Hochschule für Angewandte Wissenschaften Hamburg

Hamburg University of Applied Sciences

Masterarbeit

Eike Christian Falkenberg

Entwicklung einer Messaging Middleware für Code on Demand Anwendungen auf mobilen Plattformen

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Eike Christian Falkenberg

Entwicklung einer Messaging Middleware für Code on Demand Anwendungen auf mobilen Plattformen

Masterarbeit eingereicht im Rahmen der Masterprüfung
im Studiengang Master Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

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

Abgegeben am 15. September 2008

Eike Christian Falkenberg

Thema der Masterarbeit

Entwicklung einer Messaging Middleware für Code on Demand Anwendungen auf mobilen Plattformen

Stichworte

Mobile Computing, Code on Demand, Middleware

Kurzzusammenfassung

Im Rahmen dieser Arbeit wurde eine Middleware entwickelt, um reichhaltige Code on Demand Anwendungen auf mobilen Plattformen als Ersatz für lokal installierte Anwendungen zu ermöglichen.

Die entwickelte Middleware ermöglicht es, mobile Code on Demand Anwendungen in einem Browser auszuführen, selbst wenn keine Verbindung zum Internet verfügbar ist. Durch die angebotene Nachrichtenreplikation können installierte Anwendungen Nachrichten an die Middleware senden, die bei Wiederverfügbarkeit einer Internetverbindung an einen entfernten Webserver übertragen werden. Zusätzlich stellt die Middleware lokale Ressourcen über eine Schnittstelle zur Verfügung und ermöglicht es installierten Code on Demand Anwendungen so, auf diese zuzugreifen.

Eike Christian Falkenberg

Title of the paper

Development of a messaging middleware for code on demand applications on mobile platforms

Keywords

Mobile Computing, Code on Demand, Middleware

Abstract

A middleware to make the replacement of locally installed applications with enriched code on demand applications possible, has been developed within this work.

With the developed middleware it becomes possible to run code on demand applications within a browser, even if there is no network connection available. Because of the message replication, installed applications could send messages to the middleware, which will be later replicated to a web server when a network connection is available. Additionally, the middleware offers installed code on demand applications the opportunity to access local resources through it's interface.

Für meinen Vater

Danksagungen

Diese Master Arbeit stellt den Abschluss meines Informatik Studiums an der Hochschule für Angewandte Wissenschaften Hamburg dar. Es war eine spannende Zeit, in der ich vieles lernen durfte und viele interessante und nette Menschen kennengelernt habe.

Besonders danken möchte ich meinem betreuenden Prüfer Prof. Dr.-Ing. Martin Hübner. Seine Hilfe, seine Geduld und sein guter Rat haben bedeutend zu dieser Arbeit beigetragen. Auch Herrn Prof. Dr. Olaf Zukunft möchte ich an dieser Stelle danken, der mir in vielen Situationen des Studiums eine Hilfe war und auch Zweitprüfer dieser Arbeit ist.
Zu danken habe ich auch meinen Kommilitonen Fatih Keles, Jan Napitupulu und Thomas Schmidt für all das, was wir dank unseres Projektes UbiZoo erleben und lernen durften und dass ich sie heute zu meinen Freunden zählen darf.

Der größte Dank gilt meiner Mutter, die nun hoffentlich so stolz auf mich ist, wie ich es auf sie bin.

Ein ganz besonderer Mensch zum Schluss: meine Frau, ohne die ich das alles nie geschafft hätte.

Inhaltsverzeichnis

1 Einleitung **1**
1.1 Motivation . . . 1
1.2 Zielsetzung . . . 5
1.3 Bestehende Arbeiten und wissenschaftliche Einordnung . . . 7
1.4 Inhaltlicher Aufbau der Arbeit . . . 9

2 Grundlagen **10**
2.1 Mobile Plattformen . . . 10
2.1.1 Hardware Plattformen . . . 10
2.1.2 Software Plattformen . . . 11
2.2 Windows Mobile . . . 15
2.2.1 Konzept . . . 15
2.2.2 Architektur . . . 15
2.2.3 Programmierung . . . 17
2.2.4 Microsoft .NET Compact Framework . . . 18
2.3 Webanwendungen . . . 20
2.3.1 Klassische Webanwendungen . . . 20
2.3.2 Dynamische Webanwendungen . . . 20
2.3.3 JavaScript . . . 21
2.3.4 Rich Internet Applications . . . 24
2.3.5 Ajax . . . 24
2.3.6 XML-RPC . . . 27
2.3.7 Mobile Webanwendungen . . . 29
2.4 Gesicherte Webanwendungen . . . 29
2.4.1 Basic Access Authentication . . . 29
2.4.2 Digest Access Authentication . . . 30
2.5 Web Services . . . 34
2.6 Representational State Transfer . . . 35
2.7 JavaScript Object Notation . . . 37
2.8 Existierende Lösungsansätze . . . 39
2.8.1 (Google) Gears . . . 40
2.8.2 Adobe Integrated Runtime (AIR) . . . 40
2.8.3 Mozilla Prism . . . 40
2.8.4 Mojax . . . 41
2.8.5 Yahoo Go! . . . 41

2.8.6 HTML 5 41

3 Analyse **42**
3.1 Beispielanwendung 42
3.2 Meta Architektur 45
3.3 Anwendungsfälle 47
3.3.1 Use Case 1: Installation der Middleware 47
3.3.2 Use Case 2: Starten der Middleware 49
3.3.3 Use Case 3: Installation einer Anwendung 50
3.3.4 Use Case 4: Ausführen einer Anwendung 51
3.3.5 Use Case 5: Konfiguration der Middleware 52
3.3.6 Use Case 6: Setzen des Middleware Kontext Status 53
3.3.7 Use Case 7: Starten der Synchronisation 54
3.3.8 Use Case 8: Stoppen der Middleware 55
3.4 Funktionale Anforderungen 56
3.4.1 Kontextwechsel 57
3.4.2 Installation und Konfiguration von Anwendungen 57
3.4.3 HTTP-Proxy 58
3.4.4 XML-HTTP Messaging Proxy 60
3.4.5 Applikations Cache 61
3.4.6 Credential Store 63
3.4.7 Zugriff auf lokale Ressourcen 65
3.4.8 Persistierungsmechanismen 67
3.4.9 Daten Replikation 68
3.5 Nichtfunktionale Anforderungen 70
3.5.1 Performance 71
3.5.2 Energieeffizienz 73
3.5.3 Skalierbarkeit 75
3.5.4 Sicherheit 76
3.5.5 Portierbarkeit 78
3.5.6 Benutzbarkeit 78
3.5.7 Datenintegrität 78

4 Entwurf **81**
4.1 Module der Middleware 81
4.1.1 HTTP Interface 82
4.1.2 Request Dispatcher 83

4.1.3 HTTP Proxy . . . 83
4.1.4 Applikations Cache . . . 84
4.1.5 XML-HTTP Messaging Proxy . . . 86
4.1.6 Resource Dispatcher . . . 86
4.1.7 Credential Store . . . 86
4.1.8 Persistierungs Modul . . . 87
4.1.9 Synchronisierungs Modul . . . 88
4.1.10 Konfigurations Modul . . . 89
4.2 Sichten . . . 93
4.2.1 Module . . . 94
4.2.2 Ablauf einer Anfrage (Aktivität) . . . 95
4.2.3 Ablauf einer Anfrage (Sequenz) . . . 96
4.2.4 Daten Sicht . . . 97
4.3 Application Programming Interface . . . 99
4.3.1 Application API . . . 101
4.3.2 Resource API . . . 101
4.3.3 Messaging API . . . 105

5 Realisierung **107**
5.1 Technologien . . . 107
5.2 Verwendung des Prototyps . . . 108
5.2.1 Starten der Middleware . . . 108
5.2.2 Startseite . . . 108
5.3 Beispielanwendungen . . . 109
5.3.1 UbiFlickr . . . 109
5.3.2 UbiBlog . . . 111
5.4 Integration der Middleware . . . 111

6 Fazit **113**
6.1 Zusammenfassung . . . 113
6.2 Ergebnisse . . . 114
6.2.1 Offline Fähigkeit und Zugriff auf relevante Ressourcen . . . 115
6.2.2 Plattformunabhängigkeit . . . 116
6.2.3 Transparente Einbettung in vorhandene Strukturen . . . 116
6.2.4 Prototyp . . . 117
6.3 Ausblick . . . 117

Glossar **119**

Literaturverzeichnis **121**

Abbildungsverzeichnis

1 Tag Cloud zu O'Reillys Web 2.0 Definition von Ajit Jaokar 2
2 Hype Cycle for Emerging Technologies 2007 (Quelle: Gartner Group) 6
3 HP iPAQ hw6915 - PDA oder Smartphone? 10
4 Windows Mobile 6.1 Startbildschirm 11
5 Android Startbildschirm 12
6 Android System Architektur (Quelle: Android SDK) 13
7 Windows Mobile Kernel Übersicht (Quelle: (winMobKernel, 2008)) 17
8 Microsoft Visual Studio 2008 mit GUI Builder 18
9 Arbeitsweise der Common Language Runtime von .NET 19
10 Klassischer Server-Roundtrip 22
11 Einsatz von JavaScript zur Vermeidung von Server Roundtrips 23
12 Tabellenkalkulation als Rich Internet Application: Google Spreadsheet 24
13 Asynchrone Aufrufe mit Ajax 26
14 XML-RPC (Quelle: http://www.xmlrpc.com/) 27
15 Übersicht der Infrastruktur für Web Services 34
16 JSON - Object (Quelle: json.org 37
17 JSON - Object (Quelle: json.org 38
18 JSON - Value (Quelle: json.org 38
19 Wordpress Oberfläche zum verfassen eines Blog Eintrags 43
20 Blog Eintrag mit Middleware Beispielanwendung 44
21 Umfeld der Middleware 46
22 Integration der Middleware in das mobile System 46
23 Use Case 1: Installation der Middleware 47
24 Use Case 2: Starten der Middleware 49
25 Use Case 3: Installation einer Anwendung 50
26 Use Case 4: Ausführen einer Anwendung 51
27 Use Case 5: Konfiguration der Middleware 52
28 Use Case 6: Setzen des Middleware Kontext Status 53
29 Use Case 7: Starten der Synchronisation 54
30 Use Case 8: Stoppen der Middleware 55
31 Kommunikation mit HTTP-Proxy 59
32 Kommunikation mit XML-HTTP Messaging Proxy 61
33 Zugriff auf lokale Ressourcen mit einer Middleware Lösung 67
34 Zugriffsmatrix der Middleware 76

35 Keine Asynchronen Nachrichten an externe Server 77
36 Vielseitiges Umfeld der Middleware 79
37 Sicht auf die Module der Middleware 81
38 Einbettung des Sitzungsschlüssels durch den HTTP Proxy 84
39 Aktivitätsdiagramm für eine Anfrage an die Middleware 95
40 Sequenzdiagramm der Middleware im offline Kontext 97
41 Entity-Relationship-Diagramm Applikation 99
42 Prototyp - Staren der Middleware 109
43 Prototyp - Beispielanwendung UbiFlickr 110
44 Prototyp - Beispielanwendung UbiBlog 111

