Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Philipp Gillé

Eine Untersuchung zum Datenaustausch in mobilen Umgebungen

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Philipp Gillé

Eine Untersuchung zum Datenaustausch in mobilen Umgebungen

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Olaf Zukunft
Zweitgutachter: Prof. Dr. Martin Hübner

Eingereicht am: 6. Dezember 2013

Philipp Gillé

Thema der Arbeit
Eine Untersuchung zum Datenaustausch in mobilen Umgebungen

Stichworte
SOAP, REST, OData, Binärprotokolle, BaaS, Backend-as-a-Service, Backend, SOA, Web Service, WCF Services, ASP.NET Web API, WCF Data Services, Apache Thrift, Azure Mobile Services

Kurzzusammenfassung
Diese Arbeit behandelt den Datenaustausch in mobilen Umgebungen. Es werden grundlegende Technologien hierfür vorgestellt, sowie deren Eigenschaften gegenübergestellt. Anschließend werden einige Implementierungen der Technologien ausgewählt, mit denen beispielhaft eine Anwendung entwickelt wird. Anhand dieser Anwendung werden Messungen durchgeführt, deren Ergebnisse zusammen mit den gemachten Erfahrungen eine abschließende Bewertung der Technologien ermöglichen.

Philipp Gillé

Title of the paper
A study about data exchange in mobile environments

Keywords
SOAP, REST, OData, Binary Protocols, BaaS, Backend-as-a-Service, Backend, SOA, Web Service, WCF Services, ASP.NET Web API, WCF Data Services, Apache Thrift, Azure Mobile Services

Abstract
This document discusses data exchange in mobile environments. Relevant technologies are presented, whose properties are compared afterwards. Subsequently some implementations of these technologies are chosen, with which an examplary application is developed. With this application some benchmarks are conducted. Together with the gathered experiences this makes a closing rating possible.

Inhaltsverzeichnis

1. Einleitung

Innerhalb der letzten Jahre hat die Verbreitung moderner mobiler Endgeräte wie z.B. Smartphones stark zugenommen [Gar13]. Gleichzeitig wird für die kommenden Jahre ein starker Anstieg des Datenverkehrs in mobilen Netzwerken prognostiziert [Cis13]. Viele Anwendungen für diese Endgeräte verwenden auf zentralen Servern laufende Back-Ends um Daten zu speichern und zu verwalten. Dadurch kommen den Technologien, die dem Austausch dieser Daten dienen, eine tragende Rolle zu.

1.1. Motivation

An die bestehenden Technologien für den Datenaustausch werden in mobilen Umgebungen bestimmte Anforderungen gestellt. Beispielsweise sind aufgrund niedrigerer Datenübertragungsraten als in herkömmlichen Netzwerken häufig eine schnelle Verarbeitungsgeschwindigkeit und ein geringes Datenübertragungsvolumen gefragt. In anderen Szenarien wiederum sind Sicherheit und Interoperabilität wichtiger.

Einige der Technologien, welche für den Datenaustausch in herkömmlichen Netzwerken zum Einsatz kommen, lassen sich bei der Entwicklung mobiler Anwendungen nicht verwenden. So ist beispielsweise in der Klassenbibliothek von .NET für Windows Store Apps gegenüber der .NET Klassenbibliothek für Windows Desktop Applikationen der Namespace System.Data nicht mehr vorhanden, sodass nicht mehr per ODBC auf Datenbanken zugegriffen werden kann. Stattdessen werden z.B. Web Services eingesetzt, über welche die mobilen Anwendungen mit ihren Back-Ends kommunizieren können.

Web Services können auf verschiedene Arten implementiert werden. Mit der Entwicklung von serviceorientierten Architekturen (SOA) wurde SOAP (Simple Object Access Protocol) populär. Aufbauend auf der Architektur des Webs wurde später REST (Representational State Transfer) entwickelt, das bei vielen Schnittstellen zu öffentlich zugänglichen Systemen SOAP ersetzt hat.

Neben diesen beiden weit verbreiteten Web Services gibt es weitere Technologien, welche für den Datenaustausch mobiler Anwendungen mit Back-Ends eingesetzt werden können. Viele der zuvor genannten „RESTful“ APIs unterscheiden sich in einigen Punkten voneinander, wie z.B. das Datenformat der ausgetauschten Daten, welche die Ressource repräsentieren, auf die zugegriffen wurde. Hier ist der offene Standard OData ein Versuch diese Art von Schnittstellen zu vereinheitlichen. Ein Ansatz, die Geschwindigkeit der Datenübertragung zu erhöhen, sind Binärprotokolle. Und an Anwendungsentwickler, welche sich nicht mit der Entwicklung von Back-Ends beschäftigen möchten oder können, richten sich Backend-as-a-Service Provider. Diese stellen die serverseitige Infrastruktur schlüsselfertig bereit.

Die Stärken und Schwächen der zuvor genannten Technologien, sowie ihre Eignung für den Einsatz in mobilen Umgebungen besser kennen zu lernen ist der Antrieb für diese Arbeit.

1.2. Ziele und Abgrenzung

Hauptziel der Arbeit ist es, anhand einer konkreten Beispielanwendung ausgewählte Technologien für den Einsatz in Back-Ends im Kontext von mobilen Umgebungen unter verschiedenen Gesichtspunkten zu vergleichen. Der Vergleich soll dabei auf theoretischen und praktischen Erkenntnissen beruhen. Letztendlich sollen Empfehlungen für populäre Einsatzszenarien ausgesprochen werden.

Die zu entwickelnden Systeme für die Beispielanwendung werden so weit entwickelt, bis sie einen Vergleich ermöglichen. Ein Einsatz in produktiven Umgebungen ist nicht vorgesehen.

1.3. Gliederung

In Kapitel 2 (S. 4 ff.) werden gängige Technologien für den Datenaustausch vorgestellt. Dabei werden ihre theoretischen Eigenschaften betrachtet. In Kapitel 3 (S. 22 ff.) werden diese anschließend gegenübergestellt. In Kapitel 4 (S. 30 ff.) wird die Entwicklung einer Testanwendung beschrieben. Zuerst werden hier die einzusetzenden Technologieimplementierungen ausgewählt, sowie die Kriterien ermittelt, unter denen ein späterer Vergleich stattfinden soll. Daraufhin werden die Anforderungen analysiert und für die Systeme Entwürfe aus fachlicher (Konzept) und technischer Sicht (Design) erstellt. Anschließend werden die Systeme entsprechend implementiert. Nach der Implementierung werden Messungen z.B. der Bearbeitungsdauer von Anfragen durchgeführt. Das Kapitel 5 (S. 77 ff.) enthält die Messergebnisse

und Beschreibungen der gemachten Erfahrungen. Das letzte Kapitel, Kapitel 6.1 (S. 91 ff.), fasst die Arbeit zusammen und gibt einen Ausblick auf mögliche zukünftige Weiterentwicklungen der betrachteten Technologien.

2. Grundlagen

In diesem Kapitel wird zuerst eine Einführung in die zeitliche Entwicklung von Systemen zum Speichern von Daten, sowie Methoden für den Zugriff auf diese Systeme gegeben. Anschließend werden die aktuellen Technologien im Einzelnen vorgestellt, welche in dieser Arbeit näher betrachtet werden. Außerdem wird auf die Besonderheiten von mobilen Umgebungen eingegangen.

Was in den folgenden Unterkapiteln und in der gesamten Arbeit als „Technologien“ für den Datenaustausch bezeichnet wird, sind teilweise Spezifikationen, Architekturen, konkrete Protokolle oder nur Konzepte. Diese lassen sich jedoch auf vergleichbare Art und Weise für die Entwicklung und den Einsatz eines Back-Ends verwenden, so dass diese einheitliche Bezeichnung als Oberbegriff verwendet wird.

2.1. Einführung

Nach der Entstehung von relationalen Datenbankmanagementsystemen (RDBMS) in den 1970er Jahren etablierte sich SQL als Sprache für den Zugriff auf diese Systeme. Teil von SQL ist eine Data Manipulation Language (DML) genannte Operationsmenge, welche vier Operationen enthält: Select, Insert, Update, Delete. Diese Menge an Operationen wurde 1983 in [Mar83] als „CRUD“ bezeichnet, was für Create, Read (oder Retrieve), Update und Delete steht. CRUD spielt im weiteren Verlauf dieser Arbeit noch eine Rolle. Für den Zugriff auf Datenbanken über das Netzwerk etablierte sich ODBC. Dies ist eine Datenbankschnittstelle, welche SQL als Datenbanksprache verwendet und es mit Hilfe von sog. ODBC Treibern ermöglicht, eine Unabhängigkeit zwischen einer Anwendung und einem DBMS zu erreichen.

Später kamen objektorientierte, Key-Value-, dokumentorientierte und Graphdatenbanken auf. Diese werden als NoSQL Datenbanken bezeichnet [Fow12]. Für den Zugriff werden Erweiterungen zu SQL und objektrelationale Mappings (ORM) verwendet.

Um nicht nur entfernte Daten- sondern auch Prozeduraufrufe zu ermöglichen, wurden Remote Procedure Calls (RPC) als Paradigma und XML-RPC als Implementierung dessen entwickelt. Diese bot bereits die Möglichkeit, Prozeduraufrufe in XML zu encodieren und HTTP für den Transport in Netzwerken zu verwenden [Mel10, S. 108]. Das enthaltene vereinfachte Typsystem wird jedoch in [Mel10, S. 111] als aufwändig für den Programmierer empfunden.

Als RPC den Anforderungen an verteilte Systeme nicht mehr genügte, wurden Middlewares wie CORBA und RMI entwickelt [Mel10, S. 2]. CORBA hatte jedoch praktische Nachteile wie eine anfangs unvollständige Spezifikation, und dass nicht immer eine gute Kompatibilität zwischen den ORB-Implementierungen verschiedener Hersteller gegeben war [Mel10, S. 75].

Die zuvor genannten Technologien sind teilweise Programmiersprachenspezifisch, benötigen eine zusätzliche Sprache oder einen Treiber für die Kommunikation, machen Client und Server voneinander abhängig oder haben andere Schwächen wie eine sich häufig ändernde Spezifikation. Um Client und Server voneinander zu entkoppeln und von Dritten entwickelte IT-Systeme integrieren zu können, wurden Service-orientierte Architekturen (SOA) entwickelt.

Definition aus [Mel10, S. 9]:

> Service-orientierte Architekturen, kurz SOA, sind das abstrakte Konzept einer Software-Architektur, in deren Zentrum das Anbieten, Suchen und Nutzen von Diensten über ein Netzwerk steht. Diese Dienste werden plattformübergreifend von Applikationen oder anderen Diensten genutzt. Ein wesentlicher Vorteil einer SOA ist die Unabhängigkeit von der jeweiligen Implementierung. Dies ermöglicht eine funktionale Zerlegung der Anwendung und erleichtert eine prozessorientierte Betrachtungsweise. Teilprozesse oder Dienste können über das Netzwerk angesprochen werden. Im Idealfall ist sogar eine einfache Integration ganzer Anwendungen möglich.

Eine SOA bietet folgende grundlegende Merkmale der Dienste:

- Verteiltheit
- Lose Kopplung (Einbindbarkeit zur Laufzeit)
- Auffindbarkeit über einen Verzeichnisdienst
- Prozessorientierung (im Sinne von Geschäftsabläufen)

- Verwendung von Standards
- Trennung von Schnittstelle und Implementierung
- Hohe Wiederverwendbarkeit dank hoher Kapselung

Eine Umsetzung des Konzepts von SOA sind Web Services, auf die in einem separaten Unterkapitel genauer eingegangen wird.

2.2. Web Services

In [LR07] wird unterschieden zwischen „big“ und „RESTful“ Web Services, wobei die zu ersterem zugehörigen Technologien, wie auch in [JW10], als „WS-*“-Stack bezeichnet werden.

2.2.1. WS-* Stack

Mit dem WS-* Stack umgesetzte Web Services (in diesem Unterkapitel einfach als Web Services bezeichnet) stellen die technische Instanz einer SOA dar [Mel10, S. 4 u. weitere].

Eine der konkretesten der vielen Definitionen des Begriffs Web Service ist folgende vom W3C [HH13]:

> A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

Ein Web Service besteht aus den drei Komponenten Kommunikation, Dienstbeschreibung und Verzeichnisdienst. Diese werden zurzeit mithilfe der folgenden Spezifikationen beschrieben [Mel10, S. 63]:

- SOAP (Simple Object Access Protocol)
- WSDL (Web Service Description Language)
- UDDI (Universal Description, Discovery and Integration protocol)

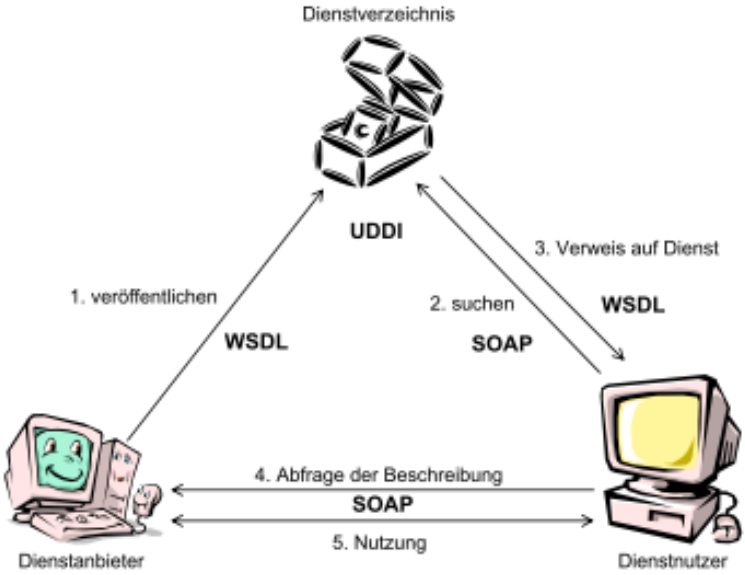

Abbildung 2.1.: Web Service Komponenten [Quelle: [Mel10, S. 64]]

SOAP beschreibt das XML-basierte Nachrichtenformat der Kommunikation und dessen Einbettung in ein Transportprotokoll. WSDL ist eine XML-basierte Beschreibungssprache, um Web Services zu beschreiben. UDDI beschreibt einen Verzeichnisdienst für Web Services. Abbildung 2.1 stellt die Beziehungen der Komponenten zueinander dar.

Damit entspricht das Zusammenspiel der Web Service Komponenten dem Aufbau einer SOA. Definitionen des Begriffs Web Service erwähnen die Komponente UDDI oftmals nicht. Dies liegt daran, dass sie kein notwendiges Kriterium für Web Services sind [Mel10, S. 63].

Im Folgenden werden die drei Komponenten mit ihren Eigenschaften und zusätzliche Erweiterungen detaillierter vorgestellt. Mehr über die Architektur von Web Services lässt sich auf der entsprechenden Webseite des W3C erfahren [DB+13].

SOAP

SOAP ist das Nachrichtenformat von WS-* Web Services. Es ist eine Weiterentwicklung von XML-RPC. Es beschreibt das XML-basierte Nachrichtenformat zur Kommunikation mit Web Services und dessen Einbettung in ein fast beliebiges Transportprotokoll [Mel10, S. 84]. SOAP stand bei der Entwicklung für “Simple Object Access Protocol”, wird mittlerweile jedoch

weder als einfach, noch als geeignet für den Zugriff auf Objekte empfunden [Mel10, S. 84]. Teilweise wird der Begriff SOAP synonym mit WS-* bzw. „big“ Web Services verwendet.

Generell dient eine SOAP-Nachricht einer Ein-Weg-Übertragung, allerdings ist SOAP auch für komplexere Szenarien wie Request/Response, Request/Multiple-Response und Konversationen vorgesehen [Mel10, S. 2]

Neben Sender und Empfänger gibt es Intermediäre. Dies sind Zwischenstationen, die Aufgaben übernehmen können wie das Loggen von SOAP Nachrichten oder die Nachrichten von einem Transportsystem auf ein anderes übertragen können (beispielsweise von HTTP auf SMTP).

Neben textbasierten Nachrichten lassen sich auch binäre Dateien mit SOAP verschicken. Dies ist mit den drei Ansätzen „XML-binary Optimized Packaging“, „SOAP Message Transmission Optimization Mechanism“ und „Resource Representation SOAP Header Block“ möglich.

Es gibt zwei Typen von Kommunikationen für SOAP-Nachrichten. Für einen konversationsorientierten Nachrichtenaustausch gibt es den Typ „document-style“. Er dient dem Austausch von Dokumenten oder Daten zwischen zwei Anwendungen mit beliebigem XML-Inhalt ohne Beschreibung der Semtantik [Mel10, S. 93]. Daneben gibt es den Typ „rpc-style“, welcher speziell für die Verwendung von SOAP-Nachrichten für Funktionsaufrufe und entsprechende Rückantworten gedacht ist und eine besondere Syntax erfordert. Dieser RPC-orientierte Kommunikationstyp wird in Web Services am häufigsten verwendet [Mel10, S. 96].

Beim SOAP RPC-Request wird eine Methode mit ihren Argumenten aufgerufen. Da objektorientierte Sprachen Überladungen von Methoden erlauben und die Unterscheidung anhand der Datentypen stattfindet, muss es möglich sein, über die Informationen in der Nachricht die korrekte Methode eindeutig auszuwählen. Dies wird mit Metadaten erreicht, welche eine Typisierung ermöglichen. Das XMLSchema (http://www.w3.org/2001/XMLSchema.xsd) enthält sehr viele Datentypen. Für SOAP wurde dies mit http://www.w3.org/2003/05/soap-encoding eingeschränkt, so dass 44 einfache und zwei komplexe Datentypen zur Verfügung stehen. Bei der Implementierung eines Web Services muss darauf geachtet werden, welche Datentypen von der zu verwendeten Laufzeitumgebung bereits unterstützt werden [Mel10, S. 99].

Der Transport einer SOAP-Nachricht muss nicht zwingend per HTTP erfolgen. Die Spezifikation schreibt kein Protokoll vor, sondern nur, wie eine Nachricht mit einem Protokoll übertragen

werden soll. In den meisten Fällen wird aufgrund der hohen Verfügbarkeit (auf verschiedenen Plattformen) jedoch HTTP gewählt. HTTP wird für die Übertragung der Nachricht dabei als Tunnel genutzt. Wird eine höhere Übertragungssicherheit benötigt, wird in [Mel10] z.B. WebSphereMQ empfohlen.
Es gibt auch Ansätze auf ein Protokoll für den Transport zu setzen, welche eine Ebene tiefer im OSI-Modell angesiedelt ist. So gibt es beispielsweise „SOAP-over-UDP“ [OAS09]

Damit der Service-Aufruf den Service finden kann, werden zwei Elemente benötigt: Eine URL (im Falle des Transportprotokolls HTTP) und der vollqualifizierte Klassennahme der Klasse, in welcher die aufrufbare Methode enthalten ist. Beides zusammen ergibt den Service-Endpunkt.

Die SOAP-Spezifikation schreibt keine Übermittlungsform für die Nachrichten vor. Theoretisch können Nachrichten sowohl synchron (hier: es wird eine Antwort erwartet), als auch asynchron (hier: es wird keine Antwort erwartet) verschickt werden. Da HTTP, welches ein Request/Response-Protokoll ist, nur als Transportprotokoll für die SOAP-Nachricht verwendet wird, ist das Antwortverhalten auf Anwendungsprotokollebene davon unabhängig.

Listing A.2 zeigt ein Beispiel eines SOAP Requests über HTTP (Quelle: [Hor10, Kapitel 4]).

WSDL

Die Web Services Description Language ist eine der Standardisierung beim W3C unterliegende XML-Sprache [Mel10, S. 116] und ermöglicht sowohl die abstrakte Beschreibung von Web Service Schnittstellen mit ihren Operationen, als auch die konkrete Beschreibung technischer Informationen zum Aufruf des Dienstes. Zusammen mit SOAP bildet WSDL die Grundlage für interoperable Web-Service-Umgebungen [Mel10, S. 115]. Eine semantische Beschreibung der Dienstfunktionalität ist nicht Aufgabe der WSDL. Allerdings gibt es hierfür Erweiterungen [Mel10, S. 116].

Einige Entwicklungsumgebungen ermöglichen es, aus einer Klasse mit Methoden ein WSDL-Dokument automatisch zu erzeugen oder auf Konsumentenseite aus einem solchen Dokument die Stubs für die Nutzung der Methoden zu erzeugen.

Im binding-Element des WSDL-Dokuments wird festgelegt, welches Protokoll für den Nachrichtenaustausch verwendet wird [Mel10, S. 121]. Dies ist nicht auf SOAP beschränkt, sondern kann z.B. auch eine REST-konforme Schnittstelle sein.

UDDI und WS-Inspection

UDDI (Universal Description, Discovery and Integration) und WS-Inspection sind Verzeichnisdienste für Web Services. Verzeichnisdienste bieten eine durchsuchbare Übersicht über sämtliche Ressourcen in einem Netzwerk und machen diese Übersicht Benutzern und Applikationen zugänglich. In [Mel10] werden Verzeichnisdienste als zentrale Komponente in Service-orientierten Architekturen gesehen und als zwingend notwendig, sobald die Architektur „mehr als nur eine sehr geringe Anzahl von Diensten“ umfasst.

Verzeichnisdienste sollen jedoch nicht nur die oben genannten Möglichkeiten bieten, sondern auch Antworten auf die Fragen geben, welche Qualität ein Web Service bietet, wie die jeweilige Abrechnung erfolgt und wer die Verantwortung für einen Web Service trägt. Hierfür existiert ein an Telefonbücher angelehntes Konzept mit White Pages für Informationen über Unternehmen, Yellow Pages für eine Kategorisierung der Web Services, Green Pages für Informationen zu einzelnen Web Services und Service Type Registration für Informationen in maschinenlesbarer Form [Mel10, S. 146].

Web Service-Erweiterungen

Es gibt viele Erweiterungen für WS-* Web Services. Sie beschreiben meist bestimmten Bereichen zugeordnete Standards. Dadurch lässt sich die Funktionalität der Web Services erweitern und dabei gleichzeitig die Interoperabilität mit anderen Web Service-Systemen sicherstellen, welche die genutzten Standards ebenfalls unterstützen. Einige Bereiche und zugehörige Erweiterungen seien hier aufgelistet:

- Messaging: WS-Notification, -Adressing, -Transfer, ...
- Metadata Exchange: WS-Policy, -Discovery, -Inspection, -MetadataExchange
- Sicherheit: WS-Security, -Trust, -SecurityPolicy, -Privacy, -Federation, -Authorization
- Transaktionen: WS-Context, -Coordination, -Transaction

Ein Vergleich von Produkten, welche den WS-Transaction Standard implementieren wird in [Tew08] durchgeführt.

2.2.2. REST

REST steht für “REpresentational State Transfer” und wurde 2000 von Roy Thomas Fielding in seiner Dissertation [Fie00] beschrieben. REST ist im Gegensatz zu SOAP kein Protokoll,

sondern ein Architekturstil, der definiert, wie existierende Web-Protokolle verwendet werden können, um unter anderem möglichst einfach Web Services zu realisieren [Mel10, S. 111]. Es fokussiert sich damit auf Anwendungen im World Wide Web. Wenn eine Anwendung konform zum REST-Architekturstil ist, wird sie als RESTful bezeichnet. Sie verwaltet beliebige Ressourcen, deren Repräsentation über eine URL zugreifbar sein muss. [Mel10, S. 111]

In einer Antwort auf eine Anfrage an eine entsprechende Anwendung können Verweise auf weitere Ressourcen enthalten sein. Folgt ein Client einer solchen Ressource, ändert er seinen Zustand. Daher rührt die Bezeichnung representational state transfer. [Mel10, S. 112]

Eine Anfrage ist ein HTTP-Request und unterstützt entsprechend lediglich die durch HTTP gegebenen Methoden. Die Methoden von HTTP 1.1 sind in [W3C99, Section 9] näher beschrieben. Alle Methoden beziehen sich stets auf eine Ressource. Die vier grundlegenden Methoden entsprechen dem in Unterkapitel 2.1 (S. 4) erwähnten CRUD-Schema: POST (zum Erzeugen), GET (zum Abfragen), PUT (zum Verändern durch Überschreiben) und DELETE (zum Löschen einer Ressource). Erweitert wurden die Methoden außerdem um PATCH (zum Verändern von Teilen einer Ressource) [LD10], welches jedoch nicht Bestandteil von HTTP 1.1 ist.

Alle Anfragen bestehen aus einer URL und einem der möglichen HTTP-Methoden. Nur die Anfragen der Methoden POST, PUT und PATCH enthalten einen HTTP-Body. Dieser kann Daten in beliebiger Form (z.B. ein XML-Dokument, JSON oder Binärdaten) enthalten. Da bei einer GET-Anfrage keine Informationen als Payload mitgeschickt werden können, müssen diese in der URL enthalten sein. Häufig werden hier Informationen zur Authentifizierung und Autorisierung verwendet. Da HTTP zustandslos ist, müssen diese entweder jedesmal mitgeschickt werden, oder es wird auf Anwendungsebene eine Session verwendet, beispielsweise mithilfe von URI-Parametern und Cookies, die im HTTP-Header mitgeschickt werden [Coo13].

Beispiele für HTTP-Anfragen sind im Anhang in A.3.1 (S. 102) aufgelistet.

Durch das Anfrage/Antwort-Verhalten von HTTP sind theoretisch keine asynchronen Aufrufe möglich. Diese können in der Praxis jedoch mit 2 oder mehr Anfragen dadurch erreicht werden, dass in der Antwort auf die erste Anfrage ein Verweis auf einen Job (Ressource) zurückgegeben

wird. Dort kann der Status des Jobs abgefragt werden und wenn der Job abgearbeitet ist, liegt dort das Ergebnis oder ein Verweis auf das Ergebnis [LR07, S. 228].
Auch Zwei-Phasen Commits und verteilte Transaktionen sind möglich über einen Umweg mit einer für die Transaktionen angelegten Ressource [LR07, S. 231], [Fla09a]).

Es galt lange als Nachteil von REST, dass es nicht möglich war, automatisch Stubs für die Clients von RESTful Web Services zu generieren [Mel10], jedoch gibt es mittlerweile Beschreibungssprachen unter anderem für diesen Zweck (siehe 2.2.2 (S. 14 ff.)).
Eine mit TLS verschlüsselte Übertragung der Daten ist ohne zusätzliche Erweiterungen durch die Nutzung von HTTPS möglich.

Als Hauptmerkmal dafür, ob ein Web Service als RESTful bezeichnet werden kann oder nicht, wird der Ort der Unterbringung der Information über die Methode („method information“) und den Geltungsbereich („scoping information“) gesehen [LR07, Preface und Intro].
Die Methodeninformationen sollten bei einem RESTful Web Service in der HTTP Methode enthalten sein [LR07, S. 79]. Soll eine Ressource angefragt werden, wird GET verwendet. Soll eine Ressource angelegt werden, soll POST verwendet werden. Zum Löschen DELETE u.s.w.
Der Geltungsbereich soll in der Host-URL des HTTP-Umschlags enthalten sein [LR07, S. 79]. Die dort angegebene URL soll auf eine spezifische Ressource zeigen, auf welche die Methode angewendet werden soll.
Welche Information dabei Methodeninformation oder Information zum Geltungsbereich ist, wird vom Servicedesign festgelegt [LR07, S. 12].
In [LR07] findet noch folgende Unterscheidung statt: Wenn die Methodeninformation nicht der HTTP Methode entspricht, gilt der Web Service nicht als *RESTful*. Wenn die Information zum Geltungsbereich nicht in der Host-URL enthalten ist, gilt die *Architektur* des Web Services nicht als *ressourcenorientiert*.

Die HTTP-Methoden reichen aus, um sämtliche Interaktionen mit Ressourcen im Web zu beschreiben [LR07, S. 53]. Mehr Informationen dazu, warum WS-* und REST den gleichen Zweck erfüllen können und damit vergleichbar sind, liefert [LR07, Kapitel 5, „map any kind of action to the uniform interface“].
Im sog. „human web“ werden nur GET und POST verwendet.

Wichtig bei der Verwendung von RESTful Web Services sind Content-Types. Eine URI einer Ressource soll möglichst nicht auf eine bestimmte Repräsentationsform zeigen. Es

ist z.B. möglich, eine Ressource sowohl in XML, als auch in JSON zu repräsentieren. Dies ermöglicht Content-Negotiation, bei welcher der Client und der Server aushandeln, welche Content-Types jeweils unterstützt werden und die Datenübertragung dem Ergebnis der Verhandlung entsprechend aussieht.

In den folgenden drei Unterkapiteln werden zu REST zugehörige Themen beschrieben.

ROA

ROA steht für „resource-oriented architecture“. Der Begriff wurde zwar bereits vor Veröffentlichung von [LR07] verwendet, jedoch dort aufgegriffen und genauer definiert und beschrieben als Beispiel für eine gute Implementierung eines RESTful Web Services. Fielding beschreibt in seiner Arbeit lediglich, wann eine Architektur RESTful ist und welche Merkmale es gibt. Mit der ROA wird eine konkrete Architektur beschrieben, welche die Merkmale erfüllt und auf Best Practices aufbaut [LR07, Preface].

Die ROA ist dabei ausdrücklich nicht als Gegenstück zur SOA positioniert, sondern beschreibt Komponenten im Detail, während die SOA eine Anwendung mit einer abstrakteren Sichtweise beschreibt [LR07].
Die Ressourcenorientierte Architektur soll die RPC-Style Architektur ersetzen, welche in Web Services verwendet wird, die SOAP + WSDL verwenden. Ein Unterscheidungsmerkmal liegt darin, dass die RPC-Style Architektur Algorithmen und die ROA Daten exponiert [LR07].
Daneben wird in [LR07] noch REST-RPC-Hybrid als Mischform genannt, welche von den Dienstanbietern zwar als RESTful bezeichnet werden, jedoch noch Elemente von RPC-Style Architekturen beinhalten (z.B. Methodeninformationen in der URI).

Definiert werden vier Konzepte und vier Eigenschaften einer ROA.
Konzepte [LR07, S. 105]:

- Ressourcen
- Namen und Adressen (URIs) der Ressourcen
- Repräsentation der Ressourcen
- Links zwischen den Ressourcen

Eigenschaften [LR07, S. 79], [LR07, S. 216]:

- Adressierbar [LR07, S. 84]
- Zustandslos [LR07, S. 86]
- Verbunden
- Einheitliches Interface

Beschreibungssprachen

WADL Als Gegenstück zu WSDL, welches bei mit SOAP umgesetzten Web Services verwendet wird, gibt es für RESTful Web Services WADL (Web Application Description Language). Eine WADL-Datei beschreibt die HTTP Anfragen, welche man an einen Web Service stellen kann: Welche URIs es gibt, welche Daten diese URIs erwarten und welche Daten sie zurückschicken. Mit einer WADL-Bibliothek kann eine WADL-Datei verarbeitet und daraus die möglichen Aufrufe in der entsprechenden nativen Programmiersprache modelliert werden. [LR07, S. 47].

Auch negative Eigenschaften von WADL sind in [LR07, S. 290] beschrieben.

WSDL 2.0 Mit WSDL 2.0 ist es ebenfalls möglich, RESTful Web Services zu beschreiben [Man08].

Hypermedia

Als Hypermedia wird eine logische Erweiterung zu Hypertext bezeichnet, welche mit Text, Bildern, Audio, Video und Hyperlinks ein nichtlineares Informationsmedium darstellt.
Hypermedia Formate sind beispielsweise XHTML 4, 5 und WADL. Auch JSON kann für Hypermedia verwendet werden [Kel13]. Diese Formate können neben der Repräsentation einer Ressource auch Hypermedia Links auf weitere Ressourcen enthalten.
Mit diesen Verweisen lassen sich die zu einer Ressource zugehörigen legalen Interaktionen bekanntmachen. Dies ermöglicht es, von einem Service-Einstiegspunkt ausgehend ganze Businessprozesse darzustellen. Die Beschreibungen der zeitlichen Abfolgen und Interaktionsmöglichkeiten werden als „Domain application protocols“ (DAP) bezeichnet [JW10, S. 95].
Das Vorgehen wird unter dem Begriff HATEOAS („Hypermedia as the engine of application state“) zusammengefasst. Dadurch, dass der Client den Zustand der Applikation steuern kann, kann der Web Service zustandslos sein, was z.B. für eine bessere Skalierbarkeit sorgt.

Werden eigene Hypermedia-Typen definiert, muss im HTTP Header der korrekte Content-Type angegeben werden. Während es hierfür standardisierte Formate wie „application/xml“, „image/jpeg“ und weitere gibt, existiert die Möglichkeit, benutzerdefinierte Formate anzugeben, wie beispielsweise „application/vnd.beispiel+xml“, wobei „vnd“ für Vendor (Englisch für Anbieter) steht.

Hypermedia wird ausführlich in [JW10] beschrieben.

2.3. Weitere Technologien

Weitere Technologien für den Datenaustausch in Netzwerken werden nicht auf die gleiche Weise mit dem Begriff Web Service verbunden, wie es mit WS-* und REST der Fall ist. Diese werden im Folgenden genannt.

Atom

Atom bezeichnet die zusammengehörigen Technologien Atom Syndication Format (ASF) [MN05] und Atom Publishing Protocol (APP) [JG07].
Während das ASF eine XML-Sprache ist, welche für Web-Feeds mit chronologisch geordneten Einträgen und damit Ressourcenlisten verwendet wird, sowie auch für einzelne Ressourcen verwendet werden kann, ist das APP ein auf HTTP basierendes Protokoll zur Erstellung und Aktualisierung von Ressourcen im Web.
Atom wurde als Alternative zu RSS entwickelt und von der IETF als Standard veröffentlicht (ASF 2005, APP 2007).
Atom wird in [LR07] und [JW10] als vorbildlich bezüglich seiner REST-Konformität bezeichnet. Es wird an der Möglichkeit für Abfragen (Querys) bei Atom Feeds gearbeitet. Hierfür soll die Feed Item Query Language (FIQL) eingesetzt werden [Not07].

