] ist eine von der Evrythng Ltd. entwickelte Plattform zur Integration jeglicher Konsumprodukte in das Web. Einer der Gründer ist unter anderem der Web-of-Things-Begründer Dominique Guinard. Im Zentrum steht die sogenannte „Smart Product Active Digital Identity“, die jedem Ding oder Gegenstand eine eindeutige Identität zuweist, um damit interagieren zu können. Die cloudbasierte Plattform unterstützt die Bereitstellung von Informationen in Echtzeit sowie Erweiterbarkeit über sogenannte „Applications“. Die Plattform bietet darüber hinaus viele Features sowie eine mächtige REST-Schnittstelle, Client Libraries und Wrapper-Bibliotheken.

4.2.2 WoTKit

WoTKit[3] ist ein Web-of-Things Framework, welches von Michael Blackstock und Rodger Lea am Media and Graphics Interdisciplinare Center an der University of Columbia entwickelt und nun durch ihre Firma, Sense Tecnic System, verbreitet wird. Der Fokus von WoTKit liegt auf der Bereitstellung eines leichtgewichtigen Toolkits für die Entwicklung von IoT-Applikationen mittels Web Technologien und Rapid Application Development. Angelegt als Platform-as-a-Service (PaaS), dient das System als Aggregator für Sensordaten, als Dashboard, zur entfernten Steuerung und um Daten aufzubereiten. Darüber hinaus kann die integrierte RESTful API auch für die Entwicklung von eigenen Applikationen verwendet werden.

4.2.3 ThingSpeak

ThingSpeak[4] versteht sich als eine Open Data Plattform zur einfachen Integration von Dingen ins Internet. Die Plattform wurde von ioBridge Inc. entwickelt. Um mit ThingSpeak kommunizieren zu könnnen, muss zuerst ein sogenannter „Channel“ angelegt werden. Über diesen „Channel“ kann dann

[2] https://evrythng.com
[3] http://wotkit.sensetecnic.com/wotkit
[4] https://thingspeak.com

ein Device, eine App oder ein Thing Daten an ThingSpeak senden. Während es möglich ist mittels Plugins Erweiterungen innerhalb der Plattform im Sinne einer Anwendung zu entwickeln, gibt es auch bereits einige vorinstallierte Anwendungen wie ThingTweet, welche den aktuellen Sensorwert und Status eines Things auf Twitter postet.

4.2.4 Paraimpu

Paraimpu[5] ist ein soziales Tool zur Integration von Dingen und zur Erstellung von personalisierten WoT-Anwendungen. Es erlaubt physische oder virtuelle Dinge mit der Plattform zu verbinden, verwenden und zu teilen. Der Name Paraimpu ist aus dem sardinischen Wort Paralimpu abgeleitet. Ein Paralimpu bezeichnet eine Person, welche als Vermittler eine Hochzeit zwischen zwei Personen arrangiert. Im Kontext von Web-of-Things meint Paraimpu einen Vermittler, um unterschiedliche „Entitäten“ zu verbinden. Im Gegensatz zu den bisher vorgestellten Plattformen steht bei Paraimpu, neben der einfachen Einbindung von Dingen, die Möglichkeit zum Teilen von Dingen auf sozialen Plattformen wie Facebook oder Twitter im Vordergrund.

4.3 Ergebnis

Die Tabelle 4.1 fasst das Ergebnis der Evaluierung in Form eines Vergleichs, anhand der von den Autoren identifizierten Kriterien, zusammen. Nachfolgend werden die Ergebnisse anhand der einzelnen Kriterien detaillierter erläutert.

4.3.1 Abbildung komplexer Things

Als eines der wichtigsten Kriterien für die Evaluierung galt die Abbildung komplexer Things zur realitätsnahen Darstellung von Automaten. Die Autoren der vorliegenden Arbeit mussten feststellen, dass keine der untersuchten Plattformen eine Möglichkeit zur Strukturierung von Sensoren und Aktuatoren anbietet. Trotzdem unterstützen Paraimpu und WoTKit die Definition von Aktuatoren. WoTKit erlaubt mit der Actuator Control API angebundene Things mittels URL Callbacks, LongPolling oder WebSockets anzusteuern. Als soziale Web-of-Things-Plattform versteht Paraimpu darüber hinaus unter Aktuatoren auch Ausgabekanäle zur Weitergabe von Informationen an Twitter, Facebook, usw.

4.3.2 Business Rules

WoTKit und Paraimpu erlauben die Definition von einfachen Geschäftsregeln auf Basis von Event-Condition-Action Regeln. Paraimpu erlaubt die

[5] https://www.paraimpu.com

	Evrythng	*ThingSpeak*	*Paraimpu*	*WoTKit*
Komplexe Things	Nein	Nein	Nein	Nein
Business Rules	Nein	Nein	Ja	Ja
Erweiterbarkeit	Applications	Plugins	Nein	Processor
Templates	Nein	Nein	Nein	Nein
Datenformate	JSON	JSON	JSON, CSV, XML	JSON
Visualisierung	Ja	Charts	Ja	Ja
Web Services	REST	REST	REST	REST

Tabelle 4.1: Ergebnisse der Evaluierung in tabellarischer Form.

Definiton einer Geschäftsregel unter Angabe des Sensors, Aktuators sowie der Verbindung. Innerhalb dieser Verbindung können mittels Filterbedingungen einfache Regeln definiert werden. WoTKit hingegen ist mit einem ereignisgesteuerten Verarbeitungssystem ausgestattet, sodass mittels Pipes und Modules Sensordaten verarbeitet, gefiltert und weitere Aktionen ausgeführt werden können. Evrythng und ThingSpeak erlauben keine regelbasierte Verarbeitung.

4.3.3 Erweiterbarkeit

Geht es um die Erweiterbarkeit, so bieten drei von vier Plattformen die Möglichkeit eigene Bedürfnisse abzubilden. Evrythng kann insofern erweitert werden als Applications darin erstellt werden können. Mit diesem Feature ist es möglich externen nativen oder webbasierten Anwendungen Zugriff auf die Produkte und Things zu ermöglichen. ThingSpeak erlaubt ebenfalls die Entwicklung eigener Anwendungen mit HTML, CSS und JavaScript. Bei WoTKit ist die Erweiterung mithilfe von Apps, Pipes und Connectors möglich. Paraimpu erlaubt momentan keine Erweiterungen.

4.3.4 Templates

Betrachtet man hingegen die Unterstützung von Templates, so muss man feststellen, dass keine der Plattformen auch nur ansatzweise eine Unterstützung bietet. Abgesehen von Templates ist auch eine einfache Wiederverwendung eines Things mit Copy & Paste bei allen Plattformen nicht möglich.

4.3.5 Unterstützte Datenformate

Als Datenformat für die Übertragung mittels HTTP erlauben alle Plattformen die Anwendung von JSON. Vorreiter ist dabei Paraimpu mit einer zusätzlichen Unterstützung für Comma-separated Values (CSV) und XML.

4.3.6 Visualisierung

Positiv hervorzuheben ist die Tatsache, dass beinahe alle Plattformen Sensorwerte sowohl in Listenform als auch Diagrammform darstellen. Einzig und alleine ThingSpeak ermöglicht nur eine Darstellung in Diagrammform.

4.3.7 Web Services

Bei Betrachtung der angebotenen Web Services oder Web APIs lässt sich eine Gemeinsamkeit feststellen: Alle Plattformen setzen auf die leichtgewichtige Implementierung von REST-fähigen HTTP-Services. SOAP Services sind dabei für alle Plattformen keine Alternative. Die Plattform Evrythng bietet zusätzlich Wrapper-Bibliotheken für mehrere Programmiersprachen (JavaScript, Node.js, Java und Android) zur einfacheren Interaktion mit den Services.

4.4 Schlussfolgerung

Die Autoren sind nach Durchführung der Evaluierung der Auffassung, dass das entwickelte Szenario aus Abschnitt 3.4 durch aktuelle WoT-Plattformen nicht ausreichend unterstützt wird. Den Autoren zufolge wurden die bestehenden WoT-Plattformen intentional nicht für komplexe Things entworfen. Aufgrund der festgestellten Unzulänglichkeiten, vor allem der fehlenden Unterstützung zur Abbildung komplexer Things, entschlossen sich die Autoren zur Entwicklung einer eigenen generischen Plattform in Form eines Prototypen, welche das vorgeschlagene Szenario adäquat unterstützen soll.

In Abschnitt 5.2 werden hierfür von den Autoren Ziele und Anforderungen definiert und umgesetzt. Die entwickelte Plattform erhebt dabei nicht den Anspruch auf Vollständigkeit. Die zu entwickelnde Plattform soll lediglich als prototypische Implementierung die mögliche Umsetzung einer solchen Smart Vending Plattform aufzeigen.

4.5 Zusammenfassung

Um eine Aussage über die Unterstützung des Smart Vending Szenarios in aktuellen WoT-Plattformen treffen zu können, definierten die Autoren zunächst Evaluierungskriterien, die für eine akzeptable Smart Vending Lösung erforderlich sind. Anhand dieser Kriterien wurden dann vier Plattformen

(Evrythng, ThingSpeak, Paraimpu und WoTKit) ausgewählt und hinsichtlich Kriterienerfüllung analysiert. Die Ergebnisse der Evaluation ließen die Autoren zu dem Punkt kommen, dass heute verfügbare WoT-Plattformen für das entwickelte Smart Vending Szenario nicht geeignet sind. Deshalb entschieden sich die Autoren eine eigene prototypische Plattform zu entwerfen, die das Smart Vending Szenario ausreichend abbildet. Diese Eigenentwicklung mit den von den Autoren definierten Zielen und Anforderungen ist Gegenstand des nachfolgenden Kapitel 5.

Kapitel 5

Überblick über die Web-of-Things-Plattform WoTCloud

Die im vorigen Kapitel durchgeführte Evaluation bestehender WoT-Plattformen ließ die Autoren zum Schluss kommen, dass diese Plattformen ungeeignet sind für das entwickelte Smart Vending Szenario. Die Autoren entschieden sich für die Entwicklung einer eigenen, prototypischen Plattform, um die genannten Unzulänglichkeiten zu berücksichtigen. Dieses Kapitel ist der Beschreibung der Ziele, Anforderungen sowie der technischen Umsetzung gewidmet.

5.1 Ziele und Nicht-Ziele

Auf Basis des in Kapitel 3 entwickelten Smart Vending Szenarios und der in Kapitel 4 durchgeführten Evaluierung definierten die Autoren übergeordnete Ziele für den zu entwickelnden Prototyp, namens "WoTCloud":

- Prototypische Implementierung einer generischen Web-of-Things Plattform zur Überwachung und Steuerung von Automaten
- Implementierung als Software-as-a-Service Lösung auf Basis von Microsoft Azure
- Anwendung von aktuellen Webtechnologien
- Abbildung des Smart Vending Szenarios aus Abschnitt 3.4 mit den unten genannten Einschränkungen
- Unterstützung der Aspekte: Abbildung von komplexen Things, Templates, Business Rules und Multi-Tenancy

Unter generisch verstehen die Autoren dabei die Möglichkeit unterschiedlichste Automaten abbilden bzw. einbinden zu können. Auf Basis der oben genannten Ziele werden nun in Abschnitt 5.2 Anforderungen abgeleitet bzw.

definiert.

Im Sinne einer Systemabgrenzung werden nachfolgend auch explizit die Nicht-Ziele des zu entwickelnden Prototyps aufgelistet.

- Umsetzung einer voll funktionsfähigen, vollständigen und marktreifen WoT-Plattform
- Einbindung des Endkonsumenten (beispielsweise mittels Smartphone App)
- Setzen von Security-Maßnahmen
- Garantierte Performance
- Physische Anbindung von Automaten an die Plattform inkl. Auseinandersetzung mit Automatenhardware bzw. -protokollen

5.2 Anforderungen

Die nun folgenden Anforderungen wurden von den Autoren zur Umsetzung des Smart Vending Szenarios definiert und hierfür natürlichsprachig dokumentiert. Die Konstruktion einer Anforderung erfolgte dabei auf Basis der von Klaus Pohl [43] vorgestellten Satzschablone.

Eine Satzschablone ist "ein Bauplan für die syntaktische Struktur einer einzelnen Anforderung" [43]. Die Anforderungsdokumentation mittels Satzschablonen ist einfach anzuwenden und liefert eine hohe Qualität in Bezug auf Vollständigkeit. Die einhergehenden Nachteile der Dokumentation mittels natürlicher Sprache können im Falle der vorliegenden Masterarbeit vernachlässigt werden, da die Plattform ohnehin nur von den beiden Autoren entwickelt wurde.

A01 - Registrierung als Tenant Die Plattform muss einem nicht registrierten Operator die Möglichkeit bieten sich am System zu registrieren und dabei folgende Daten aufnehmen:

- Firmenname
- Benutzername
- Passwort

Der angelegte Benutzer ist aus Gründen der Einfachheit der einzige Benutzer eines Automatenbetreibers.

A02 - Login als Tenant Die Plattform muss einem registrierten Operator die Möglichkeit bieten sich im System einzuloggen. Zur Authentifizierung sollen Benutzername und Passwort verwendet werden.

A03 - Thing Übersicht Die Plattform muss einem angemeldeten Operator die Möglichkeit bieten im System existierende Things übersichtlich anzeigen zu können.

A04 - Thing anlegen Die Plattform muss einem angemeldeten Operator die Möglichkeit bieten im System Things anlegen zu können. Bei der Anlage von Dingen müssen folgende Felder eingegeben werden können:

- Name
- Beschreibung
- Längengrad des Standorts
- Breitengrad des Standorts

Zusätzlich muss es bei der Anlage möglich sein, komplexe Things, Sensoren (Anforderung A08) und zugehörige Aktuatoren (Anforderung A15) zu spezifizieren. Unter komplexen Things versteht man jene Things, die selbst wiederum aus Things bestehen können. Die Anlage muss dabei auf Basis eines Templates oder als Neuanlage erfolgen können.

A05 - Thing editieren Die Plattform muss einem angemeldeten Operator die Möglichkeit bieten im System existierende Things editieren zu können. Es können alle Felder, die bei der Anlage spezifiziert wurden, geändert werden.

A06 - Thing löschen Die Plattform muss einem angemeldeten Operator die Möglichkeit bieten zugehörige Things löschen zu können.

A07 - Thing anzeigen Die Plattform muss einem angemeldeten Operator die Möglichkeiten bieten bei Auswahl eines Things die dazugehörigen Detailinformationen abrufen zu können. Hierbei soll die Beschreibung, eine Map mit der geografischen Position, sowie alle zugehörigen Sensoren angezeigt werden.

A08 - Sensor anlegen Die Plattform muss einem angemeldeten Operator die Möglichkeit bieten zu einem (komplexen) Thing einen Sensor hinzuzufügen. Hierfür werden folgende Felder benötigt:

- Name
- Datentyp (bool, int, decimal oder String)

A09 - Sensor löschen Die Plattform muss einem angemeldeten Operator die Möglichkeit bieten einen Sensor löschen zu können.

A10 - Sensorwerte anzeigen Die Plattform muss einem angemeldeten Operator die Möglichkeit bieten die letzten 30 Sensorwerte in Form einer Liste und als Liniendiagramm aufbereiten bzw. darstellen zu können.

A11 - Sensorwert erzeugen Die Plattform muss einem angemeldeten Operator die Möglichkeit bieten einen Sensorwert hinzuzufügen.

A12 - Template anlegen Die Plattform muss einem angemeldeten Operator die Möglichkeit bieten private sowie öffentliche Vorlagen erstellen zu können. Die benötigten Felder bzw. Funktionen ergeben sich analog aus Anforderung A04.

Bedeutung: Templates sollen dazu verwendet werden komplexe Dinge wie Automaten mit unterschiedlichen Schächten und Sensoren (z.B. Sielaff Robimat XL) vordefinieren zu können, um diese anschließend mehrfach wiederverwenden zu können. Wird eine Vorlage als öffentlich definiert, können andere Operatoren diese Vorlage ebenfalls wiederverwenden.

A13 - Templateübersicht Die Plattform muss einem angemeldeten Operator die Möglichkeit bieten private sowie öffentliche Vorlagen übersichtlich anzeigen zu können.

A14 - Template löschen Die Plattform muss einem angemeldeten Operator die Möglichkeit bieten private sowie öffentliche Vorlagen löschen zu können.

Hinweis: Aus Gründen der Einfachheit kann ein Operator seine eigenen (privaten) als auch öffentliche Vorlagen jederzeit löschen.

A15 - Aktuator anlegen Die Plattform muss einem angemeldeten Operator die Möglichkeit bieten zu einem (komplexen) Thing einen Aktuator hinzuzufügen. Hierfür werden folgende Felder benötigt:

- Name
- URI

Bedeutung: Mittels Aktuatoren kann schreibend auf ein Thing zugegriffen werden. Aktuatoren werden unter anderem für die Anforderung A18 benötigt.

A16 - Aktuator ausführen Die Plattform muss einem angemeldeten Operator die Möglichkeit bieten zu einem (komplexen) Thing zugehörigen Aktuator unter Angabe eines Wertes auszuführen. Der spezifizierte Wert wird dabei nicht von der Plattform validiert, da es sich sowohl um elementare Typen (string, int, bool, usw.) aber auch komplexe Datenstrukturen (XML, JSON, usw.) handeln kann.

A17 - Aktuator löschen Die Plattform muss einem angemeldeten Operator die Möglichkeit bieten einen Aktuator löschen zu können.

A18 - Geschäftsregel anlegen Die Plattform muss einem angemeldeten Operator die Möglichkeit bieten zu einem (komplexen) Thing eine Geschäftsregel hinzuzufügen. Hierfür werden folgende Felder benötigt:

- Angabe ob Regel datengetrieben oder eventgetrieben
- Zeitangabe (hh:mm) bei eventgetriebener Regel
- Auswahl eines Sensors
- Operator (kleiner, kleiner gleich, größer, größer gleich, gleich und ungleich)
- Vergleichswert
- Angabe ob Aktion, Aktuator oder E-Mail
- Auswahl eines Aktuators bei Aktion Aktuator
- Übermittlungswert bei Aktion Aktuator
- Empfängeradresse bei Aktion E-Mail
- Textnachricht bei Aktion E-Mail

Nach erfolgreicher Erstellung ist die Regel aktiv und kann nur über eine Löschung deaktiviert werden.

A19 - Geschäftsregel löschen Die Plattform muss einem angemeldeten Operator die Möglichkeit bieten eine Geschäftsregel löschen zu können.

5.3 Konzeptuelles Modell für komplexe Dinge

Einer der Hauptaspekte der Plattform ist die Abbildung von komplexen Dingen. Ein komplexes Ding könnte beispielsweise ein Getränkeautomat (Sielaff Robimat) sein. Dieser Getränkeautomat besteht neben allgemeinen Sensoren (z. B. Außentemperatur, Lokation,...) aus bis zu 48 Schächten. Jeder dieser Schächte kann wiederum als eigenes Ding angesehen werden und besteht aus Sensoren (z. B. Füllstand, Schachttemperatur, Preis,...) und Aktuatoren. Aus dieser Darstellung ergibt sich eine Unterscheidung zwischen komplexen und simplen Dingen, wobei ein komplexes Ding mehrere simple Dinge subsumiert (siehe Abbildung 5.1). Prinzipiell könnte eine beliebige hierarchische Struktur ermöglicht werden, indem ein komplexes Ding eine Vielzahl an abstrakten Dingen als Selbstreferenz zulassen würde. Die Autoren haben aber die Hierarchie auf zwei Stufen begrenzt, da die Entwicklung eines generischen Webclients zur Erfassung und Verwaltung von Dingen ansonsten den Rahmen dieser Arbeit sprengen würde.