Listings

1 Eine XML-RPC Anfrage . . . 28
2 Basic Access Authentication . . . 30
3 WWW-Authenticate Response Header Definition . . . 30
4 WWW-Authenticate Response Header . . . 31
5 Authorization Request Header Definition . . . 32
6 response-digest und entity-digest Definition . . . 32
7 Konkrete Anfrage eines Clients mit Authorization Header . . . 33
8 SOAP Nachricht an einen Web Service . . . 36
9 Aufruf eines REST Services . . . 37
10 Lesbare Darstellung eines JSON Objekts . . . 39
11 Blog Eintrag erstellen mittels XML-RPC Aufruf über Blogger API . . . 45
12 code und style Elemente des Installations Deskriptors . . . 62
13 XHTML Dokument aus den transformierten Daten . . . 63
14 Unnötige Objekterzeugung durch Stringkonkatenation . . . 72
15 Ajax Template des Applikation Cache . . . 85
16 Flash Template des Applikations Cache . . . 85
17 Installations Deskriptor einer Anwendung . . . 91
18 Basis Adresse des API . . . 100
19 Sitzungsschlüssel als API Parameter . . . 100
20 Laden einer Anwendung von der Middleware . . . 101
21 Ein Kontakt als JSON Objekt . . . 102
22 Ein Termin als JSON Objekt . . . 102
23 Ein in JSON transformiertes GPS Exchange Format Dokument . . . 105
24 Ein JSON Nachrichten Objekt . . . 105

1 Einleitung

1.1 Motivation

Als Sir Timothy John Berners-Lee 1989 das WorldWideWeb erfand(LeeW3, 1989), basierte es zunächst auf statischen HTML Dokumenten. Das Paradigma war simpel und unidirektional: HTML Entwickler schreiben Dokumente, legen diese auf Web Server, welche wiederum die HTML Dokumente über das HTTP an Clients ausliefern. Das WWW wurde in seiner Anfangszeit primär im wissenschaftlichen Umfeld genutzt und stand nur einem kleinen Kreis zur Verfügung.
Seither hat sich das Internet wesentlich weiterentwickelt. Die Anzahl der Internetnutzer ist stark gestiegen; zum Zeitpunkt der Erstellung dieser Arbeit nutzt fast jeder zweite deutsche Bundebürger das Internet, mit steigender Tendenz (expWeb, 2007).
Außerdem ist das Internet dynamischer, persönlicher und interaktiver geworden. 2005 machte Tim O'Reilly mit seinem Text *What Is Web 2.0* (ReillyWeb2, 2005) das Web 2.0 bekannt. Web 2.0 ist keine neue Technologie, sondern eine Sammlung von bereits bekannten Technologien, die in einem Paradigma zusammengefasst werden. Zu Beginn war Web 2.0 ein sehr unpräziser Begriff, O'Reilly schrieb dazu einmal:

> »I think Web 2.0 is of course a piece of jargon, nobody even knows what it means«

O'Reillys Definition des Web 2.0 hat Ajit Jaokar grafisch sehr ansehnlich umgesetzt und zeigt den technologischen Kontext, in dem sich das Web 2.0 befindet. 1.

Aus O'Reillys Sicht gibt es zwei elementare Veränderungen im Web 2.0. Dies ist zum einen die Sichtweise auf das Internet. In der neuen Sichtweise wird das Internet als Anwendungsplattform verstanden, auf dem Dienste angeboten werden. Die zweite wesentliche Veränderung ist das Nutzerverhalten. Von Tagebüchern im Internet, über Foto Plattformen bis hin zu Freundschaftsnetzwerken ist fast jeder Internetnutzer vom passiven Verbraucher zum aktiven Erzeuger von Inhalten des Internets geworden. Man muss inzwischen kein HTML Entwickler mehr sein, um eigenen Inhalt im Internet veröffentlichen zu können.

Doch nicht nur das Verhalten der Nutzer im Internet hat sich in den letzten Jahren weiterentwickelt, auch die Endgeräte haben sich verändert. Bis vor ein paar Jahren wurde das Internet fast ausschließlich mit Browsern auf Desktop PCs genutzt. Auch heute ist dies noch der primäre Anteil, jedoch drängen mehr und mehr mobile Endgeräte in den Vordergrund. Eine aktuelle Studie von Forrester Research[1] zeigt die rasante Entwicklung und prognosti-

[1] http://www.forrester.com/rb/research; Zugriffsdatum: 2008-05-01

Abbildung 1: Tag Cloud zu O'Reillys Web 2.0 Definition von Ajit Jaokar

ziert drei Mal so viele mobile Internetnutzer in den nächsten 5 Jahren (ForresterMobWeb, 2008).

Sogenannte Personal Digital Assistants (PDA) haben sich vom elektronischen Notizblock zum mobilen Mini-Büro entwickelt. Ein aktuelles Gerät beherrscht meist mehrere Funktechnologien und kann so über diverse Wege auf das Internet zugreifen. Weiterhin bieten diese Geräte die Möglichkeit, das Global Positioning System (GPS) für die Positionsbestimmung zu nutzen um so Routenplanungen zu ermöglichen.

PDAs sind jedoch keine abgeschotteten Systeme, die ausschließlich die Nutzung der mitgelieferten Anwendungen erlauben. PDAs bieten die Möglichkeit, ähnliche wie Personal Computer, eigene Anwendungen zu entwickeln und auf dem Gerät laufen zu lassen. Das Angebot, eigene Anwendungen für PDAs zu entwickeln, ist besonders im Unternehmenseinsatz relevant. So erhalten Mitarbeiter eines Unternehmens mobilen Zugriff auf relevante Applikationen. Zu jedem Systemtyp bieten die Anbieter der PDAs passende Entwicklungsumgebungen, Compiler und diverse Bibliotheken, um die Entwicklung von eigenen Anwendungen so komfortabel wie möglich zu gestalten.

Zu dem Zeitpunkt der Erstellung dieser Arbeit konnten sich die Hersteller von mobilen Plattformen nur auf wenige gemeinsame Standards einigen. Diverse Programmiersprachen,

Laufzeitumgebungen und Betriebssysteme machen die Entwicklung von mobilen Anwendungen zu einem komplexen Unterfangen. Die Programmiersprachen, Betriebssysteme und Programmierparadigmen sind so unterschiedlich, dass eine einheitliche Entwicklung nicht möglich ist und so eine aufwändige, zeitraubende und vor allem teure Mehrfachentwicklung notwendig wird.
Die softwaretechnische Vielfalt ist nur ein Teil des Problems, auch die Hardware der Endgeräte weisen eine große Vielfalt auf. Es gibt unterschiedliche Bildschirmgrößen, unterschiedliche Bildschirm-Seitenverhältnisse, Geräte mit und ohne berührungsempfindlichen Display, Geräte mit und ohne Tastatur und noch viele weitere Unterschiede.
Die Homogenität, die in den 80er Jahren den quasi-Standard „IBM-PC kompatible“ zum Durchbruch verhalf, ist auf mobilen Geräten noch nicht gegeben. Nach Meinung des Autors ist eine ähnliche Vereinheitlichung in den nächsten Jahren nicht absehbar, da jeder Anbieter darauf bedacht ist, seine Plattform als Alleinstellungsmerkmal zu bewahren.

Betrachtet man die schier unüberwindbare Komplexität der Vielfalt von mobilen Plattformen, erscheint die Entwicklung von Software auf mobilen Geräten, die auf vielen Plattformen läuft, als nahezu unmögliche Aufgabe. Doch die Lösung ist denkbar simpel und vom großen Bruder, dem Desktop PC abgeschaut: Web Anwendungen oder auch Web Services. Die Erfahrungen von Desktop PCs haben gezeigt, dass das Problem der unterschiedlichen Plattformen am ehesten durch browserbasierte Anwendungen zu handhaben ist. Bei einer Webanwendung sollte es irrelevant sein, ob der Anwender ein Linux, Mac oder ein Windows Betriebssystem verwendet. Webanwendungen basieren auf offenen Standards und robusten Implementierungen.
Diesen Lösungsansatz kann man auch auf mobile Plattformen übertragen, da die gängigen PDAs alle einen Internetbrowser mitbringen. Eine Webanwendung verhält sich im besten Fall auf allen Plattformen, ohne spezielle Anpassungen, gleich.

Die Erfahrungen auf den Desktop PCs haben gezeigt, dass klassische HTML Webanwendungen zwar durch ihre Plattformunabhängigkeit bestechen, jedoch das statische Konzept dieser nicht ausreicht, um lokale Anwendungen abzulösen. Daher bieten aktuelle Webanwendungen nicht mehr nur starre HTML Formulare und serverseitig generierte HTML Dokumente, sondern versuchen mit zusätzlichen Technologien den Bedienkomfort einer lokalen Anwendung nachzuahmen. So eine „reichhaltige“ Webanwendung wird als RIA (Rich Internet Application) bezeichnet. RIAs ermöglichen die Entwicklung von Webanwendungen, die dem Benutzerkomfort von klassischen, lokalen Anwendungen nahe kommen. Technologische Basis für RIAs sind meist Adobe Flash und das um asynchrone XML-Kommunikation

erweitertete JavaScript, welches als AJAX (Asynchronous JavaScript and XML) bezeichnet wird.
RIAs bieten jedoch nicht nur die Möglichkeit, verbesserte Oberflächen in Webanwendungen zu integrieren, sie bringen auch ein neues Verteilungskonzept mit sich. Im Gegensatz zu einer lokalen Anwendung muss eine RIA nicht installiert werden. Bei Bedarf wird eine RIA über eine URL aufgerufen, die Anwendung vom Internet Browser heruntergeladen und lokal ausgeführt. Dieses Prinzip, Code erst bei Bedarf anzufordern wird als Code on Demand bezeichnet. RIAs gelten dem Verhalten nach als Code on Demand Anwendungen. Das Code on Demand Paradigma basiert auf dem Mobile Code Konzept. Das W3C definiert[2] Mobile Code als

> [..] code that can be transmitted across the network and executed on the other end

Ein Beispiel für Code on Demand Anwendungen sind in Webseiten eingebettete Java Applets. Ein Anwender navigiert mit seinem Browser zu einem HTML-Dokument, der Browser lädt den Java Code „on Demand“ nach und führt diesen auf der lokalen Maschine aus. Verlässt der Anwender das HTML-Dokument, endet der Lebenszyklus der Anwendung. Das Nachladen des Programmcodes findet für den Benutzer transparent im Hintergrund statt.
Das Code on Demand Konzept kann auch zum Missbrauch verwendet werden. Viele Viren, die sich über Email Anhänge verbreiten, laden ihren Code aus dem Internet nach und sind somit ebenfalls als Code on Demand Anwendungen einzustufen.
Code on Demand senkt den Aufwand und die Komplexität der Verteilung wesentlich. Da Code on Demand Anwendungen bei jedem Start aktuell sind, können neue Features schneller veröffentlicht und Fehler zeitnah beseitigt werden. Viele solcher Anwendungen werden durch die einfache Verteilungsmöglichkeit ständig weiterentwickelt und haben nie eine fertige Version. So deklariert Google seinen Email Dienst *Gmail* [3] seit nunmehr 4 Jahren als „beta“.