GData

Mit GData (Google Data Protocol) hat Google ein eigenes Protokoll entwickelt, welches ähnlich wie Atom dem Abrufen, Erstellen und Manipulieren von Ressourcen dient, welche in Feeds strukturiert sind. Version 1.0 wurde vor der Veröffentlichung von APP fertiggestellt und Version 2.0 ist vollständig zu APP kompatibel [Goo12a].
Als Format kann sowohl ASF, als auch JSON verwendet werden. Eine Erweiterung gegenüber Atom im der gegenwärtigen Version ist die Möglichkeit, mit einer Abfrage (Query) an bestimmte Daten zu kommen [Wil08].

GData wurde für viele Services von Google verwendet, so z.B. für die API von Google Calendar, Google Contacts, Google Maps und andere [Goo12b]. Inzwischen sind jedoch viele der APIs als „deprecated“ (veraltet) markiert und GData wird nicht mehr verwendet.

OData

OData (Open Data Protocol) ist ein Datenzugriffsprotokoll, welches von Microsoft entwickelt und mittlerweile von der Standardisierungsorganisation OASIS weiterentwickelt wird. Es wurde entworfen um REST-basierte Datenservices zu erstellen und CRUD-Zugriff auf Ressourcen über URIs via HTTP zu ermöglichen [Mic13c, Kapitel 1.3 - Overview].

OData baut auf ASF und APP auf und erweitert diese [ODa13a, Kapitel 5 - Use of Atom] um ein Datenmodell (Entity Data Modell, EDM), welches typisiert oder nicht typisiert sein kann. Das EDM wird mit einer Common Schema Definition Language (CSDL) beschrieben [ODa13d].
OData Anfragen entsprechen den Merkmalen von Requests an RESTful Web Services. Als Datenformat kann ASF und JSON verwendet werden [ODa13f, 1.1 Representation formats and content type negotiation]. Wie im Unterkapitel 2.2.2 (S. 10 ff.) zu ROA und Hypermedia beschrieben, enthält ein von einem OData Dienst zurückgegebenes Dokument Links zu den von dort aus erreichbaren Ressourcen. Über die URI lassen sich Abfragen an den Service schicken, welche z.B. Filter- und Sortierungsangaben enthalten, sodass man neben Datensammlungen auch Teil- oder einzelne Datensätze erhalten kann, sowie auf einzelne Werte [ODa13g] zugreifen kann.

Durch die Standardisierung von OData besteht eine einheitliche Technologie um RESTful Web Services zu erstellen. Eine mögliche Folge dessen ist, dass OData Clients nur minimal geändert werden müssen, um mit OData-APIs anderer Web Services zu arbeiten. Die Firmen, die an der Weiterentwicklung des Standards beteiligt sind, erhoffen sich davon das Entstehen eines Ökosystems, was für jeden daran beteiligten Web Service von Vorteil sein soll:

> OData provides a way to break down data silos and increase the shared value of data by creating an ecosystem in which data consumers can interoperate with data producers in a way that is far more powerful than currently possible, enabling more applications to make sense of a broader set of data.

OData wird in einigen offenen Web-APIs verwendet, wie z.B. von netflix und ebay, sowie in Produkten wie z.B. IBM WebSphere, SAP NetWeaver Gateway, Sharepoint und SQL Server 2012 [ODa13e].

Besonders geeignet ist OData für die Offenlegung von ganzen relationalen Datenbanken, sodass auf diese per HTTP zugegriffen werden kann, ohne z.B. ODBC zu verwenden. Dies ist im Kontext dieser Arbeit besonders relevant.

Binärprotokolle

Die Verwendung von Binärprotokollen (teilweise auch „Data Interchange Protocols“ genannt) soll den Datenaustausch mithilfe einer schnelleren Daten(de-)serialisierung und kleineren Datenpaketen als z.B. bei Verwendung von XML als Datenübertragungsformat performanter machen.
Wie bei CORBA werden Daten und Schnittstellen mit einer IDL (Interface Description Language) beschrieben und mit einem Compiler Code für die Nutzung mit einer oder mehreren Programmiersprachen erzeugt.

Während XML und JSON als menschenlesbare Datenformate auf der einen Seite stehen, stehen dem gegenüber Binärprotokolle wie Protocol Buffers, BERT, BSON, Apache Thrift, Message Pack, Cisco's Etch, Caucho's Hessian, ZeroC's Internet Communications Engine (ICE), Apache Avro und andere.
Zwei dieser Binärprotokolle, welche in [Gup11] verglichen werden, seien im Folgenden beschrieben:

Protobuf

Protobuf (Protocol Buffers) ist wie GData eine Technologie von Google. Sie dient dem Encoding strukturierter Daten in einem effizienten und erweiterbaren Format und wird bei Google für nahezu alle internen RPC Protokolle und Formate verwendet [Goo13].
Es wurde entwickelt um gegenüber XML einfacher in der Benutzung, drei- bis zehnmal kleiner und 20 bis 100 mal schneller zu sein. Genutzt wird es seit 2001 und der Quellcode wurde 2008 veröffentlicht. [Var08], [Goo12c], [Goo12d]
Es wird z.B. verwendet bei Google in der internen Kommunikation, Open Streem Maps [OM13], ActiveMQ [HC13] und Riak [Tec13].

Apache Thrift

Apache Thrift besteht aus einer IDL für das Definieren und Erstellen von Datentypen und Services, sowie aus einem Compiler für unterschiedliche Programmiersprachen (C#, Erlang, Java, Python, Ruby und weitere [Fou12b]). Die Implementierung wurde 2007 von Facebook

öffentlich beschrieben.

Es wird z.B. verwendet bei Facebook in der internen Kommunikation, Cassandra (in der Vergangenheit - inzwischen wird das Cassandra Binary Protocol verwendet, siehe [Geh13], [Ell13]), Hadoop und HBase. Weitere Beispiele sind auf der Webseite von Apache Thrift genannt [Fou12a].
Es wird auch z.B in der öffentlichen API von Evernote verwendet, was relevant im Hinblick auf das Thema „Datenaustausch in mobilen Umgebungen“ ist. [Eng11]

Message Broker / Enterprise Integration

Protokolle wie AMQP, OpenWire und STOMP dienen zwar ebenfalls dem Datenaustausch, allerdings liegt hier der Fokus auf Message Broker Anwendungen wie z.B. ActiveMQ, Message-oriented Middlewares, Enterprise Service Bus und wird daher in dieser Arbeit nicht näher betrachtet.

2.4. BaaS

BaaS steht für Backend-as-a-Service und beschreibt den Dienst eines Anbieters, die gesamte Verwaltung eines Back-Ends auf einem Server zu übernehmen. Dies unterscheidet sich grundlegend von den zuvor genannten Technologien für den Datenaustausch, die mit Entwicklungsaufwand für ein serverseitiges System verbunden sind.

BaaS baut auf IaaS (Infrastructure-as-a-Service, Verwaltung von Hardwareressourcen durch den Anbieter) und PaaS (Platform-as-a-Service, Verwaltung einer Umgebung) auf. Je nach Sichtweise ist BaaS eine Form von oder baut zusätzlich auf SaaS (Software-as-a-Service, Anbieten einer Software als Dienst) auf. Diese Begriffe stehen für unterschiedliche Grenze zwischen den Bereichen, welche vom Dienstleister auf der einen und vom Nutzer auf der anderen Seite verwaltet werden. Abbildung 2.2 stellt dies grafisch dar.

In den meisten Fällen sind die Dienste der BaaS-Anbieter explizit an Entwickler mobiler Anwendungen gerichtet. Die Anbieter betreiben in der Regel einen Server, auf welchem eine Datenbank und eine Software betrieben wird, welche Datenspeicher und eine Benutzerverwaltung (mit Authentifizierung und Autorisierung) bietet, sowie oftmals Möglichkeit, mit serverseitigem Code eigene Businesslogik umzusetzen, eine Integration mit sozialen Netzwerken, Push-Notifications, ein CDN (Content-Delivery-Network), Datenanalyse und

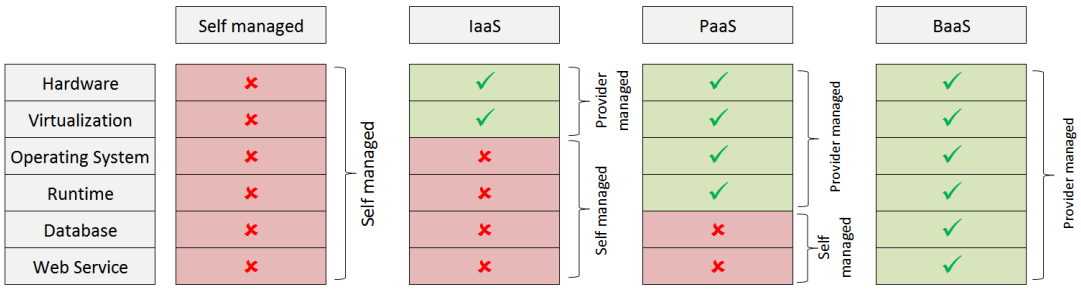

Abbildung 2.2.: as-a-Service Vergleich

Statistiken, Sicherheit (Verschlüsselung, Datenzugriffsberechtigungen, Backup), sowie API-Versionierung. Die Funktionalität wird Entwicklern mit Software Development Kits (SDK) und / oder APIs zur Verfügung gestellt.

Einige BaaS-Anbieter sind Parse.com (Mitte 2013 von Facebook akquiriert [Suk13]), Microsoft mit Azure Mobile Services, der Deutsche Anbieter apiOmat, Salesforce mit database.com und Firebase, welches ein Echtzeit-Back-End bietet.

2.5. Mobile Umgebungen

Bei der Entwicklung von Anwendungen für mobile Umgebungen existieren einige im Kontext dieser Arbeit relevante Einschränkungen und Kompensierungsansätze. Diese seien im Folgenden genannt.

2.5.1. Einschränkungen

Mobile Umgebungen haben einschränkende Eigenschaften, welche sich auf den Datenaustausch auswirken, sodass sich einige Technologien besonders dafür eignen, andere wiederum nur begrenzt von Nutzen sind. Diese Eigenschaften lassen sich an mobilen Endgeräten festmachen und sind nachstehend angegeben:

- Begrenzte Rechenkapazität / CPU-Geschwindigkeit
- Begrenzter Hauptspeicher
- Geringe und schwankende Netzwerkgeschwindigkeit
- Instabile Netzwerkverbindung (Verbindungsabbrüche)

2.5.2. Kompensierungsansätze

Es gibt folgende Möglichkeiten diese Limitierungen zu kompensieren:

Die Begrenzungen von Rechenkapazität und Hauptspeicher lassen sich durch Verlagern von Business-Logik und Berechnungen auf Server ausgleichen. Indem nur die Ergebnisse zum Client transferiert und dort auf einer nicht aufwändigen Oberfläche angezeigt werden, lässt sich die mobile Anwendung auf die Funktion als Benutzerschnittstelle reduzieren. Bei der Entwicklung von Algorithmen, welche auf dem Endgerät ausgeführt werden müssen, kann auf eine hohe Effizienz und geringe Komplexität geachtet werden. Weiterführende Literatur hierzu ist z.B. [Heu03].

Die Kompensierung der geringen Geschwindigkeit der Netzwerkverbindung lässt sich z.B. durch eine geringe Menge oder Größe von übertragenen Daten erreichen. Dies kann einerseits durch Datensparsamkeit erreicht werden. Andererseits können dazu die Daten bei der Serialisierung in ein kompakteres Übertragungsformat gebracht, ein effizienteres Übertragungsprotokoll verwendet oder bereits serialisierte Daten komprimiert werden. Binärprotokolle und die Komprimierung mit z.B. gzip sind hierfür Ansätze. Die Verwendung von gzip bei RESTful Web Services ist in [LR07, S. 247] beschrieben. Es muss darauf geachtet werden, dass eine komplexere (De-)Serialisierung wiederum zu höherem Rechenaufwand führen kann.
Eine weiterer Ansatz das Problem zu mildern ist Caching. Das Caching kann client- und serverseitig geschehen. Serverseitig kann die Art der Übertragung die Möglichkeit bieten, Antworten auf Anfragen in Knotenpunkten des Netzwerks zu speichern, so z.B. mit einem Proxy-Server bei HTTP GET-Anfragen. Bei zustandsverändernden Anfragen wie HTTP POST ist dies nicht möglich. Damit ist diese Art von Caching generell nicht möglich, wenn z.B. SOAP über HTTP verwendet wird. Clientseitig kann die Anwendung die zuvor erhaltenen Daten speichern und neue Anfragen nur dann durchführen, wenn die bisherigen Daten als veraltet angesehen werden. Das Thema Caching in mobilen Umgebungen wird näher in [Bar94] behandelt.
Auch eine Möglichkeit die Geschwindigkeit des Datenaustauschs zu erhöhen sind Kommunikationsverbindungen, welche über einen längeren Zeitraum aufrecht erhalten werden. Dabei entfällt der Schritt bei einer anstehenden Datenübertragung, dass erst eine Verbindung aufgebaut werden muss.
Verbindungen wie WebSockets ermöglichem es einem Server, Daten an einen Client zu schicken, ohne dass dieser zuvor eine entsprechende Anfrage gestellt haben muss [IF11]. Auch dies kann zu einer geringeren Menge an zu übertragenden Daten und damit zu einer höheren

Geschwindigkeit für den Erhalt von Informationen führen.

Das clientseitige Vorhalten zuvor erhaltener Daten dient auch dem Ausgleich von Verbindungsabbrüchen. Dazu werden bei unbekanntem Status, ob Daten veraltet sind oder nicht, oder auch wenn diese zuvor als veraltet gekennzeichnet wurden, weiterhin die vorhandenen Daten angezeigt.

Eine Kompensierung aller oben genannten Einschränkungen, welche nicht von der Entwicklung einer Anwendung ausgeht, ist die Weiterentwicklung von Technologien durch Hersteller mobiler Endgeräte und Netzwerkausrüster. Die Rechen- und speicherkapazität von modernen mobilen Endgeräten nimmt konstant zu, sodass die Grenze zu herkömmlichen Computersystemen verschwimmt. Auch die Geschwindigkeiten von mobilen Internetzugängen wird stetig erhöht. So sind im LTE-Mobilfunknetz Downloadgeschwindigkeiten möglich, welche die durchschnittliche Downloadgeschwindigkeit aller Internetzugänge in Deutschland übersteigt [Spi12], [Kan12].
Doch bei neuen Gerätekategorien wie intelligenten Uhren, Brillen, und Kleidungsstücken, welche weniger leistungsstarke Komponenten enthalten, gelten die Einschränkungen weiterhin.

Einige der hier genannten Punkte wie z.B die Effizienz der Algorithmen können nur vom Entwickler berücksichtigt werden, andere wie z.B. die Unterstützung von Komprimierung hängen von der verwendeten Technologieimplementierung (z.B. bestimmtes WS-* Framework) ab. Die Optimierungen durch den Entwickler wirken sich auf alle Technologien gleichermaßen aus, so dass diese für einen Vergleich nicht relevant sind.

3. Gegenüberstellung

Nach der Vorstellung der aktuellen Technologien zum Datenaustausch können diese nun in ihren Eigenschaften gegenübergestellt werden. Bei diesen Eigenschaften handelt es sich um z.B. durch eine Spezifikation festgelegte Eigenschaft (z.B. Datenformat) und nicht um daraus resultierende (z.B. Geschwindigkeit). Für letztere wird im weiteren Verlauf dieser Arbeit eine Testanwendung entwickelt.
Bei dieser Gegenüberstellung handelt es sich demnach nicht um eigens untersuchte Eigenschaften bzw. Funktionen, sondern um entweder durch die Spezifikation gegebene, durch bereits von Dritten untersuchte oder anderweitig bekannte.

Die Gegenüberstellung findet für folgende Technologien statt:
WS-* und REST werden betrachtet, weil diese für den Zweck des Datenaustauschs weit verbreitet sind. OData erweitert Atom, während GData nicht mehr verwendet wird, sodass hier OData als spezifische Technologie mit einbezogen werden kann. Für Binärprotokolle wird diese Kategorie verwendet und gemeinsame Eigenschaften unterschiedlicher Implementierungen betrachtet. Ebenso wird BaaS als Kategorie mit verglichen.

Zwar lässt sich eine Gegenüberstellung zwischen Protokollspezifikationen, Architekturstilen und Service-Konzepten nicht auf die gleiche Weise durchführen wie z.B. zwischen zwei Implementierungen einer Protokollspezifikation. Aber durch den Einsatz für den gleichen Zweck (Datenaustausch in Netzwerken) lässt sich ermitteln, welche Technologie welche Anforderungen an diesen Zweck auf welche Weise erfüllt.

Die Informationen entstammen dem Kapitel 2 (S. 4 ff.) und den Quellen [Mel10], [LR07], [JW10], [Fie00], [Fin13], [CP08], [Pre02], [Bay02], [Sch11], [Pau08a], [Pau08b], [Fla09b].

Im Einzelnen werden folgende Eigenschaften und Kategorien betrachtet, da diese als relevant für die Entwicklung von Applikationen in mobilen Umgebungen erachtet werden:
Protokolle, Sicherheit, Caching, Daten, Interoperabilität, Nachrichtenaustausch und Vorge-

	WS-*	REST	OData	Binär	BaaS
Anwendungsp.	SOAP	HTTP	OData	abh.	i.d.R. HTTP
Transportp.	HTTP, SMTP, TCP, ...	TCP	HTTP	TCP	TCP

Tabelle 3.1.: Gegenüberstellung Protokolle

hensweise bei der Implementierung.

Um eine tabellarische Übersicht zu ermöglichen wird Binärprotokolle mit „Binär“ abgekürzt. Wenn sich Eigenschaftsausprägungen bei unterschiedlichen Implementierungen oder Anbietern unterscheiden, wird dies „div.“ für „diverse“ abgekürzt.

3.1. Protokolle

Die genutzten Protokolle auf Anwendungs- und Transportebene werden verglichen, da sie bereits einige Eigenschaften mit sich bringen. So lässt sich z.B. bei Verwendung von HTTP auch HTTPS verwenden, wodurch ohne komplexe Eigenentwicklungen eine verschlüsselte Datenübertragung ermöglicht wird. Weiterführende Literatur zu Protokollen sind z.B. [W3C99], [Kle08] und [oSC81].

Tabelle 3.1 zeigt die Gegenüberstellung.

Da bei WS-* das gängigste Transportprotokoll HTTP ist, wird bei den folgenden Gegenüberstellungen hiervon ausgegangen. Das gleiche gilt für HTTP als Anwendungsprotokoll von BaaS.

3.2. Sicherheit

Das Thema Sicherheit umfasst sehr viele Aspekte. Hier wird nur auf die in Tabelle 3.2 gegenübergestellten eingegangen. Weiterführende Literatur sind z.B. [ea05] und [Tan12].

Erläuterungen zur Tabelle:
Die Verschlüsselung mit TLS findet bei Verwendung von HTTPS statt und ist mit den gängigen HTTP Servern ohne eigenen Entwicklungsaufwand möglich. Sie wird in den Fällen von WS-*, REST und OData demnach zwar nicht von den Technologien selbst durchgeführt, da diese

	WS-*	REST	OData	Binär	BaaS
Verschlüsselung	(TLS)	(TLS)	(TLS)	/	TLS
Authentifizierung	/	/	/	/	API-Key
weitere Mechanismen	WS-Security, WS-Trust	/	/	/	i.d.R. ACLs

Tabelle 3.2.: Gegenüberstellung Sicherheit

jedoch das verwendete Protokoll beeinflussen oder wie REST damit verwoben sind, ist dies mit aufgeführt.
HTTP bietet zwar HTTP Basic Authentication und HTTP Digest Authentication an ([ea99]), jedoch muss die Logik selbst implementiert werden, welche entscheidet ob eine Anfrage eines Benutzers angenommen oder abgelehnt wird.
ACL steht für Access Control List und ermöglicht es, den Zugriff auf einzelnen Daten auf bestimmte Benutzer zu beschränken und deren Zugriffsrechte zu beschränken.

3.3. Skalierbarkeit

Skalierbarkeit bezeichnet die Fähigkeit eines Systems, mit einer vermehrten Anzahl an Anfragen umgehen zu können. Diese Fähigkeit lässt sich durch vertikale und horizontale Skalierung erreichen. Vertikale Skalierung ist z.B. das Erhöhen der Rechengeschwindigkeit durch ersetzen eines Prozessors durch einen leistungsstärkeren. Horizontale Skalierung bezeichnet hingegen z.B. das Erhöhen der Rechengeschwindigkeit durch Hinzufügen eines weiteren Servers, so dass die Last durch eingehende Anfragen auf mehr Server verteilt wird. In modernen Systemen, mit denen die Kommunikation über das Internet stattfindet, wird i.d.R. horizontal skaliert, da sich damit u.a. die Möglichkeit ergibt, Server bei Lastspitzen zu- und ansonsten abzuschalten.

Bei einer Erhöhung der Anzahl an Anfragen an einen Server gibt es zwei Flaschenhälse: Die Schnittstelle, welche die Anfragen annimmt und bearbeitet, sowie die Datenhaltungsschicht, welche z.B. auf eine Datenbank zugreift. Bei dieser Gegenüberstellung wird nur die Skalierbarkeit bzgl. ersterem betrachtet, da letztere nicht abhängig von der verwendeten Schnittstellentechnologie ist.

Zwei Eigenschaften, welche eine horizontale Skalierung ermöglichen bzw. vereinfachen sind Caching und Zustandslosigkeit. Unter Caching wird hier das Vorhalten von Antworten auf häufig erhaltene Anfragen verstanden. Die Zustandslosigkeit betrifft die Sitzung auf

	WS-*	REST	OData	Binär	BaaS
Caching	/	✓	✓	/	?
Zustandslos	(✓)	✓	✓	?	✓

Tabelle 3.3.: Gegenüberstellung Caching

Protokollebene, nicht den Zustand von Daten.

Eine detailliertere Einführung in das Thema Skalierbarkeit bietet entsprechende Literatur wie z.B. [DK13].
Tabelle 3.3 zeigt die Gegenüberstellung.

Erläuterungen zur Tabelle:
Wird HTTP als Transportprotokoll für WS-* verwendet, wird immer HTTP POST genutzt. Da HTTP POST zustandsverändernd ist, kann hier nicht gecacht werden. Bei REST ist Caching wie bei Webseiten möglich und kann durch Conditional GET, Partial GET und „Look-Before-You-Leap Requests“ [LR07] noch verbessert werden. Bei BaaS hat man keine Kontrolle darüber, ob Caching verwendet wird oder nicht. Es wird davon ausgegangen, dass die Anbieter Caching verwenden.
Mit WS-* sind zustandsbehaftete und -lose Interaktionen möglich (siehe WS-Resource). Bei REST sind die Interaktionen mit Ressourcen generell zustandslos. Zustände können durch Techniken wie URI-Parameter und Cookies ermöglicht werden, jedoch entspricht diese im Web häufig verwendete Möglichkeit nicht den Vorgaben einer RESTful API. I.d.R. überlässt der Server dem Client die Verwaltung von Zuständen, was durch HATEOAS ermöglicht wird (siehe Unterkapitel 2.2.2 (S. 14)). Bei Binärprotokollen ist dies implementierungsabhängig. Bei BaaS sind i.d.R. nur zustandslose Interaktion möglich, da man keine serverseitige Kontrolle über z.B. URI-Parameter oder Cookies hat.

3.4. Daten

Einige der Technologien sind auf ein Übertragungsformat beschränkt oder an ein Schema gebunden. Relevant für die Entwicklung mit stark typisierten Programmiersprachen ist auch, ob die entsprechenden Typen ebenfalls von der Datenübertragungsmethode unterstützt werden. Ebenso ist wichtig, dass Binärdaten wie Bilder und Audiodateien übertragen werden

	WS-*	REST	OData	Binär	BaaS
Übertragungsf.	XML	XML, JSON, ...	ASF, JSON	Binär	i.d.R. JSON
Schema	SOAP	/	OData	div.	/
Typen	✓	(✓)	✓	(✓)	✓
Binärdaten	✓	✓	✓	✓	✓

Tabelle 3.4.: Gegenüberstellung Daten

können.

Tabelle 3.4 zeigt die Gegenüberstellung.

Erläuterungen zur Tabelle:
Bei Binärprotokollen wird die Serialisierung von innerhalb der Programmiersprache verwendeten Objekten von der Binärprotokoll-Bibliothek übernommen. Ob dabei ein bestimmtes Schema verwendet wird, ist abhängig vom spezifischen Binärprotokoll.
Eine Typunterstützung hängt bei REST davon ab, welches Übertragungsformat verwendet wird. Beim häufig verwendeten JSON ist nur eine sehr geringe Unterstützung von Datentypen gegeben [jso13]. Auch bei Binärprotokollen unterstützt die IDL i.d.R. nur wenige Datentypen.

3.5. Interoperabilität

Unter Interoperabilität fallen die Eigenschaften, welche beim Einsatz in heterogenen Systemen wichtig sind. Dies sind das Vorhandensein einer Beschreibungssprache für die automatische, maschinellen Integration eines unbekannten Dienstes zur Laufzeit, sowie eines Verzeichnisdienstes. Auch die Art des Interfaces fällt darunter. Ein weiterer Punkt sind standardisierte Erweiterungen. Wie sich die Dienste bei der Entwicklung von Clients einsetzen lassen ist ebenfalls relevant.

Tabelle 3.5 zeigt die Gegenüberstellung.

	WS-*	REST	OData	Binär	BaaS
Beschr.	WSDL	WADL, WSDL 2	CSDL	/	/
Verz.	UDDI	/	/	/	/
Interf.	Ein Service-punkt, var. Prozeduren	Var. Ressourcen, jede Ressource adressierbar, Uniform Inter-face (HTTP)	Var. Ressourcen, jede Ressource adressierbar, Uniform Inter-face (HTTP), Abfragen (Querys)	Ein Service-punkt, var. Prozeduren	div.
Erw.	viele	/	/	/	/
Entw.	Toolkits	HTTP Client	HTTP Cli-ent, Client Bibliotheken	Compiler-abhängig	SDK, HTTP Client

Tabelle 3.5.: Gegenüberstellung Interoperabilität

	WS-*	REST	OData	Binär	BaaS
Zust.	✓	/	/	/	/
QoS	✓	/	/	/	/
Trans.	✓	✓	(/)	/	/
Fehler	✓	entspr. HTTP	entspr. HTTP	/	/

Tabelle 3.6.: Gegenüberstellung Interoperabilität

3.6. Nachrichtenaustausch

Unter dieser Kategorie wird gegenübergestellt, ob Mechanismen für die sichere Zustellung von Nachrichten existieren, Qualitätsangaben (QoS) machbar sind, Transaktionen möglich sind und ob es standardisierte und frei definierbare Fehlermeldungen gibt.
Tabelle 3.6 zeigt die Gegenüberstellung.

Erläuterungen zur Tabelle:
Für OData gibt es keinen als Standard festgelegten Weg, wie Transaktionen durchgeführt werden können. Wenn die Anfragen nicht an ein verteiltes System, sondern nur an mehrere Ressourcen eines Systems gerichtet sind, lässt sich jedoch eine „Alles-oder-nichts“-Transaktion mit OData Batch durchführen [ODa13b].

	WS-*	REST	OData	Binär	BaaS
Entw.	Code-First, Contract-First	Code	oft DB-Freigabe, sonst Code	Contract-First	div.

Tabelle 3.7.: Gegenüberstellung Interoperabilität

3.7. Entwicklung

In Tabelle 3.7 wird die Vorgehensweise bei der Implementierung gegenübergestellt.

Erläuterungen zur Tabelle:
Bei REST sind WSDL / WADL als Contract und entsprechende Tools noch nicht ausgereift. Anfangs müssen z.B. die Ressourcen identifiziert und das URI-Design und Payload-Format festgelegt werden. Bei OData kann eine ganze Datenbank freigegeben werden. Für einen granularer definierten Dienstzugriff müssen Anpassungen im Programmcode vorgenommen werden. Bei BaaS werden i.d.R. Tabellen über eine Verwaltungs-Webapp erstellt. Einige Dienste bieten daraufhin ein entsprechend generiertes SDK für den Zugriff von Clientseite an. Andere bieten eine generische API.

Weiteres

Eine Bewertung wird an dieser Stelle noch nicht vorgenommen. Auf die Gegenüberstellung wird sich erst in Unterkapitel 5.5 (S. 87) bezogen.

Zur objektiven Gegenüberstellung von Implementierungsaufwand, Geschwindigkeit und anderen Aspekten kann erst nach Entwicklung der Beispielanwendung und entsprechenden Benchmarks etwas gesagt werden. Weitere Kriterien werden im Unterkapitel 4.1.1 (S. 31 ff.) genannt.

Es gibt Versuche, die Komplexität von SOAP und REST objektiv gegenüberzustellen. So wird in [CP08] nach einer Untersuchung darauf geschlossen, das man bei einem RESTful Web Service zwar weniger Entscheidungen treffen muss, bei welchen es außerdem weniger Alternativen gibt (was als „freedom from choice“ gegenüber „freedom of choice“ bezeichnet wird), dafür jedoch einige Funktionen nur mit eigener Implementierung ermöglicht werden.

Weitere Vergleiche zwischen WS-* und REST finden sich in [LR07, Kapitel 10] und [JW10, S. 375].

4. Entwicklung einer Testanwendung

Um weitergehende Gegenüberstellungen und Vergleiche zu ermöglichen, werden die im Kapitel Gegenüberstellung genannten Technologien beispielhaft implementiert.

Die folgenden Unterkapitel basieren auf den getrennten Softwareentwicklungsphasen, wie sie in [Ste00] identifiziert werden. In der Analyse werden Vorüberlegungen angestellt, die eine grobe Richtung für die zu entwickelnden Systeme angeben. Im Unterkapitel Konzept wird vermittelt, **was** das System leisten soll - es werden konkrete Anforderungen genannt, Anwendungsfälle aufgelistet, die fachliche Architektur erstellt und damit Inhalte eines Lastenheftes festgelegt. Im Unterkapitel Design wird beschrieben, **wie** die zuvor genannten Anforderungen erfüllt werden - es wird auf die technische Architektur eingegangen sowie Klassen- und Sequenzdiagramme angegeben und damit Inhalte eines Pflichtenheftes bestimmt. Im Unterkapitel Implementierung und Test wird beschrieben, wie die Systeme entsprechend der vorherigen Angaben entwickelt und getestet wurden, sowie welche Optimierungsmöglichkeiten bestehen. Da der Vergleich den Datenaustausch betrifft, wird weder auf die mobile Anwendung, noch auf den zentralen Dienst der Fokus gelegt, sondern beide gleichwertig behandelt. Bei einem Datenaustausch existieren immer ein Sender und ein Empfänger und die verwendete Technologie betrifft beide Seiten.

4.1. Analyse

Die beispielhaft zu entwickelnden Systeme sollen möglichst repräsentativ sein, damit die spätere Bewertung einen weiten Geltungsbereich hat. Dazu wurden als Gemeinsamkeiten vieler mobiler Applikationen folgende Eigenschaften ermittelt: Es gibt eine grafische Benutzeroberfläche, welche Daten anzeigt. Die Daten werden von einem Server abgerufen. Es gibt Daten, die der Benutzer erstellen, verändern oder löschen kann. Eine entsprechende Aktion wird dem Server mitgeteilt, so dass der Server als zentraler, stets aktueller Datenspeicher agiert. Dies sind die Minimalanforderungen an die Systeme bzgl. deren Funktionalität. Da die Beispielanwendung nicht für den Produktiveinsatz gedacht ist und eine hohe Komplexität

einen Vergleich erschwert, ist eine weitere Anforderung, dass die Systeme möglichst simpel sind.

Ein populärer Ansatz für den Vergleich unterschiedlicher Technologien anhand der Entwicklung einer einfachen Anwendung ist TodoMVC. Dies ist eine Webseite, welche verschiedene Javascript-Frameworks miteinander vergleicht, indem es eine To-do-Anwendung in allen Frameworks implementiert.
Da diese Art von Anwendung die zuvor identifizierten Minimalanforderungen erfüllt und gleichzeitig sehr simpel ist, wird dieser Ansatz übernommen. In diesem Fall wird sowohl die Client- als auch die Serverseite der To-do-Anwendung in den fünf Technologien implementiert, welche im Kapitel Gegenüberstellung ausgewählten wurden. Zusätzlich dazu soll es eine Möglichkeit geben, die Kommunikationsanfragen von Seiten der Anwendung automatisch durchzuführen, sowie währenddessen auf Client- und Serverseite Messungen durchzuführen.

4.1.1. Vergleichskriterien

Da das zu entwickelnde System das Ziel hat, die unterschiedlichen Technologien zu vergleichen, müssen hierfür Vergleichskriterien festgelegt werden, damit die Softwareentwicklung zielgerichtet stattfinden kann.

Mögliche Vergleichskriterien

Mögliche Vergleichskriterien sind zusätzlich zu den in Kapitel 3 (S. 22 ff.) betrachteten Eigenschaften diejenigen, welche sich durch Messungen bestimmen lassen:
Antwortzeiten, Datenübertragungsvolumen, Bearbeitungszeiten, Verfügbarkeit (Uptime), maximale Anzahl an Clients, deren Anfragen parallel bearbeitet werden können und die CPU-Auslastung von Client und Server. Auch der Implementierungsaufwand ist objektiv bestimmbar.