Auf dieser Grundlage ergibt sich das in Abbildung 5.1 dargestellte konzeptuelle Modell. Ausgehend von einem Kunden (Tenant), der im Smart

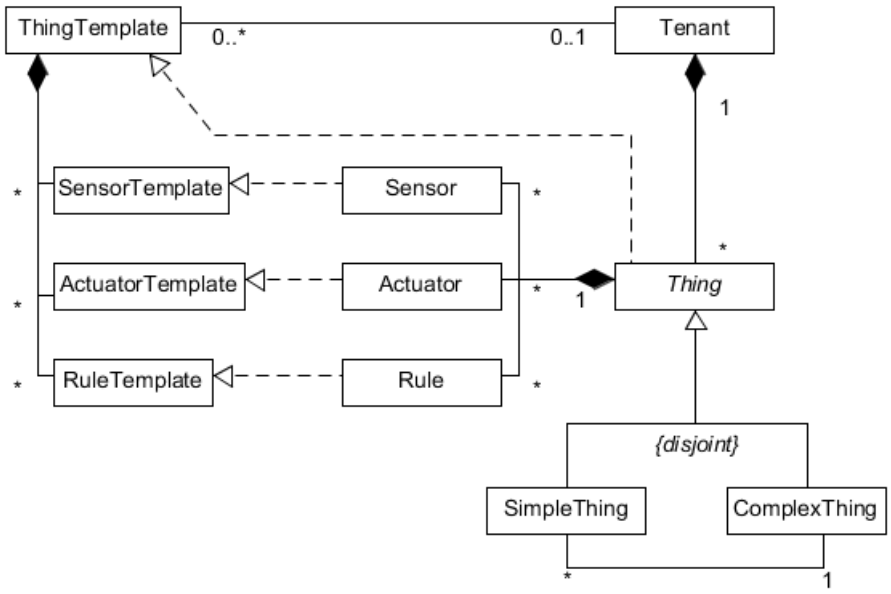


Abbildung 5.1: Konzeptionelles Modell von WoTCloud.

Vending Szenario einen Operator darstellen würde, gibt es eine abstrakte Basisklasse Thing und zwei konkrete, disjunkte Spezialisierungen: *SimpleThing* und *ComplexThing*. Ein komplexes Ding (*ComplexThing*) kann wiederum aus einer Menge von simplen Dinge bestehen. Jedes Ding ist eindeutig genau einem Tenant zugewiesen und kann nur in Kombination mit diesem Tenant existieren. Sensoren, Aktuatoren und Rules sind der abstrakten Basisklasse Thing zugeordnet. Folglich kann sowohl für simple als auch für komplexe Dinge eine Menge an Sensoren, Aktuatoren und Regeln definiert werden, die wiederum eine Existenzabhängigkeit zu den Things aufweisen. Darüber hinaus gibt es auch noch Templates, die entweder von einem spezifischen Tenant erstellt wurden oder öffentlich zugänglich sind und eine Art Schablone für Things darstellen. So können auch für Templates Sensoren, Aktuatoren und Regeln definiert werden.

Der aus dem konzeptuellen Modell abgeleitete logische Entwurf findet sich im Anhang A.

5.4 Spezifische Aspekte

In Folge werden kurz die wesentlichen, besonderen Aspekte der Plattform beschrieben. Diese Konzepte werden im zweiten Teil bzw. der begleitenden Masterarbeit detailliert erläutert.

5.4.1 Business Rules

Geschäftsregeln dienen der Abbildung von Geschäftslogiken eines Unternehmen. Sie erlauben es zeitgerecht auf eintretende Ereignisse in einer vordefinierten Art und Weise zu antworten und somit eine Verbindung von passiven Datenbeständen und der dynamischen Welt zu ermöglichen. Unsere WoTCloud-Plattform erlaubt das Erstellen von einfachen, atomaren Event-Condition-Action(ECA)-Regeln, um Teile der Geschäftslogiken (z. B. die Anforderung eines Vending-Operators über niedrige Füllstandsmengen informiert zu werden) abzubilden. Die Basis für diese reaktiven Regeln bilden die Sensordaten, die an die Plattform geliefert werden. Regeln können sowohl auf Things als auch auf Templates definiert werden. Eine Regel könnte beispielsweise wie folgt aussehen:

1. *Event:* Füllstandsänderung Schacht 1
2. *Condition:* aktueller Füllstand < 2
3. *Action:* Benachrichtige Operator

5.4.2 Multi-Tenancy

Das Konzept Multi-Tenancy (dt. Mehrmandantenfähigkeit) basiert auf dem Grundgedanken mehrere Kunden, Tenants oder Benutzer auf einer einzigen Software-Instanz zu hosten, um Kosteneinsparungen und Skalierungseffeke zu erzielen. Dies führt jedoch unweigerlich zu einer Erhöhung der Komplexität, da einerseits Isolationsmechanismen erforderlich sind, aber andererseits gewisse Ressourcen allen Beteiligten konsistent zur Verfügung gestellt werden müssen (Resource Sharing). Um diesen Anforderungen gerecht zu werden, sind insbesondere auf Datenbankebene besondere Vorkehrungen zu treffen. So besteht die Möglichkeit, dass jeder Tenant eine eigene dezidierte Datenbank erhält, alle Tenants in einer Datenbank abgelegt werden oder Mischformen realisiert werden. In diesem Zusammenhang werden Anforderungen evaluiert und unter Einbezug ökonomischer Überlegungen ein Multi-Tenancy Modell für das Deployment unserer WoT-Plattform vorgeschlagen und umgesetzt.

5.4.3 REST

Representational State Transfer, kurz REST, ist ein von Thomas Roy Fielding [19] beschriebener Architekturstil zur Entwicklung verteilter Anwendungen. REST zeigt Möglichkeiten auf, wie Kommunikationsprotokolle und Datenformate genutzt werden können, um möglichst einfach leichtgewichtige und lose gekoppelte Web Services realisieren zu können [40]. Im Zentrum von REST stehen Ressourcen, die über eine URL adressiert bzw. abgefragt werden. Zum Manipulieren dieser Ressourcen bedient sich REST der standardisierten HTTP-Operationen (PUT, POST und DELETE). Wie schon in

Kapitel 2 erwähnt, können REST-basierte Web Services als technologische Grundlage von Web-of-Things angesehen werden.

5.4.4 Templates

Eine wesentliche Anforderung unserer Plattform ist die einfache Integration und Verwendung von Things. In Zusammenhang mit unserem Smart Vending Szenario wird oft eine Vielzahl an gleichartigen Things angelegt. So möchte beispielsweise ein befragter Operator (siehe Abschnitt 3.2.1) seine 4000 Automaten, die in 4 unterschiedliche Automatentypen untergliedert sind, einfach und schnell anlegen. Durch sogenannte Templates können Referenzmodelle eines bestimmten Automatentyps erstellt und wiederverwendet werden. Ein Template spezifiziert also ein abstraktes Ding mit all seinen Sensoren und Aktuatoren. Weiters sollen auch Regeln bereits auf dieser abstrakten Ebene definiert werden können. Diese Templates ermöglichen es, Dinge basierend auf Vorlagen einfach und schnell anzulegen und stellen somit ein wesentliches Unterscheidungsmerkmal von WoTCloud dar.

5.5 Umsetzung

Nachfolgend wird nun die technische Realisierung von WoTCloud diskutiert bzw. ein Einblick in die drei entwickelten Schichten gegeben.

5.5.1 Systemarchitektur

Die Architektur der entwickelten Web-of-Things-Plattform basiert auf einem Client/Server Interaktionsmodell. Dieses Interaktionsmodell ist insofern gegeben, als das Web - im Sinne der Kernstandards/-protokolle HTTP, HTML und URIs - im Wesentlichen auf einem Client/Server-Modell basiert. Dieses Interaktionsmodell wurde konkret in einer 3-Layer Architektur umgesetzt (siehe Abbildung 5.2). Der Data Layer kümmert sich um den Zugriff und die Persistierung der Daten sowie der Abstraktion der dahinterliegenden konkreten Technologien. Der Service Layer bietet sämtliche Datenrepräsentation über Webservices an, welche vom Client - der Presentation Layer - entsprechend konsumiert und aufbereitet wird.

Die physische Verteilung (Tiers) dieser drei logischen Schichten (Layers) kann unterschiedlich erfolgen. Theoretisch könnten alle drei Layer auf einer physischen Instanz deployed werden (1-Tier). Im konkreten Fall sind jedoch sowohl Data- als auch Service Layer auf Microsoft's Cloud Plattform Azure getrennt deployed und können auch einzeln horizontal und vertikal skaliert werden. Somit ergibt sich ein 3-Tier Architekturszenario.

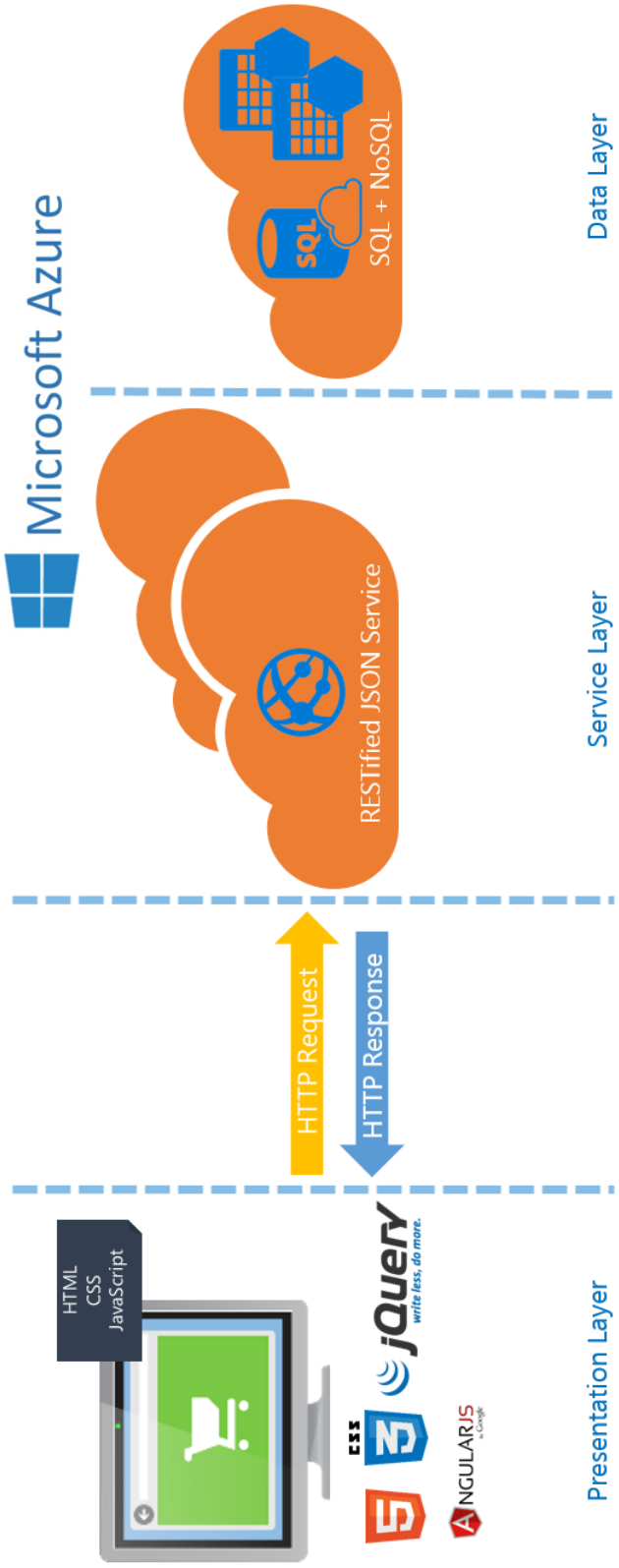


Abbildung 5.2: Systemarchitektur von WoTCloud.

5.5.2 Data Layer

Nachdem die WoT-Plattform von Beginn an vollständig als Software-as-a-Service Lösung konzipiert wurde, musste eine Entscheidung hinsichtlich Datenarchitektur und Speichertechnologien getroffen werden. Hierfür entwickelten die Autoren zuerst das nötige UML-Klassendiagramm zur Abbildung der Entitäten und Assoziationen. Anschließend wurde eine hybride Datenarchitektur auf Basis von Microsoft Azure für Datenzugriff und Datenhaltung entwickelt.

Datenmodell

Das Datenmodell entspricht dem konzeptuellen Modell aus Abschnitt 5.3. Abbildung 5.1 skiziert das entwickelte Datenmodell als Klassendiagramm der UML und stellt die Struktur der verwendeten Entitäten dar.

Hybride Datenarchitektur

Für die Umsetzung des oben genanten Entitätsmodells wurde eine hybride Datenarchitektur mit den von Microsoft Azure gebotenen Diensten realisiert. Konkret wurden die Dienste Azure SQL Database sowie Azure Table Storage für die Realisierung eingesetzt.

Microsoft Azure SQL Database ist ein relationales Datenbankmanagementsystem (RDBMS) in der Cloud und unterstützt klassische relationale ACID-Transaktionen [41]. Im Wesentlichen basiert dieser Service auf der SQL Server Engine. Azure SQL Database eignet sich somit zur Speicherung von schematisch, strukturierten Daten unter Sicherstellung der referenziellen Integrität.

Im Gegensatz dazu ist *Azure Table Storage* ein flexibler, cloudbasierter Key/Value-Speicher zur Speicherung von schemalosen Applikationsdaten [41]. Diese Speichertechnologie verzichtet auf Konzepte wie Joins, Stored Procedures und referenzielle Integrität und ermöglicht die Speicherung von Tupeln mit unterschiedlicher Struktur in einer Tabelle. Dieser Speicheransatz zielt auf die kostengünstige und performante Speicherung von großen Datenmengen ab, wobei die Daten denormalisiert sind [37].

Aufgrund der Tatsache, dass beide Ansätze jeweils beachtliche Vorteile bieten, ließ die Autoren zum Schluss kommen, eine hybride Architektur unter Verwendung beider Speicherdienste zu implementieren. Das in Abbildung 5.1 skizzierte Datenmodell könnte dabei vollständig in *Azure SQL Database* implementiert werden. Betrachtet man jedoch die Vielzahl an erwarteten Write-Transaktionen von konkreten Sensorwerten, welche durch einen angebundenen Automaten versendet werden, würde man hierbei sehr schnell an die Grenzen (max. Datenbankgröße[1]: 500 GB) stoßen. Deshalb

[1] http://azure.microsoft.com/de-de/pricing/details/sql-database

wird für die performante Speicherung und den Zugriff auf Sensorwerte *Azure Table Storage* verwendet, da die Sensorwerte ohnehin kein striktes relationales Schema mit referenzieller Integrität aufweisen. Ein weiterer Grund für den Einsatz von *Azure Table Storage* ist das kostengünstigere Preismodell für große Datenmengen.

5.5.3 Service Layer

Der Service Layer als zentraler Baustein der Plattform wurde als Sammlung von REST-basierten Webservices entwickelt. Hauptaufgabe der Services ist es, Repräsentationen von Ressourcen (z. B. Sensordaten für ein bestimmtes Thing) aufzubereiten und zur Verfügung zu stellen. Der Service Layer dient sowohl als zentraler Datenhub für Sensoren, als API für spätere Anwendungsszenarien (z. B. jegliche Form von Apps (Smartphone, Tablet, etc.), Integrationen in andere Softwarekomponenten, etc.) sowie auch als Basis für den Webclient, der selbst ausschließlich auf diesen Webservices basiert.

Technologisch wurde der Service Layer mit Microsoft's ASP.NET Web API[2] und dem .NET Framework realisiert. Als Entwicklungsumgebung für den Service Layer verwendeten die Autoren Visual Studio[3] von Microsoft. Neben C#, .NET und ASP.NET Web API wurden noch folgende Frameworks/Toolkits in Form von NuGet-Packages eingesetzt: Newtonsoft JSON, .NET Entity Framework (O/R-Mapper) und OWIN[4].

Newtonsoft JSON ist dabei ein JSON-Serialisierer für das Microsoft .NET-Framework. Entity Framework ermöglicht eine objektrelationale Zuordnung von Objekten zu relationalen Daten und kapselt somit wesentlich den Datenzugriff. OWIN umfasst Komponenten zum Entkoppeln der eigentlichen Webapplikation vom zugrundeliegenden Webserver und vereinfacht damit Portierungen.

Webservices

Wie bereits erwähnt, basiert der Server auf einzelnen, voneinander unabhängigen Webservices. Dies impliziert, dass alle Things vollständig im Sinne einer ROA als Ressourcen modelliert und in eine URI-Struktur eingegliedert wurden. So ist beispielsweise der Zugriff via HTTP GET auf ein einzelnes Thing über folgende URI möglich:

http://.../api/*{tenantId}*/things/*{thingId}*

Prinzipiell sind diese Webservices an den von Roy Fielding entworfenen Architekturstil REST angelehnt (Zustandslosigkeit, lose Kopplung, HTTP-Methoden, URIs) [19], setzen aber nicht alle Konzepte (vor allem Hypermedia

[2] http://www.asp.net/web-api/
[3] https://www.visualstudio.com/
[4] Open Web Interface for .NET - http://owin.org/

as the Engine of Application State (HATEOS)) vollständig um. Somit handelt es sich nicht um RESTful-Webservices im engeren Sinn. Darüber hinaus basieren unsere Webservices vollständig auf HTTP, eine Tatsache, die in einem REST-Umfeld nicht zwingend nötig ist, jedoch meist impliziert wird [46]. Die Kommunikation des Clients mit diesen Webservices erfolgt ausschließlich über HTTP-Requests und HTTP-Responses (siehe Abbildung 5.2) und bietet als Datenformat neben XML bevorzugt JSON an.

Als Basistechnologie zur Umsetzung dieser auf REST und HTTP-basierten Webservices dient *ASP.NET Web API*. Dabei handelt es sich um ein sehr leichtgewichtiges Framework zur Entwicklung HTTP-basierter Services, welches vollständig auf dem .NET Framework von Microsoft basiert. Web API bietet einen direkten Zugriff auf die Möglichkeiten von HTTP und abstrahiert, im Gegensatz zur Windows Communication Foundation (WCF), nicht das darunter liegende Protokoll [46]. Die Tatsache, dass ASP.NET Web API auch um einiges leichtgewichtiger und somit auch weniger komplex als WCF ist, lässt es vor allem für jene Projekte, deren Kommunikation sich auf HTTP stützt, bevorzugt zum Einsatz kommen. Laut Steyer & Schwichtenberg [46] ermögliche dieses Framework auch eine Implementierung von RESTful Webservices. Die korrekte Implementierung jedoch liege vollständig in den Händen der Entwickler. So könne sich der Entwickler „gegen REST-ful URLs entscheiden, pro Ressource nur ein Datenformat erlauben oder auf den Einsatz von Hyperlinks im Rahmen der retournierten Daten verzichten“ [46].

5.5.4 Presentation Layer

Parallel zu den vorher genannten Schichten entwickelten die Autoren einen Webclient als Benutzerschnittstelle. Der Webclient zeichnet dabei verantwortlich für die Präsentation der gesamten Daten an der Benutzerschnittstelle.

Als Entwicklungsumgebung für den Presentation Layer verwendeten die Autoren WebStorm[5] von JetBrains. Neben HTML, CSS und JavaScript wurden für das User Interface folgende Frameworks/Toolkits bzw. Technologien eingesetzt: AngularJS, Twitter Bootstrap, jQuery, Telerik Kendo UI, Angular Google Maps, Angular UI Router und einige mehr.

Single Page Anwendung (SPA)

Der Webclient wurde als sogenannte „Single Page“-Anwendung konzipiert, sodass die Applikation im Wesentlichen aus einer einzigen HTML-Seite besteht und deren Inhalte dynamisch nachgeladen werden.