Durch die Migration lokaler Anwendungen zu browserbasierten Code on Demand Anwendungen entstehen neue Herausforderungen. Das Offensichtlichste ist die nicht mehr gewährte Verfügbarkeit der Anwendung. Da leichtgewichtige Browseranwendungen mindestens beim Start der Anwendung eine bestehende Verbindung zu einem Webserver benötigen, sind diese offline nicht verfügbar. Dies ist besonders im Unternehmens Einsatz nicht akzeptabel. Ein weiteres Problem resultiert aus den Sicherheitsbeschränkungen des

[2] http://www.w3.org/MobileCode/; Zugriffsdatum: 2008-03-18
[3] http://www.google.com/mail; Zugriffsdatum: 2008-03-17

Browsers. Aufgrund diverser Angriffe durch Schadsoftware wie Viren lassen aktuelle Browser die Webanwendung in einer sogenannten Sandbox laufen. Dies soll verhindern, dass Webseiten die Integrität des lokalen Systems gefährden. Dieses Konzept ist sehr sinnvoll und verhindert viele Sicherheitsprobleme mit Webbrowsern, bringt aber auch Probleme bei dem Einsatz von Code on Demand Anwendungen mit sich. Die Beschränkungen bedeuten, dass Code on Demand Anwendungen weder auf Hardwarekomponenten zugreifen können, noch auf das lokale Dateisystem. Die Beschränkungen von Webanwendungen durch die Browser verhindern bisher oftmals die Migration lokaler Anwendungen zu leichtgewichtigen Code on Demand Anwendungen.

Bei der Bewertung aktueller Technologien bietet die Gartner Group[4] mit ihrem *Hype Cycle for Emerging Technologies* eine gute Orientierung für die Bewertung der Marktreife. Im Jahr 2007 hat die Gartner Group das Web 2.0 Paradigma in das *Trough of Disillusionment* (deutsch: Tal der Desillusionierung) geschickt (siehe Abb. 2) und prognostiziert deren Marktreife erst in den nächsten 2-5 Jahren. Mobile Computing Konzepte, besonders aus dem Bereich Location Awareness sind hingegen schon weiter, sie haben es in den Bereich der *Slope of Englightenment* (deutsch: Steigenden Erkenntnis) geschafft. Aus Sicht des Autors ist die langfristige Marktreife des Web 2.0 Paradigmas nur durch die Verbindung mit Mobile Computing erreichbar. Aus diesem Grunde wird im Rahmen dieser Arbeit eine Plattform entworfen, die die effiziente und robuste Zusammenführung von Code on Demand und Mobile Computing ermöglicht.

1.2 Zielsetzung

Im Rahmen dieser Arbeit soll eine Middleware entworfen werden, um die zuvor beschriebenen Probleme beim Einsatz von Code on Demand Anwendungen als Ersatz für lokale Anwendungen auf mobilen Geräten zu beheben.
Die beiden Hauptprobleme, die nicht garantierte Verfügbarkeit der Anwendung und die beschränkten Zugriffsmöglichkeiten, sollen durch eine Zwischenschicht zwischen Browser und dem lokalen System in Form einer Middleware gelöst werden. Die Middleware soll lediglich als Speicher, Zugriffs und Kommunikationskomponente dienen, die Code Ausführung und Darstellung findet hingegen durch einen Webbrowser statt.
Damit Code on Demand Anwendungen ohne eine Verbindung zu dem entsprechenden Webserver funktionsfähig bleiben können, muss ein entsprechender Cache als zwischenspeicher verfügbar sein. Die zu entwickelnde Middleware wird solch einen Cache anbieten

[4] http://www.gartner.com/; Zugriffsdatum: 2008-05-18

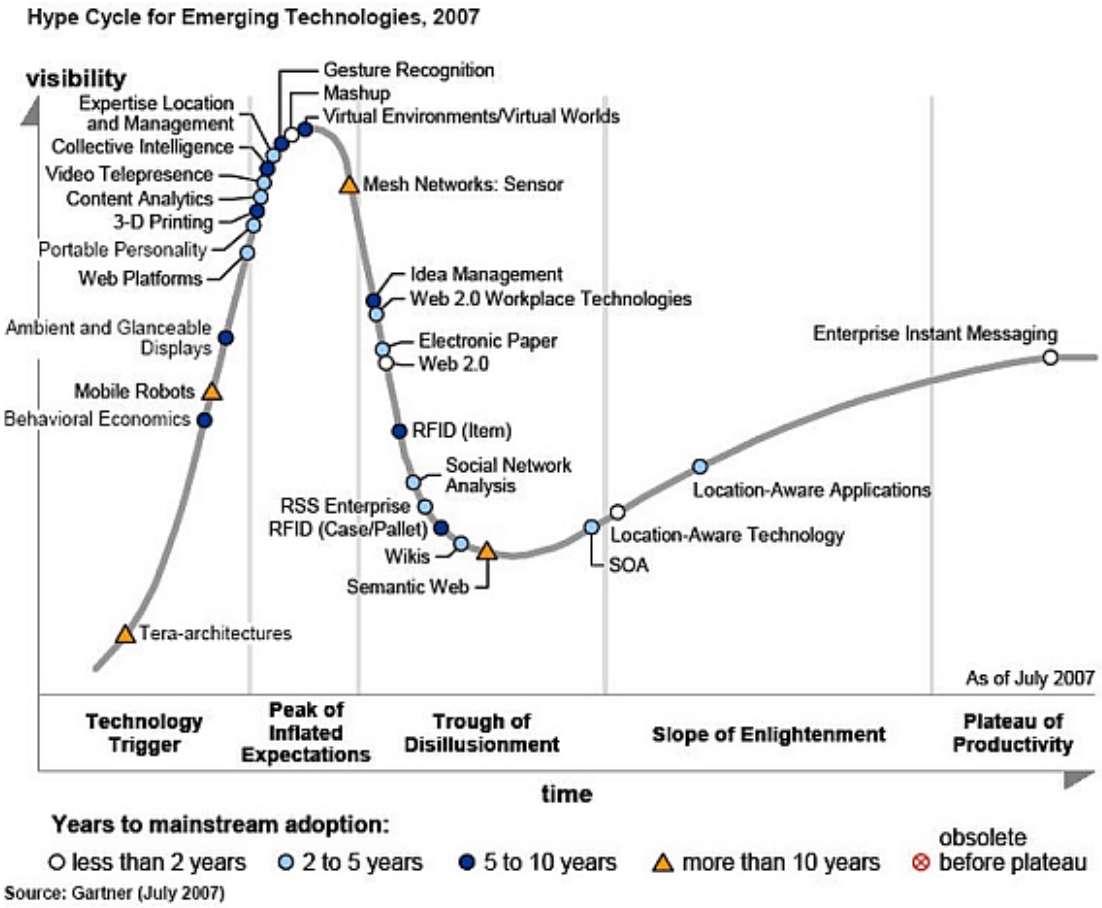


Abbildung 2: Hype Cycle for Emerging Technologies 2007 (Quelle: Gartner Group)

und es ermöglichen, Anwendungen in diesen Cache zu laden. Somit werden Webanwendungen auch ohne Verbindung zu dem Webserver verfügbar. Weiterhin ist entscheidend, dass die Anwendung nicht nur ausgeführt werden kann, sondern eine lokale Datenspeicherung ermöglicht wird. Der Cache wird die Möglichkeit bieten, Daten lokal vorzuhalten, diese zu manipulieren und später zu synchronisieren.
Die Middleware wird Schnittstellen zur Verfügung stellen, um auf lokale Subsysteme, wie beispielsweise ein GPS Modul oder die Telefonfunktionen zugreifen zu können. Diese Schnittstellen werden konfigurierbar sein und mit anwendungsspezifischen Berechtigungen versehen werden.

Im Rahmen der Arbeit soll eine Systembeschreibung und ein Architekturentwurf für die Messaging Middleware entstehen. Die Middleware soll auf bestehenden, offenen Standards

aufbauen und sich transparent in die vorhandenen Nachrichten- und Protokollstrukturen einfügen. Wichtig für das Verständnis der Lösung ist eine klare Meta-Sicht auf das Konzept, welches in der Arbeit ausführlich beschrieben wird.
Neben der reinen Theorie, die in der Analyse und dem Systementwurf behandelt wird, wird auch ein Prototyp als Proof of Concept entwickelt. Mithilfe des Prototyps soll evaluiert werden, ob die entwickelte Lösung in der Realität praktikabel ist. Der Prototyp soll eine mögliche Umsetzung der Lösung darstellen, jedoch soll und muss die Lösung so flexibel konzipiert sein, dass sie auch auf anderen Plattformen realisiert werden kann.
Die Middleware ist zwar grundsätzlich unabhängig von der Code on Demand Technologie, bei der Implementierung des Prototyps und in großen Teilen der Arbeit wird jedoch der Fokus auf AJAX liegen. Bei relevanten Unterschieden zu anderen Code on Demand Technologien wie beispielsweise Adobe Flash, wird an entsprechender Stelle hingewiesen.

1.3 Bestehende Arbeiten und wissenschaftliche Einordnung

Ziel dieser Arbeit ist die Vereinigung des Code on Demand Konzepts mit Technologien des Web 2.0 Paradigmas und Mobile Computing.
Den Anstoß für diese Arbeit lieferten die Whitepapers der OpenAjax Alliance. Dieses Konsortium relevanter IT Unternehmen versucht, die Verwendung von Ajax zu Standardisieren um Ajax Anwendungen so robuster, austauschbarer und zukunftssicherer zu machen. Die OpenAjax Alliance bildet das Dach mehrerer spezialisierter Arbeitsgruppen. Eine dieser Arbeitsgruppen beschäftigt sich mit Mobile Ajax und hat zu dem Thema mehrere Whitepapers veröffentlicht (mobOpenAjax, 2007).

Die OpenAjax verweist mehrfach auf die *7 Rules of Mobile Data User Experience* (7rules, 2005) von Elliot Drucker als Basis für ihre Arbeiten. Im Besonderen Relevant für diese Arbeit ist der Punkt 4 von Drucker, der die Bedeutung und Motivation einer offline Fähigkeit beschreibt:

> One thing about mobile data users: they tend to move around. And sometimes they find themselves in locations (airplanes, subway systems, etc.) that even the most ubiquitous wireless data networks cannot cover. However, lack of wireless data access does not mean that a data application cannot provide useful functionality. In some cases a user may be able to anticipate the loss of

> coverage and pre-load data for later use by an application program resident in his or her mobile device.[..][5]

Die OpenAjax Alliance berücksichtigt den zitierten Punkt in den bisherigen Arbeiten nicht. In vielen klassischen lokal installierten mobilen Anwendungen jedoch ist diese Regel befolgt worden. Ein typisches Beispiel sind Email Clients auf PDAs. Diese empfangen und senden Emails sofern sie Online sind und bieten offline die Möglichkeit, Emails zu lesen und zu verfassen.

Aus Sicht der Autors birgt eine offline Funktionalität beim Einsatz von Code on Demand Technologien im Besonderen auf mobilen Endgeräten viel Potenzial. So wird es erst durch eine offline Fähigkeit sinnvoll möglich, lokal installierte Anwendungen wie beispielsweise Email Clients durch Code on Demand Anwendungen zu ersetzen. Aus diesem Grund baut die vorliegende Arbeit auf denen der OpenAjax mit dem Ziel auf, mobile Code on Demand Anwendung mit offline Funktionalität auf mobilen Geräten zu ermöglichen.