Einige subjektive Aspekte sind ebenfalls mögliche Kriterien:
Die empfundene Einfachheit die Technologien zu verstehen sowie umzusetzen, Zukunftsfähigkeit, Anpassbarkeit, Wartbarkeit und generelle Handhabung.

Auswahl Vergleichskriterien

Die im Kapitel 3 (S. 22 ff.) genannten Kriterien sind bereits untersucht und der Vergleich entsprechend gegeben. Die folgende Softwareentwicklung wird daher auf die messbaren Kriterien beschränkt. Die subjektiven Aspekte werden anschließend zwar ebenfalls betrachtet, jedoch

werden keine Softwarekomponenten für die Messung benötigt. Es wird darauf geachtet, die Ausprägungen der betrachteten Kriterien wie z.B. Wartbarkeit nicht von der Implementierung abhängig zu machen, sondern die Unterschiede zwischen den Implementierungen so gering zu halten, dass sich die Vergleiche auf die Technologien beziehen.

4.1.2. Technologieauswahl

Für die Entwicklung von beispielhaften Systemen mit den in Kapitel Gegenüberstellung verglichenen Technologien müssen eine möglichst einheitliche Plattform für Client und Server gewählt werden, sowie Frameworks für die serverseitige Implementierung, ggf. Clientbibliotheken, ein spezifisches Binärprotokoll und ein BaaS-Anbieter. Im Folgenden werden erst die Möglichkeiten und Anschließend die Auswahl beschrieben.

Clientplattform

Für die mobile Anwendung bieten sich folgende Plattformen an:

- Android
- iOS
- Windows Phone
- WinRT
- BlackBerry

Andere Plattformen wie WebOS, Sailfish OS, Firefox OS und Ubuntu Phone haben derzeit einen zu geringen Marktanteil um repräsentativ zu sein [IDC13].

Serverplattform und Framework

Serverseitig wird ein Back-End entwickelt. Für die Entwicklung von einem WS-* und einem REST-konformem Dienst bieten sich viele Frameworks für populäre Programmiersprachen ([Sof13], [Lan13]) an. Einige sind in Tabelle 4.1 aufgelistet.

Für OData existieren Bibliotheken für die server- und clientseitige Entwicklung, jedoch wird es nur von sehr wenigen Web Service Frameworks unterstützt. In [ODa13c] sind nur für Java, .NET und node.js sog. OData Producer gelistet.

	WS-*	REST
Java	JAX-WS Implementierungen wie Apache Axis2, Apache CXF	JAX-RS Implementierungen wie Apache Wink, Jersey, JBoss RESTEasy, Restlet; andere REST-Frameworks wie Spring (mit entsprechendem Modul), Play, restfulie
.NET	WCF Services, ServiceStack	ASP.NET Web API, Nancy, FubuMVC, OpenRasta
PHP	WSO2 WSF/PHP, Zend Framework	WSO2 WSF/PHP, Zend Framework, Tonic
Python	Zolera SOAP Infrastructure	Django, Flask
Ruby	soap4r, simplews	Rails, Sinatra

Tabelle 4.1.: Web Service Frameworks

Binärprotokolle und BaaS-Anbieter wurden bereits in den Grundlagen vorgestellt. Binärprotokolle bieten mit ihren Compilern die Möglichkeit, Code für eine Vielzahl von Programmiersprachen zu generieren. BaaS-Anbieter stellen entsprechende Client-Bibliotheken bereit.

Auswahl

Es sollten möglichst gleichartige Umgebungen geschaffen werden, damit Messergebnisse nicht durch Einflüsse verfälscht werden, die nichts mit der eigentlichen Technologie zu tun haben. Ein Beispiel für so einen möglichen Einfluss ist die unterschiedliche Ausführungsgeschwindigkeit von Code durch einen Interpreter und Bytecode durch eine Laufzeitumgebung. Daher sollten nur eine Plattform / Programmiersprache verwendet werden, und die für diese verfügbaren Technologieimplementierungen (Frameworks, Clientbibliotheken, ...).

Mehrere Gründe sprechen für die Entwicklung mit .NET:

- Das Unternehmen, in dessen Kooperation diese Bachelorarbeit entsteht, entwickelt größtenteils mit Technologien von Microsoft.
- Mit Technologien für .NET lassen sich alle bisher genannten Möglichkeiten realisieren.
- Mit .NET steht eine einheitliche Plattform zur Verfügung, sodass eine gute Vergleichbarkeit gegeben ist.

Als Clientplattform kommt WinRT im Rahmen einer Windows Store App für Windows 8 zum Einsatz. Ein simpler Vorteil gegenüber Windows Phone liegt darin, dass kein weiteres Gerät zusätzlich zum Entwicklungsrechner benötigt wird. Für die Implementierung der WS-* und REST Dienste werden die von Microsoft entwickelten und unterstützten Frameworks WCF Services und ASP.NET Web API ausgewählt. Als OData Produzent dient ein mit WCF Data Services entwickelter Dienst. Bei den am weitesten verbreiteten Binärprotokollen bietet Apache Thrift als einziges die Möglichkeit, Service Endpunkte zu generieren und unterstützt C#, so dass dieses verwendet wird. Als BaaS-Anbieter wird Microsoft mit seinem Dienst Azure Mobile Services ausgewählt. Der Grund wird in folgendem Abschnitt erläutert, welcher auch auf das Hosting der anderen vier Dienste eingeht.

Für das Deployment der Dienste muss ebenfalls eine einheitliche Umgebung gewählt werden, damit die Vergleichbarkeit der einzelnen Technologien gegeben ist. Hierfür bietet sich ein virtueller Server eines IaaS-Anbieters an, da dies einem realistischen Szenario für das Anbieten eines Webservices entspricht. Zwar lässt sich z.B. ein WCF Services Dienst auf drei verschiedene Arten deployen (Self-hosted bei einem PaaS-Provider, auf IIS aufbauend bei einem Webseiten-Host und auf einer virtuellen Maschine bei einem IaaS-Provider), um gleiche Bedingungen zu schaffen eignet sich jedoch am besten die virtuelle Maschine, da sich hier die Implementierungen aller oben genannten Technologien nutzen lassen. Die Wahl des IaaS-Providers fällt auf Azure, da dort mit Mobile Services auch ein BaaS zur Verfügung steht, dessen Server sich in den gleichen Rechenzentren befinden, wie die Maschinen, auf denen die Virtuellen Maschinen laufen und damit die Anbindung gleich ist, was der Vergleichbarkeit zuträglich ist.

Die Auswahl von Clientbibliotheken ist durch die Verwendung von WinRT vorgegeben. Für WCF Services und WCF Data Services können Servicereferenzen verwendet werden, für ASP.NET Web API kann die in .NET Klassenbibliothek für Windows Store Apps integrierte HttpClient Klasse innerhalb einer eigenen Clientklasse verwendet werden, für Apache Thrift wird Code für den Client erzeugt und für die Verwendung von Azure Mobile Services wird eine entsprechende Bibliothek bereitgestellt.

Folgende Liste bietet eine Übersicht der Technologien und die in der folgenden Entwicklung verwendeten Technologieimplementierungen:

- Plattform: Server: .NET; Client: WinRT

- WS-*: WCF Services
- REST: ASP.NET Web API
- OData: WCF Data Services
- Binärprotokoll: Apache Thrift
- BaaS: Azure Mobile Services

4.1.3. Bestehende Lösungen

Es gibt einige Benchmarks für den Vergleich von Datenübertragungstechnologien. Nur wenige berücksichtigen dabei die Vielzahl an Technologien, die in dieser Arbeit untersucht werden. Der Vergleich in [Ani13] beschreibt die Testumgebung nur ungenau und keinen Versuchsaufbau. Ein Vergleich von Serialisierungsgeschwindigkeiten bei Verwendung von für die JVM verfügbaren Technologien findet bei [Ani12] statt. Eine Performanzmessung von SOAP gegenüber REST wird z.B. bei [Jen08] unternommen.

4.2. Konzept

In der Konzeption wird festgelegt, was die Anwendung leisten soll. Es werden demnach Anwendungsfälle, funktionale und nicht-funktionale Anforderungen angegeben, sowie die fachliche Architektur bestimmt. Das Ergebnis entspricht Inhalten eines Lastenhefts.

Da es erst im Design um die technischen Implementierungen und Details geht, wird hier noch nicht auf die unterschiedlichen Technologien eingegangen.

4.2.1. Anwendungsfälle

Um die Komplexität der Anwendungsfälle und Aktivitäten auf den wesentlichen Ablauf zu begrenzen, werden z.B. mögliche Kommunikationsprobleme, Sicherheitsaspekte und Objektflüsse nicht explizit modelliert.

Übersicht

Die mobile Anwendung soll dem Benutzer ermöglichen, eine To-do-Liste zu verwalten. Die Daten sollen auf einem zentralen Datenspeicher gehalten werden und über einen Dienst erreichbar sein, so dass von unterschiedlichen Anwendungen und Geräten auf diese Daten

zugegriffen werden kann, sowie die Möglichkeit des Datenverlustes auf dem Client durch Soft- oder Hardwarefehler oder Diebstahl ausgeschlossen wird. Diese Art des Szenarios entspricht dem vieler aktueller Anwendungen auf mobilen Geräten.

Aus diesen Grundanforderungen ergeben sich folgende Anwendungsfälle für den Benutzer auf Anwendungsseite:

1. To-do-Liste erstellen
2. To-do-Liste anzeigen
3. To-do-Item erstellen
4. To-do-Item als erledigt markieren
5. To-do-Item bearbeiten
6. To-do-Item löschen

Diese Anwendungsfälle sind an der Benutzung durch eine Person ausgerichtet, könnten in den Beschreibungen jedoch den Zusammenhang zu den dienstseitigen Aktionen beinhalten, da diese jeweils durch eine Aktion des Benutzers in der Anwendung ausgelöst werden. Damit würden separate Anwendungsfälle für den Dienst entfallen. Da es sich jedoch um völlig voneinander getrennte Systeme handelt und der Dienst auch mit anders konzeptionierten Clientanwendungen kommunizieren können soll, wird der Dienst separat modelliert. Für diese Modellierung der Anwendungsfälle wird als Sichtweise für die Interaktion nicht die des Benutzers einer Clientanwendung, sondern eines beliebigen Clients gewählt. Somit enthält „laden“ das Laden aus einem Speicher, sowie das zurücksenden an den Client, der die Anfrage gestellt hat - ähnlich dem obigen Anwendungsfall „anzeigen“, welches ein vorhergegangenes Laden vom Server beinhaltet. Ebenso ist auf Seiten des Dienstes aus fachlicher Sicht nicht zwischen „als erledigt markieren“ und „bearbeiten“ zu unterscheiden. Damit ergeben sich für den Dienst folgende Anwendungsfälle:

1. To-do-Liste speichern
2. To-do-Liste laden
3. To-do-Item speichern
4. To-do-Item bearbeiten

5. To-do-Item löschen

Zusätzlich zu der Benutzeranwendung und dem Dienst gibt es noch das Messsystem, welches ebenso separat modelliert wird. Dabei wird das Durchführen einer Messung vom Benutzer initiiert und ist nur Clientseitig relevant, während die einzelnen Messungen sowohl client- als auch dienstseitig für jeden oben genannten Anwendungsfall durchgeführt werden:

1. Messung durchführen
2. Zeit messen
3. Datenübertragungsvolumen messen
4. CPU-Auslastung messen

Grafisch dargestellt (Abb. 4.1) werden die Beziehungen deutlich.

Bei diesem Diagramm werden abweichend vom UML-Standard includes- und extend-Verbindungen von Anwendungsfällen nicht nur zu einzelnen anderen Anwendungsfällen, sondern zu Komponenten verwendet. Diese bedeuten, dass die Verbindungen zu jedem der enthaltenen Elemente gehen und dienen der Übersichtlichkeit.

Die Liste aller Anwendungsfälle mit den dazugehörigen Beschreibungen befindet sich im Anhang.

Beispiel: UC S3 To-do-Item speichern

Beschreibung: Ein To-do-Item wird gespeichert
Beteiligte Akteure: /
Verwendete Anwendungsfälle: /
Erweiternde Anwendungsfälle: UC M2 Zeit messen, UC M3 Datenübertragungsvolumen messen, UC M4 CPU-Auslastung messen
Erweiterte Anwendungsfälle: /
Auslöser: Der Server bekommt eine Anfrage zum Speichern eines To-do-Items
Vorbedingungen: Eine To-do-Liste wurde mit der Kennung gespeichert, die in dieser Anfrage für die Identifikation der Liste enthalten ist
Ergebnis: Das To-do-Item wurde in die angegebene To-do-Liste gespeichert und dem Absender der Anfrage die zugehörige Kennung zur Identifikation des To-do-Items zugesendet.
Standardablauf: Der Server bekommt eine Anfrage zum Speichern eines To-do-Items. In der

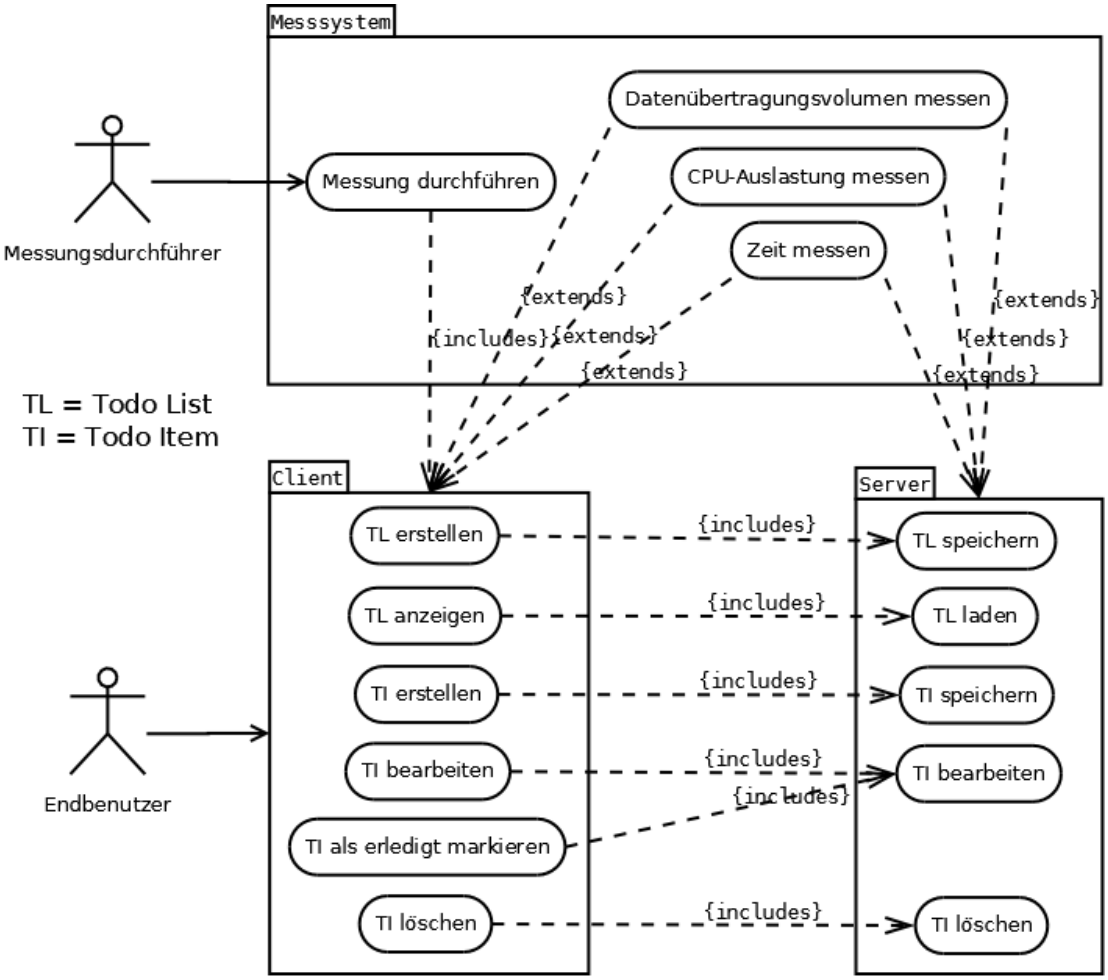

Abbildung 4.1.: Anwendungsfalldiagramm

Name	Beschreibung
FR C1	Nachdem der Benutzer die Anwendung startet, erscheint der Startbildschirm. Dort hat er die Möglichkeit zu wählen, mit welcher Technologie er eine manuelle Verwaltung einer To-do-Liste vornehmen möchte, und er kann eine Messung durchführen.
FR C2	Wenn der Benutzer eine Technologie zur manuellen Verwaltung einer To-do-Liste ausgewählt hat, wird die To-do-Liste des Benutzers angezeigt. Wenn die Anwendung zum ersten Mal gestartet wurde, wird dem Server mitgeteilt, dass eine neue To-do-Liste angelegt werden soll und es wird eine leere Liste angezeigt. Wenn die Anwendung zuvor bereits gestartete wurde, wird die zuvor verwendete To-do-Liste abgerufen und angezeigt. Dieser Vorgang ist von der gewählten Technologie unabhängig.
FR C3	Wenn eine To-do-Liste angezeigt wird, kann der Benutzer diese verwalten, d.h. To-do-Items erstellen, bearbeiten, als erledigt markieren und löschen.
FR C4	Der Benutzer hat die Möglichkeit, aus der To-do-Verwaltung zum Startbildschirm zurückzukehren.

Tabelle 4.2.: Funktionale Anforderungen Client

Anfrage ist die Kennung zur Identifizierung der Liste enthalten, in die das To-do-Item gespeichert werden soll. Der Server ermittelt eine Kennung für das To-do-Item, welches noch nicht in der Liste verwendet wird. Der Server speichert das To-do-Item in die Liste. Der Server schickt dem Absender der Anfrage die Kennung des To-do-Items.

4.2.2. Anforderungen

Die Namen richten sich nach folgendem Schema:

- FR: Functional requirement; NFR: Non-functional requirement
- C: Client; S: Server; M: Measuring system
- <number>: Fortlaufende Nummer

Client

Funktionale Anforderungen Die funktionalen Anforderungen des Clients sind in Tabelle 4.2 angegeben.

Nichtfunktionale Anforderungen Die nichtfunktionalen Anforderungen des Clients sind in Tabelle 4.3 angegeben.

Name	Beschreibung
NFR C1	Die Benutzeroberfläche ist einfach und übersichtlich.
NFR C2	Die Benutzeroberfläche ist responsiv.

Tabelle 4.3.: Nichtfunktionale Anforderungen Client

Name	Beschreibung
FR S1	Der Server bietet eine zentrale Verwaltung von To-do-Listen. Auf diese Verwaltung kann von Clients über Dienste zugegriffen werden.
FR S2	Die Dienste der unterschiedlichen zu vergleichenden Technologien laufen parallel, so dass sie nicht nacheinander manuell gestartet und beendet werden müssen.
FR S3	Die Verwaltungsdienste umfassen die Möglichkeit, To-do-Listen zu erstellen, abzurufen, sowie To-do-Items zu erstellen, zu bearbeiten und zu löschen.
FR S4	Die Dienste der unterschiedlichen Technologien greifen auf den gleichen zentralen Speicher zu, so dass einem Client immer die Verwaltung ein und derselben To-do-Liste ermöglicht wird, unabhängig von der gewählten Technologie.

Tabelle 4.4.: Funktionale Anforderungen Server

Server

Funktionale Anforderungen Die funktionalen Anforderungen des Servers sind in Tabelle 4.4 angegeben.

Nichtfunktionale Anforderungen Die nichtfunktionalen Anforderungen des Servers sind in Tabelle 4.5 angegeben.

Messsystem

Funktionale Anforderungen Die funktionalen Anforderungen des Messystems sind in Tabelle 4.6 angegeben.

Name	Beschreibung
NFR S1	Die Dienste sind jederzeit verfügbar, sofern auch der Server verfügbar ist.

Tabelle 4.5.: Nichtfunktionale Anforderungen Server

Name	Beschreibung
FR M1	Das Messsystem führt Messungen sowohl client- als auch serverseitig durch.
FR M2	Die Messung betrifft die verstrichene Zeit von einer Anfrage bis zu einer Antwort, das Datenübertragungsvolumen und die CPU-Auslastung.
FR M3	Die Messung findet über einen bestimmten Zeitraum statt, woraufhin zusammengefasste Daten angezeigt oder gespeichert werden.

Tabelle 4.6.: Funktionale Anforderungen Messsystem

Name	Beschreibung
NFR M1	Die gemessenen Werte stimmen mit den realen Werten überein.

Tabelle 4.7.: Nichtfunktionale Anforderungen Messsystem

Nichtfunktionale Anforderungen Die nichtfunktionalen Anforderungen an das Messsystem sind in Tabelle 4.7 angegeben.

Qualitätsanforderungen

In der folgenden Liste werden die Qualitätsanforderungen nach ISO/IEC 9126-1 angegeben, sowie die Beschreibung, inwiefern diese Anforderung für das zu entwickelnde System gestellt wird. Eine Reihe von Qualitätsanforderungen oder Unteranforderungen werden nicht gestellt, was in allen Fällen in folgender Besonderheit begründet ist:
Das System wird für eine einmalige Evaluation anhand beispielhafter Implementierungen entwickelt und läuft auch nur für den Zeitraum der Evaluation.

- Functionality - Die funktionalen Anforderungen sollen erfüllt werden. Sicherheit kann vernachlässigt werden, die Interoperabilität zwischen den Systemen ist entsprechend den funktionalen Anforderungen zu implementieren, das System als Ganzes hat jedoch keine Schnittstellen nach außen. Die Genauigkeit betreffend der Messergebnisse wird entsprechend NFR M1 gefordert.
- Reliability - Aufgrund der Beispielhaftigkeit ist keine Ausgereiftheit gefordert. Aus dem oben angegebenen Grund kann auch auf Fehlertoleranz und Wiederherstellbarkeit verzichtet werden.
- Usability - Das System soll nachvollziehbar sein, damit die Beurteilung transparent ist. Aus dem oben genannten Grund wird auf Erlernbarkeit, Operierbarkeit und Attraktivität verzichtet.

- Efficiency - Für den Ressourcenverbrauch kann keine allgemeine Anforderung gestellt werden, da dieser bei jeder Implementierung unterschiedlich ist und einen Aspekt für den Vergleich darstellt. Bezüglich Performanz: Eine Vergleichbarkeit ist bei einfacher Implementierung der unterschiedlichen Technologien gegeben. Würde man die Performanz einer der Implementierungen verbessern, müsste man dies bei allen tun. Da die Performanzoptimierungsmöglichkeiten von einer Technologie bereits umfangreich sind, wäre dies zu zeitaufwändig für den Rahmen einer Bachelorarbeit. Des weiteren ist fraglich, ob sich bei einer Performanzoptimierung aller Systeme die relativen Unterschiede der jeweiligen Messwerte zu den Messwerten eines nicht performanzoptimierten Systems stark unterscheiden würden.
- Maintainability - Aufgrund der Einmaligkeit der Evaluierung ist eine Wartbarkeit nicht gefordert.
- Portability - Ein Durchführen der Tests auf unterschiedlichen Betriebssystemen wäre interessant, ist jedoch nicht Ziel der Untersuchung, da nicht Betriebssysteme, sondern Datenaustauschtechnologien verglichen werden sollen.

4.2.3. Fachliche Komponenten

Die Bausteinsicht bietet eine grobe bis feine Übersicht über die Bausteine des Systems. Die Bausteine können dabei Subsysteme, Schichten, fachliche und technische Komponenten sein. In einem Komponentendiagramm (Abb. 4.2) wird eine grobe Übersicht über die entsprechend der Anwendungsfälle und Anforderungen ermittelten fachlichen Komponenten dargestellt.

4.3. Design

Im Design wird festgelegt, wie die Anwendung die fachlichen Anforderungen lösen soll. Es werden Klassendiagramme und die technische Architektur bestimmt. Das Ergebnis entspricht Inhalten eines Pflichtenhefts.

Da für die unterschiedlichen Technologien unterschiedliche Implementierungen nötig sind, hat jeder Service auf dem serverseitigen System eine eigene technische Architektur. Auf Clientseite befinden sich hingegen lediglich die Anbindungen an die unterschiedlichen Services. Diese sind weniger komplex. Damit kann die Architektur hier zusammengefasst werden.

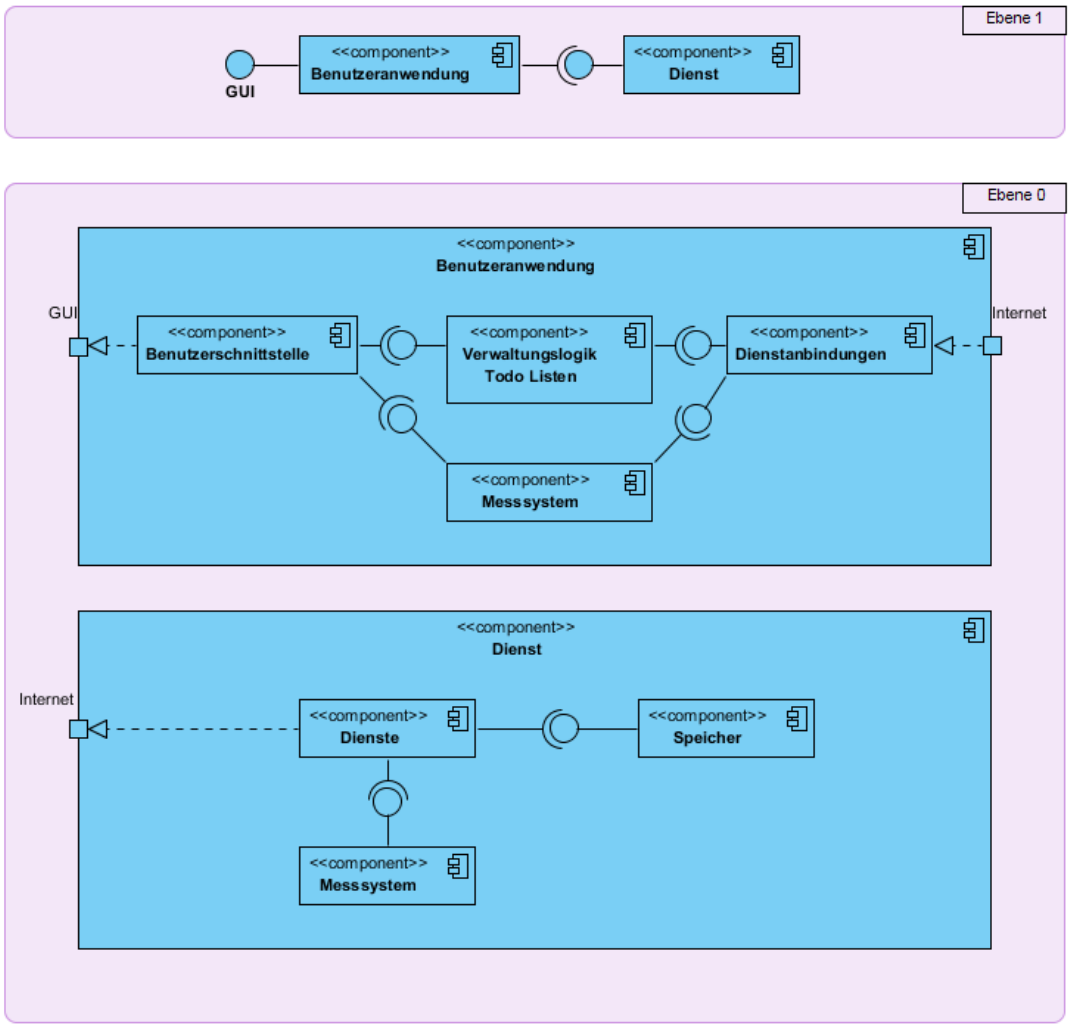

Abbildung 4.2.: Funktionale Komponenten

4.3.1. Architekturstile

Es ist sinnvoll, Architekturstile für die zu entwickelnde Anwendung festzulegen, damit z.B. sowohl Personen, die in die Entwicklung einbezogen sind, als auch diejenigen, welche nicht daran beteiligt sind, wissen, wie der Aufbau der Komponenten gestaltet ist.
In [Tea09, Kapitel 3] wird dabei unterschieden zwischen folgenden Architekturstilen:
Client/Server, Component-Based Architecture, Domain Driven Design, Layered Architecture, Message Bus, N-Tier / 3-Tier, Object-Oriented, Service-Oriented Architecture (SOA).
Es wird ausdrücklich darauf hingewiesen, dass bei Systemen i.d.R. nie nur einer der Architekturstile angewandt wird, sondern eine Kombination aus mehreren. Offensichtlich ist es für die hier zu entwickelnde Anwendung sinnvoll, SOA mit einem Service Exposer und einem Service Consumer zu verwenden, dies ist quasi bereits durch die Aufgabenstellung vorgegeben. Da jeweils mehrere unterschiedliche Service-Technologien zum Einsatz kommen ist es weiterhin sinnvoll, eine Abstraktionschicht für die Anbindung zwischen Service-Consumer und -Exposer zu definieren und in Zusammenhang mit diesem Data Access Layer die restlichen Teile ebenfalls in Schichten einzuteilen. Damit wird demnach ebenfalls eine Layered Architecture angewandt. Zuletzt können die Bestandteile der einzelnen Layer in Komponenten zusammengefasst werden.

Die genauere Unterteilung und deren grafische Darstellung wird im Unterkapitel 4.3.2 (S. 44 ff.) ersichtlich.

4.3.2. Systemarchitektur

Die Systemarchitektur beschreibt die technischen Komponenten des Systems. Diese Komponenten und ihre Beziehungen zueinander sind eine Möglichkeit, die fachlichen Komponenten umzusetzen.

Server

Die Komponenten des Servers sind in Abbildung 4.3 dargestellt.

Es gibt einen Service Layer, in welchem sich die Services als Packages befinden. Diese sind nicht als Komponenten dargestellt, da sich in ihnen die gesamte Architektur eines Services befindet. Diese müssen im Einzelnen separat beschrieben werden. Für eine grobe Übersicht über das serverseitige Systems ist eine Behandlung als Black Box zweckdienlich. Die Services

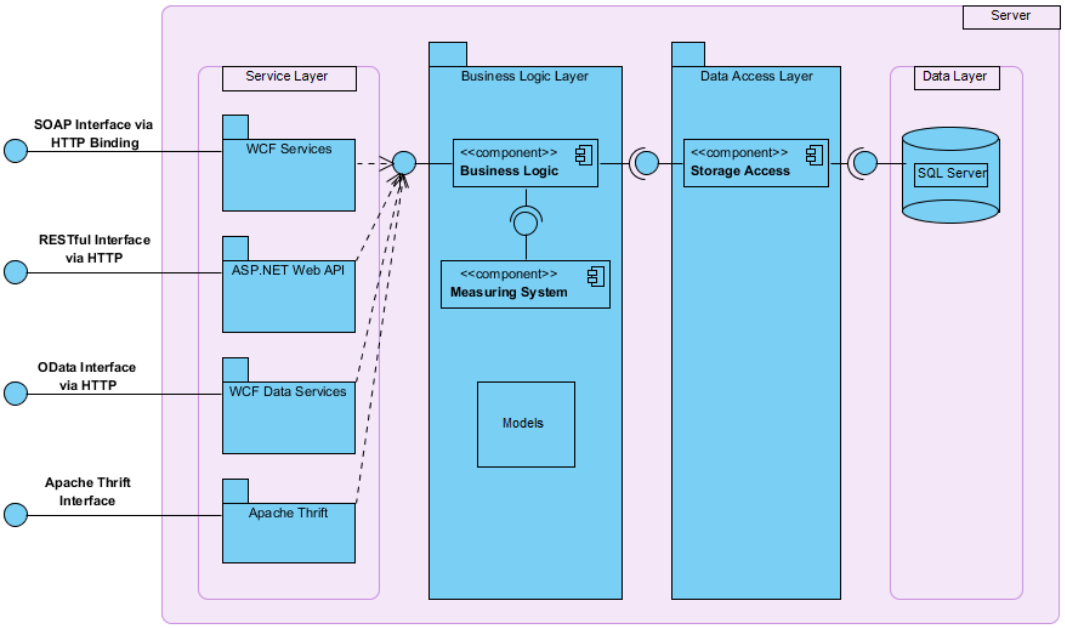

Abbildung 4.3.: Komponenten des Servers

sollen dabei parallel laufen (entsprechend FR S2). Die Dienste, die als Webseiten gehostet werden, können dazu auf Unterseiten der dem Server zugeordneten Domain betrieben werden. Der Business Layer behinhaltet die Business Logik, das Messsystem und die Business Models. Der Data Access Layer abstrahiert die Anbindung an eine Datenbank.

Azure Mobile Services ist kein eigener Bestandteil des Servers, sondern ist nur als Dienst zu verwenden, dessen Architektur von Microsoft entwickelt wurde und ist deshalb nicht in obigem Diagramm mit untergebracht.

Beispiel REST Servicekomponente

Als Beispiel für die Architektur eines eigenen einzelnen Services wird der RESTful Service, der mit ASP.NET Web API implementiert wurde, gezeigt. Die generische Architektur eines ASP.NET Web API Services ist in Abbildung 4.4 dargestellt.