In klassischen Webanwendungen liefert der Server auf Anfragen des Clients immer vollständige HTML-Dokumente aus. Dies löst beim Webclient

[5] https://www.jetbrains.com/webstorm

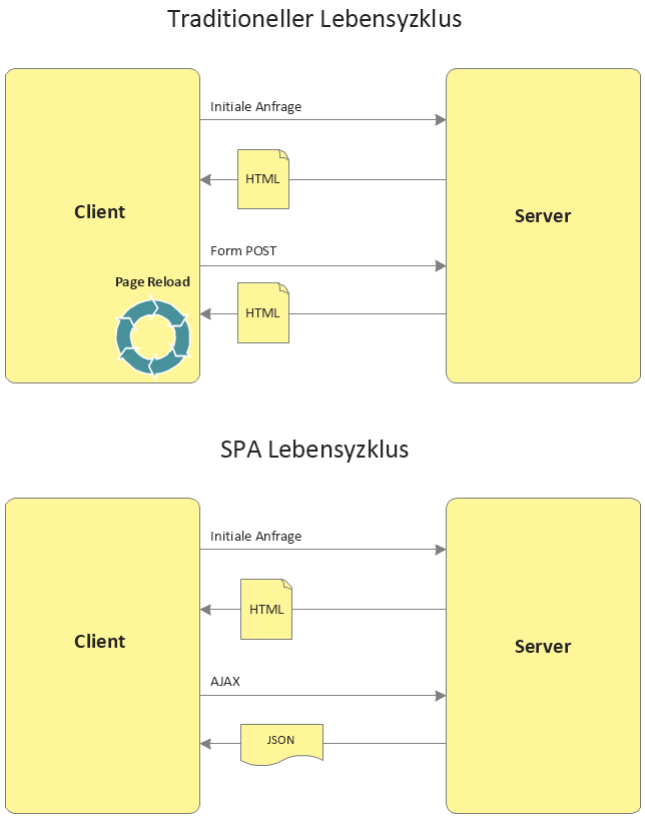


Abbildung 5.3: Der traditionelle Lebenszyklus gegenübergestellt mit dem Single-Page-App Lebenszyklus einer Seite (angelehnt an [49]).

einen sogenannten „Page Reload“ aus, der in einer Neudarstellung des retournierten HTML-Dokuments resultiert. Bei Single Page Anwendungen hingegen wird nur initial ein HTML-Dokument vollständig geladen. Alle weiteren Anfragen erfolgen über „Asynchronous JavaScript and XML“-Aufrufe. Der Server liefert anschließend eine Antwort in einem durch Content-Negotiation spezifizierten Format, in unserem Fall, als JSON-Dokument. Diese JSON-Daten werden dann vom Browser verwendet, um die Inhalte einer Seite dynamisch auszutauschen. Abbildung 5.3 stellt die Unterschiede beider Ansätze dar.

Ein wesentlicher Vorteil von SPAs ist die gesteigerte Interaktivität und Performanz der Anwendung, da anstatt von Markup-Code hauptsächlich Daten in Form von JSON ausgetauscht werden. Angular UI Router[6] stellt dabei das zentrale und unterstützende Framework zur Umsetzung dar.

[6] https://github.com/angular-ui/ui-router

5.6 Zusammenfassung

Dieses Kapitel diente der Beschreibung der grundsätzlichen Anforderungen und der Umsetzung der selbst entwickelten WoTCloud-Plattform. Es wurden Ziele definiert und konkret Anforderungen aus diesen Zielen abgeleitet, die aus den Ergebnissen der Befragungen aus Kapitel 3 und dem Vergleich aus Kapitel 4 resultieren. Darauf aufbauend entwickelten die Autoren unter Berücksichtigung komplexer Dinge ein konzeptuelles Modell, das auch als Datenmodell dient. Die konkrete Implementierung von WoTCloud basiert auf einer logischen Trennung in drei Schichten: Data Layer, Service Layer und Presentation Layer. Es wurde insbesondere die hybride Datenarchitektur des Data Layers, die REST-basierte Architektur des Service Layers und die Umsetzung des Presentation Layers als Single Page Application vorgestellt.

Teil II

Spezifische Aspekte der Web-of-Things-Plattform

Kapitel 6

Multi-Tenancy und skalierbare Datenhaltung

Da die von den beiden Autoren entwickelte Plattform von Beginn als cloudbasierte SaaS-Anwendung implementiert wurde, werden zunächst die Begriffe Cloud Computing sowie Software-as-a-Service (SaaS) definiert.

Aus Sicht von Salesforce [44] ist Multi-Tenancy eine wesentliche Technologie im Cloud Computing um IT-Ressourcen kosteneffizient und sicher gemeinsam zu nutzen. Hierfür beleuchtet das folgende Kapitel Multi-Tenancy als organisationalen Ansatz zur gemeinsamen und effizienten Nutzung von IT-Ressourcen und dessen Umsetzung anhand des Shared Table Ansatzes in WoTCloud.

Weiters wird Sharding als horizontaler Fragmentierungsansatz im Kontext von Cloud Computing diskutiert, da vertikale Skalierung nur eingeschränkt in der Cloud umgesetzt werden kann. Multi-Tenancy und Sharding sind insofern für WoTCloud von Bedeutung, als die Plattform für die globale Nutzung durch Vending Betreiber vorgesehen wurde.

Aus diesem Grund wird auf eine mögliche Umsetzung von Sharding in Microsoft Azure SQL Database mithilfe von Elastic Scale eingegangen. Elastic Scale ist dabei eine zusätzliche Komponente für Azure SQL Database zum bedarfsgerechten horizontalen Skalieren der Datenschicht. Der Autor der vorliegenden Arbeit untersucht inwiefern eine mögliche Implementierung mittels Elastic Scale erfolgen kann und welche Anpassungen nötig sind.

6.1 Rahmenbedingungen

In Folge werden die beiden Begriffe *Cloud Computing* und *Software-as-a-Service* erläutert, die für das Verständnis dieser Arbeit erforderlich sind.

6.1.1 Cloud Computing

Aufgrund der Vielzahl an Definitionen von *Cloud Computing* wurde jene des National Institute of Standards and Technology (NIST) übernommen. Diese Definition ist insofern zutreffend, da sie Cloud Computing im weiteren Sinne beschreibt.

> „Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models.“ [39]

Aus der Definition des NIST gehen wesentliche Merkmale hervor, die Cloud Computing im weiteren Sinne definieren und in Folge kurz diskutiert werden.

Wesentliche Merkmale

Das NIST [39] und Baun et al. [6] definieren folgende Merkmale von Cloud Computing:

- *Selbstbedienung (On-demand self-service)*: Die Bereitstellung von Rechenzeit, Netzwerkspeicher oder andere IT-Ressourcen soll automatisch und ohne menschliche Interaktion auf Seite des Cloud Providers erfolgen.
- *Breitbandzugriff (Broad network access)*: Die angebotenen Dienste sollen über ein Breitbandnetzwerk (z. B. Internet) zur Verfügung gestellt werden und mittels Standardmechanismen und -protokollen zugänglich sein, sodass auch heterogene Clients (z. B. Desktops, Tablets, Smartphones, etc.) darauf zugreifen können.
- *Gemeinsame Nutzung physischer Ressourcen (Resource pooling)*: Die Bündelung von Ressourcen in einem Pool ermöglicht dem Cloud Provider, mehreren Kunden gleichzeitig physische und virtuelle Ressourcen zuzuordnen. Je nach Bedarf und Nachfrage werden die Ressourcen dabei dynamisch zugeteilt und bei Freigabe wieder an den Pool zurückgegeben.
- *Rapide Elastizität (Rapid elasticity)*: Mit der Unterstützung von Selbstbedienung sollen Rechenleistung und Speicher zeitnahe und flexibel bereitgestellt werden. Einer nutzungsbasierten Skalierung steht somit nichts im Weg. Dem Benutzer wird dabei der Eindruck vermittelt, als wären die zur Verfügung stehenden Ressourcen unbegrenzt.
- *Überwachung und Messbarkeit (Measured service)*: Im Sinne einer nutzungsbasierten Abrechnung (engl. Pay-per-Use) der konsumierten Diens-

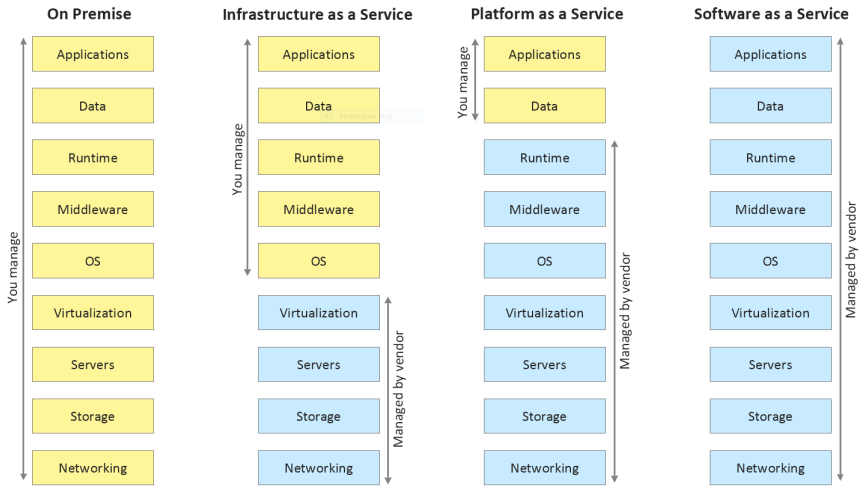


Abbildung 6.1: Service Modelle von Cloud Computing (angelehnt an [25]).

te ist eine genaue Messung der tatsächlichen Ressourcennutzung ausschlaggebend. Die Ressourcennutzung kann überwacht, kontrolliert und an den Provider sowie die Kunden berichtet werden.

6.1.2 Software-as-a-Service

> „Bei der Nutzung von SaaS wird den Kunden eine Standardsoftwarelösung als Dienstleistung über das Internet zur Verfügung gestellt. Der SaaS-Anbieter ist für Betrieb und Wartung der mehrmandantenfähigen Software verantwortlich. Dabei erzielen die Anbieter keine Lizenzeinnahmen. Vielmehr zahlen die Anwender für die gemieteten Softwarekomponenten und Serviceleistungen Nutzungsgebühren, die in der Regel monatlich, quartalsbasiert oder jährlich anfallen.“ [12]

SaaS ist ein Servicemodell [39] von Cloud Computing, das sich von *Infrastructure-as-a-Service (IaaS)* und *Platform-as-a-Service (PaaS)* aufgrund des Abstraktionsgrades der zugrunde liegenden technischen Infrastruktur unterscheidet. Wie in Abbildung 6.1 ersichtlich, stellt der Cloud Provider bei Software-as-a-Service den umfassenden Betrieb der Software sicher. Dem Nutzer wird dabei völlig transparent die Endanwendung zur Verfügung gestellt, wobei dieser keinen Einfluss auf die zugrunde liegende Infrastruktur nehmen kann. Die Voraussetzungen für die Nutzung einer SaaS-Anwendung sind dabei oft nur minimal, d. h. Breitbandnetzwerk + Webbrowser.

Stellvertretend für kommerzielle SaaS-Lösungen können folgende Anwendungen angeführt werden: Salesforce Sales Cloud, Microsoft Office 365, Goo-

gle Apps for Work und Microsoft Dynamics CRM Online.

6.2 Grundlagen von Multi-Tenancy

In diesem Abschnitt werden Begrifflichkeiten rund um Multi-Tenancy definiert sowie Eigenschaften und Herausforderungen diskutiert.

6.2.1 Definition

Die Autoren Bezemer & Zaidman definieren *Multi-Tenancy* (dt. Mehrmandantenfähigkeit) wie folgt:

> „A multi-tenant application lets customers (tenants) share the same hardware resources, by offering them one shared application and database instance, while allowing them to configure the application to fit their needs as if it runs on a dedicated environment.“ [53]

Weiters definieren Bezemer & Zaidman einen *Tenant* als:

> „A tenant is the organizational entity which rents a multi-tenant SaaS solution. Typically, a tenant groups a number of users, which are the stakeholders in the organization.“ [53]

Wie aus der obigen Definition hervorgeht, geht es bei Multi-Tenancy sowohl um die gemeinsame Nutzung von Applikations- und Datenbankinstanz als auch um die Anpassbarkeit bzw. Konfigurierbarkeit einer Anwendung an Anforderungen des Tenant.

Die vorliegende Masterarbeit betrachtet Multi-Tenancy ausschließlich auf Datenbankebene und vernachlässigt bewusst die gemeinsame Nutzung der Applikationsinstanz sowie das Anpassen der Software an Organisationsbedürfnisse.

Nachfolgend wird Multi-Tenancy von zwei traditionelleren Ansätzen abgegrenzt.

Multi-Tenant versus Multi-User

Nach Bezemer & Zaidman [53] nutzen in einer Multi-User Anwendung alle Tenants diesselbe standardisierte Anwendung ohne Möglichkeit einer Anpassung an spezielle Bedürfnisse. Im Gegensatz zu einer Multi-Tenant Applikation kann ein Tenant bei diesem Typ von Anwendung das User Interface, Datenstrukturen und Workflows nicht verändern. In der Regel haben auch alle Tenants die gleiche Dienstgütevereinbarung (SLA).

Typische Beispiele für Multi-User Anwendungen sind: Facebook, Twitter, Instagram, Xing, LinkedIn, usw.

Multi-Tenant versus Multi-Instance

Wird jedem Tenant eine eigene bzw. dezidierte Applikations- und/oder Datenbankinstanz bereitgestellt, so spricht man von Multi-Instance bzw. Single-Tenancy [53]. Betrachtet man die Anpassbarkeit dieses Typs, so ist festzustellen, dass eine Anpassung der Applikations- und/oder Datenbankinstanz einfach umzusetzen ist, da Tenants isoliert von anderen betrieben werden und Auswirkungen somit nur auf den zu ändernden Tenant begrenzt sind. Nachteilig sind jedoch die hohen Wartungs- bzw. Administrationskosten, zumal Updates und Fehlerbehebungen auf allen isolierten Instanzen nachgezogen werden müssen.

6.2.2 Eigenschaften

Die Autoren Bezemer & Zaidman [53] und Cai et al. [13] definieren folgende Eigenschaften von Multi-Tenancy unabhängig von Applikations- und Datenbankinstanz.

Anpassbarkeit

Ein weiteres wichtiges Charakteristikum von Multi-Tenancy ist die Anpassung bzw. Konfigurierbarkeit [53] der Software durch den Tenant. Im Falle von Single-Tenancy kann hierfür der Source Code individuell für jeden Tenant mittels Branches[1] abgeändert werden. Bei einer Multi-Tenant Anwendung verwenden jedoch alle Tenants diesselbe Codebasis, was den Entwicklern nur eine Anpassungsmöglichkeit offen lässt: die Anpassbarkeit bzw. Konfigurierbarkeit zur Laufzeit (engl. runtime configurable).

Gemeinsame Nutzung der Applikations- und Datenbankinstanz

Idealerweise gibt es bei einer Multi-Tenant Lösung nur eine Applikationssowie Datenbankinstanz [53]. Eine Ausnahme stellt die Replikation der Applikationsinstanz zum Zwecke der Skalierung dar. Im Vergleich zu Multi-Instance reduziert sich die Anzahl der Instanzen drastisch, was das Deployment wesentlich vereinfacht. Vor allem repetitive Tätigkeiten wie das Ausrollen von Updates müssen nur mehr auf einigen wenigen Instanzen durchgeführt werden.

Durch diese gemeinsame Nutzung der Infrastruktur wird die Kapazität der Infrastruktur besser ausgelastet, was zu Kosteneinsparungen führt. In Abschnitt 6.3 werden hierfür Ansätze zur gemeinsamen Nutzung auf Datenbankebene vorgestellt.

[1] http://de.wikipedia.org/wiki/Versionsverwaltung

Isolation

Die Autoren Cai et al. [13] führen Isolation als weiteres wichtiges Merkmal an. Sie sind der Ansicht, dass es spezielle Artefakte in einer Anwendung gibt, die besonders schützenswert gegenüber anderen Tenants sind. Dazu zählen beispielsweise Authentifizierung und Autorisierung, Dateien, Remote Service Calls, Datenbanken und Messages. Eine Implementierung von Multi-Tenancy muss folglich diese Artefakte voneinander isolieren.

6.2.3 Herausforderungen

Abgeleitet aus den Merkmalen von Multi-Tenancy ergeben sich zu beachtende Herausforderungen, die im Folgenden erläutert werden.

Individualisierung

Der Bitkom zufolge ist eine der größten Herausforderungen die Individualisierung des Angebots [50]. Individualisierung erhöht zwar den Nutzen für den Kunden und somit auch dessen Zufriedenheit, senkt aber die Standardisierung des Angebots und die damit verbundenen Skaleneffekte [50]. Es muss dafür gesorgt werden, die Attraktivität eines SaaS-Angebots bei gleichzeitig hoher Standardisierung beizubehalten.

Performanz

Durch die gemeinsame Nutzung von Ressourcen kann es vorkommen, dass ein Tenant mehr Ressourcen, beispielsweise durch komplexere Abfragen, beansprucht als ein anderer [53]. Die Anwendung muss dabei sicherstellen, dass eine solche Beeinträchtigung der anderen Tenants nicht erfolgt.

Die Autoren Aulbach et al. [4] schlagen hierfür eine zentrale Überwachungsinstanz im Sinne eines Controllers vor, der die Einhaltung der Service-Level-Agreement (SLA) steuert, überwacht und gegebenenfalls einzelne Tenants priorisiert und ressourcenintensive Anfragen zurücksetzt.

Skalierbarkeit

Unter Skalierbarkeit nennen die Autoren Chong et al. [14] die Herausforderung auf eine gestiegene Arbeitslast bedarfsgerecht mittels Skalierung zu reagieren. Diese Zunahme wird in der Regel sowohl auf Anwendungs- aber auch auf Datenbankebene durch eine Kapazitätserhöhung bewerkstelligt. Daher müssen Anwendung sowie Datenbank für eine mögliche vertikale bzw. horizontale Skalierung konzipiert worden sein. Vertikale Skalierung meint dabei das Steigern der Leistung durch Erweiterung der Kapazitäten mittels Hardware, wobei horizontale Skalierung eine gesteigerte Leistung durch Hinzufügen von zusätzlichen Rechnern/Knoten erreicht.

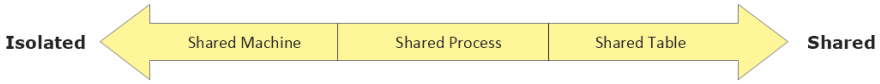


Abbildung 6.2: Isolation versus Ressourcenteilung (angelehnt an [15]).

Sicherheit

Gerade in einer Multi-Tenant Umgebung können nicht vollständig implementierte Sicherheitsmaßnahmen enorme Auswirkungen haben. Ein Defekt in der Anwendung könnte dabei zum Diebstahl der Daten aller Tenants führen oder Tenants Zugang zu sensiblen Daten des Mitstreiters gewähren (gen. Cross Tenant Access) [53].

Verfügbarkeit

Selbstbedienung als wesentliches Merkmal von Cloud Computing (siehe 6.1.1) meint die automatische Bereitstellung eines Tenants ohne menschliche Interaktion auf Seite des Anbieters. Nach erfolgter Bereitstellung möchten Tenants mehr oder weniger ihre organisationalen Bedürfnisse in der Anwendung über Anpassungen abbilden. Diese Vorgänge dürfen sich dabei nicht auf die Performanz oder Verfügbarkeit des Systems auswirken [53], da ein Ausfall des Systems zur Nichteinhaltung vereinbarter Service-Level-Agreements führen kann. Diese nicht eingehaltene garantierte Verfügbarkeit kann dem Anbieter oft teuer zu stehen kommen.

6.3 Ansätze zur Umsetzung von Multi-Tenancy auf Datenbankebene

In Folge werden Ansätze zur Umsetzung von Multi-Tenancy auf Datenbankebene diskutiert. Für die Betrachtungsweise der Applikationsinstanz sind die unten angeführten Ansätze nicht geeignet.