Weiterhin hatten die ebenfalls von der OpenAjax referenzierten *Mobile Web Best Practices 1.0* (mobWeb, 2006) des W3C Einfluss auf die Entwicklung der Middleware. Das W3C beschreibt darin diverse Paradigmen bezüglich der Entwicklung von Webanwendungen für mobile Endgeräte. Bei der Entwicklung der Middleware hatte dies Einfluss auf die Datenrepräsentation und die Nutzbarkeit, diente aber vor allem als Grundlage für die Analyse der Anforderungen mobiler Webanwendungen.

Der Anwendungs-Fokus dieser Arbeit liegt auf dem Unternehmenseinsatz von Ajax Anwendungen auf mobilen Geräten als Ersatz für lokale Anwendungen, wie es beispielsweise im *mobileweaver ajax framework*(mobWeaver, 2008) beschrieben wird. In diesem Einsatzgebiet langen die offline Funktionalitäten alleine nicht aus, es werden noch weitere Zugriffsmöglichkeiten benötigt, beispielsweise der Zugriff auf ein GPS Modul. Um dies erfüllen zu können, bietet die Middleware auch einen indirekten Zugriff auf lokale Subsysteme des Gerätes und auf das lokale Dateisystem.

[5] Übersetzung des Autors: *Ein wichtiger Punkt über mobile Anwender: sie neigen dazu, sich zu bewegen. Manchmal befinden sie sich an Orten (Flugzeuge, U-Bahnen, etc.) die selbst die allgegenwärtigste, drahtlose Netzwerkverbindung nicht abdecken kann. Das Nichtvorhandensein einer drahtlosen Datenverbindung bedeutet aber nicht, dass eine Daten Anwendung keine sinnvolle Funktionalität bietet kann. In einigen Situationen kann der Anwender die fehlende Abdeckung vorhersehen und Daten vorab laden, welche später von einer auf dem mobilen Gerät installierten Anwendung verwendet werden können.*[..]

1.4 Inhaltlicher Aufbau der Arbeit

Das einleitende Kapitel 1 beschreibt die Motivation und die Ziele für diese Arbeit.

Das anschließende Grundlagen Kapitel 2 vermittelt die notwendigen Grundlagen für das Verständnis der Arbeit. Zunächst wird eine Übersicht über die relevanten mobilen Plattformen gegeben und diese gegeneinander abgegrenzt. Da der Prototyp der Middleware auf Windows Mobile basiert, wird auf Windows Mobile im Detail eingegangen.
Im zweiten Teilabschnitt des Kapitels werden Webanwendungen und relevante Technologien vorgestellt.

Das Kapitel Analyse 3 beschreibt zunächst eine Beispielanwendung, um den Leser ein besseres Gefühl für die praktischen Anforderungen zu geben. Folgend werden die Anwendungsfälle beschrieben und die funktionalen- und nichtfunktionalen Anforderungen an die Middleware definiert.

Im Entwurfs Kapitel 4 werden die aus der Analyse gewonnenen Erkenntnisse und Definitionen für einen Entwurf der Middleware genutzt.
Das Kapitel beginnt mit einer Modul Definition der Middleware. Die anschließenden Sichten verfeinern die Architekturbeschreibung. Den Abschluss bildet die Schnittstellendefinition der Middleware.

Im Rahmen dieser Arbeit fand eine prototypische Umsetzung als Proof of Concept statt. Der entwickelte Prototyp wird in Kapitel Realisierung 5 vorgestellt.
In Unterkapiteln werden die verwendeten Technologien, die Bedienung des Prototypen und die mitgelieferten Beispielanwendungen beschrieben. Abschließend wird am Beispiel des Prototypen die Integration der Middleware in eigene Lösungen erläutert.

Den Abschluss bildet das Fazit 6. Hier wird nach einer Zusammenfassung ein Überblick über die Ergebnisse dieser Arbeit gegeben. Das Kapitel und diese Arbeit schließt mit einem Ausblick.

2 Grundlagen

Im folgenden Abschnitt sollen die Grundlagen vermittelt werden, die für das Verständnis der folgenden Kapitel notwendig sind.

2.1 Mobile Plattformen

Im Besonderen sind im Rahmen dieser Arbeit Personal Digital Assistants (PDAs) und Smartphones als mobile Plattformen relevant. Das Problem der Vielfalt ist in der Motivation (1.1) beschrieben worden, der folgende Abschnitt wird die zum Zeitpunkt der Erstellung relevantesten Ausprägungen darstellen.

2.1.1 Hardware Plattformen

Zwei Ausprägungen von mobilen Plattformen sind für diese Arbeit im Besonderen Relevant, PDAs und Smartphones. Markante Eigenschaft eines PDAs ist der berührungsempfindliche Eingabebildschirm. Ein Smartphone hingegen besitzt keinen Touchscreen und wird ausschließlich über eine Tastatur bedient.
Inzwischen ist die Abgrenzung zwischen Smartphone nicht mehr scharf und beide Begriffe werden oft synonym verwendet. Hinzu kommt, dass es immer mehr hybride Geräte gibt, die sowohl Touchscreen als auch externe Tastatur besitzen (wie das HP iPAQ auf Abbildung 3).

Abbildung 3: HP iPAQ hw6915 - PDA oder Smartphone?

Im Folgenden wird nicht mehr zwischen Smartphones und PDAs unterschieden, beide Typen werden gemeinsam als mobile Plattformen oder mobile Endgeräte bezeichnet. Für das Konzept der Middleware ist es nicht relevant, ob die Eingabe über eine Tastatur oder einen Touchscreen erfolgt.

2.1.2 Software Plattformen

Als Software Plattformen sollen hier die relevanten Betriebssysteme für die mobilen Endgeräte vorgestellt werden. Die bereits beschriebene Vielfalt ist in diesem Bereich sehr ausgeprägt, daher werden hier nur einige der relevantesten vorgestellt.

2.1.2.1 Windows Mobile

Windows Mobile ist ein auf Windows CE basierendes 32-Bit Betriebssystem. Windows CE wurde als kompaktes Betriebssystem für eingebettete Systeme und mobile Endgeräte konzipiert. Windows CE hat zwar eine an das Desktop Betriebssystem angelehnte Oberfläche, jedoch einen komplett anderen Kernel, daher laufen Desktop Anwendungen nicht auf Windows CE und somit auch nicht auf Windows Mobile.

Windows Mobile ist eine Erweiterung von Windows CE, die speziell für Smartphones

Abbildung 4: Windows Mobile 6.1 Startbildschirm

und Pocket PCs (Microsofts Bezeichnung für PDAs mit Telefon Funktionalität) entwickelt wurde. Die prägnantesten Unterschiede sind zum einen die Telefon Funktionalität und zum anderen die auf kleine Displays optimierte Oberfläche. Weiterhin liefert Windows Mobile ein Office Paket mit und setzt so klar den Fokus auf einen geschäftlichen, mobilen Kontext. Windows Mobile ist normalerweise nur in Verbindung mit einem Gerät erhältlich und ist nicht im Einzelhandel zu beziehen.

Der Prototyp der im Rahmend dieser Master Arbeit entwickelten Middleware ist auf Basis von Windows Mobile realisiert. Aus diesem Grunde wird auf die Details von Windows Mobile in einem eigenen Abschnitt (2.2) eingegangen.

2.1.2.2 Android

Die Open Handset Alliance gab Ende 2007 bekannt, dass sie ein Betriebssystem für mobile Plattformen entwickeln wird. Zu der Open Handset Alliance gehören Unternehmen wie

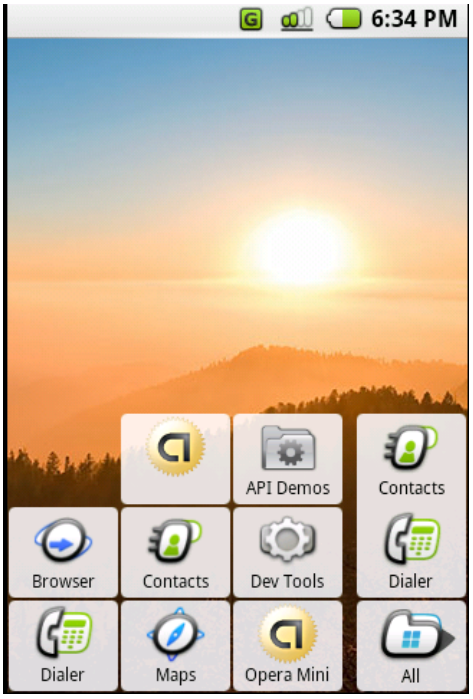


Abbildung 5: Android Startbildschirm

HTC, LG, Motorola und Samsung Electronics. Initiator und treibende Kraft der Open Handset Alliance ist Google Inc., dies ist auch der Grund, warum Android oft fälschlicherweise als

Google Android bezeichnet.
Die Open Handset Alliance definiert Android als:

> Android is a software stack for mobile devices that includes an operating system, middleware and key applications.

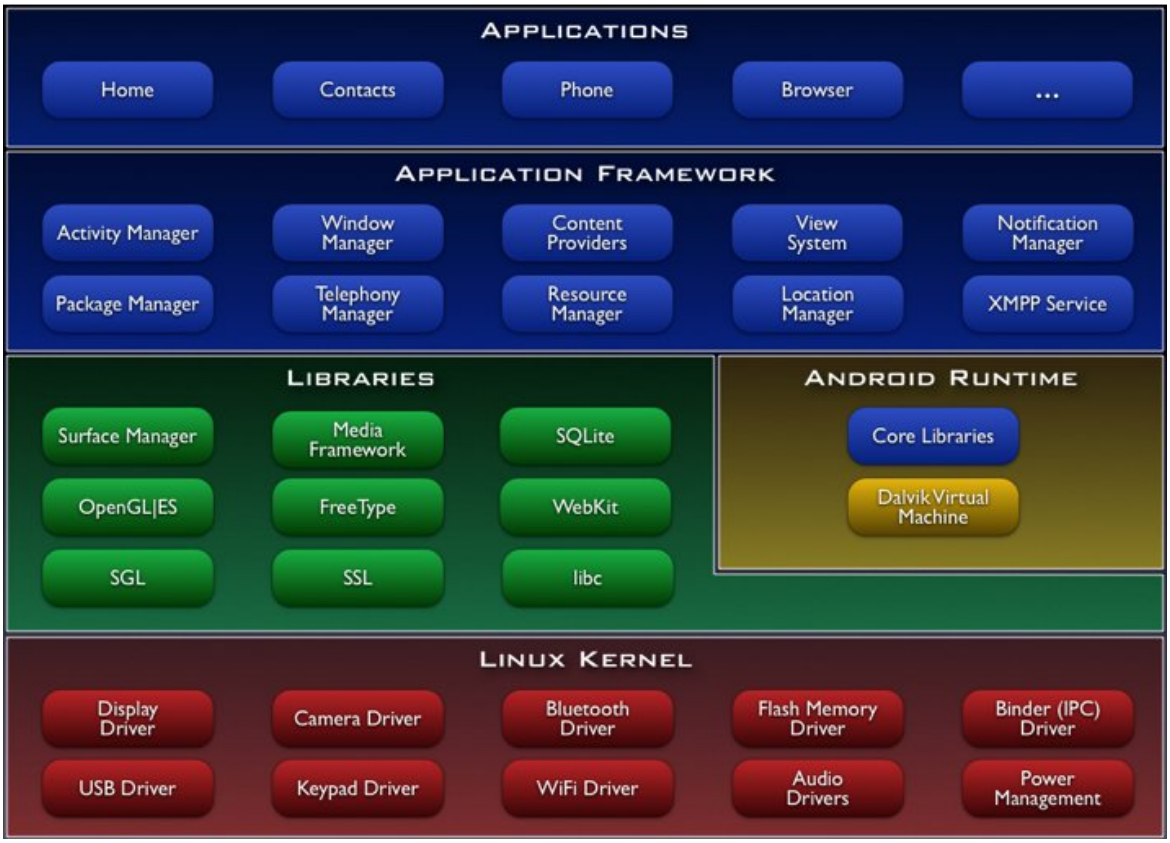


Abbildung 6: Android System Architektur (Quelle: Android SDK)

Eine grafische Darstellung der System Architektur von Android ist im Schaubild 6 gegeben.
Für Prozess- und Speicherverwaltung, ebenso als Hardware Abstraktions Schicht, setzt Android auf einen Linux Kernel der Version 2.6. Weiterhin ermöglicht der Linux Kern den Netzwerk Zugriff.
Auf dem Linux Kern aufbauend, nutzt Android eine spezielle Java Virtual Machine *Dalvik*, die erweitert um einige Bibliotheken als *Android Runtime* bezeichnet wird. Dalvik ist im Gegensatz zu der Java VM von Sun Microsystems nicht Stack- sondern Register basiert.