Im Falle der hier zu entwickelnden Anwendung wird als Host dabei ASP.NET Web Hosting verwendet. Der Message Handler Pipeline Layer ist ein Bestandteil des Systems, an dem in diesem Fall keine Implementierungen vorgenommen werden müssen. Im Controller Handling Layer werden konkrete Controller implementiert. Es wird i.d.R. für jede Ressource eines RESTful Services ein Controller implementiert, welcher die jeweils unterstützten HTTP-Methoden implementiert. Im Fall der hier zu entwickelnden Anwendung existieren sowohl To-do-Listen, als auch To-do-Items, wobei die To-do-Items fester Bestandteil der Listen sind. Weiterhin gibt es Benutzer, welchen genau eine To-do-Liste zugeordnet ist. So ergeben sich folgende Routing-Templates:

- „{controller}“
- „{controller}/{userId}“
- „{controller}/{userId}/item/{itemId}“

Die Ressourcen lassen sich mit zwei Controllern beschreiben:

- UserController
- TodoListController

Der User Controller dient der Anlegung von Benutzern. Die weiteren Aufgaben übernimmt der TodoListController. Mit parametrisierten Methoden im Controller kann dieser sowohl die

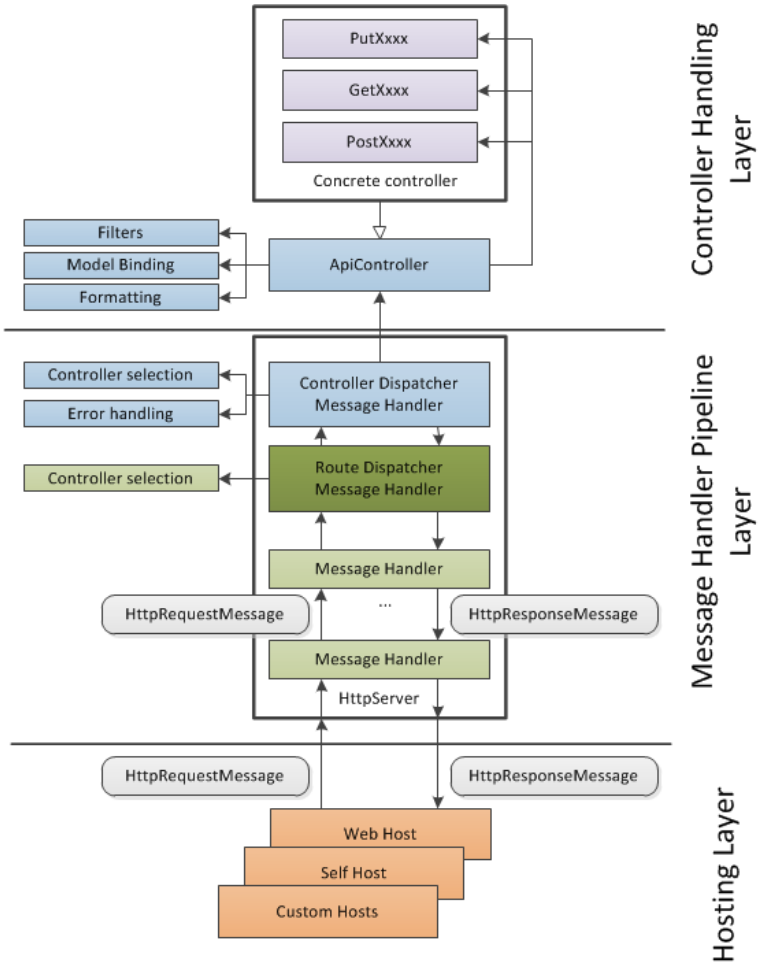

Abbildung 4.4.: Generische Architektur eines ASP.NET Web API Projekts [Quelle: [ea13]]

Verwaltung von To-do-Listen, als auch die Verwaltung von To-do-Items übernehmen. Dies bildet die Beziehung zwischen To-do-Listen und -Items besser ab als zwei separate Controller. Dies geschieht transparent, so dass die Schnittstelle den REST-Prinzipien treu bleibt und erstens der Geltungsbereich in der Uri enthalten ist, sowie zweitens nicht auf Uri-Parameter zurückgegriffen wird, ohne dass dies notwendig ist.

Client

Aus Kapitel 4.1.2 (S. 32 ff.) geht hervor, dass eine Windows Store App als Service-Comsumer entwickelt werden soll. Als Design-Pattern für Windows Store Apps wird häufig MVVM (Model-View-Viewmodel) verwendet (siehe Abb. 4.5).

Hiermit sind eine Vielzahl von Vorgaben verbunden, wie die Anwendung aufgebaut sein soll und was die jeweiligen Komponenten enthalten sollen. Diese Vorgaben sind sinnvoll für größere Anwendungen mit mittlerer bis hoher Komplexität. Für die einfache To-do-Anwendung wird dieses Level an Abstraktion nicht nur nicht benötigt, sondern würde die Entwicklung verzögern, ohne dabei einen ausgleichend großen Vorteil zu bringen. Aus diesem Grund wird darauf verzichtet, die Vorgaben des MVVM-Patterns im Detail einzuhalten. Insbesondere wird hier auf das ViewModel verzichtet, weil das Databinding nur mit einer To-do-Liste stattfindet, welches die einzige Entität ist, und eine Abstraktion an dieser Stelle nicht nötig ist. Aus diesen Überlegungen ergeben sich die in Abbildung 4.6 dargestellten Layer und Komponenten.

Die Anzeige der To-do-Liste, sowie die oberste Ebene der Interaktion mit dem Benutzer (Handler für Clickevents) befinden sich im Presentation Layer. Die Benutzeroberfläche ist deklarativ in XAML beschrieben. Im XAML wird die To-do-Liste per Databinding eingebunden. Die Clickevent Handler befinden sich im sog. Code-Behind der entsprechenden XAML-Dateien. Im Code-Behind sollten keine GUI-Manipulationen vorgenommen werden, sowie auch andersweitig der Code so schlank wie möglich gehalten werden. Dies gilt als Best Practice beim MVVM-Pattern.

Die Businesslogik, das Messsystem und die Models als Entitäten befinden sich im Business Logic Layer. Je nach vorhergegangener Benutzerinteraktion verwendet die Businesslogik das Messsystem oder reicht die Befehle, welche vom Presentation Layer kommen, direkt an den Data Access Layer weiter.

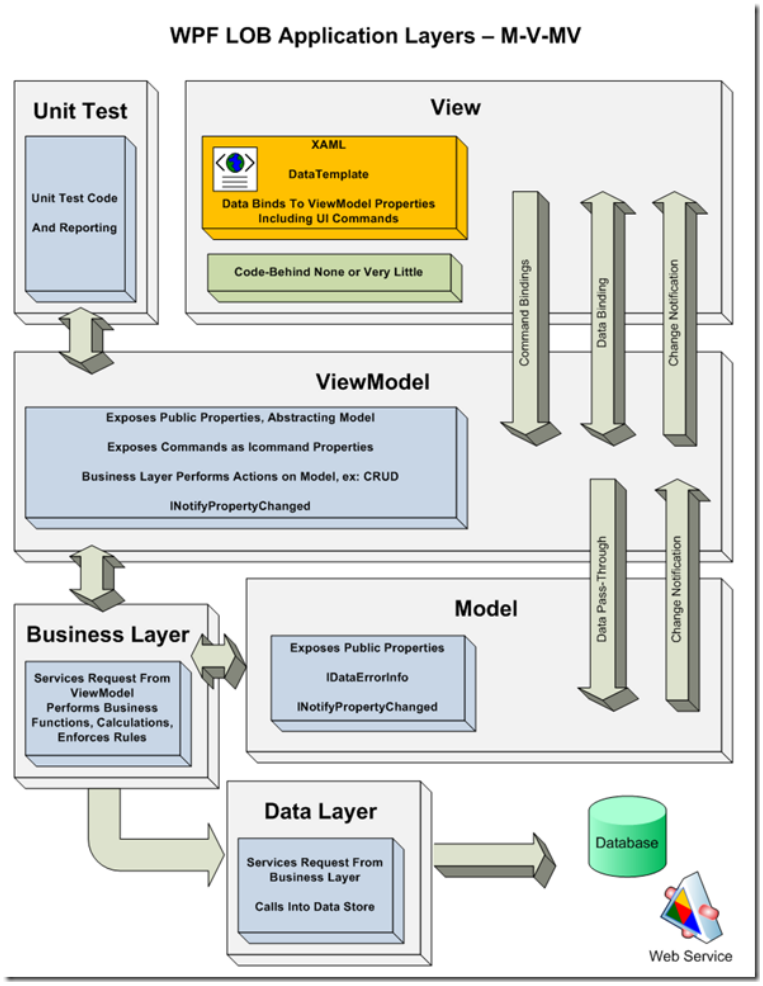

Abbildung 4.5.: Darstellung des MVVM Entwurfsmusters [Quelle: [Shi10]]

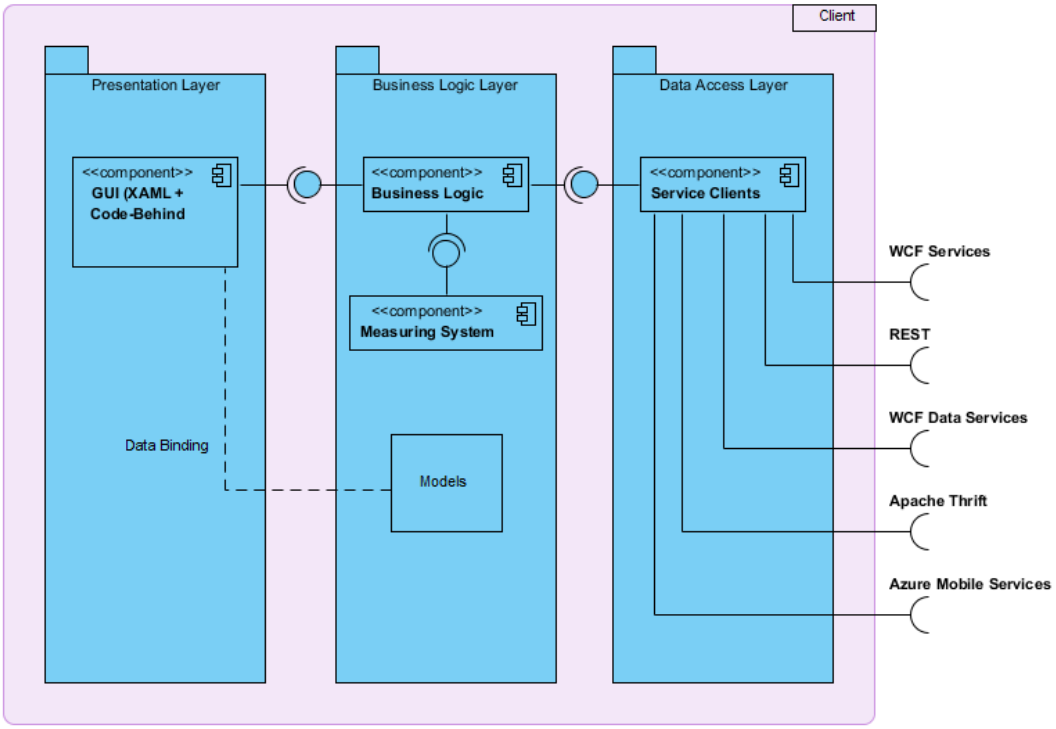

Abbildung 4.6.: Komponenten des Clients

In diesem Data Access Layer befinden sich die Anbindungen an die unterschiedlichen Services. Damit im Code-Behind oder Messystem nicht jede einzelne Anbindung mit ihren jeweiligen Besonderheiten einzeln angesprochen werden muss, wird ein abstrahierendes Interface angeboten.

4.3.3. Objektorientierter Entwurf

Server Klassen

Bei der Erstellung des Klassendiagramms (Abb. 4.7) wird die zuvor erstellte Systemarchitektur berücksichtigt. In diesem Fall wird in zwei Punkten davon abgewichen.
1) Es ist neben der SqlTodoStorage auch eine FileTodoStorage Implementierung vorgesehen. Dies ermöglicht es später bei der Implementierung der Testanwendung prototypisch vorzugehen und zuerst eine auf Dateisystemebene arbeitende Persistenzschicht zu verwenden, da eine den SQL Server verwendende Persistenzschicht als mit höherem Entwicklungsaufwand verbunden angesehen wird.
2) Es wird keine explizite Klasse für die Business Logik erstellt, da zwischen Service Aufruf Handling - welcher Teil des Services ist - und Datenzugriff für den Betrieb der Testanwendung und dem Durchführen von Messungen keine weiteren Aktivitäten stattfinden müssen. Die Implementierung von Business Regeln macht Sinn, sobald die Anwendung produktiv betrieben werden soll. Aktuell ist jedoch ein direkter Zugriff vom Service Layer auf den Data Access Layer zweckdienlicher.

Für die Integration des Messsystems wird eine Fassade verwendet. Das Fassaden Entwurfsmuster sieht eine Klasse vor, welche nach außen eine einfache Schnittstelle bietet und intern Delegationen an mehrere weitere Klassen vornimmt. Im Fall der Testanwendung implementiert die Fassade das ITodoStorage Interface, sodass in den Services sowohl die einfachen Implementierungen FileTodoStorage sowie SqlTodoStorage ohne Messsystemintegration, als auch die Fassade verwendet werden kann, ohne dass dies im Service gesondert berücksichtigt werden muss.

Die Austauschbarkeit wird durch eine Factory ermöglicht. Das Factory Entwurfsmuster dient der Zentralisierung der Auswahl von Implementierungen von Interfaces. In den Services wird einer Variable vom Typ ITodoStorage mit dem Aufruf der Methode GetInstance(string service) die in der Factory bestimmte Implementierung zugewiesen. Das Argument der GetInstance

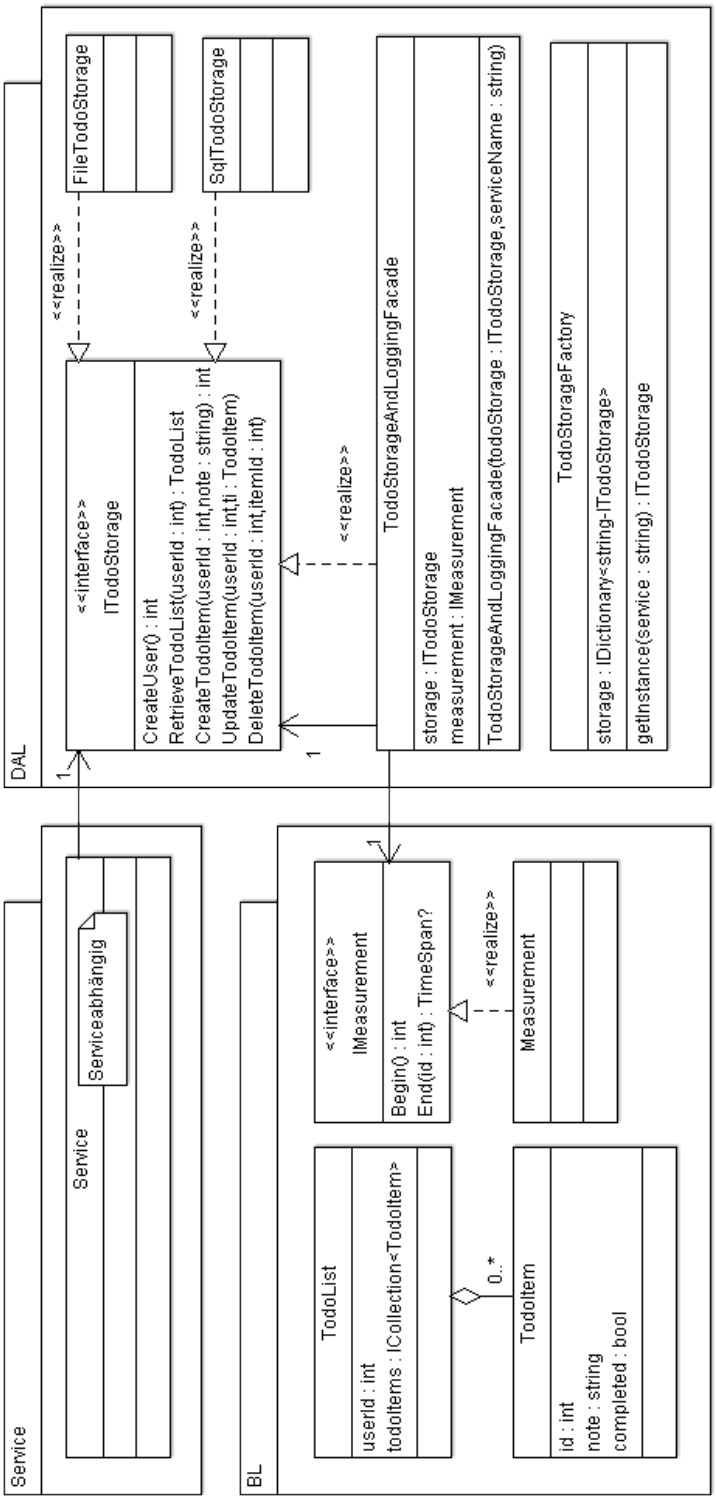

Abbildung 4.7.: Klassendiagramm Server

Methode dient dazu, sowohl innerhalb der Factory eine unterschiedliche Behandlung der Services zu ermöglichen, als auch den Servicenamen der Fassade mitzuteilen, so dass z.B. dem Namen entsprechende Logdateien erstellt werden können. Fortan kann an zentraler Stelle in der Factory die Auswahl der ITodoStorage Implementierung geändert werden, welches sich auf alle Services auswirkt, ohne deren Code verändern zu müssen.

Die Delegationen der Fassade bestehen aus den Aufrufen der Messmethoden des Messsystems sowie den Methoden einer ITodoStorage Implementierung. Welche Implementierung hier verwendet wird, wird über den Konstruktor der Fassade gesteuert, indem eine ITodoStorage Implementierung als Parameter erwartet wird. Da die Fassade in der Factory erzeugt wird, ist auch diese Steuerung zentral möglich. Die Übergabe einer Interfaceimplementierung über einen Konstruktor nennt sich „Inversion of Control“ oder „Dependency Injection“, da nicht mehr innerhalb der Klasse über die Auswahl einer Implementierung entschieden und auch keine Factory aufgerufen werden muss, sondern die die Klasse verwendende Komponente über die Implementierung entscheidet.

Das Dictionary in der Factory dient der Vorhaltung von zuvor erzeugten, zu einem Service zugehörigen Implementierungen. Fragt ein Service nach einer ITodoStorage Implementierung und ist der Servicename noch nicht enthalten, wird ein neues Objekt erzeugt, andernfalls das vorhandene zurückgegeben. Dies hat z.B. bei ASP.NET Web API den Vorteil, dass in beiden Controllern eine Fassade angefordert werden kann und dabei nur ein Objekt besteht. Im Web API Projekt müsste sonst eine zusätzliche statische Klasse erstellt werden, dabei soll die Servicekomponente in sich abgeschlossen sein und keine Aufgaben für die Persistenz übernehmen. Sollte in Zukunft ein Controller hinzugefügt werden, welcher ein separates Objekt benötigt, so ist dies ebenso möglich.

Client Klassen

Die Komponenten und Klassen im Client (siehe Abb. 4.8) sind ähnlich aufgebaut wie die des Servers. Anstelle eines Interfaces in der DAL für den Zugriff auf eine Persistenzschicht gibt es hier ein Interface für den Zugriff auf die Services. An Stelle der frei wählbaren ITodoStorage Implementierungen im Server müssen hier fünf verschiedene Implementierungen für die fünf unterschiedlichen Services entwickelt werden.

Ein weiterer Unterschied ist das Fehlen einer Factory. Es ist vorgesehen, dass der Benutzer wählen kann, über welchen Servicetyp mit dem Back-End kommuniziert wird. Im Messsystem

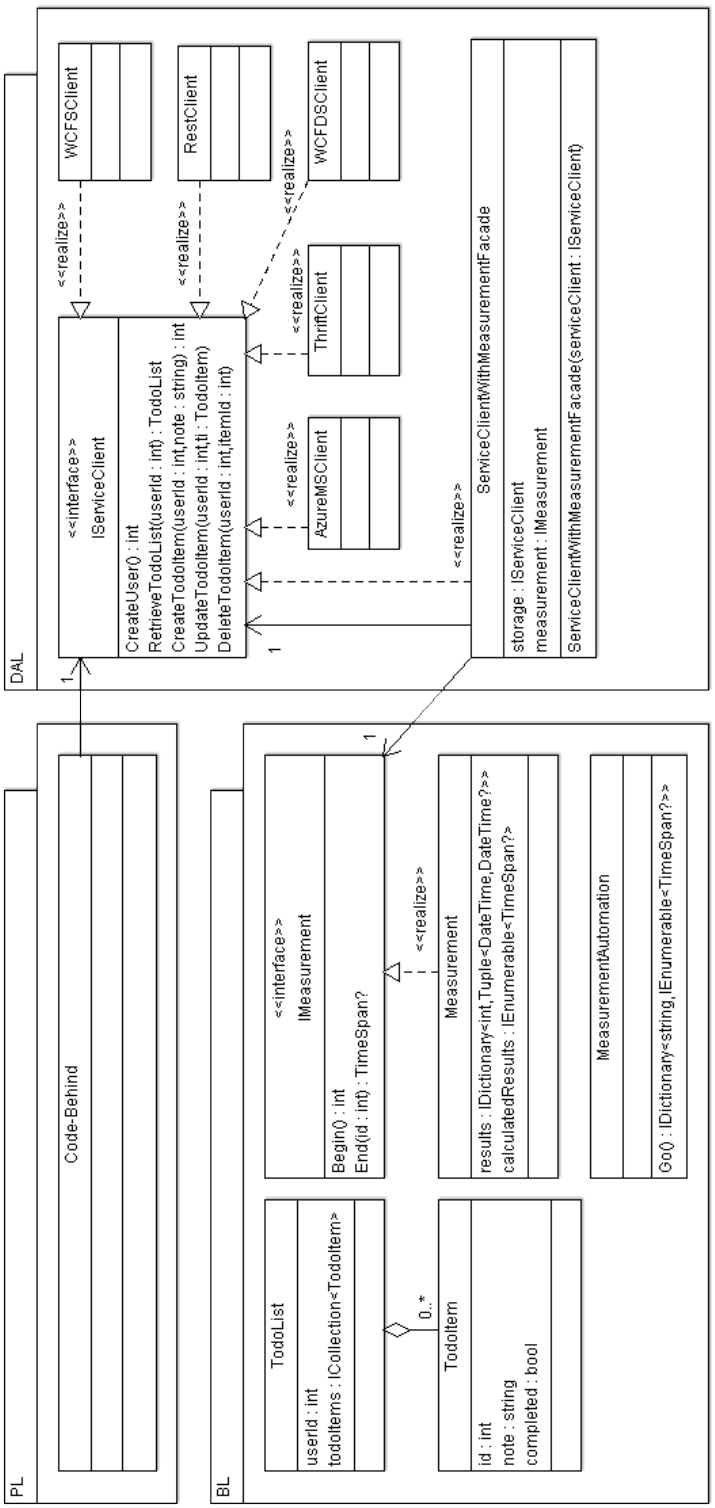

Abbildung 4.8.: Klassendiagramm Client

werden ohnehin sämtliche Implementierungen verwendet. Eine Factory würde zwar einen späteren Austausch einer vorhandenen IServiceClient Implementierung erleichtern, dies ist jedoch, anders als bei der Ersetzung des FileTodoStorage durch den SqlTodoStorage auf dem Server, nicht vorgesehen.

Die IMeasurement Implementierung wurde um zwei Eigenschaften erweitert, da clientseitig nicht dauerhaft geloggt, sondern Messungen über die Dauer eines Benchmarks gesammelt und anschließend verwendet werden.
Die MeasurementAutomation Klasse dient der automatisierten Durchführung eines Benchmarks.

Beispiel interner Ablauf

Der Ablauf eines Benchmarks auf Clientseite sei beispielhaft im Sequenzdiagramm in Abbildung 4.9 skizziert.

Der Benchmark kann aus weiteren Methodenaufrufen bestehen. Vom ServiceClient aus finden Aufrufe an den Service statt, welcher nicht mehr Teil des Clientsystems sind. Die Messwerte sind als Property (zur Veröffentlichung eines Attributs) im Measurement Objekt enthalten, welches wiederum als Property in der Fassade enthalten ist. So wird von der MeasurementAutomation auf die Ergebnisse zugegriffen. Dieser Zugriff findet erst nach Ende des Benchmarks statt.

Serverseitig wird hingegen dauerhaft geloggt. Der Ablauf sei beispielhaft im Diagramm in Abbildung 4.10 skizziert. Das Logging findet dabei nach jedem Rückkehr der End() Methode statt.

4.3.4. Datenbankmodellierung

Eine Datenbank wird anhand der identifizierten Entitäten modelliert. In diesem Fall sind dies die drei Entitäten Benutzer, To-do-Liste und To-do-Item. Das Modell ist in Abbildung 4.11 gezeigt.

Zu dem Benutzer werden jedoch keine Informationen gehalten außer einer ID. Außerdem hat jeder Nutzer eine To-do-Liste und zu jeder To-do-Liste gibt es nur einen Benutzer. Dadurch können Benutzer und Liste „verschmolzen“ und in einer Tabelle gehalten werden.

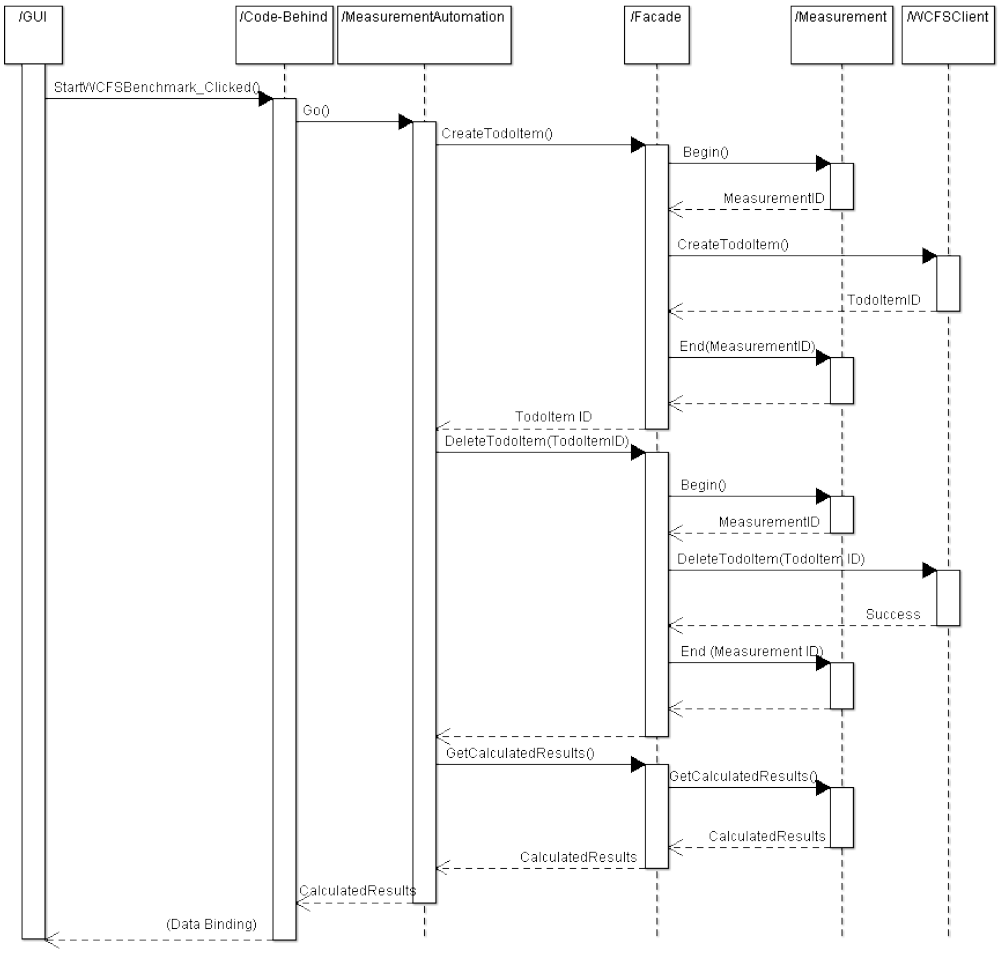

Abbildung 4.9.: Sequenzdiagramm eines beispielhaften Vorgangs im Client

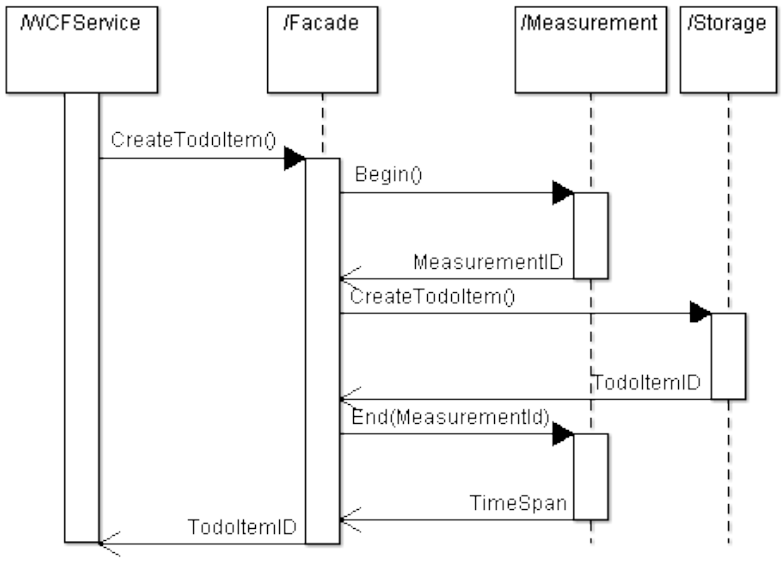

Abbildung 4.10.: Sequenzdiagramm eines beispielhaften Vorgangs im Server

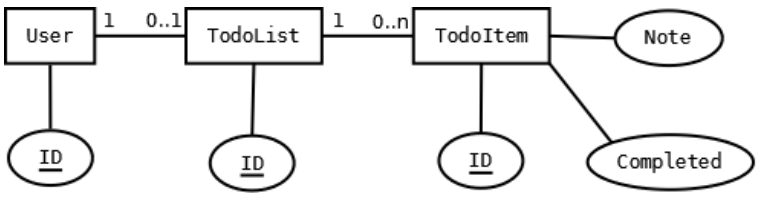

Abbildung 4.11.: ER-Diagramm entsprechend Modellierung

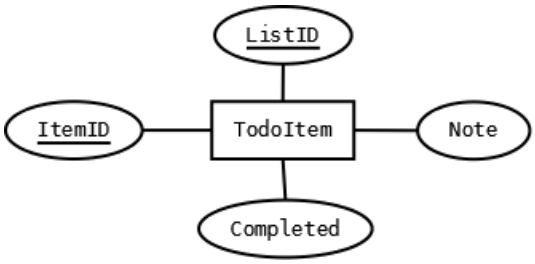

Abbildung 4.12.: ER-Diagramm nach Vereinfachung

ListID	ItemID	Note	Completed
1	1	Herr Prof. Dr. Zukunft kontaktieren	true
1	2	Bachelorarbeit zuende schreiben	false
2	1	Termin vereinbaren	true
2	2	Feedback geben	false

Tabelle 4.8.: Beispielausprägungen Datenbank

Es ist noch eine weitere Vereinfachung und Optimierung möglich. In der Datenbank müsste nun für das Umsetzen der Relation in der TodoList Tabelle neben der Spalte der Listen-ID eine Spalte mit zugehörigen TodoItems gespeichert sein. Bei einer SQL-Anfrage würden hierüber dann die Informationen aus der TodoItem Tabelle abgerufen - es finden demnach zwei Lookups über unterschiedliche Indizes statt.
Da die To-do-Liste keine weiteren Informationen außer einer ID hält, entstehen keine Redundanzen, wenn auch diese beiden Tabellen zusammengefasst werden. Damit existiert nur eine Tabelle mit ListID und ItemID gemeinsam als Primärschlüssel. Die zuvor genannte Spalte entfällt, sowie der doppelte Lookup. Das Ergebnis in Form eines ER-Diagramms ist in Abbildung 4.12 gezeigt.

Beispielausprägungen sind in Tabelle 4.8 angegeben.

4.3.5. Grafische Benutzeroberfläche

Ein Mockup der grafischen Benutzeroberfläche, welches die Anforderungen FR C1, FR C2 zum Teil, FR C3, FR C4 und NFR C1 erfüllen soll, ist in Abbildung 4.13 dargestellt.