Jeder der in Folge betrachteten Ansätze weißt einen unterschiedlichen Grad der Ressourcenteilung bzw. Isolation auf (siehe Abbildung 6.2). Die Isolation gibt dabei an, inwiefern die Tenants physisch voneinander getrennt sind. Ressourcenteilung hingegen definiert den Grad der Nutzung von Skaleneffekten durch die gemeinsame Nutzung der Datenbankinstanz [15]. Eine höhere Ressourcenteilung kann dabei nur zulasten einer geringeren Isolation und umgekehrt erreicht werden.

6.3.1 Shared Machine

Der Ansatz mit der höchstmöglichen Isolationsstufe ist der Shared Machine-Ansatz [15]. Hierbei erhält jeder Tenant bzw. Betreiber (Vending Machine

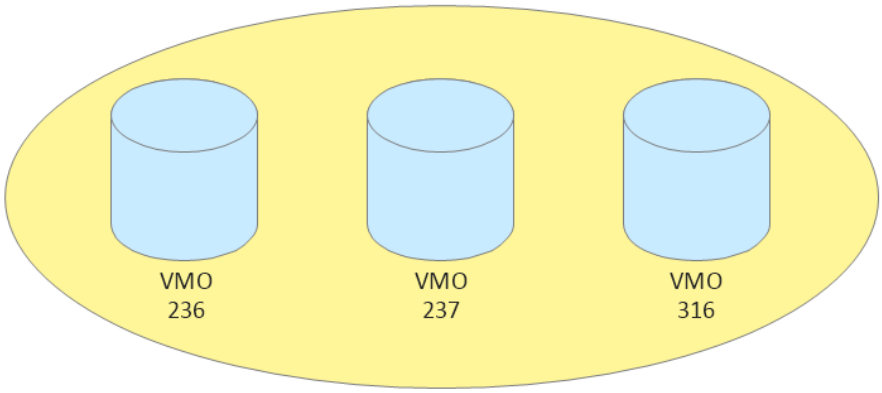


Abbildung 6.3: Shared Machine Ansatz (angelehnt an [15]).

Operator (VMO)) eine dezidierte Datenbank auf einem gemeinsamen Datenbankserver (siehe Abbildung 6.3). Folglich teilen sich die Tenants bei diesem Ansatz nur den Datenbankserver untereinander. Dieser Ansatz eignet sich für Szenarien, bei denen Datensicherheit [15] ein wichtiges Kriterium darstellt. Beispiele dafür sind Bank-, Versicherungs- und Gesundheitswesen.

Anpassungen eines Tenants sind bei diesem Ansatz simpel umzusetzen, da die Erweiterungen aufgrund der physischen Trennung nur eine Datenbank und dessen Schema tangieren. Der hohe Grad an Isolation wirkt jedoch der Ressourcenteilung stark entgegen, was bedeutet, dass auch bei niedriger Arbeitslast gewisse Grundfunktionalitäten CPU und Speicherplatz für sich reservieren (gen. Overhead). Die Ausnutzung von Skaleneffekte ist somit nur eingeschränkt möglich. Darüber hinaus ist die Anzahl der Tenants durch die maximale Anzahl der Datenbanken pro Server beschränkt [34].

6.3.2 Shared Process

Im Unterschied zu Shared Machine befinden sich beim Shared Process [15] alle Tenants in derselben Datenbank, jedoch in unterschiedlichen Tabellen. Jeder Tenant besitzt für sich ein eigenes Schema wie in Abbildung 6.4 dargestellt. Dieser Ansatz vertraut auf die physische und logische Trennung in der Anwendung.

Wie auch schon bei Shared Machine sind Anpassungen bzw. Erweiterungen leicht umsetzbar, da nur das Schema des gewünschten Tenants verändert werden muss. Da nur mehr eine Datenbank für die Tenants existiert, werden Skaleneffekte auf Kosten der verringerten Isolation erzielt. Schwieriger hingegen gestaltet sich die Wiederherstellung von einzelnen Tenants aus einem Backup, da nur das Schema bzw. die einzelnen Tabellen des Tenants wiederhergestellt werden dürfen.

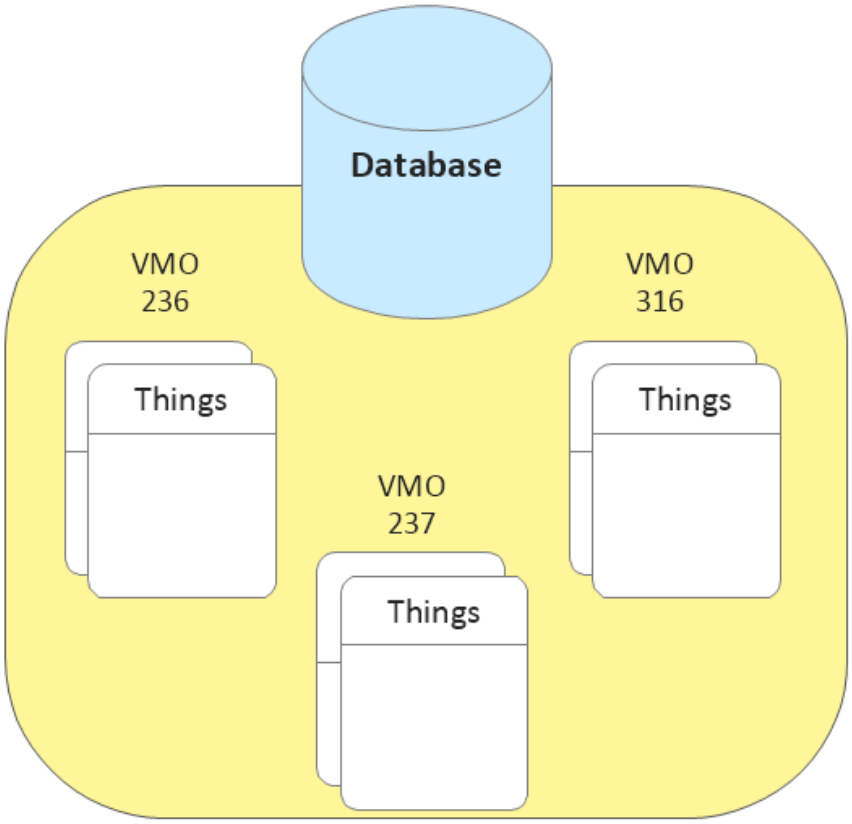


Abbildung 6.4: Shared Process Ansatz (angelehnt an [15]).

6.3.3 Shared Table

Nach den Autoren Chong et al. [15] ist der Shared Table Ansatz jener Ansatz mit der höchsten Ressourcenkonsolidierung und niedrigster Isolation. Hierbei teilen sich die Tenants Datenbankserver, Schema und auch die Tabellen (siehe Abbildung 6.5). Voraussetzung für das Funktionieren dieses Ansatzes ist die Zuweisung von Tupeln zu einer sogenannten `TenantId`. Diese `TenantId` identifiziert beliebige Tupel (gen. Rows) eindeutig bzw. ermöglicht die Zuordnung zu einem Tenant.

Auch bei diesem Ansatz ist aufgrund der fehlenden physischen Isolation wiederum die Anwendung für die Isolation verantwortlich. Folglich bringt dieser Ansatz den höchsten Entwicklungsaufwand mit sich, da es sogenannte Cross-Tenant Zugriffe zu verhindern gilt und Anpassungen/Erweiterungen aufgrund der gemeinsamen Tabellen aufwendiger zu gestalten sind [15]. Dieser Ansatz eignet sich vor allem in cloud-basierten Software-as-a-Service Anwendungen, bei denen Skaleneffekte wesentliche Treiber des Geschäfts-

TenantID	Name	IsActive
236		
237		
316		
389		

TenantID	ThingID	Name	CreateDate
236	77	Robimat 75	2015-06-01
236	78	FK 230	2015-05-20
237	79	FK 230	2015-06-10
389	80	FK 170	2015-06-12

Abbildung 6.5: Shared Table Ansatz (angelehnt an [15]).

modells sind.

Der Shared Table Ansatz kann mit verschiedensten weiteren Tabellenlayouts [5] umgesetzt werden, die jedoch in dieser Arbeit nicht diskutiert werden.

Für die zu entwickelnde Web-of-Things-Plattform *WoTCloud* wurde dieser Ansatz aufgrund der Skaleneffekte als geeignet identifiziert und in Folge auch umgesetzt. Es wird angenommen, dass die Plattform zukünftig von einer Vielzahl an Betreibern genutzt wird, die jeweils ein unterschiedliches Lastverhalten erzeugen. Auch die aufgezeigte Umsetzung (in Abschnitt 6.4) mittels Elastic Scale bedingt die Umsetzung des Shared Table Ansatzes um eine flexible Reorganisation der Daten zur Laufzeit zu ermöglichen. Die Skalierung dieses Ansatzes wird im Wesentlichen durch das Entwurfsmuster Sharding realisiert und im nächsten Abschnitt beschrieben bzw. komplementiert.

6.3.4 Sharding

> „Sharding is a horizontal scaling strategy in which resources from each shard (or node) contribute to the overall capacity of the sharded database. Database shards are said to implement a shared nothing architecture that simply means that nodes do not share with other nodes; they do not share disk, memory, or other resources.“ [52]

Sharding ist laut Wilder [52] ein fortgeschrittenes Entwurfsmuster zum horizontalen Skalieren bzw. Fragmentieren (auch gen. *Scale-out*) von Daten. Konkret wird dabei eine einzige Datenbank in zwei oder mehrere physische Datenbanken (gen. Shards) aufgeteilt. Alle einzelnen Shards haben dabei nur das Datenbankschema gemein und teilen sich sonst keine Ressourcen (auch gen. *shared-nothing-architecture)*. Aufgeteilt werden die einzelnen Tupel an-

TenantID	Name	IsActive	UserName
1	Vending GmbH	True	mmustermann@vending.at
2	ABC Vending	False	psteiner@abcvending.at
3	Fabrikam Inc.	True	mbonifac@fabrikam.com
...	...	...	...
100	Winery Corp.	True	svladislav@winery.com
160	Getränke OG	True	rmarius@getraenke.de

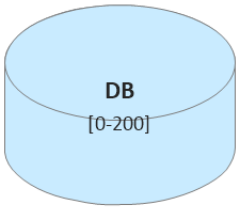


Abbildung 6.6: Die Datenbank vor dem Einsatz von Sharding.

hand eines *Sharding Keys*, der einen Datensatz disjunkt zu einem Shard zuweist.

In Abbildung 6.7 wurde hierfür beispielhaft die `TenantID` als Sharding Key herangezogen. Der erste Shard enthält die Kunden mit den IDs 0-100, wohingegen der zweite Shard Kunden mit den IDs 100-200 speichert. Der Sharding Key kann dabei willkürlich festgelegt werden und auch alphanumerische Typen umfassen.

Für eine Applikation ist die Anzahl der dahinterliegenden Datenbanken transparent. Die Applikation verbindet sich lediglich mit der Root-Datenbank, die den richtigen Shard ausfindig macht.

In der Vergangenheit konnte Sharding aufgrund fehlender Unterstützung in Datenbankmanagementsystemen (DBMS) nur mittels aufwendigen Eigenentwicklungen im Applikationscode sichergestellt werden. Heutzutage stellen fortgeschrittene DBMS diese Skalierungsinfrastruktur out-of-the-box zur Verfügung.

Motivation

In der Vergangenheit wurde eine Datenbank meist auf einem einzigen Rechnerknoten betrieben. Die Performanz oder Leistung, ausgedrückt durch den Durchsatz, kann nur durch Aufstocken der zugrundeliegenden Hardware (gen. *Scale-up* in Abbildung 6.9) erfolgen. Die zu bewältigende Transaktionslast ist somit auf die Kapazität des Datenbankservers beschränkt. Engpässe können dabei Prozessor, Hauptspeicher, Festplattengeschwindigkeit oder die

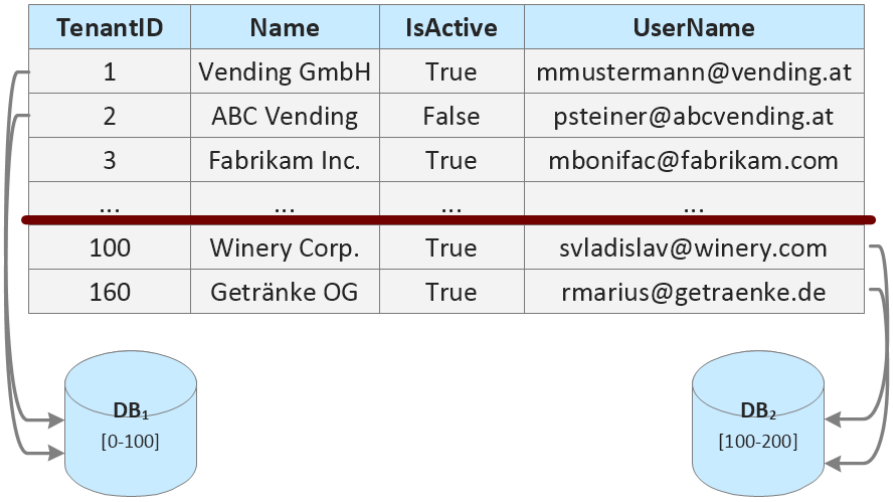


Abbildung 6.7: Die Datenbank unter Verwendung von Sharding.

Netzwerkbandbreite darstellen.

Ab einem gewissen Punkt kann aber die Performanz durch Kapazitätserweiterungen nicht mehr weiter gesteigert werden. Erschwerend kommt hinzu, dass Cloud Provider meist sog. „Commodity Hardware“ einsetzen. Anstatt kostspielige Computer mit High-End Hardware zu betreiben, wird bei Commodity Hardware eine Vielzahl an handelsüblichen Rechnern mit hoher Komponentenstandardisierung verwendet, die im Bedarfsfall einfach ersetzt werden können. Commodity Hardware unterstreicht damit die Wichtigkeit von Sharding, da vertikales Skalieren in der Cloud nur begrenzt möglich ist [52].

6.4 Mögliche Umsetzung in WoTCloud mit Elastic Scale

In diesem Kapitel wird untersucht, inwieweit der in Abschnitt 6.3.3 erwähnte Shared Table Ansatz unter Berücksichtigung von Sharding in einem heute aktuellen cloud-basierten relationalen Datenbanksystem unterstützt wird. Hierfür wurde das im Oktober 2014 vorgestellte Produkt Elastic Scale für Microsoft Azure SQL Database näher untersucht. Auf eine vollständige Umsetzung wurde dabei im Rahmen dieser Masterarbeit verzichtet, da zum Zeitpunkt des Verfassens nur eine Public Preview Version vorlag und das Microsoft Entity Framework nur unzureichend[2] unterstützt wurde. Entity Framework ist ein objektrelationaler Mapper, der relationale Daten in .NET-

[2] http://stackoverflow.com/questions/27528988/executing-an-elastic-scale-multi-shard-query-on-the-database-context

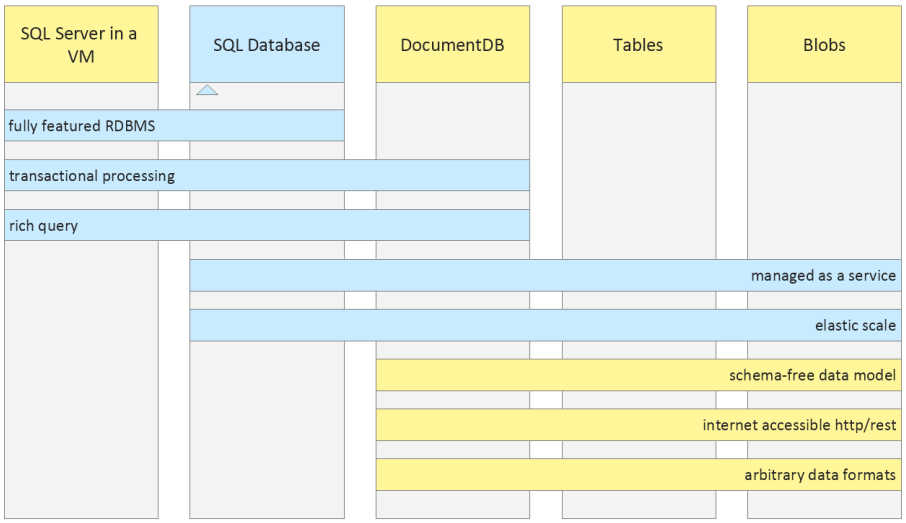


Abbildung 6.8: Verfügbare Speichertechnologien in Microsoft Azure (angelehnt an [2]).

Objekten abbildet und damit wesentlicher Bestandteil des Data Layer. Mithilfe einer beispielhaften Umsetzung soll jedoch ein Ausblick über zukünftige Entwicklungen in diesem Bereich gegeben werden.

Zur besseren Einordnung wird in Abbildung 6.8 eine Übersicht über sämtliche Azure Storage Technologien gezeigt. Für die vorliegende Arbeit ist dabei ausschließlich SQL Database von Bedeutung, da die IaaS Variante (SQL Server in einer virtuellen Maschine) kein Elastic Scale ermöglicht.

6.4.1 Einleitung

In Microsoft Azure ist das horizontale Skalieren der Web Apps[3] durch die einfache Änderung des `InstanceCount` im Portal möglich. Als Herausforderung galt diese horizontale Skalierung bis vor kurzem für die Datenschicht (Data Layer). *Elastic Scale* wurde von Microsoft mit dem Ziel entwickelt, den Anwendungsentwickler beim Implementieren von skalierbaren Datenbanklösungen maßgeblich zu unterstützen. Der Entwickler kann sich somit wieder auf seine Kernkompetenzen, das Entwickeln von Anwendungscode, konzentrieren und auf Elastic Scale als Skalierungsinfrastruktur für Datenbanken zurückgreifen. Elastic Scale ist dabei die konkrete Implementierung des vorgestellten Shardings auf Datenbankebene, durch das die Komplexität des Skalierens vom Applikationscode in die Datenbank verlagert wird.

Ein weiteres Motiv für die Entwicklung von Elastic Scale war die begrenz-

[3] http://azure.microsoft.com/en-us/services/app-service

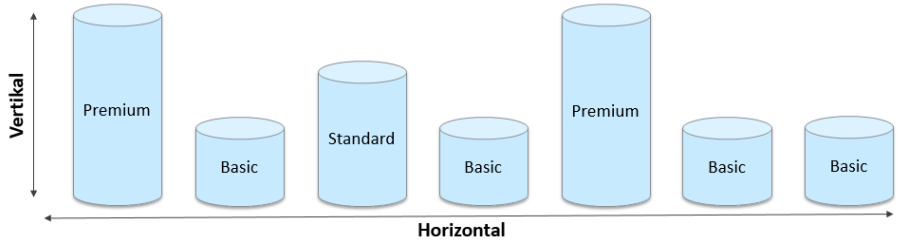


Abbildung 6.9: Horizontales und vertikales Skalieren in Azure SQL Database.

te Datenbankgröße in Azure SQL Database. Die maximale Datenbankgröße[4] in Azure SQL Database ist auf 500 GB begrenzt. Sollte diese Grenze nicht ausreichen, müssten, ohne Einsatz von Sharding und Elastic Scale, Produktivdaten vom System entfernt und archiviert werden. Durch die horizontale Fragmentierung mittels Elastic Scale kann diese Einschränkung umgangen werden.

In Abbildung 6.9 sind beide Skalierungsansätze von Azure SQL Database dargestellt, wobei für die nachfolgenden Ausführungen die horizontale Skalierung mittels Elastic Scale von Bedeutung ist. Die vertikale Skalierung erfolgt durch Ändern des Service Tiers[5] (derzeit Basic, Standard und Premium). Dabei wird die Leistung einer Datenbank durch Aufstocken der Hardware gesteigert.

6.4.2 Architektur

Die Architektur (siehe Abbildung 6.10) von Elastic Scale zeigt dabei eine klare Rollenverteilung auf: Auf der einen Seite ist der Anwendungsentwickler, der für die Entwicklung der Anwendung mithilfe der Client Library verantwortlich ist und auf der anderen Seite der Administrator verantwortlich für den späteren Betrieb der Anwendung. Letzterer verwendet hierfür die Management Services, die im Wesentlichen das Split-Merge-Tool beinhalten. Für den Entwickler steht die Interaktion von meist nur einem Shard im Vordergrund, da meist nur Daten eines Tenants gelesen bzw. geschrieben werden. Die derzeit vorliegende Version besteht aus der *Elastic Database Client Library* sowie den *Management Services*.