Dalvik ist durch das Ansprechen von Registern auf mobilen Geräten wesentlich effizienter. Für Dalvik kompilierter Code ist nicht auf einer normalen Java VM ausführbar und Vice versa.
Neben der Android Runtime liefert Android einige relevante Bibliotheken mit. Beispielsweise werden SQLite als Relationales Datenbank Management System, Webkit als Browser Komponente, FreeType für Vector Schriftarten und SSL für gesicherte Verbindungen über ungesicherte Netzwerke mitgeliefert. Diese laufen nicht auf der Android Runtime, sondern sind klassische C-Bibliotheken.
Basierend auf der Android Runtime und den mitgelieferten Bibliotheken befindet sich das eigentliche Android, das Application Framework. Dieses stellt diverse Bibliotheken und Schnittstellen zur Programmierung bereit.
Die fundamentalen Anwendungen in Android sind in Java geschrieben und sind gleichgestellt mit Software von Drittanbietern. Bei den meisten bestehenden mobilen Betriebssystemen ist dies nicht der Fall. So lässt sich beispielsweise bei Windows Mobile das Programm *Kontakte* nicht durch eine eigene Kontakt-Anwendung ersetzten; dies ist bei Android möglich.

Android ist zum Zeitpunkt der Erstellung noch nicht auf physikalischen Geräten verfügbar. Es existiert ein Software Development Kit (SDK), welches neben einigen Beispielanwendungen und Tutorials auch einen Emulator und eine Integration in die freie Entwicklungsumgebungsplattform Eclipse bietet.

2.1.2.3 Symbian OS

Symbian OS ist Smartphone Betriebssystem Firma Symbian[6].
Symbian OS ist ein Multitasking fähiges 32-Bit Betriebssystem und baut auf dem von der Firma Psion entwickelten Betriebssystem EPOC. Symbian OS ist stark verbreitet und wird unter anderem in Telefonen der Firmen Nokia, Sony Ericsson und Samsung eingesetzt.
Es gibt im Gegensatz zu Windows Mobile oder Android nicht eine Oberfläche, sondern mehrere. Die bekanntesten Oberflächen sind zum einen das von Nokia entwickelte und eingesetzte *S60* und zum anderen das primär von Sony Ericsson eingesetzte *UIQ*.
Anwendungen für Symbian OS werden primär in den Sprache C++ und Java entwickelt, weitere Sprachen wie Python sind verfügbar.
Bis 2010 soll Symbian OS und dessen Oberflächen unter die Eclipse-Lizenz, also Open Source gestellt werden. Die relevanten Firmen wie Nokia, Sony Ericsson und Samsung haben hierzu die Symbian Foundation beschlossen.

[6] http://www.symbian.com/; Zugriffsdatum 2008-08-03

2.2 Windows Mobile

Windows Mobile wurde bereits kurz in Abschnitt 4 vorgestellt. Da der in Rahmen dieser Master Arbeit entwickelte Prototyp auf Basis von Windows Mobile entwickelt wurde, wird folgend noch etwas detaillierter auf Windows Mobile eingegangen.

2.2.1 Konzept

Microsoft sieht in Windows Mobile nicht bloß eine Plattform für mobile Endgeräte, sondern die mobile Komponente einer ganzheitlichen Unternehmens Lösung.
Eine der korrespondierenden Komponenten ist der *Microsoft Exchange Server*, für die zentrale Speicherung von (Push-)Email, (Push-)Kontakte und (Push-)Kalender. Eine weitere Komponente ist der *Microsoft Internet Security and Acceleration Server*, hierbei handelt es sich um eine Firewall und somit Zugriffs Schnittstelle für interne Ressourcen (winMobArch, 2008).
Für die Verwaltung von Geräten in der IT Infrastruktur eines Unternehmens bietet Microsoft die Produkte *Microsoft System Center Configuration Manager* und *Microsoft Systems Center Operations Manager* mit diesen beiden Produkten lassen sich beispielsweise IT Landschaften inventarisieren und deren Policies verwalten.
Für mobile Endgeräte existiert ein Produkt mit der Bezeichnung *System Center Mobile Device Manager*. Dieser dient der Fernadministration von mobilen Endgeräten. Ein System Administrator erhält so beispielsweise die Möglichkeit, ein gestohlenes Gerät remote zurückzusetzen (Remote Wipe).

Seit der Version 5 von Windows Mobile, liefert Microsoft im ROM der Geräte eine spezielle Version seines Datenbank Management Systems *Microsoft SQL Server (MSSQL)* aus. Durch diese Erweiterung steht auf den Geräten eine relationale Datenbank zur Verfügung, auf die mit der Abfragesprache SQL zugegriffen werden kann. Zusätzlich existieren Replikationsdienste, um eine lokale Datenbank auf einem mobilen Gerät mit einem entfernten MSSQL zu synchronisieren.

2.2.2 Architektur

Die bei der Erstellung dieser Arbeit aktuelle Version von Windows Mobile lautet 6.1 und basiert auf Windows CE 5.2. Diese Version wird auch im Prototyp verwendet.
Windows Mobile und das Windows CE auf dem es aufbaut, sind stark an das Desktop Pendant Windows angelehnt, weisen aber grundlegende Unterschiede auf. Windows XP ist aus-

schließlich für x86 Architekturen konzipiert, Windows CE hingegen ist für diverse Architekturen verfügbar (ARM, MIPS, SuperH, x86).

2.2.2.1 Kernel

Der Kernel von Windows Mobile befindet sich in `Kernel.dll` und wird als Prozess `Nk.exe` ausgeführt. Windows Mobile bietet eine auf dem Win32 API basierende Schnittstelle auf die über `Coredll.dll` zugegriffen werden kann.
Durch eine Abstraktionsschicht zur Hardware (Hardware Abstraction Layer) wird ein Betriebssystem unabhängig von der darunter liegenden Hardware. Für einen neuen Hardwaretyp muss lediglich der Hardware Abstraction Layer angepasst werden, dieser befindet sich bei Windows Mobile in `Hal.lib`.
Der eigentliche Kern von Windows Mobile ist in `Nk.lib` implementiert und bietet den Zugriff auf die Subsysteme des Betriebssystems, ebenso wie auf die Betriebssystem internen Dienste wie Prozessverwaltung und Fehlerbehandlung.
Das Schaubild 7 stellt die einzelnen Komponenten des Windows Mobile Kernels grafisch dar.

2.2.2.2 Speicher

Windows Mobile unterscheidet zwischen drei Arten von Speicher: ROM, RAM und Persistent Storage.
Der ROM ist ein Speicher auf den nur lesend zugegriffen werden kann (Read Only Memory). Im ROM befindet sich das Betriebssystem, sowie für das Hochfahren des Systems relevante Dateien und einige Anwendungen.
RAM (Random Access Memory) entspfdem Arbeitsspeicher eines PCs, wird also für das Laden und Ausführen von Code und zwischenspeichern von Daten verwendet. RAM ist ein flüchtiger Speicher.
Bis Version 5.0 lagen alle Daten des Telefons, wie beispielsweise die Emails und Kontaktdaten im RAM. Dies hatte zur Folge, dass bei einem sogenannten „Tiefenentladen“, also dem kompletten entleeren des Akkus, diese Daten verloren waren. In Version 5.0 wurden deshalb diese Daten in einen persistenten Flash Speicher verschoben (Persistent Storage). Da dieser nicht zum Schutz vor Datenverlust permanent mit Strom versorget werden muss, hat diese Änderung den positiven Nebeneffekt, dass der Energieverbrauch sinkt.

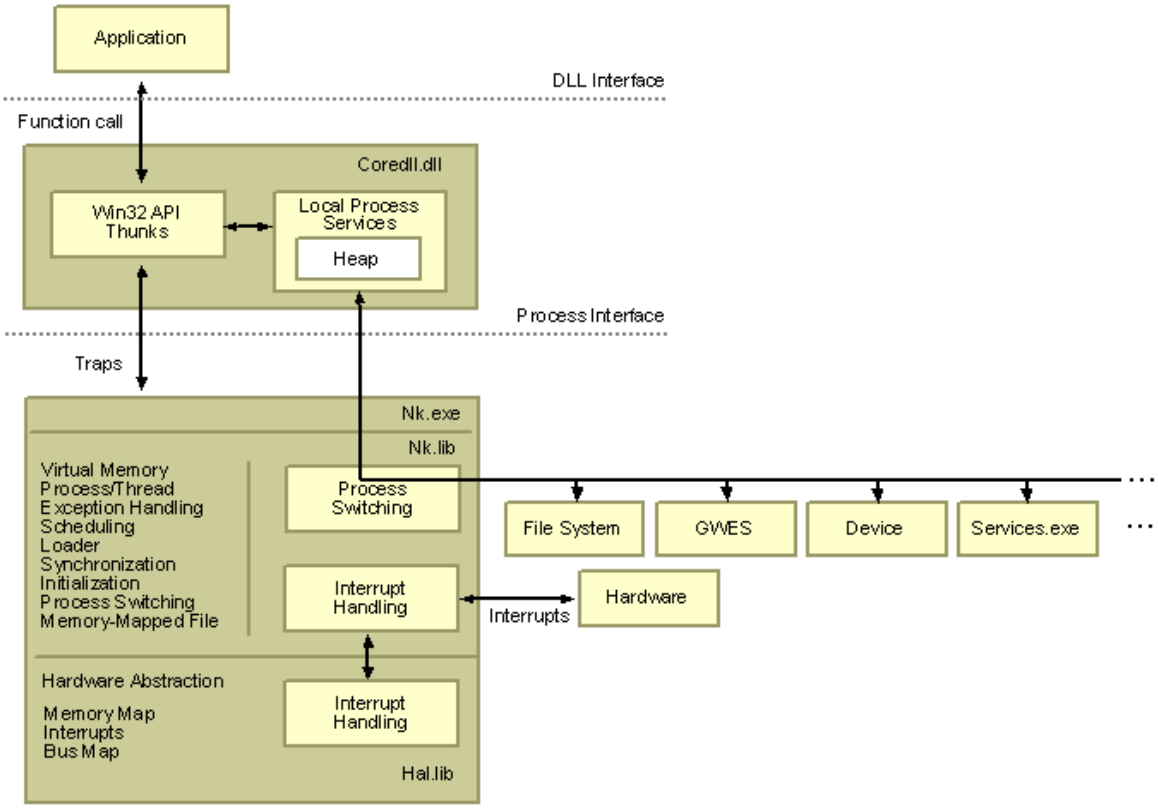


Abbildung 7: Windows Mobile Kernel Übersicht (Quelle: (winMobKernel, 2008))

2.2.3 Programmierung

Windows CE und auch Windows Mobile Geräte werden typischer weise entweder maschinennah in C/C++ oder auf Basis des .NET Compact Frameworks (2.2.4) entwickelt. Weiterhin ist auch die Ausführung von Java ME Anwendungen möglich.
Die Ausführung von Programmen auf einer virtuellen Maschine impliziert die bekannten Vor- und Nachteile: mehr Sicherheit und Stabilität für geringere Performance. Die Performance Einbußen waren in der Vergangenheit ein starker Kritikpunkt gegenüber Programmiersprachen die eine Laufzeitumgebung benötigen wie beispielsweise Java oder C#. Die Laufzeitumgebungen wurden jedoch mit jeder Version performanter und die Hardware ebenfalls effizienter, so fällt die geringere Performance durch die Laufzeitumgebung immer weniger in Gewicht.