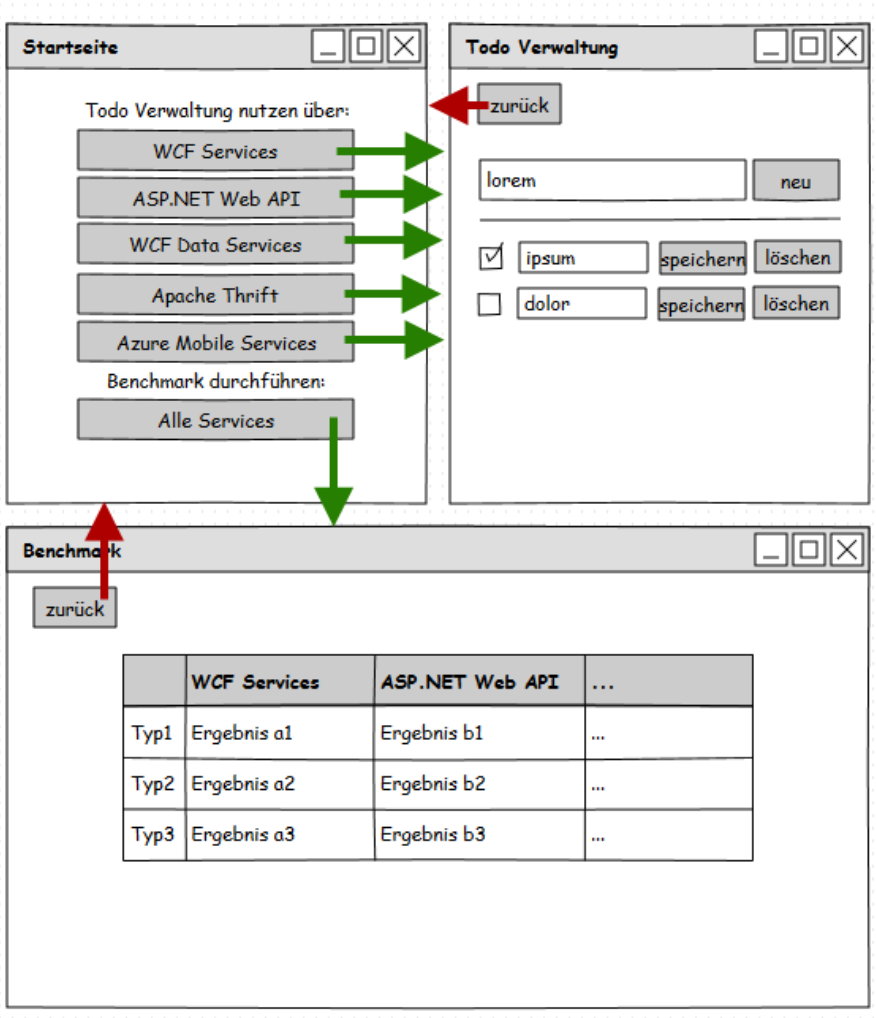

Abbildung 4.13.: Mockup der grafischen Benutzeroberfläche

Die Responsivität (entsprechend Anforderung NFR C2) kann über das Verwenden von asynchronen Methodenaufrufen mit den Schlüsselwörtern async und await erreicht werden. Async-await wurde als Sprachfeature in C# 5 eingeführt und wird bereits von vielen Methoden der .NET Klassenbibliothek für Windows Store Apps verwendet. Die Asynchronizität der Methode ist wichtig, damit die Benutzeroberfläche der App Befehle des Benutzers entgegennehmen kann, während die Servicekontaktierung im Hintergrund weiterläuft und die Benutzeroberfläche nicht „einfriert“. Bei der Implementierung der Serviceclients müssen in Methoden ggf. Tasks gestartet werden, um dies zu erreichen.

4.4. Implementierung und Test

Entsprechend Konzeption und Design wurden die Komponenten implementiert. Im Folgenden werden der Umfang der Implementierung, eine kurze Beschreibung der Einrichtung der Entwicklungs- und Laufzeitumgebung, sowie der Programmierung, wo auf Details der Implementierung und Abweichung zu Konzeption und Design eingegangen wird, beschrieben.

4.4.1. Umfang

Da sowohl eine beispielhafte mobile Applikation, als auch ein Messsystem entwickelt werden soll, ist eine den Anforderungen entsprechende vollständige Implementierung der unterschiedlichen Web Services notwendig. Es reicht demnach nicht, Stubs zu entwickeln, sondern die Implementierungen müssen inklusive Persistenz vollständig funktionsfähig sein.

Es werden keine der in Unterkapitel 2.5.2 (S. 20) genannten Optimierungsmöglichkeiten umgesetzt. Einige der Möglichkeiten sind Technologieunabhängig und würden sich auf alle Messungen gleichermaßen auswirken. Die anderen werden nicht explizit angewandt, damit der Vergleich die Standardkonfigurationen der Technologien betrifft.

4.4.2. Einrichtung

Entsprechend den Überlegungen im Kapitel 4.1 (S. 30 ff.) wird als Hardware-Plattform für den Server eine virtuelle Maschine beim IaaS-Provider Windows Azure verwendet. Um Beschränkungen bzgl. Leistung aufgrund der Hardware zu minimieren wird eine virtuelle Maschine mit der Vorkonfiguration „Mittel“ gewählt, welche 2 dedizierte Kerne, knapp 4 GB RAM, ca. 130 GB persistenter Speicher und weitere 130 GB temporärer Speicher hat. Als Betriebssystem wird Windows Server 2012 Datacenter gewählt, um Inkompatibilitäten mit

dem WCF Framework zu vermeiden. Unter Linux lassen sich zwar mit Mono .NET Programme ausführen, jedoch wird Linux nicht offiziell von dem WCF-Entwicklerteam von Microsoft unterstützt.

Auf dem Server wurde der Microsoft Webserver IIS in der aktuellen Version 8.0, sowie Web Deploy installiert und konfiguriert. IIS wird verwendet um WCF Services, ASP.NET Web API, sowie WCF Data Services zu hosten. Hier gibt es weitere Varianten, z.B. Self-Hosting und Hosting mit OWIN, jedoch entspricht das Hosten in IIS einem realistischen Szenario. Damit Web Deploy funktioniert, müssen über den Web Platform Manager weitere Komponenten bezogen werden, wofür sich das vorkonfigurierte Paket „Recommended Settings for Hosting“ eignet. Weiterhin installiert / aktiviert wurden einige nötige Features und Komponenten über den Server Manager.

Anschließend können im IIS Manager für die drei genannten Dienste Webapplikationen erstellt werden, in welche später die in Visual Studio 2012 exportierten Projekte importiert werden können. Wichtig ist hierbei, zuerst Unterseiten einer IIS Webseite zu erstellen, die jeweils auf andere Virtual Directories verwesen. Damit sind Pfade wie z.B. „ba13.cloudapp.net/TodoWcfService“ und „ba13.cloudapp.net/TodoRestService“ angelegt. Diese Unterseiten lassen sich anschließend in Webapplikationen umwandeln. Dazu muss jeder Webapplikation ein App Pool zugewiesen werden. Hierbei sollten auch für jeden Web Service eine eigene App Pool Identität erstellt werden, damit die unterschiedlichen Services parallel Requests bearbeiten können und bei Ausfall die Verfügbarkeit der anderen Services nicht beeinträchtigt [Mic13b] [Pul13].
Letztendlich sind dann z.B. über die URL „ba13.cloudapp.net/TodoWcfService/TodoService.svc“ der WCF Services Dienst verfügbar und über die URL „ba13.cloudapp.net/TodoRestService/TodoList/1“ eine Ressource des ASP.NET Web API Dienstes ansprechbar. Ohne die separaten Virtual Directories und Webapplikationen könnte entweder nur ein Service zu einer Zeit laufen oder es müssten mehrere Domains mit dem Server verwaltet werden.

Als persistenter Speicher für die Daten, deren Austausch die Web Services leisten sollen, wurde Microsoft SQL Server gewählt. Dieser wurde in Version Standard 2012 installiert. Im gleichen Zug wurden eine Datenbank und Tabelle entsprechend dem im Design ermittelten Schema erstellt, ein zusätzlicher Nutzer angelegt und aktiviert, zusätzlich zur Windows Authentifizierung SQL Authentifizierung aktiviert und entsprechende Zugriffsrechte vergeben.

Der Azure Mobile Services Dienst wird nicht auf dem Azure Server eingerichtet, sondern über die Azure Management Webapp. Als Standort wurde Nordeuropa gewählt, weil sich dort auch der Azure Server befindet und für möglichst vergleichbare Messergebnisse der Standort gleich sein sollte. Für den Mobile Service muss ein zusätzlicher SQL Server eingerichtet werden, jedoch nicht als eigene Serverinstanz, sondern als Azure Dienst. Hier wäre als Standort auch Westeuropa möglich, was näher an der Clientapp liegt, jedoch spielt lediglich die Nähe zum Mobile Service eine Rolle.

In der Azure Management Webapp müssen für einige Dienste sog. „Endpunkte“ erstellt werden, welche Port und Protokoll von Diensten betreffen, die von einem Client erreichbar sein sollen.

Zum Entwickeln und Testen auf dem lokalen PC muss Visual Studio installiert, sowie IIS aktiviert sein. Hier wurde Visual Studio in der Version 2012 Premium verwendet.

4.4.3. Besonderheiten der Implementierung

Server DAL Prototyp

Damit alle Web Services auf einen Datenbestand zugreifen können, wurde zuerst diese zentrale Komponente entwickelt. Entsprechend dem Design ist dies der Data Access Layer. Er wird als separates Projekt als .NET Klassenbibliothek angelegt, um in den unterschiedlichen Web Service Implementierungen eingebunden werden zu können. Um zuerst den Fokus sobald wie möglich auf die Services legen zu können, wurde die Komponente prototypisch entwickelt. Dafür wurde keine Funktionalität für Messungen, sowie vorerst nur die FileTodoStorage Implementierung fertiggestellt. Diese legt für jeden neuen Benutzer eine Datei namens „<UserId>.json“ an und speichert die zugehörige To-do-Liste in JSON-Notation in der Datei und manipuliert den Inhalt entsprechend der vorgegebenen Methoden.

Ebenso wurde die Factory entsprechend implementiert.

WCF Services

Veröffentlicht werden die Methoden:

- int CreateUser()
- TodoList GetTodoList(int userId)
- int CreateTodoItem(int userId, string note)

- bool UpdateTodoItem(int userId, TodoItem ti)
- bool DeleteTodoItem(int userId, int todoItemId)

Zwar könnten die Update- und Delete-Methoden als „fire and forget“ ohne Rückgabewert implementiert werden, jedoch kann mit dem Rückgabewert erstens dem Client mitgeteilt werden, ob die Anfrage auf dem Server erfolgreich durchgeführt werden konnte und zweitens wird es von dem clientseitigen Messsystem benötigt.

Ergebnis: Der Web Service ist unter http://ba13.cloudapp.net/TodoWcfService/TodoService.svc erreichbar. Er kann von generischen SOAP Clients verwendet werden, welche anhand der WSDL-Datei den nötigen Clientseitigen Code für die Verwendung generieren können.

ASP.NET Web API

ASP.NET Web API unterstützt lediglich „convention-based“ Routing. Erst Web API 2, welches derzeit (November 2013) nur verfügbar ist, wenn man mit Visual Studio 2013 Preview arbeitet, ermöglicht Attribute Routing [Was13]. Aber auch die aktuelle Version 1 hält sich an die Konvention von RESTful Services, Methoden nur entsprechend den HTTP Verben zu erstellen. Entsprechend dem Design dient der Pfad „/TodoList/1/item/1“ dem Zugriff auf ein To-do-Item. Diese Ressource soll geändert und gelöscht werden können. Die Änderung bezieht sich dabei sowohl auf die Notiz selbst, als auch auf das Flag, ob das To-do-Item erledigt ist oder nicht. Ein Update wird üblicherweise mit HTTP PUT vorgenommen. Dabei wird jedoch jedes mal ein ganzes Item übertragen, obwohl ggf. nur ein Teil (z.B. Completed Flag) geändert wurde.
Gerade in Anbetracht der Entwicklung mobiler Applikationen macht hier das neue HTTP Verb „PATCH“ Sinn. Damit müsste nicht mehr ein gesamtes To-do-Item übertragen werden. Allerdings grenzt das Framework hier ein: Es gibt in Web API 1 keinen Mechanismus für Teilupdates einer Ressource. Nur folgende Umwege bieten die Möglichkeit, PATCH zu verwenden oder anderweitig nicht die gesamte Ressource zu übermitteln:

- Ein sog. „Mini PUT“ (mit den Pfaden „/item/1/completed“ respektive „/item/1/note“). Dabei wird jedoch die Integrität der Ressource (To-do-Item) aufgebrochen.
- Über eine Preview Version einer OData Erweiterung der Web API lässt sich PATCH verwenden [Jam12]. Damit ist der Web Service jedoch nicht mehr unabhängig von dritten Technologien. Gerade der reine REST Service und OData sollten getrennt bleiben, um die Services hier miteinander vergleichen zu können.

- Man könnte ein eigenes Mediaformat verwenden. D.h. statt „application:json“ ein eigens definiertes Format, so dass sich anhand des Inhalts von übertragenen Daten erkennen lässt, ob und wie der Inhalt zum Aktualisieren eines Teils einer Ressource verwendet werden soll. Dies bringt jedoch Inkompatibilitäten und weitere Probleme mit sich.
- Man könnte auf die Ressource eine Action definieren („/res/1?action=increment“): Dies entspräche nicht mehr der Konvention einer RESTful Architektur (Methodeninformation als HTTP Methode), außerdem müsste HTTP POST verwendet werden und weicht vom Uniform Interface ab, sodass gesondert dokumentiert werden müsste.

Entsprechend der jeweiligen Begründung wird in der aktuellen Implementierung kein PATCH unterstützt. Es muss demnach bei einem Update ein ganzes To-do-Item übertragen werden.
In diesem Kontext ist noch zu erwähnen, dass für genau diesen Zweck (Patchen mit entsprechendem Datenformat) bei der IETF ein neuer Content-Type im April 2013 als RFC verabschiedet wurde: „application/json-patch“ [PB13], [Not12]

Immerhin eine Optimierung für mobile Dienste kann jedoch umgesetzt werden: Statt für das Erstellen eines To-do-Items einen JSON String eines kompletten To-do Item Konstrukts zu übertragen, reicht es, die Notiz als String zu übertragen. Dabei muss der Content-Type jedoch auf „application/x-www-form-urlencoded“ und nicht auf „application/json“ umgestellt werden. Als Body kann anschließend z.B. „=Beispieltext“ verwendet werden. Dies erleichtert auch die Entwicklung insoweit, da serviceseitig sonst ein ViewModel benötigt würde.

Ergebnis: Der Web Service ist unter http://ba13.cloudapp.net/TodoRestService/ erreichbar. Um an die Ressourcen zu kommen, muss der Pfad entsprechend erweitert werden (z.B. http://ba13.cloudapp.net/TodoRestService/TodoList/1). Verwendet werden kann der Service von jedem HTTP Client.

WCF Data Services

Ein WCF Data Services Projekt benötigt eine Referenz auf ein ADO.NET Entity Data Model. Dieses kann aus der SQL Datenbank generiert werden und besteht fortan als eigene .NET Klassenbibliothek.

OData reicht die Daten aus der Datenbank quasi weiter. Controller oder ähnliches müssen nicht programmiert werden. Die Schnittstelle ist mit einem Browser, anderen HTTP Clients

und OData Client Bibliotheken ansprechbar.

Da Microsoft dazu übergeht, Entwicklerprodukte nicht mit anderen Produkten gebündelt, sondern einzeln in separaten Releasezyklen zu veröffentlichen [Tu13], ist das Entity Framework auf NuGet in Version 6 vorhanden, welches nicht mehr kompatibel zu WCF Data Services ist. Daher musste eine DLL einer vorigen Version aus dem Filesystem des Entwicklerrechners herausgesucht und referenziert werden.

Ergebnis: Der Web Service ist unter
http://ba13.cloudapp.net/TodoWcfDataService/TodoDataService.svc/
erreichbar. Um an die Ressourcen zu kommen, muss der Pfad entsprechend erweitert werden (z.B. um „/Todos“). Als Client kann jeder HTTP Client, sowie für verschiedene Plattformen verfügbare OData Client Bibliotheken verwendet werden.

Apache Thrift

Wie in den Grundlagen (2 (S. 4 ff.)) genannt, lässt sich mit der IDL von Apache Thrift neben dem Datenmodell auch ein Service beschreiben. Als Grundlage für die vom Service zu veröffentlichen Methoden können die Methoden des WCF Services verwendet werden. Alles lässt sich in einer Datei todo.thrift unterbringen (siehe 4.1).

```
struct TodoItem {
        1: i32 Id,
        2: string Note,
        3: bool Completed
}

struct TodoList {
        1: i32 UserId,
        2: list<TodoItem> TodoItems
}

service TodoThriftService {
        i32 CreateUser(),
        TodoList GetTodoList(1:i32 userId),
        i32 CreateTodoItem(1:i32 userId, 2:string note),
        bool UpdateTodoItem(1:i32 userId, 2:TodoItem todoItemId),
        bool DeleteTodoItem(1:i32 userId, 2:i32 todoItemId),
```

```
18 }
```

Listing 4.1: Todo Verwaltung Models und Service in Apache Thrift IDL

Der daraus vom Compiler erzeugte Code besteht aus den Dateien TodoList.cs, TodoItem.cs und TodoThriftService.cs. In letzterer befindet sich das Interface „Iface“. Hierfür muss eine eigene Implementierung erstellt werden, welches die in der IDL definierten Methoden umfasst.

Damit der Service automatisch bei Systemstart und ohne Anmeldung eines Benutzers startet wurde die ausführbare Datei mit dem Task Scheduler entsprechend eingerichtet.

Ergebnis: Der Web Service ist unter ba13.cloudapp.net:9090 erreichbar. Mit einem Thrift Client, welcher im Gegensatz zu WCF Services jedoch nicht generisch sein kann, sondern mit dem aus der IDL-Datei generierten Code erstellt wurde, können die bereitgestellten Methoden entfernt aufgerufen werden.

Azure Mobile Services

Der Azure Mobile Services Dienst wird als einziger nicht auf dem Azure Server ausgeführt. Es muss kein Programmcode geschrieben werden, um den Service zu erstellen. Der Service ist lauffähig nach der Einrichtung entsprechend der Beschreibung in 4.4.2 (S. 60 ff.). Bei der Erstellung des Datenschemas muss berücksichtigt werden, dass Azure Mobile Services in jeder Tabelle eine Spalte mit der Bezeichnung Id verlangt, dessen Inhalt außerdem automatisch vom System verwaltet wird. Dennoch kann das Schema entsprechend dem Schema des SQL Servers auf dem Azure Server erstellt werden:
UserId | Id | Note | Completed

Ergebnis: Der Web Service ist unter ba13.azure-mobile.net erreichbar. Der Dienst kann über eine vom Service automatisch bereitgestellte REST Schnittstelle mit einem HTTP Client, sowie mit für verschiedene Plattformen verfügbaren Azure Mobile Services Client Bibliotheken verwendet werden.

Client DAL

Der clientseitige Data Access Layer ähnelt dem serverseitigen und dient der Bündelung der Funktionalitäten, welche hier für den Datenzugriff auf die Services zuständig sind. Es ist eine abgeschlossene Komponente, welche nach außen eine einfache Schnittstelle bereitstellt, mit welcher der Nutzer der Komponente unabhängig vom letztendlich verwendeten Service immer

auf gleiche Weise interagieren kann.

Dank der Abgeschlossenheit der Komponente kann sie als separates Projekt erstellt in die Visual Studio Solution für den Client eingebunden werden. Um dem Anspruch von mobilen Umgebungen gerecht zu werden, würde es Sinn machen, nicht eine herkömmliche .NET Klassenbibliothek zu erstellen, sondern eine sog. Portable Class Library (PCL). Als Ziele könnten die Plattformen .NET Framework 4, 4.0.3, 4.5, „Silverlight 4 and higher“, Silverlight 5„ „Windows Phone 7 and higher“, Windows Phone 7.5, 8, „.NET for Windows Store Apps“, sowie Xbox 360 angegeben werden. Dabei verringert sich zwar der Umfang der nutzbaren Klassen und Methoden der .NET Klassenbibliothek, mit PCLs lassen sich jedoch neben den angesprochenen Zielplattformen mit Tools von Xamarin auch mobile Applikationen für Android und iOS entwickeln. Da Apache Thrift jedoch z.B. die Klasse Stream verwendet und diese unter den von Windows Store Apps unterstützten Klassenbibliothekstypen nur beim Typ „Class Library (Windows Store Apps)“ verfügbar ist, musste darauf verzichtet werden.

Entsprechend dem Design und der eingangs genannten einfachen, nach außen bereitgestellten Schnittstelle, gibt es ein Interface „IServiceClient“, welches die in Listing 4.2 dargestellten Methoden bereitstellt. Die Verwendung von „Task<>“ als Rückgabewert stellt sicher, dass die Implementierungen asynchrone Methodenaufrufe unterstützen.

```
1 public interface IServiceClient
2 {
3     Task<int> CreateUser();
4     Task<TodoList> RetrieveTodoList(int userId);
5     Task<int> CreateTodoItem(int userId, string note);
6     Task<bool> UpdateTodoItem(int userId, TodoItem ti);
7     Task<bool> DeleteTodoItem(int userId, int itemId);
8 }
```

Listing 4.2: Service Client Interface

Vom Interface werden fünf Implementierungen für die jeweiligen Services benötigt:

- ServiceClientWcf
- ServiceClientWebApi
- ServiceClientWcfData

- ServiceClientThrift
- ServiceClientAzureMS

Eine IServiceClient Implementierung ist zwar keine explizite Fassade, jedoch finden in den Implementierungen Delegierungen statt (z.B. Serviceaufruf und ggf. Konvertierung, Parsing etc.), welche vor dem Interfacenutzer verborgen bleiben.

ServiceClientWcf Für die Nutzung von WCF kann in Windows Store Apps eine sog. Servicereferenz hinzugefügt werden. Hierfür muss die URL der svc-Datei angegeben werden. Es können weitere Einstellungen vorgenommen werden, z.B. welche Listen-Implementierung für Listen verwendet werden soll. Interessant ist hier die Möglichkeit, eine ObservableCollection zu verwenden, weil sich diese direkt für das Binden an XAML-Komponenten eignet. Nach dem Hinzufügen wird der Client-Code generiert mit den vom Service bereitgestellten Methoden sowie Model-Klassen, welche INotifyPropertyChanged implementieren (ebenfalls von Nutzen für Data Binding).

Da Rückgabewerte und Argumente der Interfacemethoden Modelklassen verwenden, welche nicht den generierten Modelklassen des WCF Clients entsprechen, müssen die Objekte konvertiert werden. Da ein Eingriff in die von WCF generierten Klassen nicht möglich ist, gibt es die herkömmliche Möglichkeit, eine statische Utility-Methode zu erstellen, oder das Sprachfeature von C# der sog. Extension Methods. In diesem Fall wird letztere Variante verwendet. Dabei sind vier Extension Methods notwendig - je für die Konvertierung von und nach WCF und lokaler To-do-Liste sowie Item. Als Beispiel ist die Konvertierung „Local TodoList to WCF TodoList“ in Listing 4.3 angegeben.

```
1  public static TodoWcfService.TodoList ToWcfTodoList(
2          this TodoListModel input)
3  {
4      ICollection<TodoWcfService.TodoItem> targetTodoList =
5          new List<TodoWcfService.TodoItem>();
6      foreach (TodoItemModel sourceTodoItem in input.TodoItems)
7      {
8          targetTodoList.Add(sourceTodoItem.ToWcfTodoItem());
9      }
10     return new TodoWcfService.TodoList() {
11         TodoItems = targetTodoList.ToList<TodoWcfService.TodoItem>(),
12         UserId = input.UserId
13         };
```

```
14 }
```

Listing 4.3: Konvertierung von lokalem TodoList Model zu WCF TodoList

REST Client Für den Datenzugriff auf die serverseitige Schnittstelle gibt es in NuGet eine Bibliothek namens RESTSharp, welche HTTP Zugriff und (De-)Serialisierung in lokale Modelklassen übernimmt. Die Bibliothek ist jedoch nur für herkömmliche .NET und Portable Class Libraries verfügbar und nicht für „Class Libraries (Windows Store apps)“. Die Funktionalität musste entsprechend nachgebildet werden. Hierfür wird die Klasse HttpClient verwendet. Für die (De-)Serialisierung wird die aus NuGet bezogene Bibliothek Newtonsoft.Json verwendet. Die bereitgestellte (De-)Serialisierung ermöglicht es, als Quelle bzw. Ziel die lokalen Modelklassen zu verwenden, sodass keine weitere Konvertierung benötigt wird, wie es beim Client für den WCF Services Dienst der Fall ist.

WCF Data Services Client Bei WCF Data Services lässt sich wie bei WCF Services eine Servicereferenz hinzufügen und damit Client Code erzeugen. Dementsprechend muss kein Code für das durchführen von Anfragen mit einem HttpClient geschrieben werden.

Als Referenz auf den entfernten Dienst dient ein sog. Servicekontext. In der Dokumentation des OData Clients [Mic13e] wird die Methode serviceContext.SaveChanges() genannt, welche zuvor gemachte Änderungen, die erst gesammelt werden können, zum Web Service überträgt. Diese Methode steht im OData Client für Windows Store Apps jedoch nicht zur Verfügung, was nicht dokumentiert ist. Eine eigene Implementierung schafft jedoch Abhilfe (siehe Listing 4.4).

```
1 private async Task<DataServiceResponse> SaveChanges(
2         TodoWcfDataService.TodoDBEntities context)
3 {
4     var asyncResult = context.BeginSaveChanges(null, null);
5     return await Task.Factory.FromAsync(asyncResult,
6         ar => context.EndSaveChanges(ar));
7 }
```

Listing 4.4: Eigene SaveChanges Implementierung

Anstatt der Methode BeginSaveChanges eine Callback-Methode als Argument zu übergeben, wird in FromAsync direkt per Lambda-Ausdruck definiert, was beim Beenden der BeginSaveChanges-Methode geschehen soll.

Ein weiteres wichtiges Detail ist, dass das Vormerken von Änderungen, z.B. durch die Methode serviceContext.UpdateObject(object entity), zu einem Fehler führt, wenn die Entität nicht zuvor entweder dem Context hinzugefügt oder mit einem Query abgefragt wurde. [Mic13a] Wie bei WCF Services sind die generierten Modelklassen nicht zu den lokalen Modelklassen kompatibel, sodass entsprechende Konverter Extension Methods geschrieben werden mussten.

Apache Thrift Client Die Apache Thrift Bibliothek aus NuGet lässt sich in Windows Store Apps nicht verwenden. Es gibt jedoch ein Open Source Projekt, welches Apache Thrift auf die WinRT Plattform portiert hat (http://thriftwinrt.codeplex.com/), jedoch nur einen Maintainer, nach dem initialen Upload keinen Commit und keine Dokumentation hat.

Aufgrund der genutzten Klasse Stream war es nicht möglich, eine PCL zu erstellen, sondern es musste eine „Class Library (Windows Store apps)“ erstellt werden. Nach Anpassungen im Quellcode ließ sich dieses Projekt kompilieren und im Client DAL Projekt referenzieren sowie verwenden.

Für diesen Serviceclient mussten entsprechend Unterkapitel 4.3.5 (S. 58) die Methoden explizit asynchron implementiert werden. (siehe Listing 4.5).

```
1 public async Task<int> CreateTodoItem(int userId, string note)
2 {
3    return await Task.Run(() => _client.CreateTodoItem(userId, note));
4 }
```

Listing 4.5: Task starten zur Ermöglichung eines asynchronen Aufrufs

Wie bei WCF Services und WCF Data Services sind die Modelklassen nicht zu den lokalen Modelklassen kompatibel, sodass entsprechende Konverter Extension Methods geschrieben werden mussten.

Azure Mobile Services Client Für die Verwendung von Azure Mobile Services lässt sich eine Client Bibliothek über NuGet beziehen. Die Verwendung ähnelt der OData Client Bibliothek. Hier fällt auf, dass OData Unterstützung in Azure Mobile Services implementiert wurde [Mic13d].

Bei einem Query gibt Azure Mobile Services standardmäßig nur 50 Werte aus. Dies ist für die Verwendung in mobilen Umgebungen sehr hilfreich, da die Menge der übertragenen Daten damit begrenzt wird. Die Anzahl der Entitäten, welche man mit einem Query erhalten möchte, lassen sich frei wählen.

Das im Unterkapitel 4.4.2 (S. 60 ff.) beschriebene Schema reicht zwar aus, um To-do-Listen zu verwalten, jedoch nicht für die Benutzererstellung. Durch das bisherige Testen mit der serverseitigen FileTodoStorage Implementierung anstelle der SqlTodoStorage Implementierung ist dies noch nicht aufgefallen. Beim Design wurde davon ausgegangen, dass eine Zeile in einer Tabelle ohne To-do-Item ID dafür verwendet werden könnte, einen Nutzer anzulegen. Jedoch ist dies erstens bei Azure Mobile Service im Speziellen durch die Beschränkung, dass die To-do-Item ID vom System verwaltet wird und nie leer ist und zweitens durch nötige zusätzliche Berücksichtigungen im Code, der jedoch alleinig für die To-do-Verwaltung zuständig sein soll, nicht möglich bzw. nicht optimal. Gelöst wurde dies durch eine weitere Tabelle.

Wie bei WCF Services, WCF Data Services und Apache Thrift sind die Modelklassen nicht zu den lokalen Modelklassen kompatibel, sodass entsprechende Konverter Extension Methods geschrieben werden mussten. Dadurch, dass die To-do-Item ID von Azure Mobile Services verwaltet wird, ergibt sich hier jedoch eine Besonderheit, die unter „Probleme“ beschrieben wird.

Windows Store App

Die grafische Oberfläche wurde mit XAML, der Beschreibungssprache für grafische Oberflächen mit WPF, Silverlight und Windows Store Apps, beschrieben. Im Code-Behind werden die Buttonclick Events gehandhabt.
Eine Besonderheit ist das Editieren eines To-do-Items. Im entsprechenden Handler lässt sich über den DataContext des Buttons herausfinden, um welches To-do-Item es sich handelt. In dem entsprechenden Objekt ist jedoch die Notiz noch nicht aktualisiert. Um an die geänderte Notiz zu kommen, muss durch den XAML Visual Tree iteriert werden [Nix13]. Der Code in Listing A.4 veranschaulicht dies.

Funktionalität für Messungen wurde vorerst nicht implementiert.

Messsystem

Das Messsystem wurde lediglich als Logging- und Zeitdifferenzberechnung implementiert. Die weiteren geforderten Kennwerte (CPU-Auslastung und Datenübertragungsvolumen) können einfacher mit bestehenden Programmen gemessen werden. Clientseitig wird die Zeit von Beginn einer Servicezugriffsoperation bis zur verarbeiteten Antwort, serverseitig von Beginn einer Datenzugriffsoperation bis zur Antwort gemessen. Clientseitig wird auf die Verarbeitung (z.B. Konversion) gewartet, da die Bedingungen sonst zwischen den unterschiedlichen Services nicht gleich sind (eine interne Messung im WCF Services Client ist beispielsweise gar nicht möglich).

Das Messsystem wurde als eigenständiges Projekt als Portable Class Library so implementiert, dass es sowohl im Server, als auch im Client verwendet werden kann. Da der Server nicht weiß, wann ein Benchmarking stattfindet und wann ein normaler Nutzer mit der App arbeitet, wird hier kontinuierlich gemessen. Im Client hingegen muss nicht gemessen werden, wenn die App vom Benutzer zum Zweck der To-do-Verwaltung verwendet wird. Es gibt jedoch Benchmarking Durchgänge, bei welchen die Messungen am Ende gesammelt ausgewertet werden.

Ein Logging mit Schreibvorgängen auf das Dateisystem kann in dieser PCL nicht implementiert werden, da die Schnittstellen für Datei Ein- und Ausgabe auf den unterschiedlichen Zielplattformen (Windows Store App und .NET 4.5) unterschiedlich ist (bei Windows Store Apps kann beispielsweise nur in einem Sandboxbereich geschrieben werden, so lange der Benutzer nicht ausdrücklich mit einem vom System bereitgestellten Dateiauswahldialog ein Ort auf dem Dateisystem ausgewählt wurde) und die PCL nur gemeinsam genutzten Code enthalten kann.

Windows Store App Erweiterungen

Messsystem Integration Nachdem das Messsystem erstellt wurde, konnte es dem Klassendiagramm aus Unterkapitel 4.3.3 (S. 53) entsprechend mit der Fassade integriert werden.

Des Weiteren wurde eine grafische Oberfläche für das Benchmarking erstellt, sowie eine Automatisierung der Serviceaufrufe. Diese Automatisierung simuliert die Verwendung der App durch einen Benutzer und stellt in Kombination mit dem Messsystem einen Benchmark dar (siehe Listing A.5).

Server DAL Erweiterungen

Messsystem Integration Dem Klassendiagramm aus Unterkapitel 4.3.3 (S. 51 ff.) entsprechend wurde das Messsystem mit einer Fassade integriert.

SqlTodoStorage Als letzter Teil der Implementierung wurde serverseitig neben der FileTodoStorage Implementierung die SqlTodoStorage Implementierung entwickelt.

Optimalerweise würde hier ein ORM wie Entity Data Framework genutzt werden. Ein entsprechendes Entity Data Model Projekt ist durch den OData Service auch bereits vorhanden. Jedoch musste aus Zeitgründen darauf verzichtet werden.

Analog zum Azure Mobile Services Schema wurde nachträglich eine Tabelle für die Nutzerverwaltung erstellt. Als Folge musste das Entity Data Model Projekt, welches im OData Service Projekt verwendet wird, aktualisiert werden.

Probleme bei der Implementierung

FileTodoStorage Bei der FileTodoStorage Implementierung ist bei der Nutzung durch WCF ein Fehler aufgetreten. Da die interne Methode zum Speichern eine ganze Liste als Argument erwartet hat, wurde für das Hinzufügen eines To-do-Items zuvor diese Liste abgerufen, um dieser anschließend ein Item hinzuzufügen. Beim Aufrufen der Methode Add(), ruft dies jedoch eine NotSupportedException hervor mit der Nachricht „Collection was of a fixed size“. Eine Lösungsmöglichkeit ist, eine FixupCollection zu verwenden. In diesem Falle ist dies jedoch keine Option, da dies ein WCF-spezifischer Fix wäre und der DAL WCF-unabhängig sein soll. Die Liste muss daher kopiert werden, was eine negative Auswirkung auf die Bearbeitungsdauer hat und insbesondere in mobilen Umgebungen nachteilig ist.

WCF Data Services Bei der Implementierung von WCF Data Services auf Client- und Serverseite traten viele Fehler und Probleme auf. Diese im einzelnen zu beschreiben wäre an dieser Stelle nicht zielführend, das Auftreten der Probleme im Allgemeinen für eine spätere Bewertung jedoch relevant.