Elastic Scale unterstützt dabei folgende .NET Typen [33] zur Definition als Sharding Key: integer, long, guid, byte[], datetime, timespan und datetimeoffset.

Durch das zentrale Management der einzelnen Shards erlaubt Elastic Scale folgendes: Es kann eine globale Anwendung entwickelt und betrieben

[4] https://msdn.microsoft.com/en-us/library/azure/dn741336.aspx
[5] https://msdn.microsoft.com/en-us/library/azure/dn741336.aspx

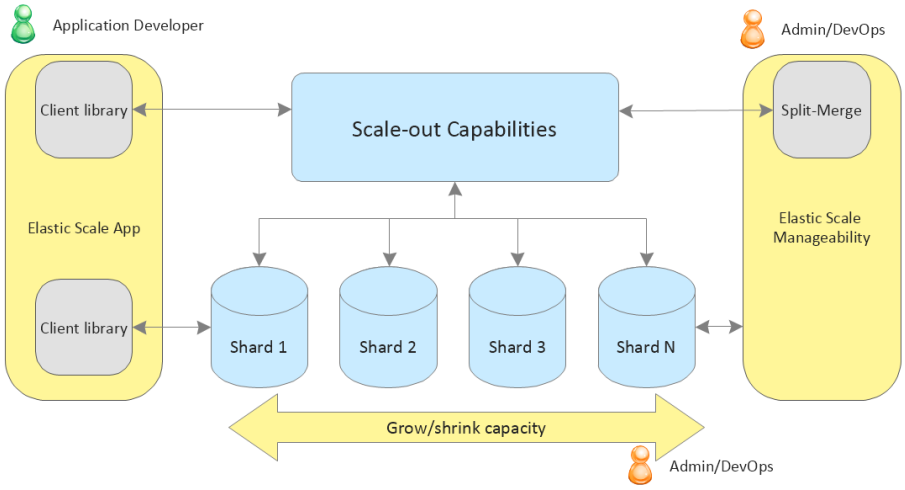


Abbildung 6.10: Die Architektur von Elastic Scale unter Berücksichtigung der Rollen Anwendungsentwickler und Administrator (angelehnt an [31]).

werden, wobei die einzelnen Shards (Datenbanken) in Microsoft Rechenzentren in der Nähe des Kunden gespeichert werden können. Somit finden (bei Vorhandensein eines Rechenzentrums im jeweiligen Land) auch regulatorische Anforderungen hinsichtlich Speicherort des Kunden Berücksichtigung. Früher musste solch eine globale Anwendung mittels Multi-Instance mehrfach betrieben werden, um die Daten in der Nähe des Kunden zu halten.

6.4.3 Wesentliche Bestandteile

In diesem Abschnitt werden die wichtigsten Bestandteile von Elastic Scale und beispielhafte Auswirkungen auf WoTCloud erläutert.

Shard Map Management

Das *Shard Map Management* ist jene Komponente, die verschiedenste Metadaten [33] über die einzelnen Shards speichert. Dieses Feature ist Teil der Elastic Scale Client Library. Der Entwickler kann mithilfe dieser Komponente die einzelnen Datenbanken als Shards registrieren, die Mappings (List oder Range) für die einzelnen Datenbanken festlegen und die Metadaten verwalten. Vor Elastic Scale mussten die genannten Funktionalitäten von den Entwickler selbst zur Verfügung gestellt werden, was zu einem hohen Entwicklungsaufwand führte.

Um die Zuordnungen der einzelnen Sharding Keys zu den Shards abzuspeichern, gibt es in Elastic Scale die Shard Map. Diese unterhält für List und Range Mappings eine Zuordnung des Sharding Keys zur assoziierten Datenbank (Shard). Im Falle der List Map werden für jede Ausprägung des

Key	*Shard Location*
1	Database_A
3	Database_B
4	Database_C
6	Database_B

Tabelle 6.1: List Mapping in Elastic Scale (angelehnt an [33]).

Key Range	*Shard Location*
[1,50]	Database_A
[50,100]	Database_B
[100,200]	Database_C
[400,600]	Database_C

Tabelle 6.2: Range Mapping in Elastic Scale (angelehnt an [33]).

Sharding Keys die einzelnen Shards definiert. Wie in Abbildung 6.1 ersichtlich, ist das List Mapping also explizit, da für jede Ausprägung ein Shard angegeben werden muss, bevor Lese- und Schreibetransaktionen dazu gestartet werden können.

Im Gegensatz dazu wird beim Range Mapping (Abbildung 6.2) eine Zuordnung in der Map immer als durch das Wertepaar `[LowValue, HighValue]` spezifiziert. Der `LowValue` repräsentiert dabei den kleinsten Key und der `HighValue` den höchsten exklusiven Key in der Range [33]. Beispielsweise beinhaltet die Range [0,50] die Werte bzw. Tenants 0 bis 49.

Beide der genannten Ansätze ermöglichen dabei die mehrfache Zuordnung von Shards zu individuellen Ausprägungen oder Ranges (in Abb. 6.1 Datenbank B und in Abb. 6.2 Datenbank C). Durch diesen Umstand ist es möglich, mittels Elastic Scale sowohl Single-Tenancy als auch Multi-Tenancy (siehe Shared Table 6.3.3) zu unterstützen. Beispielsweise wäre die Implementierung des Geschäftsmodells Freemium[6] durch Umsetzung einer hybriden Form ohne Weiteres möglich. Dieses Geschäftsmodell könnte mit Elastic

[6]Beim Freemium wird ein Basisprodukt meist kostenfrei angeboten, wohingegen die vollständige Version kostenpflichtig ist.

Scale folgendermaßen realisiert werden: Ein Nutzer der vollständigen Version erhält einen dezidierten Shard (Datenbank), wobei ein Nutzer der freien Version sich den Shard mit mehreren Nutzern teilt.

Zur Speicherung all dieser Metadaten (Mappings und Shards) verwendet die Shard-Map-Komponente eine eigene und zusätzliche Datenbank, auch als Management-Datenbank bezeichnet. Eine Anwendung benötigt lediglich die Verbindung zu dieser Management-Datenbank für alle Transaktionen. Dieser `ConnectionString` ist in Listing 6.1 dargestellt.

```
<connectionStrings>
<addname="WoTCloudManagement" connectionString=
"Data Source=[server].database.windows.net;
Integrated Security=False;
Initial Catalog=WoTCloud.Management;
User Id=[login]@[server];
Password=[password];
Trusted_Connection=False;
Encrypt=true;"
providerName="System.Data.SqlClient"/>
</connectionStrings>
```

Listing 6.1: Der ConnectionString zur ShardMapManager Datenbank.

Data Dependent Routing

Da die Anwendung nur einen Verweis auf die Management-Datenbank besitzt (siehe Listing 6.1) und damit nur transitiv Kenntnis über die dahinterliegenden Datenbanken hat, impliziert, dass die eingehenden Anfragen bzw. Transaktionen auf Basis des Sharding Keys an die richtige Datenbank weitergeleitet werden müssen. Genau dies wird durch *Data Dependent Routing* erledigt. Sobald der für die Transaktion benötigte Shard ausfindig gemacht wurde, wird eine Verbindung zu diesem aufgebaut und die Transaktion ausgeführt [31]. Dieser Infrastrukturcode musste früher ebenfalls selbst implementiert werden und wird nun vollständig durch die Client Library angeboten.

Diese Komponente ist auch wesentlich für die Vermeidung von Cross-Tenant Zugriffen (siehe Abschnitt 6.2.3) verantwortlich, da sie nur mit dem angegebenen Sharding Key assoziierte Daten zurückliefert.

```
using (SqlConnection conn = ShardMap.OpenConnectionForKey(
        shardingKey,
        credentialsConnectionString /* Credentials Only */ ,
        ConnectionOptions.Validate));
{
  using (SqlCommand cmd = new SqlCommand()
    {
    cmd.Connection = conn;
    cmd.CommandText = "SELECT Id, Name, Description, Location FROM
    Things";

```

```
11      SqlDataReader sdr = cmd.ExecuteReader();
12
13      // Daten abfragen
14    }
15 }
```

Listing 6.2: Der Verbindungsaufbau mit OpenConnectionForKey zum eigentlichen Shard.

In Listing 6.2 ist der Verbindungsaufbau zum eigentlichen Shard über die Shard Map dargestellt. Die Methode `OpenConnectionForKey` in Zeile 1 stellt hierfür mithilfe des Sharding Keys die Verbindung mit dem gewünschten Shard her.

Multi-Shard Query

Mit *Multi-Shard Querying* ermöglicht Elastic Scale, eine Abfrage unter Einbeziehung mehrerer Shards durchzuführen. Das SQL-Statement wird dabei ausgehend vom Shard Map Manager an alle Shards geschickt und von diesen ausgeführt. Die einzelnen Ergebnisse (Results) werden dann mittels `UNION ALL` in eine gesamte Ergebnismenge zusammengefügt und zurückgegeben. Multi-Shard Query ist ebenfalls ein Bestandteil der Client Library.

```
1 using (MultiShardConnection conn = new MultiShardConnection(myShardMap.
      GetShards(), credentialsConnectionString))
2 {
3   using (MultiShardCommand cmd = conn.CreateCommand())
4     {
5       cmd.CommandText = "SELECT Id, Name, Description, Location FROM
      Things";
6         cmd.CommandType = CommandType.Text;
7         cmd.ExecutionOptions = MultiShardExecutionOptions.
      IncludeShardNameColumn;
8         cmd.ExecutionPolicy = MultiShardExecutionPolicy.PartialResults;
9
10        using (MultiShardDataReader sdr = cmd.ExecuteReader())
11        {
12            while (sdr.Read())
13            {
14                var c1Field = sdr.GetString(0);
15                var c2Field = sdr.GetString(1);
16                var c3Field = sdr.GetString(2);
17                ...
18            }
19        }
20    }
21 }
```

Listing 6.3: Beispielhafte Abfrage unter Einbeziehung aller Shards mittels Multi-Shard Query.

Das Listing 6.3 zeigt, wie eine Multi-Shard Query mithilfe von `MultiShardConnection`, `MultiShardCommand` und `MultiShardDataReader` ausge-

führt werden kann. Zuerst wird eine `MultiShardConnection` geöffnet unter Einbeziehung aller Shards (Zeile 1). Anschließend wird ein `MultiShardCommand` erzeugt, der das Statement der Abfrage beinhaltet (Zeile 3). Zusätzlich kann mit der Option `MultiShardExecutionOptions.IncludeShardNameColumn` angegeben werden, dass das Ergebnis der Abfrage den Shard Namen beinhalten soll. Zuletzt wird der `MultiShardCommand` ausgeführt und das Ergebnis mittels `MultiShardDataReader` ausgelesen (Zeile 10).

Split-Merge Tool

Zusätzlich zu den genannten Funktionalitäten der Client Library gibt es noch das *Split-Merge Tool* [32] als Teil der Elastic Scale Management Services. Die flexible und bedarfsgerechte Verteilung der einzelnen Tenants auf die Shards erfordert das Hinzufügen bzw. Reduzieren von Shards zur Laufzeit. Diese Aufgabe erledigt der Administrator, indem er auf Basis der Nutzung weitere Shards hinzufügt oder entfernt. Das Split-Merge Tool führt diese Verteilung von Daten zur Laufzeit durch und ist damit das Management-Tool für die horizontale Skalierung. Es sei angemerkt, dass das Split-Merge Tool nur dann verwendet werden muss, wenn bereits bestehende Daten in den einzelnen Shards vorhanden sind. Hingegen erfordert die Anlage eines Shards und späterer Befüllung mittels Client Library keine Anwendung des Split-Merge Tools.

Das von Microsoft als customer-hosted Service zur Verfügung gestellte Cloud Service muss in Azure selbst installiert und betrieben werden. Nach erfolgreichem Deployment können dann Split bzw. Merge Operationen über das mitgelieferte Web User Interface oder über PowerShell[7] durchgeführt werden.

Nachfolgend werden die beiden Operationen Split und Merge erläutert. Für beide Operationen gilt, dass die zu transferierenden Datensätze während der Operation offline und für das Data Dependent Routing nicht zur Verfügung stehen [32].

Split Mittels *Split* werden Daten eines Shards, unter Angabe bei welchem Sharding Key gesplittet werden soll, aufgeteilt. Wie in Abbildung 6.11 ersichtlich, wird die DB5 beim Key 450 gesplittet. Alle Daten mit dem Key kleiner als 450 bleiben in dem alten Shard vorhanden. Die Tupel mit einem Sharding Key von größer gleich 450 werden in den neuen Shard transferiert. Split wird häufig dann durchgeführt, wenn ein Shard zuviel beansprucht wird und dies zu nicht mehr zufriedenstellenden Antwortzeiten führt.

Hinweis: Bevor diese Operation ausgeführt werden kann, muss der neue Shard bereits im Vorfeld inkl. leeren Tabellen (Schema) existieren. Dies kann wiederum mit der Elastic Scale Client Library erreicht werden.

[7] http://de.wikipedia.org/wiki/Windows_PowerShell

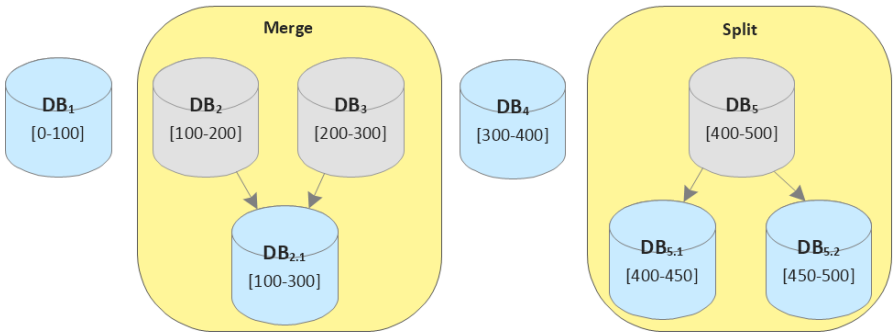


Abbildung 6.11: Schematische Darstellung der Split und Merge Operation.

Merge Im Gegensatz zu Split wird mit *Merge* genau das Gegenteil erreicht: Ist ein Shard zu wenig ausgelastet, wird dieser mit einem anderen, ebenfalls minder ausgelasteten Shard, wiedervereint. In Abbildung 6.11 werden hierfür die Daten von DB3 in DB2 zusammengeführt.

Hinweis: Die leer gewordene Datenbank DB3 wird nicht automatisch nach abgeschlossener Operation vom Split-Merge Tool gelöscht. Diese Aufgabe obliegt dem Administrator.

6.4.4 Nötige Anpassungen

Bevor Elastic Scale in WoTCloud eingesetzt werden kann, müssen Data und Business Layer gewissen Änderungen unterworfen werden. Nachfolgend werden diese Änderungen auszugsweise skizziert.

Data Layer

Zuerst muss ein *Sharding Key* (siehe 6.3.4) ausgewählt werden, der die horizontale Aufteilung der Daten fortan bestimmt. Der Autor entschied sich im Falle von WoTCloud für die `TenantId`, da diese auch die Umsetzung eines hybriden Modells ermöglichen würde. Unter einem hybriden Modell wird dabei die Möglichkeit verstanden, mehrere Tenants in einer gemeinsamen Datenbank zu verwalten, aber auch parallel Tenants in dezidierten Datenbanken zu halten. Da in unserem relationalen Schema alle Tabellen dem Sharding unterworfen sind (alle Tabellen sollen horizontal aufgeteilt werden), müssen diese zukünftig den Sharding Key beinhalten. Dies macht eine Erweiterung des relationalen Schemas der einzelnen Tabellen, wie beispielhaft für Tabelle `Sensors` in 6.4 dargestellt, notwendig.

Hierfür wird der zusammengesetzte Schlüssel `PK_Sensors` um den Sharding Key `TenantId` (Zeile 2) und um eine Fremdschlüsselbeziehung `FK_Sensors_TenantId` (Zeile 15) erweitert.

```
1 CREATE TABLE [Sensors](
```

```
2         [TenantId] [int] NOT NULL, --Sharding Key hinzugefügt
3         [ThingId] [int] NOT NULL,
4         [Id] [int] IDENTITY(1,1) NOT NULL,
5         [Name] [nvarchar](50) NOT NULL,
6     [DataType] [int] NOT NULL,
7     [CreateDate] [datetime] NOT NULL,
8     [ChangeDate] [datetime] NOT NULL,
9     [RowVersion] [timestamp] NOT NULL
10     CONSTRAINT [PK_Sensors] PRIMARY KEY CLUSTERED (
11        [TenantID] ASC, --Sharding Key auch Teil des Schlüssels
12        [ThingID] ASC,
13        [Id] ASC
14     ),
15     CONSTRAINT [FK_Sensors_TenantId] FOREIGN KEY (
16        [TenantId]
17     ) REFERENCES [Tenants]([TenantId]),
18     CONSTRAINT [FK_Sensors_ThingId] FOREIGN KEY (
19        [ThingId]
20     ) REFERENCES [Things]([ThingId])
21 )
```

Listing 6.4: Angepasstes Schema der Tabelle Sensors für die Verwendung mittels Elastic Scale.

Service Layer

In dem entwickelten Service Layer sind die Änderungen hingegen weitreichender und werden in diesem Abschnitt nur kurz angeschnitten. Zuerst müssen die Packages mittels NuGet zum Service Projekt hinzugefügt werden. Anschließend muss der Service Layer sukzessive anhand der obig beschriebenen Bestandteile (6.4.3) angepasst werden.

Im Zentrum steht dabei die SchemaInfo API, mit welcher die Shard Map die Strukturinformationen über die Tabellen erhält (siehe Listing 6.5). Es müssen dabei alle Tabellen, die mittels Sharding horizontal aufgeteilt werden sollen, registriert werden (Zeile 5 bis 7). Hierfür muss für jede Tabelle jeweils der zurvor eingeführte Sharding Key registriert werden.

```
1 // Create the schema annotations
2 SchemaInfo schemaInfo = new SchemaInfo();
3
4 // Sharded tables
5 schemaInfo.Add(new ShardedTableInfo("dbo", "Tenants", "TenantId"));
6 schemaInfo.Add(new ShardedTableInfo("dbo", "Things", "TenantId"));
7 schemaInfo.Add(new ShardedTableInfo("dbo", "Sensors", "TenantId"));
8 ...
9
10 // Publish
11 smm.GetSchemaInfoCollection().Add(Configuration.ShardMapName, schemaInfo
       );
```

Listing 6.5: Die SchemaInfo API der Elastic Scale Client Library zur Spezifikation der zu fragmentierenden Tabellen.

6.5 Zusammenfassung

Multi-Tenancy ermöglicht die kosteneffiziente und gemeinsame Nutzung von IT-Ressourcen und ist damit ein wesentlicher Erfolgsfaktor für viele Software-as-a-Service Anwendungen. Im Vergleich zu traditionelleren Ansätzen werden dabei Nutzer von ein und derselben Applikations- bzw. Datenbankinstanz bedient. Einerseits entstehen Skaleneffekte sowie Kosteneinsparungen andererseits entstehen aus der gemeinsamen Nutzung heraus aber auch zu lösende Herausforderungen.

Zur Umsetzung von Multi-Tenancy auf Ebene der Datenbank wurden drei mögliche Ansätze vorgestellt. Die gezeigten Ansätze weisen dabei jeweils eine unterschiedlich ausgeprägte Kombination von Isolation und Ressourcenteilung auf. Beim Shared Table Ansatz werden alle Tenants in denselben Tabellen gespeichert, was die effizienteste Ressourcenteilung zur Folge hat. Aufgrund der zu erzielenden Skaleneffekte wurde der Shared Table Ansatz in WoTCloud umgesetzt. Weiters wurde Sharding als horizontaler Fragmentierungsansatz für die Cloud vorgestellt, dass den Shared Table Ansatz vervollständigt bzw. skalieren lässt.