Für die Entwicklung von Programmen für sein Windows Mobile bietet Microsoft das Visual Studio. Auch wenn die Entwicklung von C/C++ oder .NET Anwendungen für Windows

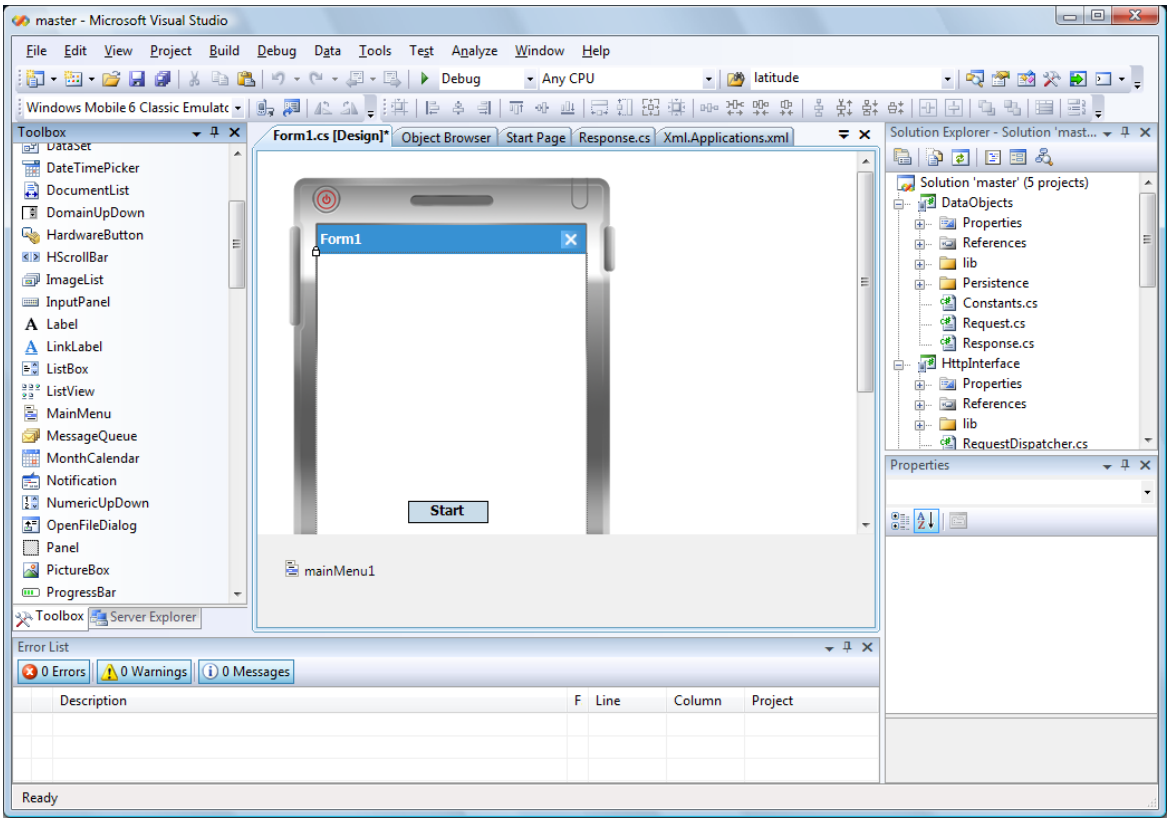


Abbildung 8: Microsoft Visual Studio 2008 mit GUI Builder

Mobile grundsätzlich auch in einem beliebigen Editor und Kommandozeilen Tools erfolgen kann, bietet die Integrierte Entwicklungsumgebung viele Vorteile. Visual Studio bietet die üblichen Funktionen einer integrierten Entwicklungsumgebung, bietet zudem aber einen grafischen Designer für Oberflächen von Windows Mobile Anwendungen, sowie diverse Windows Mobile Emulatoren, die sich direkt aus dem Visual Studio starten lassen. Durch die integrierten Emulatoren lassen sich Anwendungen für mobile Geräte entwickeln, die phsysisch für den Entwickler nicht verfügbar sind. Weiterhin bietet das Visual Studio die Möglichkeit, Anwendungen die auf verbundenen Geräten oder in einem der Emulatoren ausgeführt werden, zu debuggen.

2.2.4 Microsoft .NET Compact Framework

Das .NET Framework besteht aus einer Laufzeitumgebung, einer umfangreichen Klassenbibliothek, sowie diversen Werkzeugen für die Entwicklung, Wartung und Verteilung von Anwendungen. Die von Microsoft unterstützten Programmiersprachen für .NET ist zum

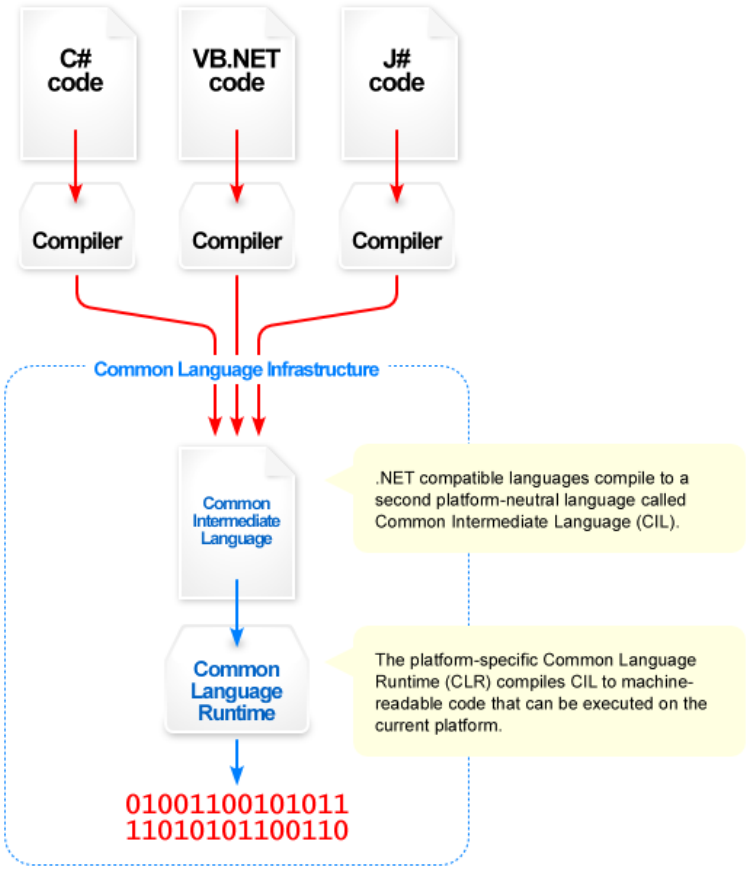


Abbildung 9: Arbeitsweise der Common Language Runtime von .NET

einen C# und zum anderen Visual Basic .NET (VB.NET). Es existieren weitere .NET Compiler und .NET ist grundsätzlich unabhängig von der Programmiersprache. Die .NET Compiler übersetzen den Programmcode nicht wie herkömmliche Compiler in Maschinencode, sondern in einen Zwischencode der als (Microsoft) Common Intermediate Language (CIL) bezeichnet wird. Der CLI Code wird zur Laufzeit des Programms von der Laufzeitumgebung, die als Common Language Runtime (CLI) bezeichnet wird, in Maschinencode übersetzt und ausgeführt (siehe Abbildung 9).

Das Microsoft .NET Compact Framework (.NET CF) ist eine Teilmenge des .NET Frameworks speziell für mobile Geräte wie PDAs mit dem Betriebssystem Windows CE oder dem darauf aufbauenden Windows Mobile.
Grundsätzlich bietet das .NET CF dieselben Eigenschaften, die auch das komplette .NET Framework bietet: eine Laufzeitumgebung und eine umfangreiche Klassenbibliothek, wo-

bei diese im Umfang gegenüber dem kompletten .NET Framework beschränkt ist. Die Einschränkungen lassen sich bei Bedarf oftmals durch Bibliotheken von Drittanbietern abfedern, wie beispielsweise dem Smart Device Framework von OpenNETCF Consulting[7].

2.3 Webanwendungen

Webanwendungen bieten die anwendungslogische Basis des heutigen Internets. Die zunächst statischen Webinhalte wurden mit der Zeit immer dynamischer und interaktiver. Die technologische Basis dazu wird in dem folgenden Abschnitt näher erläutert.

2.3.1 Klassische Webanwendungen

Der für diese Arbeit interessante Teil beginnt, als die ersten Webseiten aus statischen HTML-Dateien bestanden. Diese wurden von einem HTML-Entwickler auf dessen lokalen Rechner entwickelt und anschließend auf den Webserver übertragen. Die Webseitenbesucher konnten nichts anderes machen, als sich diese Datei auf den eigenen Rechner herunterzuladen und im Browser anzeigen zu lassen. Die einzige weitere Funktionalität waren Verlinkungen zu anderen Webseiten, die der Benutzer durch Anklicken des Links besuchen konnte.