Azure Mobile Services Die zuvor genannte Besonderheit durch die vom System verwaltete To-do-Item ID: Die To-do-Items sind in einigen Anwendungsfällen nicht eindeutig identifizierbar. In der CreateTodoItem Methode wird beim Service ein To-do-Item erstellt, ohne explizit eine ID zu nennen (weil dies nicht möglich ist). Die ID wird jedoch benötigt, um sie im lokalen

To-do-Item Pendant zu halten, damit bei einer späteren Änderung des Items genau dieses identifiziert werden kann. Es muss nach dem Erstellen eines To-do-Items demnach das selbe Item abgefragt werden, um daraus die mittlerweile entfernt hinzugefügte ID auszulesen. Bei der Abfrage an die Liste können als einschränkende Kriterien jedoch nur User ID, To-do-Item Notiz und Completed Wert verwendet werden. Nach dem Filtern sind so ggf. mehrere To-do-Items übrig. Um das Problem zu lösen, müsste die lokale Liste durchgegangen werden um zu prüfen, welche ID dort noch nicht vorhanden ist. Diese ID könnte dann von der CreateTodoItem Methode zurückgegeben werden. Von der Methode aus ist jedoch kein Zugriff auf die lokale Liste vorgesehen. Eine weitere Möglichkeit wäre, zu Beginn der Methode die Liste vom Service abzufragen und die IDs zu speichern und später zu prüfen. Dies geht jedoch mit einer erhöhten Datenübertragung einher.
Da der Fall nur eintritt, wenn User ID, To-do-Item Notiz und Completed Wert inhaltsgleich sind, wird dieses Problem für die Messung im Rahmen dieser Bachelorarbeit als geringfügig angesehen und so belassen.

Messsystem Beim Testen des Messsystems fielen ungewöhnliche Werte auf dem Server auf. Daraufhin wurde ein Testprogramm (siehe Listing lst:measurement-check) zum Untersuchen der Werte geschrieben. Es misst die Zeit von Beginn bis Ende einer Schreiboperation auf die Festplatte. Auf dem Client lieferte das Programm folgende Werte in Millisekunden (mehrere Schreiboperationen nacheinander):
5,0037 1,001 0,9976 1,001 1,001 2,002 0,9956 2,002 1,0003 0,9996
Auf dem Server hingegen ergaben sich folgende Werte:
0 0 0 0 0 0 0 15,6122 0 0
Als mögliche Ursache wird ein Caching-Mechanismus vermutet. Nach Umstellung der ITodo-Storage Implementierung vom Dateisystem auf den SQL Server veränderten sich die Messungen nur unerheblich.

Mögliche Optimierungen

Für die Vereinheitlichung der Interfaces und Vereinfachung der Entwicklung erwarten die UpdateTodoItem Methoden von WCF Services und Apache Thrift als Argument ein ganzes To-do-Item, wie es beim RESTful Web Service aufgrund der Ressourcenorientierung nötig ist. Es ist jedoch möglich, die Datenübertragungsmenge und (De-)Serialisierungsgeschwindigkeit zu verbessern, indem die Methode aufgetrennt wird auf zwei Methoden für einmal ein Update einer Notiz und zum anderen Update der Completed Markierung. Dann müssten statt einem

ganzen Objekt, nur ein String oder Boolean übertragen werden.

Converter Methoden mussten im Laufe der Entwicklung für alle Serviceclients geschrieben werden mit Ausnahme des REST Clients. Für eine bessere Vergleichbarkeit sollte auch hier eine Konvertierung stattfinden, auch wenn diese nicht nötig ist. Es lässt sich nicht sagen, dass das Nicht-benötigen einer Konvertierung ein (alleiniger) Vorteil der Verwendung von ASP.NET Web API ist, da bei Verwendung von jeweils nur einer Technologieimplementierung im Client in keinem Fall eine Konvertierung nötig wäre. Dieser Umstand ändert sich, sobald in der Klasse TodoItems die Liste von TodoItems z.B. eine ObservableCollection<> sein soll, da z.B. vom Apache Thrift Compiler stets nur eine einfache List<> generiert wird.

Hinsichtlich des Fokus auf mobile Umgebungen sind noch einige Optimierungen möglich. So könnte man die Services standardmäßig nur eine begrenzte Anzahl an To-do-Items liefern lassen, wenn eine To-do-Liste angefragt wird. Über ein Indexargument ließen sich weitere To-do-Items abrufen. Im Client könnte ein sog. Paging eingesetzt werden, sodass beim Scrollen durch eine Liste weitere Elemente abgerufen werden, sobald absehbar ist, dass der Benutzer in der Liste weiter scrollt.

Für eine bessere Vergleichbarkeit der vier selbst implementierten Dienste mit Azure Mobile Services, sowie für eine bessere Beurteilung bzgl. des Themas Sicherheit, müsste der Web Server so eingerichtet werden, dass er HTTPS unterstützt und die Clients müssten diese Verbindung nutzen.

4.4.4. Test

Da die Systeme beispielhaft implementiert wurden und nicht in einer Produktivumgebung eingesetzt werden, wurden bei der Entwicklung keine Unittests verwendet. Um nach einer Iteration bei der Entwicklung sicherzustellen, dass die Dienste weiterhin funktionsfähig sind, wurden manuelle Tests durchgeführt.

Der WCF Services Dienst konnte mit dem in Visual Studio 2012 mitgelieferten WcfTestClient getestet werden. Dieser erstellt automatisch Client-Funktionalität für die in der WSDL-Datei bekanntgegebenen vom Service bereitgestellten Methoden und ermöglicht es, diese über eine grafische Benutzeroberfläche zu nutzen.
Der Web API Dienst wurde mit Fiddler getestet, einem Programm zum Sniffen von HTTP Paketen. Einige Testanfragen sind in A.3.1 (S. 102) angegeben.

Der Test des OData Dienstes konnte ebenfalls mit Fiddler durchgeführt werden. Mit der Headerangabe „Accept: application/json;odata=verbose“ kann das Datenformat von XML auf JSON umgestellt werden.
Im Falle des Apache Thrift Services wurde ein kurzes Konsolenprogramm entwickelt, welches die vom Dienst angebotenen Methoden automatisch aufgerufen hat.
Der BaaS-Dienst war nach dem Anlegen von Daten über die Webapp des BaaS-Providers voll funktionsfähig.

Die Tests wurden sowohl auf der lokalen Entwicklungsmaschine, als auch auf dem entfernten Server durchgeführt.

5. Evaluierung und Bewertung

5.1. Anforderungsabgleich

In Tabelle 5.1 wird aufgeführt, welche Anforderungen aus dem Unterkapitel Konzeption 4.2 (S. 35) im Design 4.3 (S. 42) und der Realisierung 4.4 (S. 60) erfüllt wurden.

Die Anzeige von FR C2 ist im Design als GUI Mockup vorhanden. Die Listen(/User-)erstellung ist zwar im Interface im Klassendiagramm, jedoch nicht als Vorgang mit einem Sequenzdiagramm beschrieben.
FR C3 ist mit dem GUI Mockup und dem Interface im Klassendiagramm beschrieben, jedoch nur beispielhaft in einem Sequenzdiagramm erläutert.
FR S4 konnte nicht gänzlich erfüllt werden, da Azure Mobile Services einen eigenen Datenspeicher hat, welcher nicht auf dem eigenen Server verwaltet wird, sodass auf diesen nicht direkt zugegriffen werden kann. Alle anderen Services greifen jedoch auf den gleichen Speicher zu.
Bzgl. FR M2 ist nur die Zeitmessung im Design berücksichtigt und als Softwarekomponente implementiert. Datenübertragungsvolumen und CPU-Last werden mit externen Tools gemessen.
Bzgl. FR M3 findet nur clientseitig die Messung über einen bestimmten Zeitraum statt. Serverseitig werden die Messungen dauerhaft geloggt.
NFR M1: Eine Verifizierung der gemessenen Werte wurde weder im Design vorgesehen, noch implementiert. Eine Garantie, dass Messwerte der Realität entsprechen, ist nur schwierig zu erreichen. Durch das parallele Anwenden mehrerer Messsysteme, welche unterschiedlich vorgehen, hätte zumindest eine Annäherung erreicht werden können.

Für die Bewertung wurden in 4.1.1 (S. 31) gefordert: Antwortzeiten, Datenübertragungsvolumen, Bearbeitungszeiten, Verfügbarkeit (Uptime), maximale Anzahl an Clients, deren Anfragen parallel bearbeitet werden können, CPU-Auslastung von Client und Server, sowie der Implementierungsaufwand.
Gemessen wurden nur Datenübertragungsvolumen, Bearbeitungszeiten und CPU-Auslastung.

Anforderung	Design	Impl.
FR C1 (Startbildschirm, Auswahl Technologie und Messung)	✓	✓
FR C2 (To-do-Liste erstellen und anzeigen)	(✓)	✓
FR C3 (To-do-Liste Verwaltung)	(✓)	✓
FR C4 (Rückkehr zum Startbildschirm)	✓	✓
NFR C1 (GUI einfach und übersichtlich)	✓	✓
NFR C2 (GUI responsiv)	✓	✓
FR S1 (Zentrale To-do-Verwaltung über Dienste erreichbar)	✓	✓
FR S2 (Dienste laufen parallel)	✓	✓
FR S3 (Verwaltungsmöglichkeiten (CRUD))	✓	✓
FR S4 (Unterschiedliche Dienste greifen auf gleichen Speicher zu)	(✓)	(✓)
NFR S1 (Dienste verfügbar wenn Server verfügbar)	✓	✓
FR M1 (Messungen client- und serverseitig)	✓	✓
FR M2 (Messung von Zeit, Datenübertragungsvolumen und CPU-Auslastung)	(✓)	(✓)
FR M3 (Messung über bestimmten Zeitraum)	(✓)	(✓)
NFR M1 (Übereinstimmung Messwerte mit realen Werten)	/	/

Tabelle 5.1.: Erfüllung der Anforderungen

Die Antwortzeiten konnten aus den in 4.4.3 (S. 72) genannten Gründen nicht gemessen werden. Die Verfügbarkeit konnte nicht über einen längeren Zeitraum hinweg gemessen werden, da die Kosten für das Serverhosting das Budget für diese Arbeit überstiegen hätten. Während der Server angeschaltet war, gab es keine Störungen. Die maximale Anzahl an gleichzeitig bearbeiteten Anfragen konnte nicht gemessen werden, da unter Windows 8 nur eine Instanz einer Windows Store App ausgeführt werden kann und keine weiteren Testrechner zur Verfügung standen. Ein HTTP Lasttest hätte sich nur den REST und OData Dienst addressiert. Der Implementierungsaufwand wurde nicht zeitlich genau gemessen. In 5.4 (S. 85) werden jedoch Angaben zum Implementierungsaufwand gemacht.

5.2. Ermittelte Werte

Mit der Kombination aus Beispielapplikation und Messsystem lassen sich nun Messungen durchführen. Dabei werden die in Listing A.5 aufgeführten Methoden ausgeführt, welche eine durchschnittliche Sitzung eines Benutzers simulieren sollen. Somit sind nicht die einzelnen Operationen innerhalb eines, sondern immer nur zwischen den Services zu vergleichen. Dies entspricht auch dem Ziel dieser Arbeit. Des weiteren ist zu beachten, dass sich die Messwerte

auf Technologieimplementierungen (z.B. WCF Services) und nicht auf eine Technologie (z.B. WS-*) beziehen.

Es gibt zwei Besonderheiten bei Azure Mobile Services.

1. Als Übertragungsprotokoll wird standardmäßig und unveränderbar HTTPS verwendet, während die anderen Dienste die Daten mit HTTP übertragen.
2. Der Dienst wird anders als die anderen vier Dienste auf einem separaten Server mit unbekannten Eigenschaften ausgeführt.

Die Verschlüsselung des Datenverkehrs und die unterschiedliche Umgebung resultieren in einer verringerten Vergleichbarkeit des BaaS zu den anderen Diensten. Im Folgenden wird Azure Mobile Services daher angegeben, jedoch ausgeklammert.

Die Messungen wurden mit folgendem Gerät durchgeführt:
Acer W700 mit Intel Core i3-3217U 1.8 GHz, 4 GB RAM, 64 GB SSD, Betriebssystem: Windows 8 64 Bit
Die Internetverbindung erreicht bei einem Benchmark (http://www.speedtest.net) 5,7 Mbit/s Download Geschwindigkeit, 0,52 Mbit/s Upload Geschwindigkeit und eine Latenz von 68 ms.

Die folgenden Diagramme zu den Benchmarkergebnissen zeigen die gesamte Prozessdauer als Summe der Dauer der Einzelprozesse bei den untersuchten Technologieimplementierungen.

Bei Anbindung des Geräts an das Internet über WLAN ergeben sich clientseitig die in Abbildung 5.1 dargestellten Messergebnisse. Die genauen Messergebnisse sind im Anhang in Tabelle A.1 angegeben. Wiederholtes Durchführen der Messung hat zu vergleichbaren Ergebnissen geführt (Abweichung bei 20 durchgeführten Messungen geringer als 10 % bei stets gleichbleibenden Verhältnissen der Gesamtzeiten zueinander).

Eine mobile Umgebung lässt sich z.B. mit einem HTTP-Proxy wie Fiddler simulieren. Fiddler nutzt dazu ein in JavaScript geschriebenes Regelsystem, welches sich anpassen lässt. Die Standardeinstellungen für die Simulierung von Modemgeschwindigkeiten ist in Listing A.1 angegeben.

Mit diesen Einstellungen läuft der Benchmark jedoch nicht erfolgreich durch. Bei Anpassungen der Werte auf ein „request-trickle-delay“ von 100 ms und ein „response-trickle-delay“ von 50

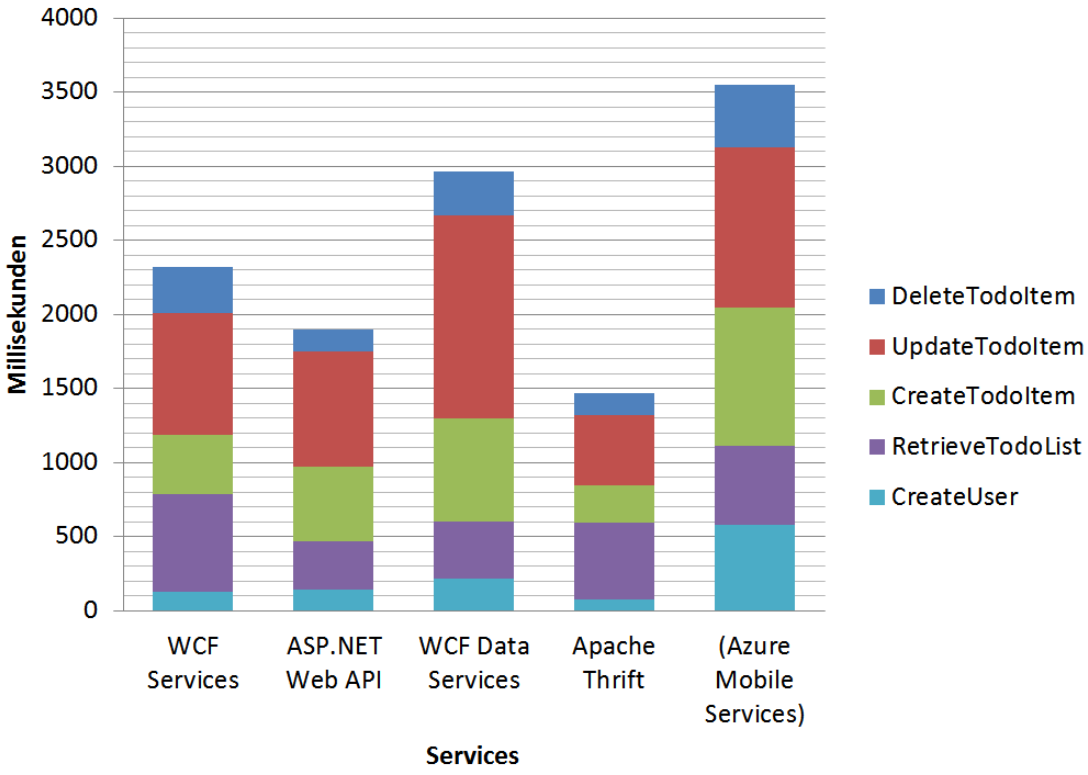

Abbildung 5.1.: Benchmarkergebnisse bei einer Internetanbindung über WLAN

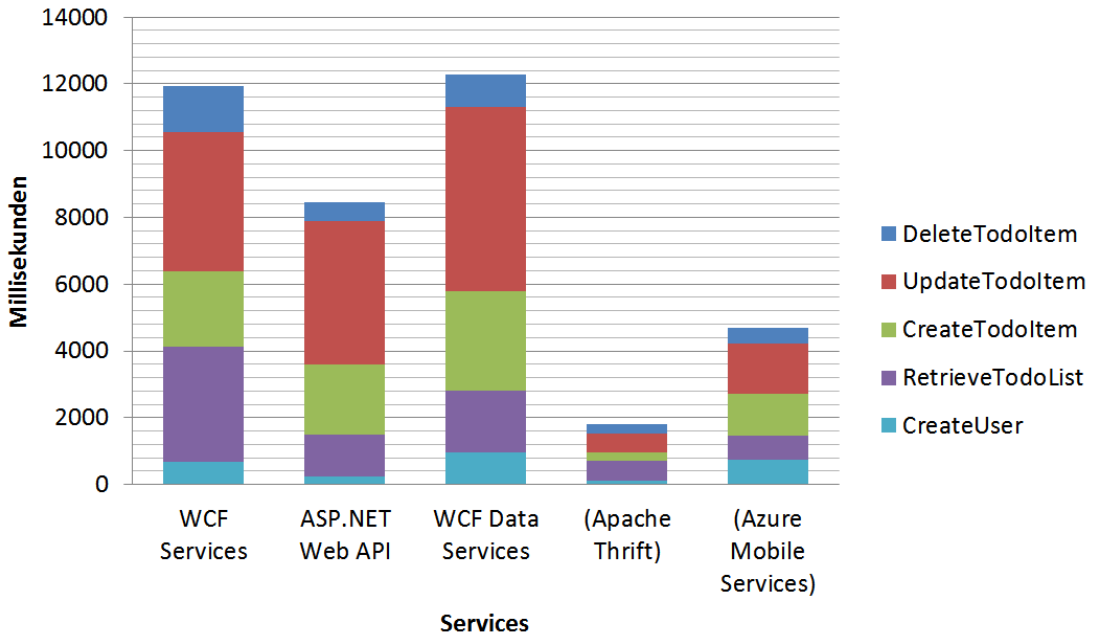

Abbildung 5.2.: Benchmarkergebnisse bei einer Internetanbindung über eine simulierte mobile Datenverbindung

ms ergeben sich die in Abbildung 5.2 dargestellten Werte. Die genauen Messergebnisse sind im Anhang in Tabelle A.2 angegeben. Bei wiederholtem Durchführen der Messung ergaben sich vergleichbare Werte.

Hierbei müssen neben den Werten von Azure Mobile Services auch die von Apache Thrift ausgeklammert werden, weil die Drosselung des Proxys sich nur auf HTTP Datenverkehr auswirkt und Apache Thrift davon ausgeschlossen ist. Auch auf den Datenverkehr mit Azure Mobile Services scheint sich die Drosselung nicht ausgewirkt zu haben. Aus Zeitgründen kann kein komplexerer Proxy wie z.B. WANem eingesetzt werden.
Erkenntnisreicher ist es auch, statt einer Simulation eine echte mobile Datenverbindung zu verwenden. Bei Anbindung des Geräts an das Internet über Mobilfunk mit EDGE-Geschwindigkeit (per WLAN-Tethering über ein Smartphone) ergeben sich jedoch clientseitig Messergebnisse, welche bei mehrfacher Durchführung so unterschiedlich sind, dass man die Daten nicht für einen Vergleich berücksichtigen kann. So benötigt CreateUser bei der Verwendung von WCF Data Services beispielsweise zwischen 1300 und 13200 ms. Oder bei ASP.NET Web API zwischen 1200 ms und 68000 ms. Auch innerhalb eines Benchmarkdurchlaufs sind die

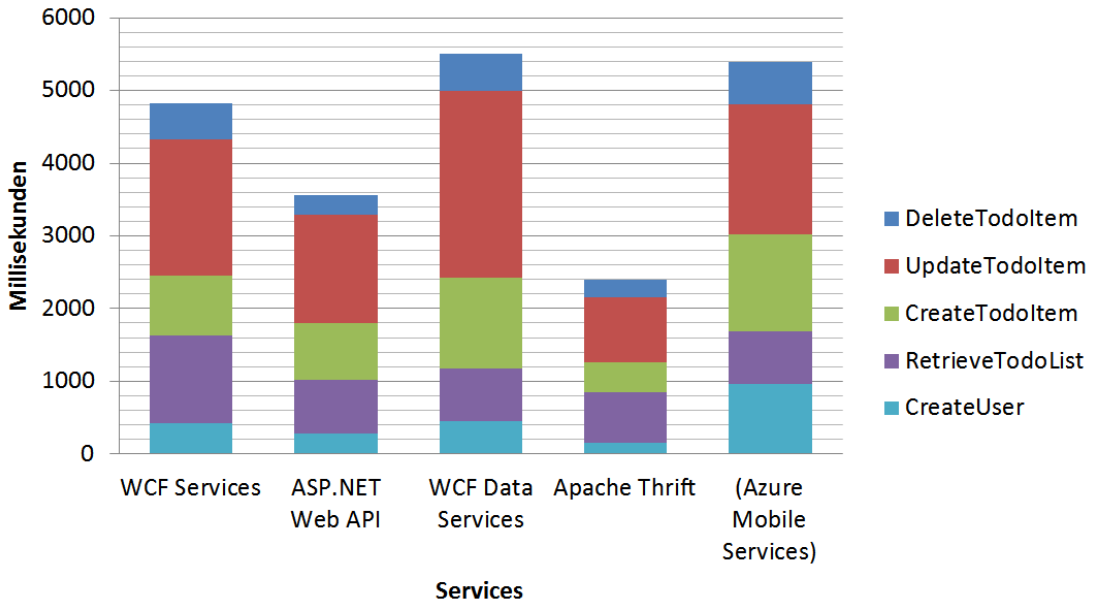

Abbildung 5.3.: Benchmarkergebnisse bei einer Internetanbindung über eine mobile Datenverbindung mit HSPA

Minimum und Maximum Angaben sehr unterschiedlich. Bei Apache Thrift unterscheiden sich die einzelnen Werte bei RetrieveTodoList und UpdateTodoItem bei einer Messung beispielsweise um das 20fache. Eine Messung ist als Beispiel im Anhang in Tabelle A.3 angegeben.

Bei Anbindung des Geräts an das Internet über Mobilfunk mit HSPA-Geschwindigkeit (per WLAN-Tethering über ein Smartphone) sind die Messergebnise bei wiederholtem Durchführen der Messung jedoch besser vergleichbar. Einzelne Messwerte schwankten zwar, das Verhältnis der addiertem Summen zueinander blieb hingegen annähernd konstant. Die Internetverbindung erreichte bei dem Benchmark, der auch bei der vorigen Messung verwendet wurde, 3,87 Mbit/s Download Geschwindigkeit, 1,59 Mbit/s Upload Geschwindigkeit und eine Latenz von 87 ms. Die Werte einer Messung sind in einem Diagramm in Abbildung 5.3 dargestellt. Die genauen Messwerte befinden sich im Anhang in Tabelle A.4.

Die Bearbeitungsdauer auf dem Server ist unabhängig von der Internetanbindung des Clients. Die Werte der Serverlogs für den Testdurchlauf, welcher die obigen clientseitigen

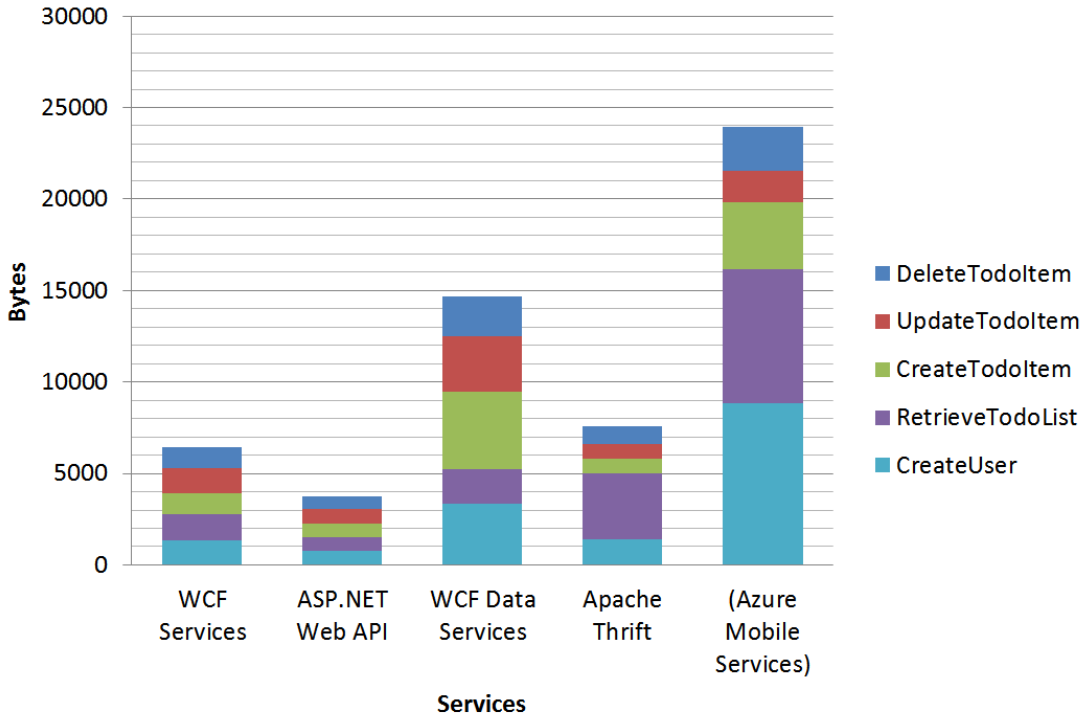

Abbildung 5.4.: Paketgrößen der Übertragungen

Testergebnisse (DSL) lieferte, sind jedoch unbrauchbar (siehe 4.4.3 (S. 74 ff.)).

Mit Wireshark lassen sich die einzelnen Pakete ermitteln, welche zwischen Client und Server ausgetauscht werden. Damit sind die Paketgrößen bekannt. Bei einheitlichen Inhalten der To-do-Items (RetrieveTodoList mit einem To-do-Item „initial“, CreateTodoItem mit einem To-do-Item "New Note", UpdateTodoItem mit dem setzen der Checkbox des To-do-Items "New Note", DeleteTodoItem mit dem Löschen des To-do-Items "New Note") ergeben sich die in Abbildung 5.4 dargestellten Werte (mit addierten PDU-Segmentgrößen und miteingerechneten SYN und ACK Paketen. Die genauen Daten mit Auftrennung nach Ein- und Ausgehenden Daten aus Sicht des Clients sind im Anhang in Tabelle A.5 angegeben.

Die CPU-Last lässt sich mit dem Ressourcenmonitor von Windows beobachten. Hiermit wird nicht die gesamte CPU-Last, sondern nur die eines einzelnen Prozesses (hier der Windows Store App) betrachtet. Dabei wurde der Benchmark jeweils drei Mal direkt hintereinander durchgeführt. Der Wert entspricht dem durchschnittlichen Wert in 60 Sekunden und ist in

WCF S	Web API	WCF DS	Thrift	(Azure MS)
0,7	0,7	1,5	0,7	1,2

Tabelle 5.2.: CPU-Last Client

Zeit	Zeit mobil	Paketgrößen	CPU-Last
1. Thrift 2. Web API 3. WCF S 4. WCF DS 5. (Azure MS)	1. Thrift 2. Web API 3. WCF S 4. (Azure MS) 5. WCF DS	1. Web API 2. WCF S 3. Thrift 4. WCF DS 5. (Azure MS)	Kein Ranking möglich (Erläuterung siehe Text)

Tabelle 5.3.: Ranglisten der Messwerte

Prozent angegeben. Die Werte sind in Tabelle 5.2 dargestellt.

Auf dem Server wurde die Messung auf ähnliche Weise durchgeführt. Hierbei müssen als Prozesse sämtliche SQL Server, IIS und Apache Thrift Prozesse beobachtet werden. Während der Durchführung der Benchmarks ließ sich hier jedoch keine Auswirkung auf die CPU-Last feststellen oder reproduzierbare Werte messen. Eine aussagekräftige Messung ließe sich nur durchführen, wenn eine Vielzahl von Clients gleichzeitig den Server beanspruchen würden. Dies lässt sich mit den vorhandenen Mitteln jedoch nicht durchführen.

5.3. Interpretation

Anhand der gemessenen Werte lassen sich die in Tabelle 5.3 angegebenen Ranglisten erstellen. Die Messwerte der CPU-Last ähneln sich dafür jedoch zu sehr.

Aus dem Unterschied der Reihenfolgen bei den Messungen zur Bearbeitungsdauer und zum Übertragungsvolumen lässt sich schließen, dass die Geschwindigkeit bei den Serviceanfragen nicht nur von der Menge der zu übertragenden Daten abhängt, sondern die Web Services

	WCF Services	ASP.NET Web API	WCF Data Services	Apache Thrift	(Azure MS)
WLAN	0,44	0,5	0,2	0,19	0,15
HSPA	0,91	0,95	0,38	0,32	0,23

Tabelle 5.4.: Berechnete Werte, nachdem die Benchmarkergebnisse und Übertragungsvolumen zueinander ins Verhältnis gesetzt wurden

selbst eine unterschiedliche Verarbeitungsgeschwindigkeit bieten.

Die gemessenen Werte lassen sich auch zueinander ins Verhältnis setzen. Teilt man die Benchmarkergebnisse (addierte Summen) durch die Übertragungsvolumen, erhält man die in Tabelle 5.4 angegebenen Werte. Diese stellen eine abstrakte Kennzahl der Effizienz dar, wie viel Zeit pro übertragene Daten benötigt wird.

Diesen Daten entsprechend lässt sich sagen, dass Apache Thrift und WCF Data Services wesentlich effizienter arbeiten als die anderen beiden Dienste. Azure Mobile Services ist von dieser Betrachtung aus zuvor genannten Gründen (TLS Verschlüsselung) ausgenommen.

5.4. Erfahrungen

Allgemeines

WCF Services, ASP.NET Web API und WCF Data Services haben gemeinsam, dass sie gehostet werden müssen. Hierfür muss sich in die Funktionsweise eines PaaS, in IIS oder einen anderen HTTP Server, Self-Hosting oder eine OWIN-Implementierung eingearbeitet werden. Bei Apache Thrift sowie Azure Mobile Services entfällt dies.

Bei Azure Mobile Services entfällt weiterhin das Implementieren einer Persistenzschicht, wie zum Beispiel in Form einer Anbindung an einen SQL Server. Auch dies ist ein weiteres separates Thema, in welches sich eingearbeitet werden muss.

WCF Services / WS-* Stack, SOAP

Die serverseitige WCF Services Komponente war dank des Visual Studio Projekt Templates einfach zu implementieren. Der von Visual Studio gelieferte Testclient ist sehr hilfreich und die Möglichkeit, in Windows Store Apps einen WCF Services Dienst als Servicereferenz hinzuzufügen und den darauffolgend automatisch generierten Client Code zu verwenden hat die

clientseitige Entwicklung ebenfalls einfach gemacht. Durch die Bereitstellung von Methoden, anstelle von Ressourcen, wodurch man den Service wie ein lokal verfügbares Objekt, welches Methoden bereitstellt, verwenden kann, muss man nicht paradigmenübergreifend entwickeln. Dies erleichtert auch die serverseitige Implementierung. Eine tiefgreifende Kenntnis des WS-* Stacks und SOAP ist nicht nötig, um einen einfachen Web Service bereitstellen zu können. Die Verwendung eines WS-* Stack Frameworks scheint unabdingbar, wenn ein entsprechender Service bereitgestellt werden soll, da der WS-* Stack und SOAP umfangreiche und komplexe Themengebiete sind.

ASP.NET Web API / REST

Bei ASP.NET Web API ist eine Umstellung in der Denkweise gefordert. Das korrekte Bereitstellen von Ressourcen und Unterstützen entsprechender HTTP Methoden erfordert fortgeschrittene Kenntnisse des REST Architekturstils und HTTP Protokolls. Wenn dieses Konzept verstanden ist, ist eine serverseitige Implementierung mit ASP.NET Web API jedoch unkompliziert. Clientseitig ist eine Zwischenschicht, welche mit einem HTTP Client die Kommunikation mit dem Server betreibt und Methoden für lokale Aufrufe bereitstellt, nahezu unabdingbar.

WCF Data Services / OData

Die Tools für die Erstellung und Komsumierung eines WCF Data Services ermöglichen es, einen grundlegenden OData-konformen Dienst bereitzustellen und zu verwenden, ohne sich mit OData befassen zu müssen. Interessant ist, dass so auf einfache Weise eine ganze Datenbank veröffentlicht werden kann. Um die Logik muss sich im Gegenzug jedoch die Clientseite befassen, wodurch hier ein erhöhter Aufwand entsteht. Für eine Kontrolle des serverseitigen Dienstes ist eine Einarbeitung in einen ORM wie z.B. Entity Framework nötig. Die Tools, sowie WCF Data Services und das ganze Thema OData generell scheinen nicht weit verbreitet zu sein, was die Lösungssuche bei auftretenden Problemen erschwert. In der Theorie scheint OData jedoch ein durchdachtes, zukunftsfähiges Konzept zu sein.