Zuletzt wurde die Unterstützung von Sharding in dem cloud-basierten relationalen Datenbanksystem Microsoft Azure SQL Database als Basis für eine skalierbare Datenhaltung für WoTCloud untersucht. Die hierfür von Microsoft entwickelte Komponente Elastic Scale setzt auf Azure SQL Database auf und ermöglicht eine bedarfsorientierte und flexible Skalierung des Shared Table Ansatzes. Elastic Scale führt alle Reorganisationsvorgänge (Split und Merge) während des Betriebs in der Produktivumgebung durch.

Durch die Anwendung von Elastic Scale kann der vorgestellte Shared Table Ansatz für WoTCloud bedarfsgerecht skaliert werden, sodass auch hybride Formen von Single-Tenancy und Multi-Tenancy ausgestaltet werden können. Aufgrund der Tatsache, dass die Infrastruktur von Microsoft Azure mittlerweile auf allen Kontinenten mit großen Kapazitäten zur Verfügung steht, ermöglicht eine weltweite Nutzung der Plattform WoTCloud.

Kapitel 7

Templates zur Wiederverwendung in WoTCloud

Wiederverwendung ist im Kontext moderner Softwareentwicklung eines der wesentlichen Konzepte zur Umsetzung und Bewältigung großer Entwicklungsvorhaben. Mittels Wiederverwendung werden vorhandene Komponenten wiederholt in Software eingesetzt, was den Time-to-Market sowie die Entwicklungskosten deutlich reduziert. Die Anwendung von Wiederverwendungstechniken (Klassen, Module, usw.) beschränkt sich dabei aber nicht nur auf die Entwicklung der Software selbst, sondern führt auch, bei Vorhandensein einer guten Usability, in der Bedienung der Software zu Produktivitätssteigerungen (Total Cost of Ownership[1]).

Auf Basis des Smart Vending Szenarios wurde eine Wiederverwendung mittels Templates für Web-of-Things-Plattformen zur vereinfachten Anlage von Automaten untersucht und auch umgesetzt.

Hierfür sollen zunächst die Begriffe Wiederverwendung und Templates näher erläutert werden, bevor auf die konkrete Umsetzung eingegangen wird.

7.1 Grundlagen der Wiederverwendung

7.1.1 Definition

Der Autor Cybulski [17] definiert *Wiederverwendung* im Kontext von Software wie folgt:

> „reuse is the use of previously acquired concepts or objects in a new situation, it involves encoding development information

[1] Total Cost of Ownership (TCO) betracht bei Investitionsentscheidungen nicht nur die Anschaffungskosten, sondern auch Kosten der späteren Nutzung.

> at different levels of abstraction, storing this representation for future reference, matching of new and old situations, duplication of already developed objects and actions, and their adaptation to suit new requirements.“ [17]
>
> „reusability is a measure of the ease with which one can use those previous concepts or objects in the new situations.“ [17]

Aus der Definition geht hervor, dass es sich um das angepasste Anwenden von bestehende Konzepten auf neue Problembereiche handelt. Die Wiederverwendbarkeit wird dabei als Maß definiert, mit welcher Einfachheit eine solche Übertragung bestehender Konzepte auf neue Situationen hin erfolgen kann.

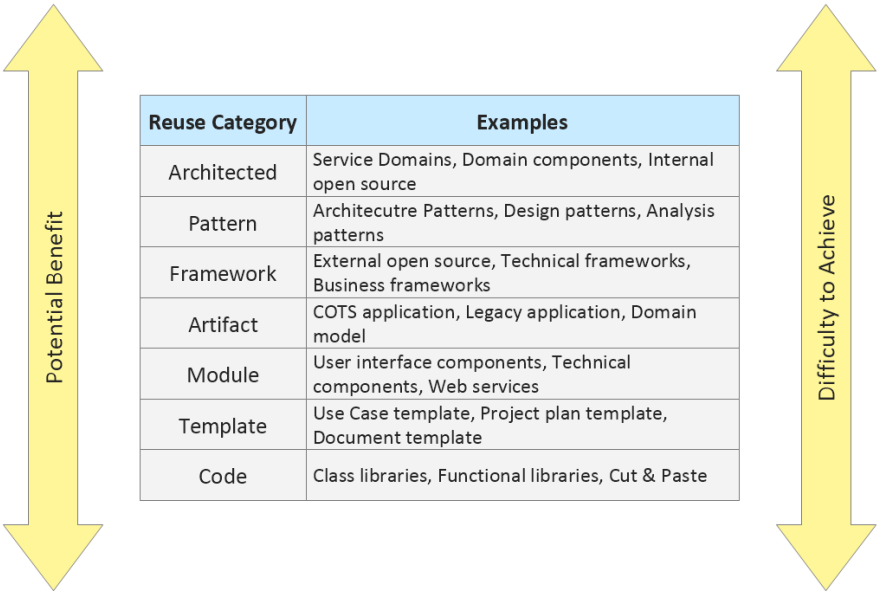
Dem Autor der vorliegenden Arbeit zufolge schränkt die obige Definition Wiederverwendung nicht auf den Kontext der Softwareentwicklung ein, sondern ermöglicht die Betrachtung von Wiederverwendungsmechanismen auch auf Ebene der Usability. Gerade im Bereich Human Computer Interaction gilt die Unterstützung des Menschen mithilfe von Informations- und Kommunikationsmitteln als oberste Prämisse. Eine solche Unterstützung kann die Eleminierung von repetitiven und unproduktiven Tätigkeiten bei Anwendung der Software darstellen.

7.1.2 Arten der Wiederverwendung

Zum besseren Verständnis der einzelnen Wiederverwendungsarten bzw. -techniken, entwickelte der Autor Scott Ambler [1] mehrere Kategorien der Wiederverwendung (siehe Abbildung 7.1). Ambler argumentiert, dass beginnend bei der Kategorie „Code“ die potenziellen Vorteile bis hin zu „Architected“ sukzessive ansteigen. Im selben Ausmaß steigt jedoch auch die Schwierigkeit, mit der ein solcher Grad an Wiederverwendung erreicht wird. So ist eine Wiederverwendung von Code und Templates relativ einfach möglich, wohingegen die Wiederverwendung mit Frameworks und Patterns aufgrund des fortgeschrittenen Wissensstandes aufwendiger ist. Nachfolgend werden die einzelnen Typen der Wiederverwendung kurz erläutert.

Typen der Wiederverwendung nach Ambler [1]:

- *Architected:* Unter Architected Reuse versteht Ambler die Wiederverwendung von gesamten Enterprise Architekturen. Die Wiederverwendung von Domainkomponenten bzw. zusammenhängenden Services stehen dabei im Vordergrund.
- *Pattern:* Mittels Pattern werden dokumentierte Problemlösungsansätze auf spezielle Anwendungsfälle übertragen und angewandt. Es wird dabei nicht nur der Quellcode wiederverwendet, sondern vielmehr die eigentliche Idee, die im Code manifestiert ist. Patterns können dabei in unterschiedlichen Phasen der Entwicklung eingesetzt werden (Analyse, Design und Architektur).



Reuse Category	Examples
Architected	Service Domains, Domain components, Internal open source
Pattern	Architecutre Patterns, Design patterns, Analysis patterns
Framework	External open source, Technical frameworks, Business frameworks
Artifact	COTS application, Legacy application, Domain model
Module	User interface components, Technical components, Web services
Template	Use Case template, Project plan template, Document template
Code	Class libraries, Functional libraries, Cut & Paste

Abbildung 7.1: Arten der Wiederverwendung in der Informations- und Kommunikationstechnologie (angelehnt an [1]).

- *Framework:* Ein Framework implementiert Basisfunktionalitäten durch eine Ansammlung von zusammenhängenden Klassen zur weiteren Verwendung. Als Beispiel kann hier das Microsoft .NET Framework genannt werden, welches sich wiederum aus einer Vielzahl an Frameworks (ASP.NET, WCF, WPF, usw.) zusammensetzt.
- *Artifact:* Die Wiederverwendung von zuvor erstellten Artefakten - wie Use Cases, Standards, domänenspezifische Modelle, Prozeduren, Guidelines oder sogar andere Anwendungen - können dazu verwendet werden, ein neues Entwicklungsvorhaben zu starten. Manchmal werden die Artefakte unverändert übernommen oder aber auch nur als Beispiel für die weitere Entwicklung herangezogen.
- *Module:* Module Reuse bezeichnet die Verwendung von vollständig gekapselten Modulen (Komponenten, Services oder Bibliotheken) in der Entwicklung der eigenen Software. Im Gegensatz zur Wiederverwendung von Code, besteht bei Module Reuse kein Zugriff auf den Quellcode.
- *Template:* Die Anwendung eines vordefinierten Layouts für essenzielle Entwicklungsartefakte wie Dokumente, Modelle und Code wird als Template Reuse bezeichnet.
- *Code:* Die Wiederverwendung von Quellcode ermöglicht das Wiederverwenden von Code in Anwendungen. Im besten Fall wird der Co-

> de im Sinne von Klassen, Funktionen und Prozeduren in einer neuen Anwendung wiederverwendet. Im schlimmsten Fall wird Code durch Copy & Paste und anschließender Adaptierung übernommen, was sich negativ auf die Wartbarkeit auswirken kann.

Die Wiederverwendung von Code mittels Copy & Paste ist eine beliebte Praxis in der Entwicklung. Die Autoren Kim et al. [35] haben hierzu eine Studie durchgeführt, um gängige Copy & Paste Praktiken besser verstehen zu können. Zukünftig sollen Entwicklungsumgebungen den Entwickler bei diesen C&P Operationen unterstützen, sowie auftretende Wartbarkeitsprobleme verhindern. Kim et al. kommen dabei zu folgendem Schluss:

> „Programmers copy an entire code snippet because it contains the structural template that they intend to reuse.“ [35]

Der Autor der vorliegenden Arbeit ist der Auffassung, dass die Schlussfolgerung von Kim et al. [35] nicht nur auf Entwickler zutrifft, sondern auch auf Anwender einer Software.

Bevor nun aus den vorgestellten Arten der Wiederverwendung Templates näher untersucht werden, werden die Vorteile durch den Einsatz von Wiederverwendung beleuchtet.

7.1.3 Vorteile durch Wiederverwendung

Ergänzend zu den vorgestellten Arten werden die Vorteile durch Wiederverwendung hinsichtlich Kosten, Produktivität, Wartbarkeit und Zuverlässigkeit betrachtet.

Kosten

Der Einsatz von vorgefertigten Komponenten führt in mehreren Softwareprodukten zu Skaleneffekte in der Entwicklung. Die zunächst höheren Entwicklungskosten für die wiederzuverwendenden Komponenten werden auf eine Vielzahl von Projekten aufgeteilt [17]. Auch durch Zukauf von Komponenten oder Bibliotheken können teure Entwicklerressourcen eingespart werden (make-or-buy).

Produktivität

Die Tatsache, dass Anwendungen wiederkehrende Funktionalitäten wie z. B. eine Benutzerverwaltung beinhalten, lässt den Schluss zu, dass durch Wiederverwendung in diesen Bereichen die Produktivität wesentlich gesteigert werden kann. Cybulski [17] zufolge führt die Entwicklung auf einem höheren Abstraktionslevel (bspw. durch Komposition von Frameworks und Services) zu einer höheren Produktivität als die Konzentration auf Implementierungsdetails.

Wartbarkeit

Systeme, die durch Komposition von bestehenden Teilen entstehen, sind für gewöhnlich auf einem höheren Abstraktionsniveau angesiedelt [17]. Dies wirkt sich positiv auf die Wartbarkeit solcher Systeme aus, da sich die Architektur sehr stark auf den Problembereich bezieht.

Zuverlässigkeit

Durch Wiederverwendung von bestehenden Frameworks und Bibliotheken wird auch die Zuverlässigkeit der neu entwickelten Anwendung gesteigert [17]. Dieses Argument stützt sich laut Cybulski [17] auf der Annahme, dass die zu verwendenden Artefakte bereits einen gewissen Software Development Life Cycle durchlaufen haben und somit bereits qualitätsgesichert wurden.

7.2 Grundlagen von Templates

Im Rahmen dieser Arbeit wurde ein Wiederverwendungsmechanismus für WoTCloud auf Basis von Templates entwickelt. Bevor auf die konkrete Umsetzung eingegangen wird, werden Templates näher definiert bzw. betrachtet.

> „Template reuse is typically a form of documentation reuse. It refers to the practice of using a common set of layouts for key development artifacts documents, models, and source code-within your organization.“ [20]

Cybulski & Linden [16] definieren ein Template folgendermaßen:

> „A need exists to produce a collection of composite artifacts similar in structure and contents.“ [16]

Die Autoren Nanard et al. [42] definieren weiters das verwandte Constructive Template:

> „It is a generic specification which makes it easier for the developer to build up actual hypermedia structure and populate it with its data.“ [42]

Den Autoren German & Cowan [23] zufolge repräsentieren beide Ansätze das Gleiche. In Folge wird daher ausschließlich der Begriff Template verwendet.

Ein Template liefert nach Cybulski & Linden [16] eine Lösung für die Produktion zusammengesetzter und gleichartiger Artefakte hinsichtlich Struktur und Inhalt. Die Autoren gehen dabei noch weiter und definieren Vor- und Nachbedingungen sowie Handlungsempfehlungen für den Einsatz von Templates.

Vorbedingung

Cybulski & Linden [16] führen als einzige Vorbedingung die Existenz von gleichartigen und zusammengesetzten Artefakten an.

Nachbedingung

Ein resultierendes Artefakt repräsentiert die Generalisierung der gleichartigen und zusammengesetzten Artefakte [16]. Dieses wird anschließend verwendet, um konkrete Ausprägungen (Instanzen) zu erzeugen.

Einsatz

Templates werden normalerweise in der Verarbeitung von Text und Bildern angewandt. Sie werden meist dazu verwendet die Struktur von Dokumenten (Outlines) oder Präsentationen vorzugeben. Auch bestehende effektive Dokumente bzw. Präsentationen können schnell in eine wiederverwendbare Vorlage umgewandelt werden und fortan verwendet werden.

Nachfolgend werden drei Nutzungsszenarien [16] für den Einsatz von Templates aufgelistet:

- Sobald eine Ansammlung von gleichartigen Objekten zum Zwecke der Wiederverwendung generalisiert wird.
- Zur Erstellung eines Dokumentenstandards, der für eine Vielzahl an gleichartigen Dokumente bereitgestellt wird.
- Zur Unterstützung beim Ausfüllen von Formularen unter Verwendung von Dokumenten oder Komponenten.

Vorteile

Cybulski & Linden [16] nennen folgende Vorteile, die aus dem Einsatz von Templates resultieren:

- Aus einem Template erzeugte Instanzen sind sowohl strukturell, inhaltlich als auch visuell konsistent.
- Ein Template generalisiert gleichartige Artefakte.
- Ein Template gruppiert wiederverwendbare Artefakte und fasst diese an einer zentralen Stelle zur weiteren Bearbeitung zusammen.

7.3 Umsetzung in WoTCloud

Abgesehen von der Wiederverwendung von Code-Snippets und Frameworks während der Entwicklung der Web-of-Things-Plattform wurde ein templatebasierter Wiederverwendungsmechanismus für die Anlage von komplexen Dingen implementiert. Die Tatsache, dass Betreiber oft eine Vielzahl an Automaten desselben Typs betreiben, lässt den Wunsch einer Generalisierung

zur simplen Anlage aufkommen. Folgender Abschnitt ist daher der technischen Umsetzung in Data und Presentation Layer gewidmet. Der Business Layer kann dabei vernachlässigt werden, da dieser das Anlegen von Things sowie Templates annähernd gleichbehandelt.

7.3.1 Ziel & Anforderungen

Zum Zwecke der Vollständigkeit werden nochmals die Ziele und die daraus abgeleiteten Anforderungen angeführt.

Im entwickelten Smart Vending Szenario (Abschnitt 3.4) wurde bezüglich Wiederverwendung nachfolgendes definiert.

Einfaches Anlegen von Automaten unter Wiederverwendung Zur einfacheren Verwaltung der gesamten Automatenflotte sollen die einzelnen Automatentypen (z. B. Kaltegetränkeautomat Sielaff Robimat 75 mit allen Schächten bzw. Sensoren) als Templates global definiert werden können. Die Neuanlage eines konkreten Automats kann dann auf Basis dieser definierten Vorlagen erfolgen. Die Vermeidung dieser repetitiven Tätigkeiten führt zur effizienteren Anlage von Automaten.

Auf Basis dieser Zielvorgabe wurden in Abschnitt 5.2 konkrete Anforderungen hinsichtlich Templates definiert:

A12 - Template anlegen Die Plattform muss einem angemeldeten Operator die Möglichkeit bieten private sowie öffentliche Vorlagen erstellen zu können. Die benötigten Felder bzw. Funktionen ergeben sich analog aus Anforderung A04.

Bedeutung: Templates sollen dazu verwendet werden komplexe Dinge wie Automaten mit unterschiedlichen Schächten und Sensoren (z.B. Sielaff Robimat XL) vordefinieren zu können, um diese anschließend mehrfach wiederverwenden zu können. Wird eine Vorlage als öffentlich definiert, können andere Operatoren diese Vorlage ebenfalls wiederverwenden.

A13 - Templateübersicht Die Plattform muss einem angemeldeten Operator die Möglichkeit bieten private sowie öffentliche Vorlagen übersichtlich anzeigen zu können.

A14 - Template löschen Die Plattform muss einem angemeldeten Operator die Möglichkeit bieten private sowie öffentliche Vorlagen löschen zu können.

Hinweis: Aus Gründen der Einfachheit kann ein Operator seine eigenen (privaten) als auch öffentliche Vorlagen jederzeit löschen.

7.3.2 Data Layer

Das in Abschnitt 5.3 gezeichnete *konzeptuelle Datenmodell* von WoTCloud zeigt auf abstrakter Ebene die Zusammenhänge eines `ThingTemplate` und eines konkreten `Thing`. Die Realisierungsabhängigkeit deutet dabei darauf hin, dass ein `Thing` ein `ThingTemplate` zur Realisierung heranziehen kann. Gleiches gilt für `Sensor`, `Actuator` und `Rule` mit ihrer zugehörigen Template Klasse.

Im *logischen Datenmodell* stellt diese Realisierungsabhängigkeit jedoch keine Assoziation dar, wie in Listing 7.1 ersichtlich. Die Plattform WoTCloud speichert Templates und Things in ein und derselben Tabelle, wobei eine Unterscheidung anhand des Diskriminatorattributs `IsTemplate` (Zeile 7) erfolgt. Da für Templates alle Felder außer `Location` von Bedeutung sind, eignet sich eine Speicherung beider Typen in derselben Tabelle. Eine Angabe der `Location` ist bei Anlage eines Templates im Webclient nicht möglich.

Weiters ist anzumerken, dass im logischen Entwurf öffentliche Templates ebenfalls einem speziellen Tenant zugeordnet sind. Dieser Umstand ergibt sich aus der Tatsache, dass kein Thing ohne zugehörigen Tenant existieren darf. Daher wurde das Flag `IsPublic` (Zeile 8) vom Typ Boolean eingeführt, um dennoch alle öffentlichen Templates zurückzubekommen. Der logische Entwurf der restlichen Entitäten befindet sich als SQL-DDL im Anhang A.

```
1 CREATE TABLE [dbo].[Things](
2   [Id] [int] IDENTITY(1,1) NOT NULL,
3   [TenantId] [int] NOT NULL,
4   [Name] [nvarchar](50) NOT NULL,
5   [Description] [nvarchar](200) NULL,
6   [Location] [geography] NULL,
7   [IsTemplate] [bit] NOT NULL, --Diskriminator
8   [IsPublic] [bit] NOT NULL,
9   [RowVersion] [timestamp] NOT NULL,
10  [CreateDate] [datetime] NOT NULL,
11  [ChangeDate] [datetime] NOT NULL,
12  [ParentId] [int] NULL,
13  [IsComplex] [bit] NOT NULL,
14  CONSTRAINT [PK_dbo.Things] PRIMARY KEY CLUSTERED
15 (
16   [Id] ASC
17 ))
```

Listing 7.1: Der logische Entwurf der Entität Things bzw. Templates in SQL-DDL.