2.3.2 Dynamische Webanwendungen

Der erste Schritt, die Webanwendungen dynamischer und aktueller zu machen, bestand in der dynamischen Generierung der Webseiten. Mit dem CGI - Common Gateway Interface (CgiW3c, 1993) wurde bereits 1993 die Möglichkeit geschaffen, einen Webserver generierte Webseiten ausliefern zu lassen. Weiterhin wurde die Möglichkeit geboten, mittels Formularen Daten vom Client an den Webserver zu übertragen und dort auszuwerten. Das CGI bietet eine Art Laufzeitumgebung und kann in nahezu jeder beliebiger Sprache programmiert werden. Ein Nachteil von CGI ist, dass es für jeden Aufruf einen eigenen Systemprozess startet und somit bei hohen Zugriffszahlen entsprechende Performanceprobleme bedingt.
Die Programme die die CGI nutzten, wurden zunächst klassisch in der Programmiersprache C, später in Perl entwickelt. Da die Entwicklung in C durchaus komplex ist und die Perl Syntax gerade für Einsteiger eine gewisse Hürde bietet, haben neuere Sprachen wie die Scriptsprache PHP oder auch Ruby großen Erfolg. Aber auch wenn Perl heute immer weniger für die Erzeugung dynamischer Webseiten verwendet wird, so läuft noch immer eine große Zahl dieser Anwendungen, z.B. Amazon.com.
Im Unternehmensbereich gibt es zwei dominierende Konzepte: Zum einen das Microsoft

[7] http://www.opennetcf.com/cf/products/sdf.ocf; Zugriffsdatum 2008-08-16

entwickelte ASP, heute ASP.NET mit dem .NET Framework und zum anderen das von Sun Microsystems entwickelte Java mit den Java Server Pages.
Hier soll keine Bewertung der einzelnen Technologien vorgenommen werden, jede hat Ihre Stärken und Schwächen. Für die weitere Arbeit ist die Bedeutung der Dynamisierung wichtig, die heutzutage ein essentieller Bestandteil des Internets ist. Ohne Webseiten Formulare und die Möglichkeit, diese Daten programmatisch auszuwerten würde es Unternehmen wie Google, Amazon und eBay heute nicht geben.

2.3.3 JavaScript

Dynamische Webseiten werden oftmals in Verbindung mit Formularen verwendet. Die Webanwendung liefert ein Formular an den Besucher, dieser füllt dieses Formular aus und schickt es an den Server zurück. Dies kann ein Webmailer, ein e-Commerce Shop oder auch ein virtuelles Gästebuch sein. Hierbei wurden zwei Probleme klar. Das erste Problem liegt, wie so oft, beim Benutzer. Menschen machen Fehler, wenn sie Formulare ausfüllen. Zahlen werden vertauscht, Buchstaben vergessen oder Felder, die ausgefüllt werden müssen, bleiben leer. Daraus resultiert das zweite Problem, der Client muss das Formular absenden, der Server empfängt die Daten, wertet sie aus und liefert die komplette Seite zurück, möglicherweise mit einer eingebetteten Fehlermeldung. Das absenden (Request) eines Formulars und die anschließende Antwort (Response) wird als Server-Roundtrip bezeichnet.
In Abbildung 10 ist ein Server-Roundtrip grafisch veranschaulicht. Der Benutzer wird aufgefordert, seine Email Adresse in ein Formular einzutragen. Er sendet die Daten durch klicken des *submit*-Buttons an den Server (Request). Der Server prüft die übergebene Emailadresse und stellt fest, dass die Email Adresse ungültig ist. Der Server platziert eine Fehlermeldung auf der Seite und sendet diese an den Benutzer zurück (Response). Der Benutzer musste die Zeit seit dem Absenden des Formulars warten und erfährt erst jetzt, dass ihm bei der Eingabe ein Fehler unterlaufen ist. Folgend korrigiert er seinen Fehler, sendet das Formular ab und muss wieder warten, bis ihm der Server nun das Formular mit einer Erfolgsmeldung zusendet.

Dieses Mankos bewusst, entwarf Brendan Eich eine integrierte Scripting Sprache für den Netscape Browser welche mit der Version 2.0B3 im Dezember 1995 vorgestellt wurde. Die Scriptsprache hieß zunächst LiveScript und wurde später zu ihrem heutigen Namen JavaScript umbenannt. Unter dem Namen ECMAScript wurde JavaScript durch die Ecma International standardisiert (EcmaScriptSpec, 2002).
JavaScript konnte in der ersten Version hauptsächlich dazu verwendet werden, Formu-

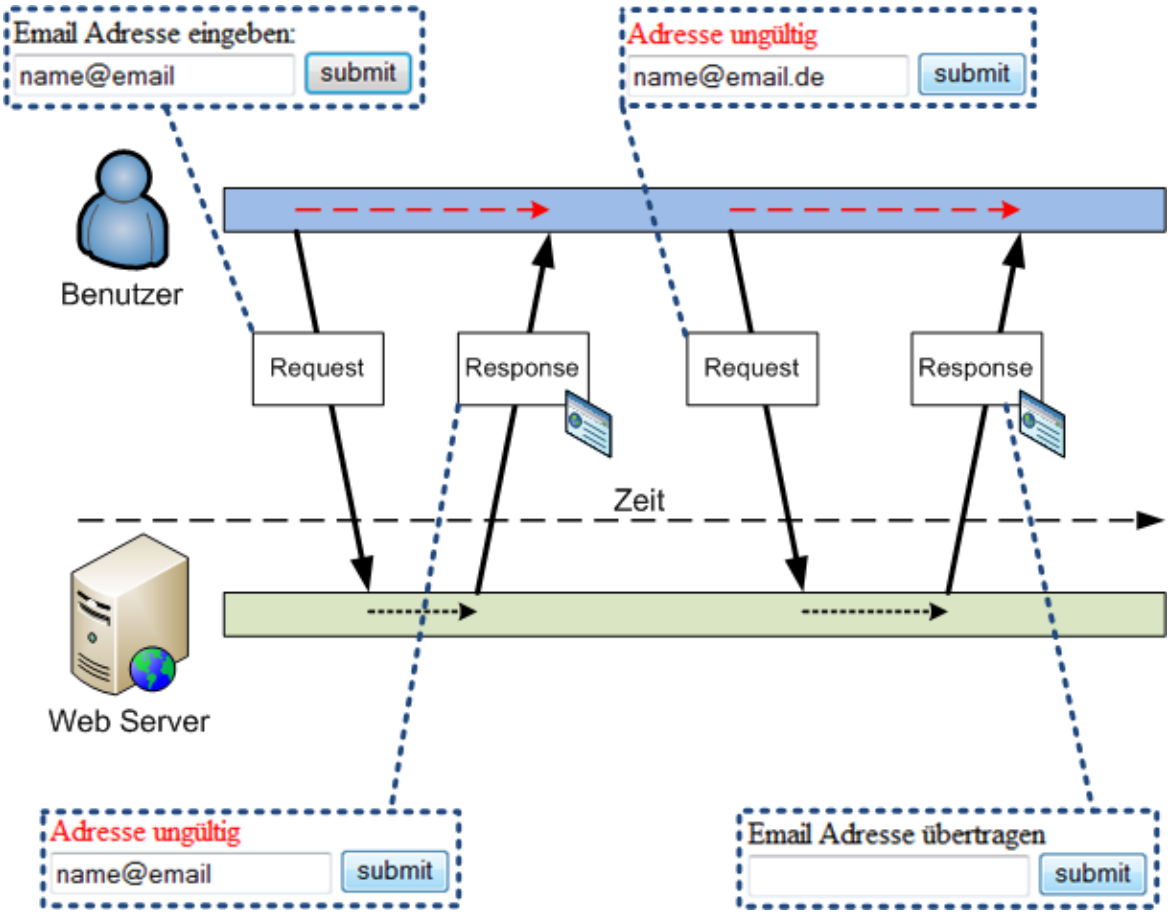


Abbildung 10: Klassischer Server-Roundtrip

lareingaben zu prüfen. Typische Fehler bei Eingaben konnten lokal abgefangen und ein unnötiger Server-Roundtrip vermieden werden. Typische Fehler sind die bereits erwähnten Rechtschreibfehler, Zahlendreher und leergelassenen Pflichtfelder. Die Abbildung 11 zeigt den veränderten Ablauf durch den Einsatz von JavaScript. Der Fehler wird bereits lokal abgefangen und der Benutzer hierüber informiert. Der Benutzer kann den Fehler korrigieren und es wird nur noch ein Server-Roundtrip benötigt.

Im den folgenden Jahren wurde JavaScript stets erweitert und ist zu einem mächtigen Werkzeug geworden. JavaScript kann inzwischen nahezu beliebige Veränderungen und Prüfungen auf der Webseite vornehmen, in die es eingebettet ist. JavaScript findet sich heute in allen großen Browsern wieder und ist der de Facto Standard, wenn es um lokales Scripting in Webanwendungen geht.

Java Script birgt allerdings auch einige Nachteile. Zum einen ist dies der Punkt der Si-

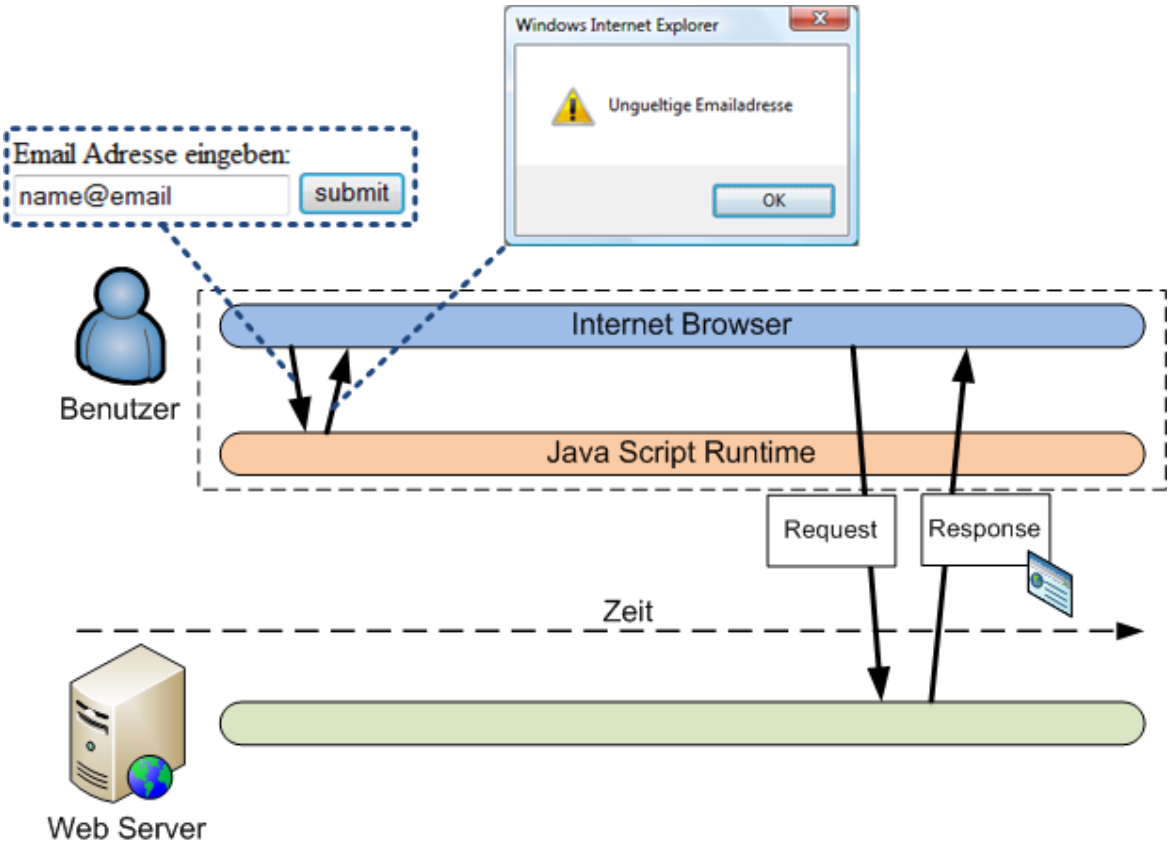
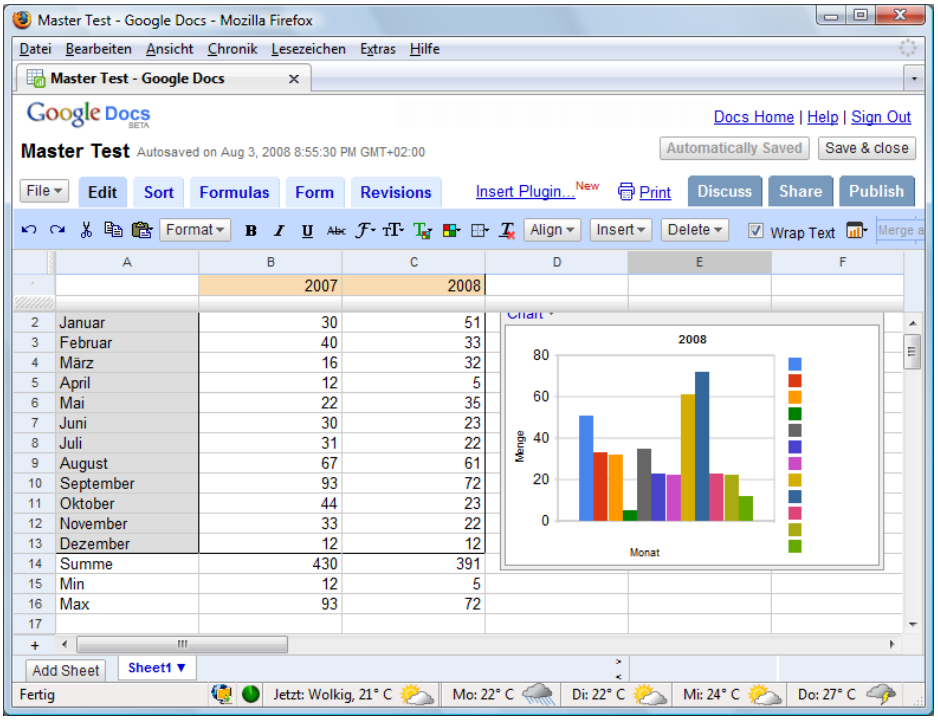