Apache Thrift / Binärprotokolle

Zwar muss man für die Verwendung von Apache Thrift die IDL erlernen und benötigt immer den zusätzlichen Schritt der Codegenerierung durch den Apache Thrift Compiler, jedoch erhält man dadurch einen funktionsfähigen Service, der noch dazu (dem Zweck von Binärprotokollen entsprechend) der performanteste ist. Das Entfallen eines Hostings erleichtert das Deployment. Ähnlich wie bei WCF Services vereinfacht das Arbeiten mit bereitgestellten Methoden die

Integration in den Client. Erschwerend ist jedoch der Umstand, dass Apache Thrift nicht offiziell WinRT unterstützt.
Zu anderen Binärprotokollen kann an dieser Stelle nichts gesagt werden, da sie sich sehr voneinander unterscheiden. Allen gemeinsam dürfte eine hohe Performanz sein, jedoch ist schon die Möglichkeit, einen lauffähigen Service automatisch zu generieren eine Besonderheit.

Azure Mobile Services / BaaS

Mit Azure Mobile Services entfällt jeglicher Aufwand, der für die Bereitstellung eines Web Services nötig ist. Lediglich das Datenmodell muss erstellt werden, was einfach in der Management Webapp von Windows Azure durchgeführt werden kann. Clientbibliotheken erleichtern die clientseitige Entwicklung. Ohne diesen Bibliotheken müsste man sich in OData einarbeiten, dessen Query Syntax verwendet wird. Ein negativer Aspekt ist jedoch die Performanz von Azure Mobile Services, welche unter allen anderen Services lag. Dies kann jedoch bei anderen Backend-as-a-Service Providern besser sein. Allen gemeinsam ist das Entfallen der Einrichtung einer serverseitigen Umgebung. Um der Anpassbarkeit eigens implementierter Web Services nicht nachzustehen, bieten die meisten BaaS-Provider die Möglichkeit an, serverseitigen Code zu schreiben. Dies ist jedoch meist auf eine bestimmte Programmiersprache wie z.B. Javascript begrenzt. Vom Provider bereitgestellte Plugins für die Anbindung an soziale Netzwerke oder das Integrieren von Push-Diensten oder einer Messaging-Umgebung erweitern die Funktionalität des BaaS bei geringem Integrationsaufwand erheblich.

5.5. Bewertung

Anhand der Informationen aus den theoretischen Eigenschaften der Technologien für den Datenaustausch, Messwerten von der Verwendung bestimmter Technologieimplementierungen und Erfahrungen bei der Implementierung lassen sich Empfehlungen für unterschiedliche Szenarien ableiten. Dabei ist zu berücksichtigen, dass die Messwerte bei anderen Technologieimplementierungen (z.B. ServiceStack anstelle von WCF Services) oder in anderen Umgebungen anders ausfallen können, sowie vor allem die Anforderungen von Szenarien oder Entwicklern sehr unterschiedlich sein können. Im folgenden werden einige mögliche Einsatzszenarien in mobilen Umgebungen beschrieben und entsprechende Empfehlungen für den Einsatz von Technologien ausgesprochen.

Inhouse Business Anwendung

Mögliche Anforderungen:

- Die Anwendung soll von Mitarbeitern eines Unternehmens eingesetzt werden. Die Anwendung soll an bestehende Systeme im Unternehmen eingebunden werden.
- Die Anwendung soll intern in einem Partnerunternehmen oder einem Unternehmen, welches Kunde des eigenen Unternehmens ist, verwendet werden. Es soll auf Daten zugegriffen werden, welche sich in den Systemen des eigenen Unternehmens befinden.

Beispiele für dieses Szenario sind ein Kassensystem für Restaurants, in welchen die Bedienung Bestellungen per Mobilgerät annimmt und abrechnet; eine Anwendung welche als Logbuch für den Außendienst eines Telekommunikationsdienstleisters dient; oder eine Anwendung für Vertriebsmitarbeiter und Geschäftsführer, die mobil z.B. für Presentationen auf aktuelle Unternehmensdaten wie Geschäftskennzahlen zugreifen wollen.

Empfehlung:
Davon ausgehend, dass in unternehmensinternen Systemen häufig bereits der WS-* Stack zum Einsatz kommt und interne Entwickler sich bereits mit der Technologie auskennen, sowie die Tatsachen, dass WS-* zahlreiche standardisierte Erweiterungen für sicherheitskritische und komplexe Anwendungsszenarien bietet, die im Businessbereich häufig vorkommen, und es an Unternehmen gerichtete Lösungen für die Integration in ESB-Systeme gibt, wird für diesen Zweck der WS-* Stack empfohlen. Die theoretischen Eigenschaften sind hierbei ausschlaggebend, während die Messwerte und die Erfahrungen nicht dagegen sprechen.

Ist eine Integration in bestehende Systeme, welche bereits mit dem WS-* Stack oder ESB arbeiten, nicht nötig, dann lässt sich auch REST verwenden. Gegen OData spricht die geringe Etabliertheit und das unausgereifte Tooling. Binärprotokolle eignen sich wegen der geringen Interoperabilität nicht so gut. Bei BaaS spricht die geringe Anpassparkeit gegen einen Einsatz in einer Inhouse Business Anwendung. Hier bieten jedoch immer mehr Anbieter eine Anpassbarkeit durch die Möglichkeit serverseitig z.B. JavaScript Code bei bestimmten Ereignissen (wie das Einfügen von neuen Daten) oder in zeitlichen Abständen auszuführen.

Öffentliche API

Mögliche Anforderung:
Es soll lediglich ein Service angeboten werden, welche beliebige andere Unternehmen oder freie Entwickler in eigenen mobilen Anwendungen einbinden können. Die Benutzung der API soll kostenfrei sein mit dem Ziel, die Reichweite des Dienstanbieters zu erhöhen oder kostenpflichtig sein mit dem Ziel, den Umsatz direkt zu steigern. Es sollen möglichst viele

Entwickler die API verwenden.

Beispiele für dieses Szenario sind ein Wetterservice, welcher seinen Dienst bekannter machen, jedoch keine Entwicklung einer Anwendung finanzieren kann oder möchte; ein soziales Netzwerk, welches durch die Integration in Produkten anderer Entwickler eine größtmögliche Penetration und Vernetzung erreichen möchte; oder ein Börseninformationsportal, welches die aktuellen Wertpapierkurse über eine kostenpflichtige API bereitstellen möchte.

Empfehlung:
Um die Anforderung zu Erfüllen, dass möglichst viele Entwickler die API verwenden, sollte sie einfach zu verstehen und zu verwenden sein, sowie sicher, schnell und ein möglichst geringes Datenübertragungs aufweisen. Als bester Kompromiss wird eine RESTful API erachtet. Zwar besteht auf Server- und Clientseite das Problem des Umdenkens von Methodenaufrufen nach Ressourcen mit Zugriff über das Uniform Interface von HTTP, jedoch sind eine einfache Sicherung mit TLS möglich, sowie die laut Messungen geringste Datenübertragungsgröße bei nur wenig längerer Bearbeitungsdauer als Apache Thrift. Außerdem sind HTTP Clients für jede verbreitete Programmiersprache verfügbar. Ebenso ermöglicht die Ressourcenorientierung das Verwenden von Hyperlinks und Präsenz im Web, was bei den anderen untersuchten Technologien, abgesehen von OData, durch den Weg über einen zentralen Servicepunkt nicht möglich ist.

Schon durch geringe Änderungen an den Anforderungen kann diese Empfehlung jedoch ihre Gültigkeit verlieren.

Open Data

Mögliche Anforderung:
Die gesamten oder ausgewählten Daten eines Systems sollen der breiten Öffentlichkeit und in maschinell verarbeitbarer Form zur Verfügung gestellt werden. Eine zugehörige mobile Anwendung soll nicht, kann jedoch von beliebigen Dritten entwickelt werden.

Beispiele für dieses Szenario sind Verkehrsmittelunternehmen, die ihre Fahrplandaten veröffentlichen wollen; öffentliche Stellen der Regierung, die gesetzlich dazu verpflichtet sind ihre Ausgaben zu veröffentlichen; oder Nichtregierungsorganisationen und Stiftungen, welche mit der Veröffentlichung von Daten zu Mithilfe animieren oder Transparenz erreichen wollen.

Empfehlung:
Wenn serverseitig keine Businesslogik nötig ist und ganze Datenbestände veröffentlicht werden sollen, eignet sich OData am besten. Die herausstellende Eigenschaft ist hier das einfache Veröffentlichen einer Datenbank in einem standardisierten Format. Die anderen betrachteten Technologien ermöglichen dies nicht oder nicht so einfach. Entsprechend den genannten Beispielen sind weder eine hohe Verarbeitungsgeschwindigkeit, noch eine geringe Datenübertragung Primärziele, welche gegen OData sprechen würden.

Back-End für App-Entwickler

Mögliche Anforderung:
Ein einzelner oder ein Team von Entwicklern mit Erfahrung in der Entwicklung von mobilen Applikationen - nicht jedoch von Back-End-Systemen - benötigt für eine Anwendung ein Back-End. Dieses dient ausschließlich für diese Anwendung als zentraler Datenspeicher.

Beispiele für dieses Szenario gibt es viele, da eine große Zahl an mobilen Anwendungen mit diesen Anforderungen entwickelt wird. Dies können eine mobile Spieleanwendung sein, welche ein Back-End für die Verwaltung von Highscores benötigt; eine Anwendung für ein allgemeines oder auf den Austausch von Fotos spezialisiertes soziales Netzwerk; eine Anwendung, welche die Newsfeeds und gelesenen Artikel des Benutzers verwaltet; oder beliebige weitere Anwendungen, die ein Back-End nur zum Speichern von Anwendungseinstellungen des Benutzers benötigen.

Empfehlung:
Die einzige Lösung für den Einsatz eines Back-Ends unter dem Umstand, dass die Entwickler keine bzw. wenig Erfahrung in diesem Bereich haben oder auch keine Zeit haben, sich mit dem Thema zu beschäftigen, ist ein Backend-as-a-Service. Auch wenn Erfahrung vorhanden ist, kann durch den Einsatz eines BaaS viel Zeit gespart und stattdessen ein Prototyp der Anwendung entwickelt werden.

6. Zusammenfassung und Ausblick

6.1. Zusammenfassung

Diese Arbeit untersuchte unterschiedliche aktuelle Technologien für den Datenaustausch. Mit entsprechenden Technologieimplementierungen für eine einheitliche Plattform wurde eine Beispielanwendung entwickelt, welche aus einem Back-End und einer mobilen Anwendung besteht, in denen die zu vergleichenden Technologien parallel zum Einsatz kommen. Damit konnte ein Vergleich durchgeführt werden.

In 2 (S. 4 ff.) wurden die theoretischen Grundlagen dieser Technologien vorgestellt und in 3 (S. 22 ff.) anhand ausgewählter Eigenschaften gegenübergestellt.

Anschließend wurde in 4 (S. 30) eine Beispielanwendung entworfen und implementiert. Hierfür wurden Implementierungen der Technologien ausgewählt, Anforderungen ermittelt, die Anwendung client- und serverseitig nach objektorientierter Methodologie entworfen und implementiert. Ebenso wurde ein Messsystem integriert.

Zuletzt wurde in 5 (S. 77) die Implementierung evaluiert sowie mit dem Messsystem und externen Tools Messungen bei Verwendung der unterschiedlichen Technologieimplementierungen durchgeführt. Anhand dieser Werte konnte eine weitere Gegenüberstellung, diesmal der praktischen Eigenschaften, durchgeführt werden. Außerdem wurden Erfahrungen bei der Implementierung genannt.
Diese drei Informationen wurden bei der Empfehlung genutzt, welche unterschiedliche mögliche Anwendungsszenarien berücksichtigt.

6.2. Ausblick

6.2.1. Arbeit

Es gibt mehrere Bereiche, welche in Folgearbeiten untersucht werden könnten, um Entscheidungen für oder gegen eine Technologie und Technologieimplementierung noch fundierter treffen zu können.
So könnten noch weitere alternative Implementierungen hinzugenommen werden. Es bieten sich beispielsweise ServiceStack als quelloffene Alternative für WCF Services und Nancy als quelloffene Alternative für ASP.NET Web API an. Dank der komponentenbasierten Entwicklung ist eine Erweiterung um diese Services sehr einfach möglich.
Aber auch eine gänzlich andere Plattform als .NET sollte als Alternative untersucht werden. So gibt es beispielsweise für jede der untersuchten Technologien auch Implementierungen für Java.

6.2.2. Technologie

Im Kapitel 2 (S. 4 ff.) wurde eine geringe Adaption von UDDI festgestellt. Während Bücher wie [Mel10] davon ausgingen, dass diese Adaption zunehmen wird, hat sich eher gezeigt, dass sie abnimmt. Ebenso ist der WS-* Stack schon vielerorts einer anderen Art Web Service gewichten: RESTful Web Services. Die Entwicklung von WADL kann dafür sorgen, dass dieser Trend weiter fortgesetzt wird. Bremsend kann sich hierauf die ereignisgetriebene SOA („SOA 2.0“) auswirken, da reaktive Programmierung immer populärer wird.

Unabhängig davon findet an unterschiedlichen Stellen eine Weiterentwicklung statt, welche eine Unterstützung von Semantik in die Technologien einbringen möchte. Hier seien OWL und RDF als Beispiele genannt.

OData als Versuch einen Standard für RESTful Web Services zu erstellen, wird kontinuierlich weiterentwickelt und in Business Produkte integriert, scheint allerdings dennoch nicht an Popularität hinzuzugewinnen.

Binärprotokolle scheinen innerhalb von Unternehmen wie Google oder Facebook weit verbreitet zu sein, die noch fehlende Interoperabilität zwischen den unterschiedlichen Formaten verhindert jedoch eine weitere Verbreitung in der Öffentlichkeit. Integrationen wie die von Protocol Buffers in ServiceStack sorgen dafür, dass die Vorteile aus der Welt des WS-*

Stacks und der Binärprotokolle kombiniert werden.

Großes Wachstumspotenzial liegt bei BaaS. Diese ermöglichen es App-Entwicklern, sich eine serverseitige Infrastruktur zu erstellen, ohne sich mit Web Services oder Web-Service ähnlichen Technologien im Einzelnen auseinandersetzen zu müssen. Diese Verringerung der Einstiegshürde, verbunden mit der steigenden Anzahl an App-Entwicklern wird voraussichtlich für einen Anstieg der Verwendung dieser Methode sorgen.

A. Anhang

A.1. Inhalt der CD-ROM

1. Verzeichnis „Abbildungen“: Die in der Bachelorarbeit verwendeten Abbildungen
2. Verzeichnis „Logs Server“: Logs der Messungen auf dem Server
3. Verzeichnis „Quellcode Apache Thrift“: Die Eingabedatei für den Apache Thrift Compiler und dessen Ausgabe
4. Verzeichnis „Quellcode Projekte“: Die Visual Studio Projektverzeichnisse aller Klassenbibliotheks-, Client- und Serverprojekte
5. Verzeichnis „Screenshots GUI“: Screenshots der Benutzeroberfläche der Anwendung
6. Verzeichnis „Screenshots Webseiten“: Screenshots der Webseiten für die Archivierung der Webseitenzustände zur Zeit des Abrufs
7. Datei „BA-2013-Philipp_Gille.pdf“: Bachelorarbeit im PDF-Format

A.2. Anwendungsfälle

Die Beschreibungsschablone ist eine vereinfachte und angepasste Variante der Anwendungsfallschablone von Alistair Cockburn.
Die Namen richten sich nach folgendem Schema:

- UC: Use Case
- C: Client; S: Server; M: Measuring system
- <number>: Fortlaufende Nummer

A.2.1. Client

UC C1 To-do-Liste erstellen

Beschreibung: Eine To-do-Liste wird erstellt
Beteiligte Akteure: Benutzer
Verwendete Anwendungsfälle: UC S1 To-do-Liste speichern
Erweiternde Anwendungsfälle: UC M2 Zeit messen, UC M3 Datenübertragungsvolumen messen, UC M4 CPU-Auslastung messen
Erweiterte Anwendungsfälle: /
Auslöser: Der Benutzer startet den Client erstmalig; Messsystem
Vorbedingungen: /
Ergebnis: Es wurde eine To-do-Liste erstellt und der Client hat sich gemerkt, dass beim nächsten Starten keine neue To-do-Liste erstellt, sondern die zuvor erstellte geladen werden soll.
Standardablauf: Der Benutzer startet den Client. Der Client prüft, ob er das erste Mal gestartet wurde. Ist dies der Fall, wird es dem Server mitgeteilt. Der Client erhält vom Server eine Kennung, mit der die To-do-liste identifiziert werden kann. Die Kennung wird lokal gespeichert.

UC C2 To-do-Liste anzeigen

Beschreibung: Eine To-do-Liste wird angezeigt
Beteiligte Akteure: Benutzer
Verwendete Anwendungsfälle: UC S1 To-do-Liste laden
Erweiternde Anwendungsfälle: UC M2 Zeit messen, UC M3 Datenübertragungsvolumen messen, UC M4 CPU-Auslastung messen
Erweiterte Anwendungsfälle: /
Auslöser: Der Benutzer startet den Client; Messsystem
Vorbedingungen: Es wurde eine To-do-Liste erstellt
Ergebnis: Auf einer Benutzeroberfläche wird die To-do-Liste des Benutzers angezeigt.
Standardablauf: Der Client wird gestartet. Er liest die Kennung der To-do-Liste aus, die zu dem Client gehört. Die Kennung wird an den Server geschickt. Der Client erhält vom Server eine To-do-Liste. Der Client zeigt diese To-do-Liste an.

UC C3 To-do-Item erstellen

Beschreibung: Ein To-do-Item wird erstellt
Beteiligte Akteure: Benutzer
Verwendete Anwendungsfälle: UC S1 To-do-Item speichern

Erweiternde Anwendungsfälle: UC M2 Zeit messen, UC M3 Datenübertragungsvolumen messen, UC M4 CPU-Auslastung messen
Erweiterte Anwendungsfälle: /
Auslöser: Der Benutzer betätigt die Schaltfläche zum Erstellen eines To-do-Items; Messsystem
Vorbedingungen: Es wurde eine To-do-Liste erstellt
Ergebnis: Es wurde ein To-do-Item erstellt; der Client hat eine Kennung für das To-do-Item erhalten
Standardablauf: Der Benutzer schreibt Text in ein entsprechendes Feld und betätigt anschließend die Schaltfläche zum erstellen eines To-do-Items. Das To-do-Item wird an den Server geschickt. Der Client erhält eine Kennung für das To-do-Item. Die angezeigte To-do-Liste wird um das To-do-Item erweitert.

UC C4 To-do-Item als erledigt markieren

Beschreibung: Ein To-do-Item wird als erledigt markiert
Beteiligte Akteure: Benutzer
Verwendete Anwendungsfälle: UC S1 To-do-Item bearbeiten
Erweiternde Anwendungsfälle: UC M2 Zeit messen, UC M3 Datenübertragungsvolumen messen, UC M4 CPU-Auslastung messen
Erweiterte Anwendungsfälle: /
Auslöser: Der Benutzer betätigt die Checkbox eines To-do-Items; Messsystem
Vorbedingungen: Es wurde eine To-do-Liste erstellt; es wurde ein To-do-Item erstellt
Ergebnis: Das To-do-Item wurde als erledigt markiert
Standardablauf: Der Benutzer betätigt die Checkbox eines To-do-Items. Der Client teilt dem Server die Kennung der Liste, des Items und den neuen Wert der Checkbox mit. Die angezeigte To-do-Liste wird entsprechend angepasst.

UC C5 To-do-Item bearbeiten

Beschreibung: Ein To-do-Item wird bearbeitet
Beteiligte Akteure: Benutzer
Verwendete Anwendungsfälle: UC S1 To-do-Item bearbeiten
Erweiternde Anwendungsfälle: UC M2 Zeit messen, UC M3 Datenübertragungsvolumen messen, UC M4 CPU-Auslastung messen
Erweiterte Anwendungsfälle: /
Auslöser: Der Benutzer betätigt die Schaltfläche zum Speichern eines vorhandenen To-do-Items; Messsystem

Vorbedingungen: Es wurde eine To-do-Liste erstellt; es wurde ein To-do-Item erstellt
Ergebnis: Das To-do-Item wurde bearbeitet
Standardablauf: Der Benutzer verändert den Text eines vorhandenen To-do-Items. Anschließend betätigt er die Schaltfläche zum Speichern dieses To-do-Items. Der Client teilt dem Server die Kennung der Liste, des Items und den neuen Text des Items mit. Die angezeigte To-do-Liste wird entsprechend angepasst.

UC C6 To-do-Item löschen

Beschreibung: Ein To-do-Item wird gelöscht
Beteiligte Akteure: Benutzer
Verwendete Anwendungsfälle: UC S1 To-do-Item löschen
Erweiternde Anwendungsfälle: UC M2 Zeit messen, UC M3 Datenübertragungsvolumen messen, UC M4 CPU-Auslastung messen
Erweiterte Anwendungsfälle: /
Auslöser: Der Benutzer betätigt die Schaltfläche zum Speichern eines vorhandenen To-do-Items; Messsystem
Vorbedingungen: Es wurde eine To-do-Liste erstellt; es wurde ein To-do-Item erstellt
Ergebnis: Das To-do-Item wurde gelöscht
Standardablauf: Der Benutzer betätigt die Schaltfläche zum Löschen eines bestimmten To-do-Items. Die zugehörige Kennung wird dem Server mitgeteilt. Aus der angezeigten To-do-Liste wird das To-do-Item entfernt.

A.2.2. Server

UC S1 To-do-Liste speichern

Beschreibung: Eine To-do-Liste wird gespeichert
Beteiligte Akteure: /
Verwendete Anwendungsfälle: /
Erweiternde Anwendungsfälle: UC M2 Zeit messen, UC M3 Datenübertragungsvolumen messen, UC M4 CPU-Auslastung messen
Erweiterte Anwendungsfälle: /
Auslöser: Der Server bekommt eine Anfrage zum Speichern einer To-do-Liste
Vorbedingungen: /
Ergebnis: Eine To-do-Liste wurde gespeichert; eine Kennung wurde dem Absender der Anfrage geschickt

Standardablauf: Der Server bekommt eine Anfrage zum Speichern einer To-do-Liste. Er wählt eine Kennung zur Identifizierung der To-do-Liste aus und speichert die Liste. Er schickt die Kennung an den Absender der Anfrage.

UC S2 To-do-Liste laden

Beschreibung: Eine To-do-Liste wird geladen
Beteiligte Akteure: /
Verwendete Anwendungsfälle: /
Erweiternde Anwendungsfälle: UC M2 Zeit messen, UC M3 Datenübertragungsvolumen messen, UC M4 CPU-Auslastung messen
Erweiterte Anwendungsfälle: /
Auslöser: Der Server bekommt eine Anfrage zum Laden einer To-do-Liste
Vorbedingungen: Eine To-do-Liste mit der entsprechenden Kennung wurde gespeichert
Ergebnis: Die To-do-Liste wurde geladen und dem Absender der Anfrage geschickt
Standardablauf: Der Server bekommt eine Anfrage zum Laden einer To-do-Liste inklusive einer Kennung zur Identifikation der Liste. Der Server lädt die Liste und schickt sie an den Absender der Anfrage.

UC S3 To-do-Item speichern

Beschreibung: Ein To-do-Item wird gespeichert
Beteiligte Akteure: /
Verwendete Anwendungsfälle: /
Erweiternde Anwendungsfälle: UC M2 Zeit messen, UC M3 Datenübertragungsvolumen messen, UC M4 CPU-Auslastung messen
Erweiterte Anwendungsfälle: /
Auslöser: Der Server bekommt eine Anfrage zum Speichern eines To-do-Items
Vorbedingungen: Eine To-do-Liste wurde mit der Kennung gespeichert, die in dieser Anfrage für die Identifikation der Liste enthalten ist
Ergebnis: Das To-do-Item wurde in die angegebene To-do-Liste gespeichert und dem Absender der Anfrage die zugehörige Kennung zur Identifikation des To-do-Items zugesendet.
Standardablauf: Der Server bekommt eine Anfrage zum Speichern eines To-do-Items. In der Anfrage ist die Kennung zur Identifizierung der Liste enthalten, in die das To-do-Item gespeichert werden soll. Der Server ermittelt eine Kennung für das To-do-Item, welches noch nicht in der Liste verwendet wird. Der Server speichert das To-do-Item in die Liste. Der Server schickt dem Absender der Anfrage die Kennung des To-do-Items.

UC S4 To-do-Item bearbeiten

Beschreibung: Ein To-do-Item wird bearbeitet
Beteiligte Akteure: /
Verwendete Anwendungsfälle: /
Erweiternde Anwendungsfälle: UC M2 Zeit messen, UC M3 Datenübertragungsvolumen messen, UC M4 CPU-Auslastung messen
Erweiterte Anwendungsfälle: /
Auslöser: Der Server bekommt eine Anfrage zum Bearbeiten eines To-do-Items
Vorbedingungen: Eine To-do-Liste mit der entsprechenden Kennung wurde gespeichert; ein To-do-Item mit der entsprechenden Kennung wurde erstellt
Ergebnis: Das angegebene To-do-Item der angegebenen Liste wurde bearbeitet
Standardablauf: Der Server bekommt eine Anfrage zum Bearbeiten eines To-do-Items. In der Anfrage ist der neue Wert für „Erledigt“ oder der neue Text enthalten, die Kennung zur Identifizierung der Liste enthalten, sowie die Kennung des To-do-Items. Der Server bearbeitet dieses To-do-Item entsprechend.

UC S5 To-do-Item löschen

Beschreibung: Ein To-do-Item wird gelöscht
Beteiligte Akteure: /
Verwendete Anwendungsfälle: /
Erweiternde Anwendungsfälle: UC M2 Zeit messen, UC M3 Datenübertragungsvolumen messen, UC M4 CPU-Auslastung messen
Erweiterte Anwendungsfälle: /
Auslöser: Der Server bekommt eine Anfrage zum Löschen eines To-do-Items
Vorbedingungen: Eine To-do-Liste mit der entsprechenden Kennung wurde gespeichert; ein To-do-Item mit der entsprechenden Kennung wurde erstellt
Ergebnis: Das angegebene To-do-Item der angegebenen Liste wurde gelöscht
Standardablauf: Der Server bekommt eine Anfrage zum Löschen eines To-do-Items. In der Anfrage ist die Kennung zur Identifizierung der Liste enthalten, sowie die Kennung des To-do-Items. Der Server löscht dieses To-do-Item.

A.2.3. Messsystem

UC M1 Messung durchführen

Beschreibung: Es wird eine Messung durchgeführt
Beteiligte Akteure: Benutzer
Verwendete Anwendungsfälle: UC C1 To-do-Liste erstellen, UC C2 To-do-Liste anzeigen, UC C3 To-do-Item erstellen, UC C4 To-do-Item als erledigt markieren, UC C5 To-do-Item bearbeiten, UC C6 To-do-Item löschen
Erweiternde Anwendungsfälle: /
Erweiterte Anwendungsfälle: /
Auslöser: Der Benutzer betätigt die Schaltfläche zum Starten einer Messung
Vorbedingungen: /
Ergebnis: Eine Messung wurde durchgeführt
Standardablauf: Der Benutzer betätigt die Schaltfläche zum Starten einer Messung. Es werden sämtliche Anwendungsfälle des Clients automatisch ausgelöst. Durch die Art der Auslösung finden die Messungen der Anwendungsfälle UC 3.2 Zeit messen, UC 3.3 Datenübertragungsvolumen messen, sowie UC 3.4 CPU-Auslastung messen statt. Die Messergebnisse werden entweder angezeigt oder gespeichert.

UC M2 Zeit messen

Beschreibung: Die Dauer einer Aktion wird gemessen
Beteiligte Akteure: /
Verwendete Anwendungsfälle: /
Erweiternde Anwendungsfälle: /
Erweiterte Anwendungsfälle: UC C1 To-do-Liste erstellen, UC C2 To-do-Liste anzeigen, UC C3 To-do-Item erstellen, UC C4 To-do-Item als erledigt markieren, UC C5 To-do-Item bearbeiten, UC C6 To-do-Item löschen, UC S1 To-do-Liste speichern, UC S2 To-do-Liste laden, UC S3 To-do-Item speichern, UC S4 To-do-Item bearbeiten, UC S5 To-do-Item löschen
Auslöser: Messsystem
Vorbedingungen: /
Ergebnis: Die Dauer der Aktion wurde gemessen
Standardablauf: Vor Beginn und nach Ende der Aktion wird die Systemzeit ausgelesen und die Differenz dem Messsystem mitgeteilt.

UC M3 Datenübertragungsvolumen messen

Beschreibung: Das Datenübertragungsvolumen einer Aktion wird gemessen
Beteiligte Akteure: /
Verwendete Anwendungsfälle: /
Erweiternde Anwendungsfälle: /
Erweiterte Anwendungsfälle: UC C1 To-do-Liste erstellen, UC C2 To-do-Liste anzeigen, UC C3 To-do-Item erstellen, UC C4 To-do-Item als erledigt markieren, UC C5 To-do-Item bearbeiten, UC C6 To-do-Item löschen, UC S1 To-do-Liste speichern, UC S2 To-do-Liste laden, UC S3 To-do-Item speichern, UC S4 To-do-Item bearbeiten, UC S5 To-do-Item löschen
Auslöser: Messsystem
Vorbedingungen: /
Ergebnis: Das Datenübertragungsvolumen der Aktion wurde gemessen
Standardablauf: Während der Aktion wird das Datenübertragungsvolumen gemessen und der Wert dem Messsystem mitgeteilt.

UC M4 CPU-Auslastung messen

Beschreibung: Die CPU-Auslastung einer Aktion wird gemessen
Beteiligte Akteure: /
Verwendete Anwendungsfälle: /
Erweiternde Anwendungsfälle: /
Erweiterte Anwendungsfälle: UC C1 To-do-Liste erstellen, UC C2 To-do-Liste anzeigen, UC C3 To-do-Item erstellen, UC C4 To-do-Item als erledigt markieren, UC C5 To-do-Item bearbeiten, UC C6 To-do-Item löschen, UC S1 To-do-Liste speichern, UC S2 To-do-Liste laden, UC S3 To-do-Item speichern, UC S4 To-do-Item bearbeiten, UC S5 To-do-Item löschen
Auslöser: Messsystem
Vorbedingungen: /
Ergebnis: Die CPU-Auslastung der Aktion wurde gemessen
Standardablauf: Während der Aktion wird die CPU-Auslastung gemessen und der Wert dem Messsystem mitgeteilt.