Die Einführung von Templates hat auf die an der Komposition von Thing beteiligten Klassen (Sensor, Actuator, Rule) keine Auswirkung. Die fehlende Assoziation zwischen Template und Thing ermöglicht es auch konkrete Instanzen eines Things abzuändern, ohne dass sich das Template verändert.

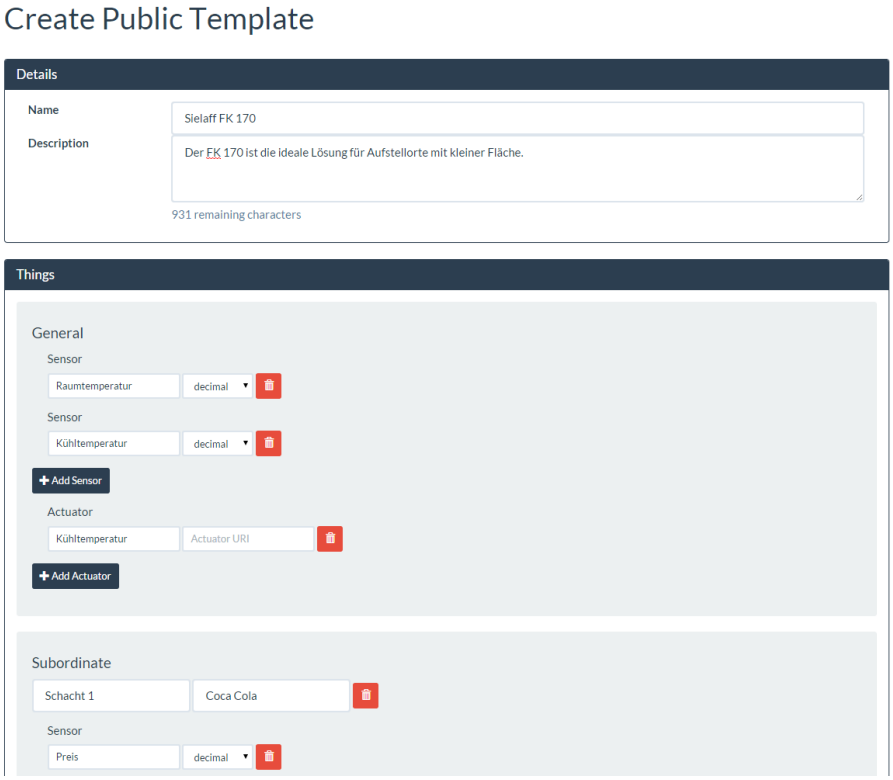


Abbildung 7.2: Anlage eines öffentlichen Templates in *WoTCloud*.

7.3.3 Presentation Layer

Die Umsetzung der Templates für komplexe Dinge im Webclient wird nachfolgend mithilfe von Screenshots dargestellt.

Um Templates wiederverwenden zu können, wurde zuerst Anforderung A12 umgesetzt, die eine Anlage von privaten sowie öffentlichen Templates erlaubt. In Abbildung 7.2 wird beispielhaft ein öffentliches Template für einen Sielaff Kaltgetränkeautomat FK 170 mit Sensoren und Schächten in WoTCloud angelegt.

Nach der Spezifizierung von Name, Beschreibung, Schächten, Sensoren und Aktuatoren werden auf einer weiteren Seite die Geschäftsregeln für den Automatentyp FK 170 angegeben. Als Beispiel wird in Abbildung 7.3 eine Regel erstellt, die bei einer nicht ausreichenden Kühltemperatur eine E-Mail an den Betreiber versendet.

Nach erfolgreicher Anlage des Templates gelangt man zur Übersicht (Abbildung 7.4), wo die privaten Templates eines Tenants, sowie die öffentlichen (tenant-übergreifenden) Templates dargestellt werden. Bei Erstellung eines öffentlichen Templates werden diese umgehend auch für alle anderen Tenants ersichtlich und verwendbar. Der Einfachheit halber kann jeder Tenant

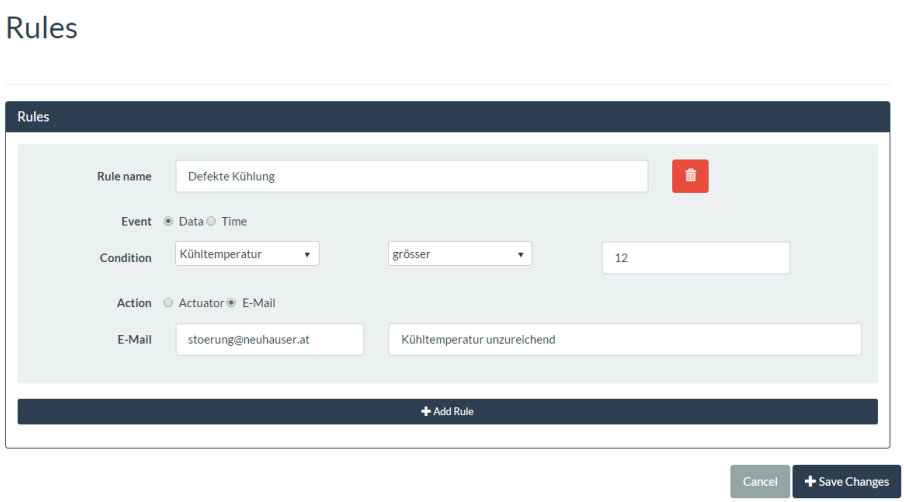


Abbildung 7.3: Hinzufügen einer Rule zu einem Template in *WoTCloud*.

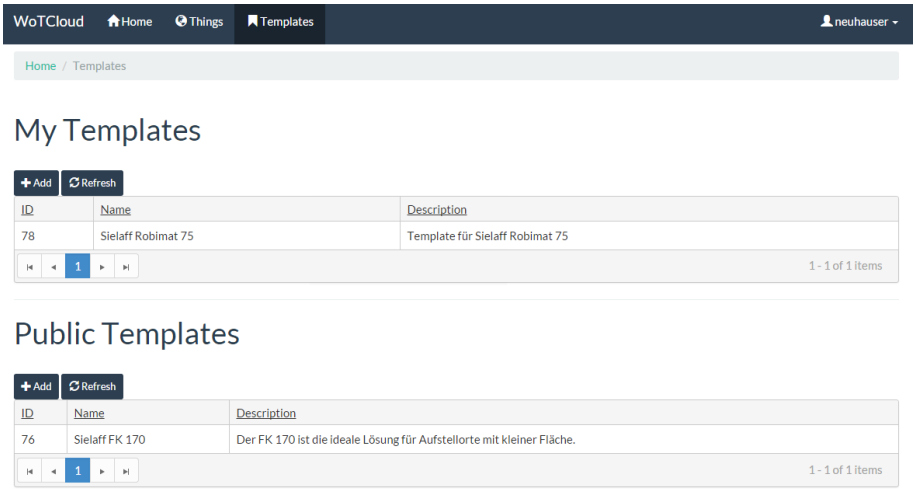


Abbildung 7.4: Übersicht aller privaten und öffentlichen Templates in *WoTCloud*.

öffentlich erstellte Templates löschen.

Zuletzt wurde die Funktionalität zur Erstellung eines konkreten Automaten auf Basis eines Templates realisiert. In Abbildung 7.5 findet sich hierfür oben die DropDownList zur Auswahl eines privaten oder öffentlichen Templates. Bei Übernahme mittels Apply wird das Template mit allen definierten Artefakten vom Service Layer geladen. Über die Zwei-Wege-Datenbindung in AngularJS wird anschließend das Webformular zum Anlegen eines Thing vorausgefüllt. Es obliegt nun dem Nutzer die Details bei der konkreten Er-

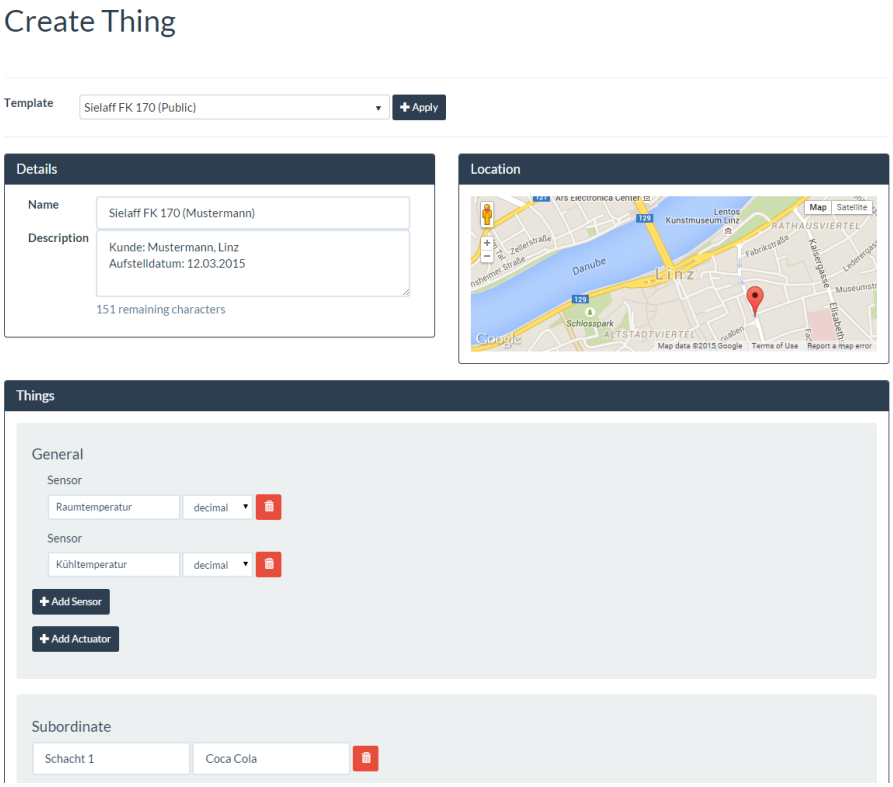


Abbildung 7.5: Anlage eines konkreten Automaten aus einem Template in *WoTCloud*.

stellung abzuändern.

7.4 Zusammenfassung

In diesem Kapitel wurde Wiederverwendung im Kontext von Software diskutiert. Es wurden dabei verschiedene Arten der Wiederverwendung in der Informations- und Kommunikationstechnologie, sowie Vorteile hinsichtlich Kosten, Produktivität, Wartbarkeit und Zuverlässigkeit betrachtet. Wiederverwendung wirkt sich nicht nur während der Entwicklung maßgeblich auf die Produktivität aus. Auch beim Endbenutzer kann, bei Vorliegen von Wiederverwendungsmechanismen in der Software, die Produktivität, durch Eliminierung von sich wiederholenden Tätigkeiten, gesteigert werden.

Aus diesem Grund wurde für WoTCloud ein template-basierter Wiederverwendungsmechanismus geschaffen, zur einfachen Anlage von vordefinierten und komplexen Things. Hierfür erfolgte zuerst eine Einleitung über Templates gefolgt von der technischen Umsetzung von Templates in Data und Presentation Layer von WoTCloud.

Kapitel 8

Fazit und Ausblick

Ziel der vorliegenden Arbeit ist es, aufgrund der zunehmenden Bedeutung von Web-of-Things, die Integration von komplexen Dingen in einer Web-of-Things-Plattform anhand eines Getränkeautomaten aufzuzeigen. Hierfür wurde von den Autoren ein Smart Vending Szenario entwickelt, mithilfe dessen eine mögliche Integration untersucht wurde. Durch Gespräche mit Marktteilnehmern konnten Bedürfnisse und Probleme bisheriger Remotetelemetrielösungen von Automaten identifiziert werden. Anschließend wurden vier aktuelle Web-of-Things-Plattformen (Evrythng, ThingSpeak, Paraimpu und WoTKit) hinsichtlich der Unterstützung des entwickelten Szenarios anhand von zuvor abgeleiteten Kriterien evaluiert. Dabei ergab sich, dass diese für das entwickelte Smart Vending Szenario aufgrund der fehlenden Unterstützung von komplexen Dingen, Geschäftsregeln und Templates nicht geeignet sind. Aus diesem Grund wurde eine eigene, prototypische Web-of-Things-Plattform namens WoTCloud entwickelt, die das entwickelte Szenario unterstützen sollte. Was die Abbildung komplexer Dinge betrifft, wurde im konzeptuellen und logischen Entwurf die Möglichkeit geschaffen, Dinge beliebig zu strukturieren.

Die von den Autoren entwickelte SaaS-Plattform *WoTCloud* wurde dabei vollständig für die Cloud konzipiert und implementiert. Um zugrunde liegende IT-Ressourcen in der Cloud aus Sicht von WoTCloud kosteneffizient nutzen zu können, wurden drei mögliche Umsetzungsvarianten von *Multi-Tenancy* diskutiert. Multi-Tenancy erzielt durch die gemeinsame Nutzung von Applikations- und Datenbankinstanz Skaleneffekte bzw. Kosteneinsparungen, indem sich mehrere Tenants ein und diesselbe Infrastruktur teilen. Die gezeigten Ansätze weisen dabei jeweils eine unterschiedlich ausgeprägte Kombination von Isolation und Ressourcenteilung auf. Es konnte festgestellt werden, dass Ressourcenteilung immer zulasten der Isolation erreicht wird. Der Shared Table Ansatz geht dabei so weit, dass sämtliche Tenants in denselben Tabellen gespeichert werden, was zwar die Ausnutzung von Skaleneffekten begünstigt, jedoch eine Trennung der Daten auf einer höheren

Anwendungsschicht bedingt. Aufgrund der Eignung wurde dieser Ansatz in WoTCloud implementiert.

Eine Umsetzung dieses Ansatzes mit der Möglichkeit der Skalierung kann mithilfe des Entwurfsmuster *Sharding* erfolgen, das den Shared Table Ansatz komplettiert und skalieren lässt. Mittels Sharding werden Tabellen anhand eines Sharding Keys horizontal fragmentiert und auf weitere Rechnerknoten verteilt, um so auftretende Engpässe bei hoher Transaktionslast zu verhindern.

Anschließend wurde die Unterstützung von Sharding in *Elastic Scale* für Microsoft Azure SQL Database untersucht. Es wurden dabei nötige Anpassungen im Data und Business Layer auszugsweise dargestellt. Auf eine vollständige Implementierung in WoTCloud wurde dabei aufgrund der noch unzureichenden Unterstützung des in WoTCloud verwendeten objektrelationalen Mappers Entity Framework verzichtet. Mithilfe von Elastic Scale ist es möglich den Data Layer von WoTCloud flexibel bei Bedarf zu skalieren. Auch hybride Formen von Single- und Multi-Tenancy können über die Range und List Mappings von Elastic Scale realisiert werden, wodurch zukünftige Geschäftsmodelle von WoTCloud wesentlich unterstützt werden. Die globale Nutzung von WoTCloud auf Basis der Microsoft Azure Infrastruktur wird ebenfalls durch die Möglichkeit der Angabe des Shard Speicherorts begünstigt. Daten werden damit möglichst in der Nähe des Kunden gespeichert, was wiederum regulatorische Anforderungen der Kunden hinsichtlich Speicherort berücksichtigt.

Weiters wurde im Smart Vending Szenarios eine Wiederverwendung mittels *Templates* vorgesehen und umgesetzt, um eine Neuanlage auf Basis eines vorher definierten Automatentyps zu unterstützen. Die Vorlage erlaubt dabei die Spezifikation von Sensoren, Aktuatoren sowie Geschäftsregeln und kann bei der Neuanlage als Referenz ausgewählt werden. Unter der Annahme, dass ein Automatenbetreiber, wie von den Marktteilnehmern geschildert, mehrere Hundert Automaten in seiner Flotte besitzt, kann die Produktivität des Betreibers mit diesem Wiederverwendungsmechanismus drastisch gesteigert werden.

Abschließend sei erwähnt, dass auch weitere Anwendungsszenarien mit WoTCloud aufgrund der erwähnten Generizität umgesetzt werden könnten, sofern Anpassungen an der Plattform durchgeführt werden.

Anhang A

Logischer Entwurf in SQL-DDL

Nachstehend findet sich der logische Entwurf sämtlicher Entitäten von WoT-Cloud in SQL-DDL. Die Felder `CreateDate` und `ChangeDate` sind aus Gründen der Nachvollziehbarkeit der Tupel in jeder Entität vorhanden. Die `RowVersion` hingegen wird von Microsoft's Entity Framework für die Lösung von Konflikten bei gleichzeitiger Änderung von Tupeln (engl. Concurrency) benötigt. Der logische Entwurf wurde von dem in Abschnitt 5.3 entwickelten konzeptuellen Entwurf abgeleitet.

A.1 Table Tenants

```
1 CREATE TABLE [dbo].[Tenants](
2   [Id] [int] IDENTITY(1,1) NOT NULL,
3   [Name] [nvarchar](50) NOT NULL,
4   [IsActive] [bit] NOT NULL,
5   [UserName] [nvarchar](50) NOT NULL,
6   [Password] [nvarchar](50) NOT NULL,
7   [RowVersion] [timestamp] NOT NULL,
8   [CreateDate] [datetime] NOT NULL,
9   [ChangeDate] [datetime] NOT NULL,
10  CONSTRAINT [PK_dbo.Tenants] PRIMARY KEY CLUSTERED
11 (
12   [Id] ASC
13 ))
```

A.2 Table Things

```
1 CREATE TABLE [dbo].[Things](
2   [Id] [int] IDENTITY(1,1) NOT NULL,
3   [TenantId] [int] NOT NULL,
4   [Name] [nvarchar](50) NOT NULL,
5   [Description] [nvarchar](200) NULL,
```

```
6   [Location] [geography] NULL,
7   [IsTemplate] [bit] NOT NULL,
8   [IsPublic] [bit] NOT NULL,
9   [RowVersion] [timestamp] NOT NULL,
10  [CreateDate] [datetime] NOT NULL,
11  [ChangeDate] [datetime] NOT NULL,
12  [ParentId] [int] NULL,
13  [IsComplex] [bit] NOT NULL,
14 CONSTRAINT [PK_dbo.Things] PRIMARY KEY CLUSTERED
15 (
16  [Id] ASC
17 ))
18
19 ALTER TABLE [dbo].[Things]  WITH CHECK ADD  CONSTRAINT [FK_dbo.
       Things_dbo.Tenants_TenantId] FOREIGN KEY([TenantId])
20 REFERENCES [dbo].[Tenants] ([Id])
21 ON DELETE CASCADE
22
23 ALTER TABLE [dbo].[Things] CHECK CONSTRAINT [FK_dbo.Things_dbo.
       Tenants_TenantId]
24
25 ALTER TABLE [dbo].[Things]  WITH CHECK ADD  CONSTRAINT [FK_dbo.
       Things_dbo.Things_ParentId] FOREIGN KEY([ParentId])
26 REFERENCES [dbo].[Things] ([Id])
27
28 ALTER TABLE [dbo].[Things] CHECK CONSTRAINT [FK_dbo.Things_dbo.
       Things_ParentId]
```

A.3 Table Sensors

```
1 CREATE TABLE [dbo].[Sensors](
2   [Id] [int] IDENTITY(1,1) NOT NULL,
3   [ThingId] [int] NOT NULL,
4   [Name] [nvarchar](50) NOT NULL,
5   [DataType] [int] NOT NULL,
6   [RowVersion] [timestamp] NOT NULL,
7   [CreateDate] [datetime] NOT NULL,
8   [ChangeDate] [datetime] NOT NULL,
9  CONSTRAINT [PK_dbo.Sensors] PRIMARY KEY CLUSTERED
10 (
11  [Id] ASC
12 ))
13
14 ALTER TABLE [dbo].[Sensors]  WITH CHECK ADD  CONSTRAINT [FK_dbo.
       Sensors_dbo.Things_ThingId] FOREIGN KEY([ThingId])
15 REFERENCES [dbo].[Things] ([Id])
16 ON DELETE CASCADE
17
18 ALTER TABLE [dbo].[Sensors] CHECK CONSTRAINT [FK_dbo.Sensors_dbo.
       Things_ThingId]
```

A.4 Table Actuators

```
CREATE TABLE [dbo].[Actuators](
  [Id] [int] IDENTITY(1,1) NOT NULL,
  [ThingId] [int] NOT NULL,
  [Name] [nvarchar](50) NOT NULL,
  [ActuatorUri] [nvarchar](255) NOT NULL,
  [RowVersion] [timestamp] NOT NULL,
  [CreateDate] [datetime] NOT NULL,
  [ChangeDate] [datetime] NOT NULL,
 CONSTRAINT [PK_dbo.Actuators] PRIMARY KEY CLUSTERED
(
  [Id] ASC
))

ALTER TABLE [dbo].[Actuators]  WITH CHECK ADD  CONSTRAINT [FK_dbo.
    Actuators_dbo.Things_ThingId] FOREIGN KEY([ThingId])
REFERENCES [dbo].[Things] ([Id])
ON DELETE CASCADE

ALTER TABLE [dbo].[Actuators] CHECK CONSTRAINT [FK_dbo.Actuators_dbo.
    Things_ThingId]
```

A.5 Table Rules

```
CREATE TABLE [dbo].[Rules](
  [Id] [int] IDENTITY(1,1) NOT NULL,
  [ThingId] [int] NOT NULL,
  [SensorId] [int] NOT NULL,
  [ActuatorId] [int] NULL,
  [Name] [nvarchar](50) NULL,
  [IsActive] [bit] NOT NULL,
  [IsTemplate] [bit] NOT NULL,
  [EventType] [int] NOT NULL,
  [EventTime] [datetime] NULL,
  [ConditionOperator] [int] NOT NULL,
  [ConditionValue] [nvarchar](25) NULL,
  [ActionType] [int] NOT NULL,
  [ActionValue] [nvarchar](200) NULL,
  [ActionEmailRecipient] [nvarchar](max) NULL,
  [CreateDate] [datetime] NOT NULL,
  [ChangeDate] [datetime] NOT NULL,
 CONSTRAINT [PK_dbo.Rules] PRIMARY KEY CLUSTERED
(
  [Id] ASC
))

ALTER TABLE [dbo].[Rules]  WITH CHECK ADD  CONSTRAINT [FK_dbo.Rules_dbo.
    Actuators_ActuatorId] FOREIGN KEY([ActuatorId])
REFERENCES [dbo].[Actuators] ([Id])

ALTER TABLE [dbo].[Rules] CHECK CONSTRAINT [FK_dbo.Rules_dbo.
    Actuators_ActuatorId]

```

```
28 ALTER TABLE [dbo].[Rules]  WITH CHECK ADD  CONSTRAINT [FK_dbo.Rules_dbo.
       Sensors_SensorId] FOREIGN KEY([SensorId])
29 REFERENCES [dbo].[Sensors] ([Id])
30 ON DELETE CASCADE
31
32 ALTER TABLE [dbo].[Rules] CHECK CONSTRAINT [FK_dbo.Rules_dbo.
       Sensors_SensorId]
33
34 ALTER TABLE [dbo].[Rules]  WITH CHECK ADD  CONSTRAINT [FK_dbo.Rules_dbo.
       Things_ThingId] FOREIGN KEY([ThingId])
35 REFERENCES [dbo].[Things] ([Id])
36
37 ALTER TABLE [dbo].[Rules] CHECK CONSTRAINT [FK_dbo.Rules_dbo.
       Things_ThingId]
```

Anhang B

Inhalt der CD-ROM

Format: CD-ROM, Single Layer, ISO9660-Format

B.1 WoTCloud.Client

In diesem Ordner befinden sich sämtliche Quellcodedateien und Bibliotheken die für die Erstellung des Webclients verwendet wurden.

Deployment

Der Webclient ist vollständig in HTML & Javascript bzw. AngularJS implementiert und läuft somit auf allen gängigen Webservern (z. B. Microsoft Internet Information Services (IIS), Apache Tomcat, etc.). Das Deployment kann in diesem Fall durch einen simplen Kopiervorgang der Dateien (beispielsweise über FTP) erfolgen.

B.2 WoTCloud.Service

In diesem Ordern befinden sich sämtliche Quellcodedateien des Service- und Data Layers von WoTCloud. Die REST-basierten Webservices wurden mithilfe von Microsoft's ASP.NET Web API erstellt. Zusätzlich finden sich im Unterordner *Scripts* alle SQL-Scripts der benötigten Entitäten. Der Data Layer basiert auf Microsoft's Entity Framework und nützt den *Code-First* Ansatz. Aus diesem Grund ist es nicht notwendig, die SQL-Skripts auf dem Datenbankserver auszuführen, da die Tabellen und Constraints beim ersten Start der Applikation automatisch erzeugt werden.

Deployment

Der Service Layer erfordert Microsoft IIS als Webserver. Alternativ dazu ist auch ein Deployment als Web App auf Windows Azure möglich. Das

Deployment kann direkt aus der Entwicklungsumgebung Visual Studio von Microsoft erfolgen (Publish to Azure bzw. Publish Web Site). Der Data Layer erfordert einen MS-SQL Server bzw. eine (cloud-basierte) SQL Azure Database von Microsoft Azure. Vor dem Start der Applikation ist der entsprechende `ConnectionString` zur Datenbank in der Datei *web.config* anzupassen.

B.3 WoTCloud.Simulator

In diesem Ordner befinden sich sämtliche Quellcodedateien des erstellten Simulators für Getränkeautomaten. Der Simulator basiert auf Microsoft's ASP.NET MVC.

Deployment

Der Simulator erfordert Microsoft IIS als Webserver. Das Deployment kann direkt aus der Entwicklungsumgebung Visual Studio von Microsoft erfolgen (Publish to Azure bzw. Publish Web Site).

Quellenverzeichnis

[1] Scott W Ambler. *Types of Reuse In Information Technology*. 2010. URL: http://www.ambysoft.com/essays/typesOfReuse.html (besucht am 23. 05. 2015) (siehe S. 68, 69).

[2] Sidney Andrews. *Data-Platform*. 2015. URL: https://github.com/Azure-Readiness/DevCamp/tree/master/Presentation/Data-Platform (besucht am 11. 05. 2015) (siehe S. 57).

[3] Luigi Atzori, Antonio Iera und Giacomo Morabito. „The Internet of Things: A Survey“. In: *Comput. Netw.* 54.15 (2010), S. 2787–2805 (siehe S. 2, 7).

[4] Stefan Aulbach u. a. „Anforderungen an Datenbanksysteme für Multi-Tenancy-und Software-as-a-Service-Applikationen“. In: *Btw*. Bd. 144. 2009, S. 544–555 (siehe S. 50).

[5] Stefan Aulbach u. a. „Multi-tenant databases for software as a service: schema-mapping techniques“. In: *Proc. ACM SIGMOD Int. Conf. on Management of Data*. SIGMOD '08. New York, NY, USA: ACM, 2008, S. 1195–1206. URL: http://doi.acm.org/10.1145/1376616.1376736$%5Cbackslash$npapers2://publication/uuid/A805CDCC-8564-4DAC-8F4A-5746DBCCA9F1 (siehe S. 54).

[6] Christian Baun. *Cloud computing\rweb-based dynamic IT services*. Springer Berlin Heidelberg, 2011, 1 online resource (ix, 97 p.) URL: http://dx.doi.org/10.1007/978-3-642-20917-8 (siehe S. 46).

[7] Michael Blackstock und Rodger Lea. „IoT interoperability: a hub-based approach“. In: *Internet of Things (IOT), 2014 International Conference on the* (2014), S. 79–84 (siehe S. 14).

[8] Michael Blackstock und Rodger Lea. „Toward interoperability in a web of things“. In: *Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication - UbiComp '13 Adjunct*. ACM. 2013, S. 1565–1574. URL: http://dl.acm.org/citation.cfm?doid=2494091.2497591 (siehe S. 14, 15).

[9] Michael Blackstock und Rodger Lea. „WoTKit: A Lightweight Toolkit for the Web of Things“. In: *Proceedings of the Third International Workshop on the Web of Things*. ACM, 2012, S. 3. URL: http://eprints.lancs.ac.uk/57475/ (siehe S. 2, 15).

[10] David L. Brock. *The electronic product code (EPC): a naming scheme for physical objects (White paper)*. Cambridge, MA, USA: Auto-ID Center, 2001, S. 21 (siehe S. 6).

[11] Bundesverband der Deutschen Vending-Automatenwirtschaft e.V. *Vending - Der Verkauf von Waren durch Automaten*. 2015. URL: http://www.bdv-vending.de/branche/vending/ (besucht am 22. 04. 2015) (siehe S. 16).

[12] Peter Buxmann, Heiner Diefenbach und Thomas Hess. „Software as a Service: die Anwendungsebene des Cloud Computing“. In: *Die Softwareindustrie*. Springer Berlin Heidelberg, 2011, S. 205–225. URL: http://link.springer.com/10.1007/978-3-642-13361-9 (siehe S. 47).

[13] Hong Cai, Ning Wang und Ming Jun Zhou. „A transparent approach of enabling saas multi-tenancy in the cloud“. In: *Proceedings - 2010 6th World Congress on Services, Services-1 2010*. 2010, S. 40–47 (siehe S. 49, 50).

[14] Frederick Chong und Gianpaolo Carraro. *Architecture Strategies for Catching the Long Tail*. 2006. URL: http://www2.cistratech.com/whitepapers/MS%5C_longtailsaas.pdf (besucht am 14. 05. 2015) (siehe S. 50).

[15] Frederick Chong, Gianpaolo Carraro und Roger Wolter. *Multi-Tenant Data Architecture*. 2006. URL: http://msdn.microsoft.com/en-us/library/aa479086.aspx (besucht am 14. 05. 2015) (siehe S. 51–54).

[16] Cybulski & Linden. „Composing Multimedia Artefacts for Reuse“. In: *Pattern languages of program design* 4 (1998), S. 1–15 (siehe S. 71, 72).

[17] Jacob L Cybulski. *Introduction to Software Reuse*. Techn. Ber. Melbourne: University of Melbourne, 1996 (siehe S. 67, 68, 70, 71).

[18] European Vending Association. *Telling the good story of coffee service and vending*. 2015. URL: http://www.vending-europe.eu/eva/an%5C_introduction%5C_to%5C_vending-update-november-2014.pdf (besucht am 22. 04. 2015) (siehe S. 16).

[19] Roy Thomas Fielding. „Architectural Styles and the Design of Network-based Software Architectures“. Diss. 2000, S. 162. URL: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm (siehe S. 11, 36, 40).

[20] K.H. Friese. *A realistic look at homœopathy*. 1993. URL: http://www.drdobbs.com/a-realistic-look-at-object-oriented-reus/184415594?cid=Ambysoft (besucht am 23. 05. 2015) (siehe S. 71).

[21] Gartner Inc. *Gartner Says 4.9 Billion Connected "Things"Will Be in Use in 2015*. 2014. URL: http://www.gartner.com/newsroom/id/2905717 (besucht am 26. 04. 2015) (siehe S. 2).

[22] W H Y Gartner und Events Consulting. *Gartner Identifies the Top 10 Strategic Technology Trends for 2013 Newsroom Gartner Identifies the Top 10 Strategic Technology Trends for*. 2013. URL: http://www.gartner.com/newsroom/id/2867917 (besucht am 26. 04. 2015) (siehe S. 1).

[23] D.M. German und D.D. Cowan. „Towards a unified catalog of hypermedia design patterns“. In: *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences*. 2000, 8 pp. (Siehe S. 71).

[24] Neil Gershenfeld. *Wenn die Dinge denken lernen*. Econ, 1999, S. 236 (siehe S. 7).

[25] Nicolai Götz. *Cloud Computing – Service-Modelle*. 2014. URL: http://dailyitproblems.azurewebsites.net/cloud-computing-service-modelle (besucht am 11. 05. 2015) (siehe S. 47).

[26] Dominique Guinard. „A Web of things application architecture“. Ph.D. ETH Zurich, 2011, S. 220. URL: http://e-collection.library.ethz.ch/view/eth:4590 (siehe S. 11–13).

[27] Dominique Guinard und Vlad Trifa. „Towards the Web of Things : Web Mashups for Embedded Devices“. In: *Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM 2009), in proceedings of WWW (International World Wide Web Conferences)*. Madrid, Spain, Apr. 2009, S. 1–8 (siehe S. 10, 11).

[28] Dominique Guinard u. a. „5 From the Internet of Things to the Web of Things : Resource Oriented Architecture and Best Practices“. In: *Architecting the Internet of Things*. Hrsg. von Dieter Uckelmann, Mark Harrison und Florian Michahelles. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, S. 97–129 (siehe S. 10, 11).

[29] Dominique Guinard u. a. „Interacting with the SOA-based internet of things: Discovery, query, selection, and on-demand provisioning of web services“. In: *IEEE Transactions on Services Computing* 3.3 (2010), S. 223–235 (siehe S. 11).

[30] Vipul Gupta, David G Simmons und Sun Labs. *Building the Web of Things with Sun SPOTs*. Manning Publications Co., 2010, S. 375 (siehe S. 12).

[31] Sidney Higa. *Azure SQL Database - elastic database tools*. 2015. URL: http://azure.microsoft.com/en-us/documentation/articles/sql-database-elastic-scale-introduction/ (besucht am 10. 05. 2015) (siehe S. 59, 61).

[32] Sidney Higa. *Scaling using the elastic database split-merge tool*. 2015. URL: http://azure.microsoft.com/en-us/documentation/articles/sql-database-elastic-scale-overview-split-and-merge/ (besucht am 11. 05. 2015) (siehe S. 63).

[33] Sidney Higa. *Shard map management*. 2015. URL: http://azure.microsoft.com/en-us/documentation/articles/sql-database-elastic-scale-shard-map-management/ (besucht am 11. 05. 2015) (siehe S. 58–60).

[34] Dean Jacobs und Stefan Aulbach. „Ruminations on multi-tenant databases“. In: *BTW Proceedings* 103 (2007), S. 514–521 (siehe S. 52).

[35] Miryung Kim Miryung Kim u. a. „An ethnographic study of copy and paste programming practices in OOPL“. In: *Proceedings. 2004 International Symposium on Empirical Software Engineering, 2004. ISESE '04.* 2004, S. 83–92 (siehe S. 70).

[36] Tim Kindberg u. a. „People , Places , Things : Web Presence for the Real World People , Places , Things : Web Presence for the Real World“. In: *Mobile Networks and Applications* 7.5 (2001), S. 365–376 (siehe S. 10).

[37] Sriram Krishnan. *Programming Windows Azure*. O'Reilly Media, 2010, S. 345 (siehe S. 39).

[38] Friedemann Mattern und Christian Flörkemeier. „Vom Internet der Computer zum Internet der Dinge“. German. In: *Informatik-Spektrum* 33.2 (2010), S. 107–121 (siehe S. 6–8).

[39] P. Mell und T. Grance. *The NIST definition of cloud computing, NIST Special Publication 800*. 2011. URL: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf (besucht am 03. 05. 2015) (siehe S. 46, 47).

[40] Ingo Melzer. *Service-orientierte Architekturen mit Web Services: Konzepte - Standards - Praxis*. 4. Aufl. Springer Spektrum, 2010, S. 382. URL: http://books.google.de/books?id=e3qnVPngoUoC%5C&lpg=PA113%5C&dq=REST%20und%20HTTP%5C&hl=de%5C&pg=PA84%5C#v=onepage%5C&q=REST%20und%20HTTP%5C&f=false (siehe S. 36).

[41] Valery Mizonov und Seth Manheim. *Azure Table Storage and Windows Azure SQL Database - Compared and Contrasted*. 2014. URL: https://msdn.microsoft.com/en-us/library/azure/jj553018.aspx (besucht am 25. 05. 2015) (siehe S. 39).

[42] Marc Nanard, Jocelyne Nanard und Paul Kahn. „Pushing Reuse in Hypermedia Design : Golden Rules , Design Patterns and Constructive Templates“. In: *ACM Conference on Hypertext & Media*. HYPERTEXT '98. New York, NY, USA: ACM, 1998, S. 11–20 (siehe S. 71).

[43] Klaus Pohl und Chris Rupp. *Basiswissen Requirements Engineering Aus- und Weiterbildung nach IREB-Standard zum Certified Professional for Requirements Engineering Foundation Level*. ISQL-Reihe. dpunkt-Verlag, 2011, S. 176 (siehe S. 31).

[44] Salesforcecom. *The Force.com Multitenant Architecture*. 2008. URL: papers2 : / / publication / uuid / 9146C1B5 - 0A89 - 4229 - BCBC - 69AA03D4F5EA (besucht am 04. 05. 2015) (siehe S. 45).

[45] Gerald Santucci. *The Internet of Things : Between the Revolution of the Internet and the Metamorphosis of Objects*. 2010. URL: http : / / cordis.europa.eu/fp7/ict/enet/publications%5C_en.html (besucht am 03. 04. 2015) (siehe S. 7).

[46] Manfed Steyer und Holger Schwichtenberg. *Moderne Webanwendungen mit ASP.NET MVC und JavaScript: ASP.NET MVC im Zusammenspiel mit Web APIs und JavaScript-Frameworks*. 2. Auflage. Köln: O'Reilly Media, 2014, S. 560 (siehe S. 41).

[47] Tops vending systems. *Sielaff Robimat*. 2015. URL: http : / / www . topsvending.be/en/products/sielaff-robimat (besucht am 28. 04. 2015) (siehe S. 16, 17).

[48] Dieter Uckelmann, Mark Harrison und Florian Michahelles. „An architeture approach towards the future internet of things“. In: *Architetureing the internet of things*. Hrsg. von Dieter Uckelmann, Mark Harrison und Florian Michahelles. Springer Berlin Heidelberg, 2011, S. 1–24. URL: http://link.springer.com/10.1007/978-3-642-19157-2 (siehe S. 2, 6–9).

[49] Mike (Microsoft) Wasson. *Single-Page Applications: Build Modern, REsponsive Web Apps with ASP.NET*. 2013. URL: https : / / msdn . microsoft . com / en - us / magazine / dn463786 . aspx (besucht am 28. 04. 2015) (siehe S. 42).

[50] Mathias Weber. *Cloud Computing - Evolution in der Technik, Revolution im Business*. 2009. URL: http://www.bitkom.org/files/documents/BITKOM - Leitfaden - CloudComputing % 5C _ Web . pdf $ %5Cbackslash $ nhttp : / / scholar . google . com / scholar ? hl = en % 5C & btnG = Search % 5C & q = intitle : Cloud + Computing + - + Evolution + in + der + Technik + , + Revolution + im + Business % 5C # 0 (besucht am 06. 05. 2015) (siehe S. 50).

[51] Erik Wilde. „Putting Things to REST“. In: *Transport* 15.November (2007), S. 1–13. URL: http://dret.net/netdret/publications%5C#wil07n (siehe S. 11).

[52] Bill Wilder. *Cloud Architecture Patterns*. Oreilly and Associate Series. Oreilly & Associates Incorporated, 2012, S. 182. URL: http://it-ebooks.info/book/947/ (siehe S. 54, 56).

[53] Andy Zaidman. „Multi-Tenant SaaS Applications : Maintenance Dream or Nightmare ? Position paper“. In: *Iwpse-Evol '10*. IWPSE-EVOL '10. New York, NY, USA: ACM, 2010, S. 88–92. URL: http://dl.acm.org/citation.cfm?id=1862393 (siehe S. 48–51).