A.3. Testdaten

A.3.1. ASP.NET Web API

In Fiddler, einem Programm zum Sniffen von HTTP Paketen, lassen sich die Requests speichern, welche zuvor im „Compose“-Tab erstellt wurden. Für die Tests während der Implementierung des ASP.NET Web API Dienstes wurden folgende Requests verwendet:

Lokal

```
============================================================

POST http://localhost:64569/User HTTP/1.1
User-Agent: Fiddler
Host: localhost:64569
Content-Length: 0

============================================================

GET http://localhost:64569/TodoList/1 HTTP/1.1
User-Agent: Fiddler
Host: localhost:64569

============================================================

POST http://localhost:64569/TodoList/1 HTTP/1.1
User-Agent: Fiddler
Host: localhost:64569
Content-Type: application/x-www-form-urlencoded

=testnote

============================================================

PUT http://localhost:64569/TodoList/1/item/1 HTTP/1.1
User-Agent: Fiddler
Host: localhost:64569
```

Content-Type: application/json

„Id“:1,„Note“:„updatednote“,„Completed“:false

==========

PUT http://localhost:64569/TodoList/1/item/1 HTTP/1.1
User-Agent: Fiddler
Host: localhost:64569
Content-Type: application/json

„Id“:1,„Note“:„updatednote“,„Completed“:true

==========

DELETE http://localhost:64569/TodoList/1/item/1 HTTP/1.1
User-Agent: Fiddler
Host: localhost:64569

==========

Remote

==========

POST http://ba13.cloudapp.net/TodoRestService/User HTTP/1.1
User-Agent: Fiddler
Host: ba13.cloudapp.net
Content-Length: 0

==========

GET http://ba13.cloudapp.net/TodoRestService/TodoList/1 HTTP/1.1
User-Agent: Fiddler
Host: ba13.cloudapp.net

==========

POST http://ba13.cloudapp.net/TodoRestService/TodoList/1 HTTP/1.1
User-Agent: Fiddler
Host: ba13.cloudapp.net
Content-Type: application/x-www-form-urlencoded

=testnote

==========

PUT http://ba13.cloudapp.net/TodoRestService/TodoList/1/item/1 HTTP/1.1
User-Agent: Fiddler
Host: ba13.cloudapp.net
Content-Type: application/json

„Id“:1,„Note“:„updatednote“,„Completed“:false

==========

PUT http://ba13.cloudapp.net/TodoRestService/TodoList/1/item/1 HTTP/1.1
User-Agent: Fiddler
Host: ba13.cloudapp.net
Content-Type: application/json

„Id“:1,„Note“:„updatednote“,„Completed“:true

==========

DELETE http://ba13.cloudapp.net/TodoRestService/TodoList/1/item/1
HTTP/1.1
User-Agent: Fiddler
Host: ba13.cloudapp.net

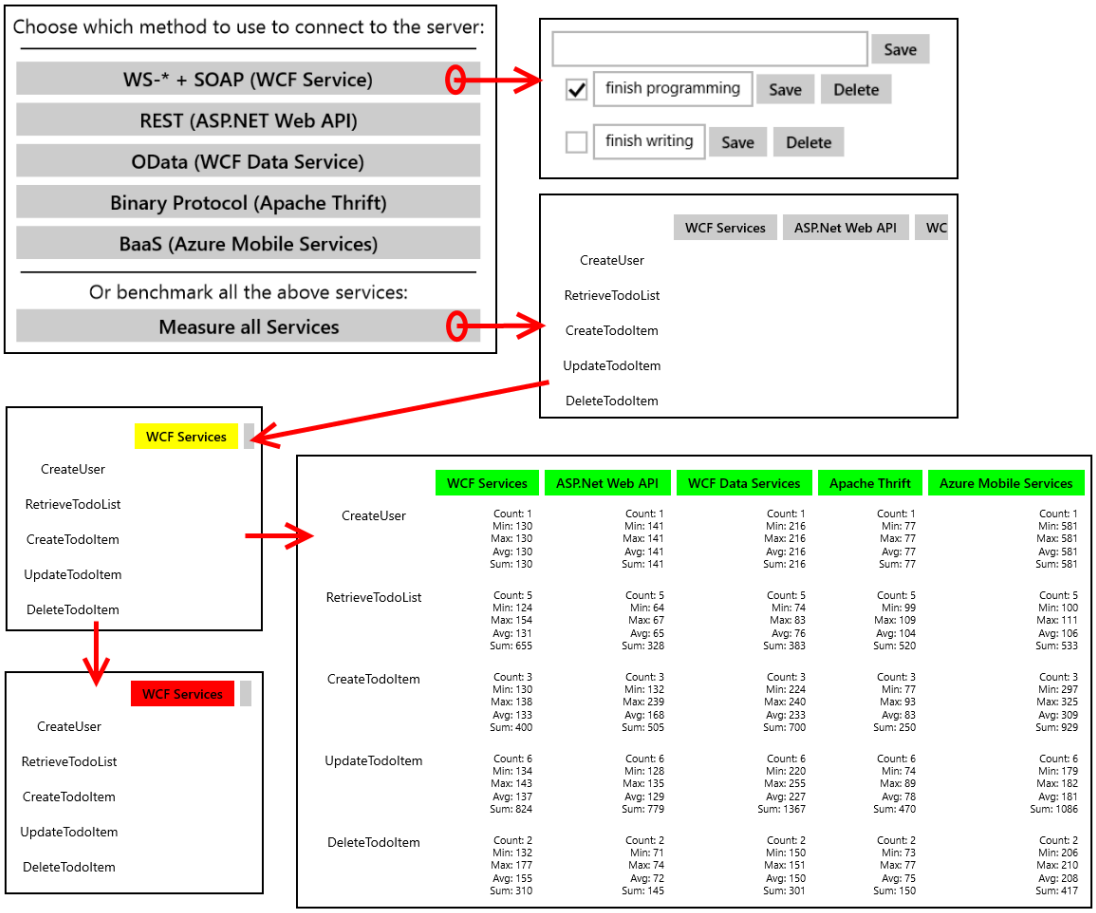

Abbildung A.1.: Betrieb des Clients

A.4. Betrieb

Clientseitig gibt es wie in vorigen Kapiteln beschrieben zwei Betriebsarten: Die App kann als To-do-Verwaltung genutzt werden, sowie zur Messung der unterschiedlichen Services. Die Auswahl findet auf dem Startbildschirm statt. Die Grafik in Abbildung A.1 veranschaulicht dies.

A.5. Listings

Der Code in Listing A.1 dient der Einstellung der Simulation einer mobilen Datenverbindung im HTTP-Proxy Fiddler.

```
// ...
// Cause Fiddler to delay HTTP traffic to simulate typical 56k modem
// conditions.
public static RulesOption("Simulate &Modem Speeds", "Per&formance")
var m_SimulateModem: boolean = false;
// ...
if (m_SimulateModem) {
    // Delay sends by 300ms per KB uploaded.
    oSession["request-trickle-delay"] = "300";
    // Delay receives by 150ms per KB downloaded.
    oSession["response-trickle-delay"] = "150";
}
// ...
```

Listing A.1: Fiddler Modemeinstellung

Listing A.2 zeigt ein Beispiel eines SOAP Requests über HTTP (Quelle: [Hor10, Kapitel 4]).

```
POST /perl/soaplite.cgi HTTP/1.0
Host: services.xmethods.net:80
Content-Type: text/xml; charset=utf-8
Content-Length: 546
SOAPAction: ""

<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <SOAP-ENV:Body>
    <ns1:BabelFish xmlns:ns1="urn:xmethodsBabelFish"
       SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <translationmode xsi:type="xsd:string">de_en</translationmode>
      <sourcedata xsi:type="xsd:string">Hallo Welt, Guten Tag</sourcedata>
    </ns1:BabelFish>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Listing A.2: Beispiel eines SOAP Requests über HTTP

Mit dem Code in Listing A.3 wurden die ungewöhnlichen Werte des Messsystems auf dem Server untersucht.

```
class Program
{
    static void Main(string[] args)
    {
        string dir = System.Environment.GetFolderPath(
                System.Environment.SpecialFolder.ApplicationData) +
                "\\TEST";
        if (!Directory.Exists(dir))
        {
            Directory.CreateDirectory(dir);
        }

        for (int i = 0; i < 10; i++)
        {
            DateTime start = DateTime.Now;
            File.WriteAllText(dir + @"\test.txt", "test");
            DateTime end = DateTime.Now;
            TimeSpan duration = end - start;
            double ms = duration.TotalMilliseconds;
            File.AppendAllText(dir+@"\log.txt", Convert.ToString(ms)+
            System.Environment.NewLine);
        }
    }
}
```

Listing A.3: Code zur Untersuchung von Werten des Messsystems

Der Code für die Iteration durch den XAML Visual Tree ist in Listing A.4 angegeben. Die AllChildren Methode ist aus [Nix13] übernommen.

```
private async void NoteEdit_Clicked(object sender, RoutedEventArgs e)
{
    var buttonContext = ((Button)sender).DataContext;
    string note = null;
    TodoItemModel ti = (TodoItemModel)buttonContext;
```

```
6       // this ti doesn't have the entered note assigned to an
7       // attribute yet.
8       // go through xaml tree to get note.
9       foreach (var listViewItem in TodoListView.Items)
10      {
11          var container =
12          TodoListView.ItemContainerGenerator.ContainerFromItem(
13                          listViewItem);
14          var children = Helpers.AllChildren(container);
15          TextBox tb = (TextBox)children[1];
16          var tbContext = tb.DataContext;
17          var ti2 = (TodoItemModel)tbContext;
18          if (ti2.Id == ti.Id)
19          {
20              note = tb.Text;
21          }
22      }
23      if (note != null)
24      {
25          ti.Note = note;
26          await _serviceClient.UpdateTodoItem(_userId, ti);
27      }
28  }
29
30  // In der statischen Helpers Klasse:
31  public static List<Control> AllChildren(DependencyObject parent)
32  {
33      var list = new List<Control> { };
34      for (int i = 0; i < VisualTreeHelper.GetChildrenCount(parent);
35          i++)
36      {
37          var child = VisualTreeHelper.GetChild(parent, i);
38          if (child is Control)
39              list.Add(child as Control);
40          list.AddRange(AllChildren(child));
41      }
42      return list;
43  }
```

Listing A.4: Durchgehen des XAML Visual Trees

Der Code in Listing A.5 ist der Code für die Messautomatisierung.

```
public static async Task<IDictionary<string, IEnumerable<TimeSpan?>>>
        Go(IServiceClient serviceClient)
{
    ServiceClientWithMeasurementFacade scwmf =
        new ServiceClientWithMeasurementFacade(serviceClient);

    int userId = await scwmf.CreateUser();
    await scwmf.RetrieveTodoList(userId);
    int todo1 =
        await scwmf.CreateTodoItem(userId, "procrastination");
    int todo2 = await scwmf.CreateTodoItem(userId, "distraction");
    int todo3 = await scwmf.CreateTodoItem(userId, "problems");
    await scwmf.RetrieveTodoList(userId);
    await scwmf.UpdateTodoItem(userId,
        new TodoAppBackend.Models.TodoItemModel()
        { Id = todo1, Completed = false, Note = "GTD" });
    await scwmf.UpdateTodoItem(userId,
        new TodoAppBackend.Models.TodoItemModel()
        { Id = todo2, Completed = false, Note = "concentration" });
    await scwmf.UpdateTodoItem(userId,
        new TodoAppBackend.Models.TodoItemModel()
        { Id = todo3, Completed = false, Note = "solutions" });
    await scwmf.RetrieveTodoList(userId);
    await scwmf.UpdateTodoItem(userId,
        new TodoAppBackend.Models.TodoItemModel()
        { Id = todo2, Completed = true, Note = "concentration" });
    await scwmf.UpdateTodoItem(userId,
        new TodoAppBackend.Models.TodoItemModel()
        { Id = todo3, Completed = true, Note = "solutions" });
    await scwmf.UpdateTodoItem(userId,
        new TodoAppBackend.Models.TodoItemModel()
        { Id = todo1, Completed = true, Note = "GTD" });
    await scwmf.RetrieveTodoList(userId);
    await scwmf.DeleteTodoItem(userId, todo2);
    await scwmf.DeleteTodoItem(userId, todo3);
    await scwmf.RetrieveTodoList(userId);

```

```
38      return new Dictionary<string, IEnumerable<TimeSpan?>>
39          {
40              { "user", scwmf.MeasurementCalculatedCreateUser },
41              { "read", scwmf.MeasurementCalculatedRetrieveTodoList },
42              { "create", scwmf.MeasurementCalculatedCreateTodoItem },
43              { "update", scwmf.MeasurementCalculatedUpdateTodoItem },
44              { "delete", scwmf.MeasurementCalculatedDeleteTodoItem }
45          };
46  }
```

Listing A.5: Messautomatisierung - Benchmark

A.6. Messwerte

Die genauen Messwerte sind wie folgt in Tabellen angegeben:

Tabelle A.1: Messung der Zeit
Tabelle A.2: Messung der Zeit mit Simulation einer mobilen Datenverbindung
Tabelle A.3: Messung der Zeit bei Benutzung einer echten mobilen Datenverbindung
Tabelle A.4: Messung der Zeit bei Benutzung einer echten mobilen Datenverbindung
Tabelle A.5: Messung der Datenpakete

	WCF S	Web API	WCF DS	Thrift	(Azure MS)
Create User	Count: 1 Time: 130	Count: 1 Time: 141	Count: 1 Time: 216	Count: 1 Time: 77	Count: 1 Time: 581
Retrieve TL	Count: 5 Min: 124 Max: 154 Avg: 131 Sum: 655	Count: 5 Min: 64 Max: 67 Avg: 65 Sum: 328	Count: 5 Min: 74 Max: 83 Avg: 76 Sum: 383	Count: 5 Min: 99 Max: 109 Avg: 104 Sum: 520	Count: 5 Min: 100 Max: 111 Avg: 106 Sum: 533
Create TI	Count: 3 Min: 130 Max: 138 Avg: 133 Sum: 400	Count: 3 Min: 132 Max: 239 Avg: 168 Sum: 505	Count: 3 Min: 224 Max: 240 Avg: 233 Sum: 700	Count: 3 Min: 77 Max: 93 Avg: 83 Sum: 250	Count: 3 Min: 297 Max: 325 Avg: 309 Sum: 929
Update TI	Count: 6 Min: 134 Max: 143 Avg: 137 Sum: 824	Count: 6 Min: 128 Max: 135 Avg: 129 Sum: 779	Count: 6 Min: 220 Max: 255 Avg: 227 Sum: 1367	Count: 6 Min: 74 Max: 89 Avg: 78 Sum: 470	Count: 6 Min: 179 Max: 182 Avg: 181 Sum: 1086
Delete TI	Count: 2 Min: 132 Max: 177 Avg: 155 Sum: 310	Count: 2 Min: 71 Max: 74 Avg: 72 Sum: 145	Count: 2 Min: 150 Max: 151 Avg: 150 Sum: 301	Count: 2 Min: 73 Max: 77 Avg: 75 Sum: 150	Count: 2 Min: 206 Max: 210 Avg: 208 Sum: 417
Summen	2319	1898	2967	1467	3546

Tabelle A.1.: Messergebnisse Zeit Client

	WCF S	Web API	WCF DS	Thrift	(Azure MS)
Create User	666	252	972	111	752
Retrieve TL	3463	1254	1851	611	705
Create TI	2258	2089	2967	232	1254
Update TI	4182	4289	5518	561	1494
Delete TI	1360	560	979	284	476
Summen	11929	8444	12287	1799	4681

Tabelle A.2.: Messergebnisse Zeit mobil

	WCF S	Web API	WCF DS	Thrift	(Azure MS)
Create User	Count: 1 Time: 1264	Count: 1 Time: 1243	Count: 1 Time: 2141	Count: 1 Time: 3320	Count: 1 Time: 6486
Retrieve TL	Count: 5 Min: 956 Max: 2941 Avg: 1754 Sum: 8773	Count: 5 Min: 581 Max: 1604 Avg: 936 Sum: 4684	Count: 5 Min: 761 Max: 2820 Avg: 1454 Sum: 7272	Count: 5 Min: 506 Max: 10834 Avg: 3345 Sum: 16725	Count: 5 Min: 602 Max: 780 Avg: 700 Sum: 3503
Create TI	Count: 3 Min: 942 Max: 1063 Avg: 989 Sum: 2967	Count: 3 Min: 1279 Max: 1837 Avg: 1486 Sum: 4459	Count: 3 Min: 1779 Max: 2342 Avg: 1980 Sum: 5941	Count: 3 Min: 625 Max: 873 Avg: 770 Sum: 2311	Count: 3 Min: 1934 Max: 3116 Avg: 2451 Sum: 7353
Update TI	Count: 6 Min: 1579 Max: 4679 Avg: 2363 Sum: 14180	Count: 6 Min: 848 Max: 2299 Avg: 1312 Sum: 7875	Count: 6 Min: 1940 Max: 5737 Avg: 3449 Sum: 20695	Count: 6 Min: 585 Max: 13681 Avg: 3064 Sum: 18388	Count: 6 Min: 721 Max: 1166 Avg: 860 Sum: 5161
Delete TI	Count: 2 Min: 1658 Max: 1859 Avg: 1759 Sum: 3518	Count: 2 Min: 580 Max: 3536 Avg: 2058 Sum: 4117	Count: 2 Min: 1863 Max: 2113 Avg: 1988 Sum: 3977	Count: 2 Min: 731 Max: 962 Avg: 847 Sum: 1694	Count: 2 Min: 1215 Max: 1482 Avg: 1349 Sum: 2698
Summen	30702	22378	40026	42438	25201

Tabelle A.3.: Messergebnisse Zeit Client Mobil Real via EDGE

	WCF S	Web API	WCF DS	Thrift	(Azure MS)
Create User	Count: 1 Time: 421	Count: 1 Time: 280	Count: 1 Time: 445	Count: 1 Time: 160	Count: 1 Time: 969
Retrieve TL	Count: 5 Min: 229 Max: 264 Avg: 241 Sum: 1208	Count: 5 Min: 140 Max: 153 Avg: 148 Sum: 741	Count: 5 Min: 130 Max: 165 Avg: 146 Sum: 731	Count: 5 Min: 117 Max: 174 Avg: 139 Sum: 696	Count: 5 Min: 132 Max: 158 Avg: 144 Sum: 723
Create TI	Count: 3 Min: 243 Max: 325 Avg: 274 Sum: 823	Count: 3 Min: 258 Max: 263 Avg: 260 Sum: 781	Count: 3 Min: 369 Max: 464 Avg: 416 Sum: 1249	Count: 3 Min: 127 Max: 143 Avg: 134 Sum: 402	Count: 3 Min: 421 Max: 452 Avg: 440 Sum: 1322
Update TI	Count: 6 Min: 229 Max: 701 Avg: 313 Sum: 1881	Count: 6 Min: 238 Max: 259 Avg: 247 Sum: 1486	Count: 6 Min: 360 Max: 670 Avg: 428 Sum: 2573	Count: 6 Min: 120 Max: 198 Avg: 149 Sum: 897	Count: 6 Min: 265 Max: 338 Avg: 299 Sum: 1795
Delete TI	Count: 2 Min: 227 Max: 270 Avg: 248 Sum: 497	Count: 2 Min: 134 Max: 137 Avg: 135 Sum: 271	Count: 2 Min: 253 Max: 257 Avg: 255 Sum: 510	Count: 2 Min: 117 Max: 124 Avg: 121 Sum: 242	Count: 2 Min: 286 Max: 297 Avg: 291 Sum: 583
Summen	4830	3559	5508	2397	5392

Tabelle A.4.: Messergebnisse Zeit Client Mobil Real via HSPA

	WCF S	Web API	WCF DS	Thrift	(Azure MS)
Create User	Out: 687 In: 674 Sum: 1361	Out: 376 In: 379 Sum: 755	Out: 1572 In: 1787 Sum: 3359	Out: 251 In: 1140 Sum: 1391	Out: 2290 In: 6531 Sum: 8821
Retrieve TL	Out: 576 In: 814 Sum: 1390	Out: 297 In: 448 Sum: 745	Out: 363 In: 1437 Sum: 1854	Out: 1975 In: 1662 Sum: 3637	Out: 1520 In: 5837 Sum: 7357
Create TI	Out: 606 In: 569 Sum: 1175	Out: 331 In: 450 Sum: 781	Out: 1377 In: 2851 Sum: 4228	Out: 319 In: 466 Sum: 785	Out: 2055 In: 1612 Sum: 3667
Update TI	Out: 778 In: 572 Sum: 1350	Out: 348 In: 448 Sum: 796	Out: 1484 In: 1589 Sum: 3073	Out: 334 In: 463 Sum: 797	Out: 942 In: 752 Sum: 1694
Delete TI	Out: 611 In: 572 Sum: 1183	Out: 314 In: 369 Sum: 683	Out: 639 In: 1509 Sum: 2148	Out: 419 In: 571 Sum: 990	Out: 1443 In: 947 Sum: 2390
Summen	5284	3760	14659	7600	23929

Tabelle A.5.: Messergebnisse Datenpaketgrößen

Literaturverzeichnis

[Ani12] Igor Anishchenko. Thrift vs protocol buffers vs avro - biased comparison. https://github.com/eishay/jvm-serializers/wiki, 2012. Zugriff: 26.11.2013.

[Ani13] Igor Anishchenko. Protocol buffers vs thrift vs avro. http://de.slideshare.net/IgorAnishchenko/pb-vs-thrift-vs-avro, 2013. Zugriff: 26.11.2013.

[Bar94] Daniel Barbara. Sleepers and workaholics: caching strategies in mobile environments. In *SIGMOD '94 Proceedings of the 1994 ACM SIGMOD international conference on management of data*, 1994.

[Bay02] Thomas Bayer. Rest web services. http://www.oio.de/public/xml/rest-webservices.htm, 2002. Zugriff: 26.11.2013.

[Cis13] Cisco. Cisco visual networking index: Global mobile data traffic forecast update, 2012–2017). http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-520862.html, 2013. Zugriff: 04.12.2013.

[Coo13] Cookiebits. session management methods reviewed. http://cookiebits.com/htm/tech.htm, 2013. Zugriff: 02.12.2013.

[CP08] Frank Leymann Cesare Pautasso, Olaf Zimmermann. Restful web services vs. „big" web services: Making the right architectural decision. http://www2008.org/papers/pdf/p805-pautassoA.pdf, 2008. Zugriff: 26.11.2013.

[DB+13] Francis McCabe David Booth, Hugo Haas et al. W3c web services architecture. http://www.w3.org/TR/ws-arch/, 2013. Zugriff: 30.11.2013.

[DK13] Michael Raith Daniel Kuhn. *Performante Webanwendungen: Client- und serverseitige Techniken zur Performance-Optimierung*. dpunkt.verlag, 2013.

[ea99] J. Franks et al. http authentication: Basic and digest access authentication. http://tools.ietf.org/html/rfc2617, 1999. Zugriff: 02.12.2013.

[ea05] Roland Bless et al. *Sichere Netzwerkkommunikation: Grundlagen, Protokolle und Architekturen*. Springer, 2005.

[ea13] Glenn Block et al. *Designing Evolvable Web APIs with ASP.NET*. O'Reilly, 2013. http://chimera.labs.oreilly.com/books/1234000001708/index.html.

[Ell13] Jonathan Ellis. Api. http://wiki.apache.org/cassandra/API, 2013. Zugriff: 26.11.2013.

[Eng11] Dave Engberg. So api together: Evernote and thrift. http://blog.evernote.com/tech/2011/05/26/evernote-and-thrift/, 2011. Zugriff: 26.11.2013.

[Fie00] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000. http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.

[Fin13] Michael Findling. Rest and soap: When should i use each (or both)? http://edn.embarcadero.com/article/40558, 2013. Zugriff: 26.11.2013.

[Fla09a] Jon Flanders. More on rest. http://msdn.microsoft.com/en-us/magazine/dd942839.aspx, 2009. Zugriff: 26.11.2013.

[Fla09b] Jon Flanders. More on rest. http://msdn.microsoft.com/en-us/magazine/dd942839.aspx, 2009. Zugriff: 26.11.2013.

[Fou12a] Apache Software Foundation. About apache thrift. http://thrift.apache.org/about/, 2012. Zugriff: 26.11.2013.

[Fou12b] Apache Software Foundation. Apache thrift features. http://thrift.apache.org/docs/features/, 2012. Zugriff: 26.11.2013.

[Fow12] Martin Fowler. Aggregateorienteddatabase. http://martinfowler.com/bliki/AggregateOrientedDatabase.html, 2012. Zugriff: 26.11.2013.

[Gar13] Gartner. Global smartphone sales q1 2009-q3 2013, by operating system. http://www.statista.com/statistics/266219/

global-smartphone-sales-since-1st-quarter-2009-by-operating-system/, 2013. Zugriff: 04.12.2013.

[Geh13] Kunz Gehrig. Thriftinterface. http://wiki.apache.org/cassandra/ThriftInterface, 2013. Zugriff: 26.11.2013.

[Goo12a] Google. Google data apis. https://developers.google.com/gdata/docs/developers-guide, 2012. Zugriff: 26.11.2013.

[Goo12b] Google. Google data apis. https://developers.google.com/gdata/docs/directory, 2012. Zugriff: 26.11.2013.

[Goo12c] Google. Protocol buffers. https://developers.google.com/protocol-buffers/, 2012. Zugriff: 26.11.2013.

[Goo12d] Google. Protocol buffers. https://developers.google.com/protocol-buffers/docs/overview, 2012. Zugriff: 26.11.2013.

[Goo13] Google. Protocol buffers. https://code.google.com/p/protobuf/, 2013. Zugriff: 26.11.2013.

[Gup11] Diwaker Gupta. Thrift vs. protocol buffers. http://old.floatingsun.net/articles/thrift-vs-protocol-buffers/, 2011. Zugriff: 26.11.2013.

[HC13] Dejan Bosanac Hiram Chirino. The activemq protocol buffers java implementation. https://github.com/apache/activemq-protobuf, 2013. Zugriff: 26.11.2013.

[Heu03] Volker Heun. *Grundlegende Algorithmen: Einführung in den Entwurf und die Analyse effizienter Algorithmen.* Vieweg, 2. edition, 2003.

[HH13] Allen Brown Hugo Haas. W3c web services glossary. http://www.w3.org/TR/ws-gloss/, 2013. Zugriff: 30.11.2013.

[Hor10] Torsten Horn. Web services mit soap (simple object access protocol). http://www.torsten-horn.de/techdocs/soap.htm, 2010. Zugriff: 04.12.2013.

[IDC13] IDC. Apple cedes market share in smartphone operating system market as android surges and windows phone gains, according to idc. http://www.idc.com/getdoc.jsp?containerId=prUS24257413, 2013. Zugriff: 26.11.2013.

[IF11] A. Melnikov I. Fette. The websocket protocol. http://tools.ietf.org/html/rfc6455, 2011. Zugriff: 02.12.2013.

[Jam12] Alex D James. Odata support in asp.net web api. http://blogs.msdn.com/b/alexj/archive/2012/08/15/odata-support-in-asp-net-web-api.aspx, 2012. Zugriff: 26.11.2013.

[Jen08] Roger Jennings. Updated sql server data services (ssds) test harness: Northwind rest and soap uploads. http://oakleafblog.blogspot.de/2008/07/updated-sql-server-data-services-ssds.html, 2008. Zugriff: 26.11.2013.

[JG07] B. de hOra J. Gregorio. The atom publishing protocol. http://tools.ietf.org/html/rfc5023, 2007. Zugriff: 26.11.2013.

[jso13] json.org. Introducing json. http://www.json.org/, 2013. Zugriff: 02.12.2013.

[JW10] Ian Robinson Jim Webber, Savas Parastatidis. *REST in Practice: Hypermedia and Systems Architecture*. O'Reilly Media, 2010.

[Kan12] Axel Kannenberg. Das internet wird weltweit schneller). http://www.heise.de/netze/meldung/Das-Internet-wird-weltweit-schneller-1664542.html, 2012. Zugriff: 04.12.2013.

[Kel13] M. Kelly. Json hypertext application language. http://tools.ietf.org/html/draft-kelly-json-hal-06, 2013. Zugriff: 30.11.2013.

[Kle08] J. Klensin. Simple mail transfer protocol. http://tools.ietf.org/html/rfc5321, 2008. Zugriff: 02.12.2013.

[Lan13] LangPop. Programming language popularity. http://langpop.com/, 2013. Zugriff: 26.11.2013.

[LD10] J. Snell L. Dusseault. Patch method for http. http://tools.ietf.org/html/rfc5789, 2010. Zugriff: 26.11.2013.

[LR07] Sam Ruby Leonard Richardson. *RESTful Web Services*. O'Reilly Media, 2007.

[Man08] Lawrence Mandel. Describe rest web services with wsdl 2.0. http://www.ibm.com/developerworks/webservices/

library/ws-restwsdl/, 2008. Zugriff: 07.2013 . Erreichbar unter http://web.archive.org/web/20130606190345/http://www.ibm.com/developerworks/webservices/library/ws-restwsdl/(Zugriff:26.11.2013).

[Mar83] James Martin. *Managing the Data Base Environment.* Prentice Hall, 1983.

[Mel10] Ingo Melzer. *Service-orientierte Architekturen mit Web Services.* Spektrum Akademischer Verlag, 4. edition, 2010.

[Mic13a] Microsoft. How to: Add, modify, and delete entities (wcf data services). http://msdn.microsoft.com/en-us/library/dd756368.aspx, 2013. Zugriff: 26.11.2013.

[Mic13b] Microsoft. Managing application pools in iis 7. http://technet.microsoft.com/en-us/library/cc753449(v=ws.10).aspx, 2013. Zugriff: 26.11.2013.

[Mic13c] Microsoft. Open data protocol (odata). http://download.microsoft.com/download/9/5/E/95EF66AF-9026-4BB0-A41D-A4F81802D92C/[MS-ODATA].pdf, 2013. Zugriff: 26.11.2013.

[Mic13d] Microsoft. Query records operation. http://msdn.microsoft.com/en-us/library/windowsazure/jj677199.aspx, 2013. Zugriff: 26.11.2013.

[Mic13e] Microsoft. Wcf data services client library. http://msdn.microsoft.com/en-us/library/cc668772(v=vs.110).aspx, 2013. Zugriff: 26.11.2013.

[MN05] R. Sayre M. Nottingham. The atom syndication format. http://tools.ietf.org/html/rfc4287, 2005. Zugriff: 26.11.2013.

[Nix13] Jerry Nixon. How to access a named control inside a xaml datatemplate (using csharp). http://blog.jerrynixon.com/2012/09/how-to-access-named-control-inside-xaml.html, 2013. Zugriff: 26.11.2013.

[Not07] M. Nottingham. Fiql: The feed item query language. http://tools.ietf.org/html/draft-nottingham-atompub-fiql-00, 2007. Zugriff: 26.11.2013.

[Not12] Mark Nottingham. Why patch is good for your http api. http://www.mnot.net/blog/2012/09/05/patch, 2012. Zugriff: 26.11.2013.

[OAS09] OASIS. Soap-over-udp version 1.1. http://docs.oasis-open.org/ws-dd/soapoverudp/1.1/os/wsdd-soapoverudp-1.1-spec-os.html, 2009. Zugriff: 26.11.2013.

[ODa13a] OData. Atom format. http://www.odata.org/documentation/odata-v3-documentation/atom-format/, 2013. Zugriff: 26.11.2013.

[ODa13b] OData. Batch processin. http://www.odata.org/documentation/odata-v3-documentation/batch-processing/, 2013. Zugriff: 03.12.2013.

[ODa13c] OData. Batch processing. http://www.odata.org/libraries/, 2013. Zugriff: 04.12.2013.

[ODa13d] OData. Common schema definition language (csdl). http://www.odata.org/documentation/odata-v3-documentation/common-schema-definition-language-csdl/, 2013. Zugriff: 26.11.2013.

[ODa13e] OData. Ecosystem. http://www.odata.org/ecosystem/, 2013. Zugriff: 26.11.2013.

[ODa13f] OData. Odata operations. http://www.odata.org/documentation/odata-v2-documentation/operations/, 2013. Zugriff: 02.12.2013.

[ODa13g] OData. Uri conventions. http://www.odata.org/documentation/odata-v2-documentation/uri-conventions/#4_Query_String_Options, 2013. Zugriff: 26.11.2013.

[OM13] OpenStreetMap-Mitwirkende. Pbf format. http://wiki.openstreetmap.org/wiki/PBF_Format, 2013. Zugriff: 26.11.2013.

[oSC81] Information Sciences Institute University of Southern California. Transmission control protocol. http://tools.ietf.org/html/rfc793, 1981. Zugriff: 02.12.2013.

[Pau08a] Cesare Pautasso. Rest vs. soap: Making the right architectural decision. http://www.jopera.org/files/soa-amsterdam-restws-pautasso-talk.pdf, 2008. Zugriff: 26.11.2013.

[Pau08b] Cesare Pautasso. Rest vs. soap: Making the right architectural decision. http://de.slideshare.net/cesare.pautasso/rest-vs-soap-making-the-right-architectural-decision-1st-international-soa-symposium-amsterdam-october-2008-presentation, 2008. Zugriff: 26.11.2013.

[PB13] M. Nottingham P. Bryan. Javascript object notation (json) patch. http://tools.ietf.org/html/rfc6902, 2013. Zugriff: 26.11.2013.

[Pre02] Paul Prescod. Roots of the rest/soap debate. http://conferences.idealliance.org/extreme/html/2002/Prescod01/EML2002Prescod01.html, 2002. Zugriff: 26.11.2013.

[Pul13] Carmelo Pulvirenti. How many web applications per application pool. http://blogs.msdn.com/b/carmelop/archive/2013/03/22/how-many-web-applications-per-application-pool.aspx, 2013. Zugriff: 26.11.2013.

[Sch11] Jens Schumann. Webservices mit java ee 6: Jax-ws und restful services. http://www.heise.de/developer/artikel/Webservices-mit-Java-EE-6-JAX-WS-und-RESTful-Services-1247464.html, 2011. Zugriff: 26.11.2013.

[Shi10] Karl Shifflett. Wpf line of business – introduction. http://karlshifflett.wordpress.com/archive/mvvm/wpf-line-of-business-introduction/, 2010. Zugriff: 04.12.2013.

[Sof13] TIOBE Software. Tiobe programming community index for november 2013. http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html, 2013. Zugriff: 26.11.2013.

[Spi12] Alexander Spier. Darf's ein bisschen schneller sein? - wie sich lte im mobilen alltag schlägt). http://www.heise.de/ct/artikel/Darf-s-ein-bisschen-schneller-sein-1722006.html, 2012. Zugriff: 04.12.2013.

[Ste00] Carl Steinweg. *Projektkompass Softwareentwicklung. Geschäftsorientierte Entwicklung von IT-Systemen.* Vieweg Verlagsgesellschaft, 2000.

[Suk13] Ilya Sukhar. The future of parse. http://blog.parse.com/2013/04/25/the-future-of-parse/, 2013. Zugriff: 01.12.2013.

[Tan12] Andrew S. Tanenbaum. *Computernetzwerke*. Pearson Studium, 5. edition, 2012.

[Tea09] Microsoft Patterns & Practices Team. *Microsoft Application Architecture Guide, 2nd Edition*. Microsoft Press, 2009.

[Tec13] Basho Technologies. Pbc api. http://docs.basho.com/riak/latest/dev/references/protocol-buffers/, 2013. Zugriff: 26.11.2013.

[Tew08] Elena Tews. Transaktionskonzepte für web services, 2008.

[Tu13] Janet I. Tu. Microsoft pitches new mantra: rapid-release product cycle. http://seattletimes.com/html/businesstechnology/2021276971_microsoftbuild27xml.html, 2013. Zugriff: 26.11.2013.

[Var08] Kenton Varda. Protocol buffers: Google's data interchange format. http://google-opensource.blogspot.de/2008/07/protocol-buffers-googles-data.html, 2008. Zugriff: 26.11.2013.

[W3C99] W3C. Hypertext transfer protocol – http/1.1. http://tools.ietf.org/html/rfc2616, 1999. Zugriff: 26.11.2013.

[Was13] Mike Wasson. Attribute routing in web api 2. http://www.asp.net/web-api/overview/web-api-routing-and-actions/attribute-routing-in-web-api-2, 2013. Zugriff: 26.11.2013.

[Wil08] Erik Wilde. Querying feeds. http://dret.typepad.com/dretblog/2008/03/querying-feeds.html, 2008. Zugriff: 26.11.2013.

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 6. Dezember 2013 Philipp Gillé