Abbildung 11: Einsatz von JavaScript zur Vermeidung von Server Roundtrips

cherheit. Sogenannte Cross-Side-Scripting Attacken (XSS) basieren auf den Webseiten untergeschobenen Java Script Fragmenten, die Unheil bei dem Webseitenaufrufer verursachen sollen. Durch solche Attacken werden primär Passwörter ausgespäht, weil Java Script Zugriff auf die Cookies des Browsers hat. Ein weiteres, großes Problem ist die mangelnde Barrierefreiheit (oder Accessibility) von Java Script. So können Vorlesesysteme nicht mit den dynamischen Veränderungen der Seite durch Java Script umgehen. Auch die Funktionalität des Zurück-Buttons ist bei Java Script nicht vorhanden oder muss selber in Java Script nachimplementiert werden.
Es bleibt zu erwähnen, dass Java Script aufgrund seines schlanken Objektmodells auch als Scriptsprache in anderen Bereichen eingebettet wird. Die weiteren Einsatzmöglichkeiten von Java Script sind jedoch nicht Teil dieser Arbeit.

2.3.4 Rich Internet Applications



	A	B	C
		2007	2008
2	Januar	30	51
3	Februar	40	33
4	März	16	32
5	April	12	5
6	Mai	22	35
7	Juni	30	23
8	Juli	31	22
9	August	67	61
10	September	93	72
11	Oktober	44	23
12	November	33	22
13	Dezember	12	12
14	Summe	430	391
15	Min	12	5
16	Max	93	72

Abbildung 12: Tabellenkalkulation als Rich Internet Application: Google Spreadsheet

Als Rich Internet Application (RIA) werden Webanwendungen bezeichnet, die einen Mehrwert gegenüber klassischen Webanwendungen bieten und dem Verhalten lokaler Anwendungen ähneln. Merkmale von RIAs sind Unterstützung von Tastaturkürzeln, Drag and Drop sowie eine konsistente Oberfläche ohne einen Neuaufbau der kompletten Seite bei Änderungen in Teilen.
Bei der Erstellung dieser Arbeit sind Ajax und Flash die Dominanten Technologien für die Erstellung von RIAs.
Mit RIAs werden immer mehr Desktop Anwendungen im Browser nachgebildet, wie beispielsweise Google Spreadsheet als Ersatz für eine lokal installierte Tabellenkalkulation (siehe Abbildung 12).

2.3.5 Ajax

Der Begriff Ajax (Asynchronous JavaScript and XML) wurde erstmals 2005 von Jesse James Garret in seinem Essay *Ajax: A New Approach to Web Applications* (AjaxApproach,

2005) erwähnt. Er beschreibt dort, wie in seinem Unternehmen Web-Anwendungen entwickelt werden, die auf dynamisch veränderten Seiten mittels Java Script und asynchronem Nachrichtenaustausch durch das XMLHttpRequest-Objekt ((XMLHttpRequest, 2007)) basieren.
Doch Garrets Unternehmen war nicht das einzige, dass diese Technologien verwendete: Google hatte bereits mit Google Suggest[8] und Google Maps[9] Ajax Applikationen ins Internet gestellt und auch A9[10] und Flickr[11] hatten Anwendungen auf Basis der Ajax-Technologie entwickelt, jedoch ohne es als solches zu bezeichnen.
Bis Ajax verwendet wurde, hatten Web-Anwendungen immer einen klaren Nachteil gegenüber lokalen Anwendungen. Lokale Anwendungen waren schneller, robuster, boten bessere Antwortzeiten und die Benutzbarkeit entsprach dem, was der Anwender gewohnt war. Einer der ersten Ansätze, dies zu ändern bot Microsoft mit seiner Webmail-Oberfläche Outlook Web Access. Aus dieser Entwicklung stammt auch das XMLHttpRequest, dass asynchrone Aufrufe aus dem Browser ermöglicht und ein entscheidender Bestandteil von Ajax ist.
Ajax ist keine neue Programmiersprache oder etwa ein Tool, sondern lediglich die Kombination bereits erprobter und robuster Werkzeuge aus dem Bereich der Webseiten Erstellung (aus (TowardsAjax, 2007)):

- The Hypertext Transfer Markup Languages: HTML, XHTML, XML
- Sprache zur Beschreibung der Darstellung eines Dokuments: Cascading Style Sheets (CSS)
- Interaktion und dynamische Darstellung: W3C Document Object Model (DOM).
- Client-seitige Sprache und Daten Validierung: Java Script
- Asynchrone Daten Übertragung und Kommunikation mit dem Server : XMLHttpRequest Objekt (XHR)
- Manipulation, Transformation und Daten Austausch: XML, XSLT, HTML, JSON, Plain Text
- Übertragungs Protokolle: HTTP, HTTPS

[8] http://www.google.com/webhp?complete=1 - Zugriffsdatum: 2008-09-04
[9] http://maps.google.com - Zugriffsdatum: 2008-09-04
[10] http://www.a9.com/; - Zugriffsdatum 2008-09-04
[11] http://www.flickr.com/ - Zugriffsdatum: 2008-09-04

- Server-seitige Sprachen: ASP.NET, Cold Fusion, JSF, JSP, Perl, PHP, Ruby und CGI Anwendungen

Der primäre Vorteil von Ajax liegt in der Kombination der Client-seitigen Sprache Java Script und der asynchronen Datenübertragung. Abbildung 13 veranschaulicht eine einfache Ajax Anwendung. Auf der Seite ist eine HTML-Tabelle dargestellt, die mithilfe eines Formulars um weitere Einträge erweitert werden kann. In diesem Beispiel muss der Anwender nicht auf den Server-Roundtrip warten, da der neue Eintrag mittels Java Script in die Tabelle hinzugefügt wird und die Übertragung der Daten asynchron im Hintergrund vorgenommen wird. Der Benutzer erhält so ein Benutzungserlebnis (User Experience), wie er es von lokalen Anwendungen gewohnt ist.

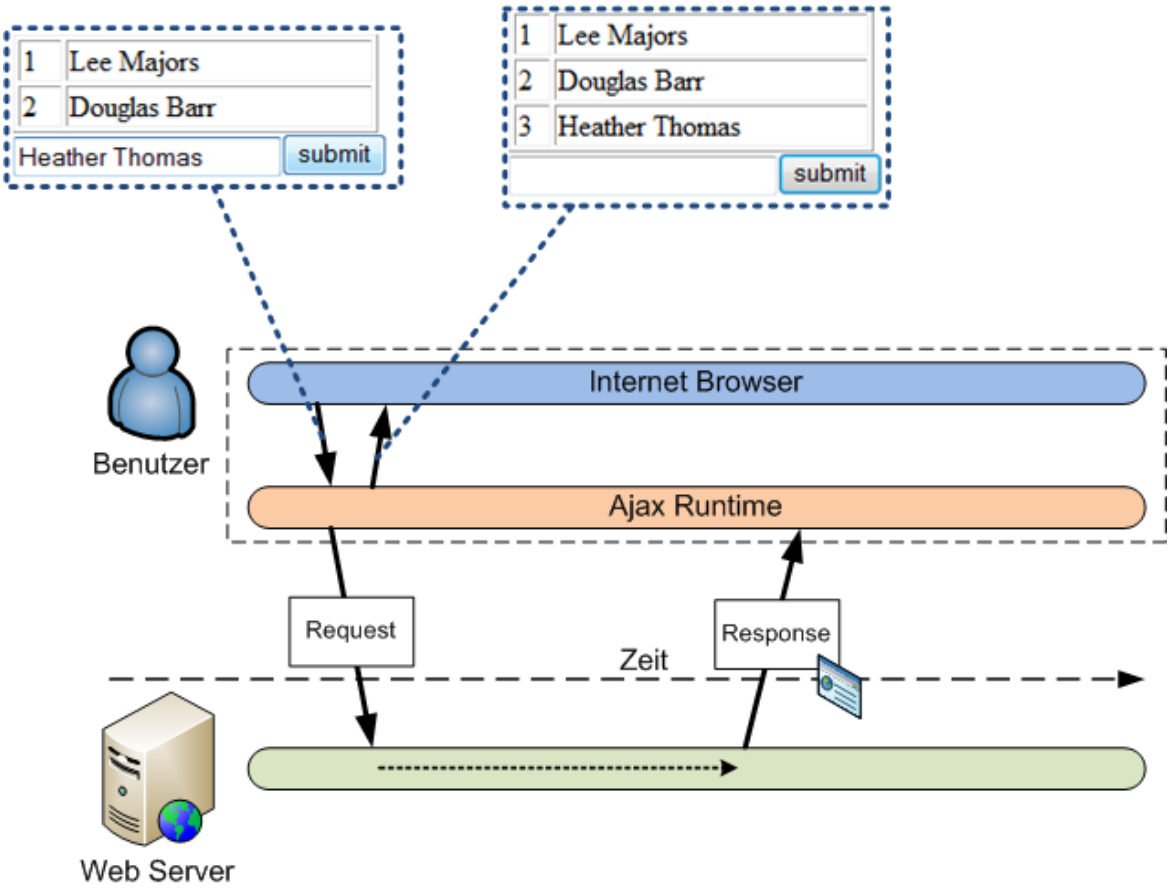


Abbildung 13: Asynchrone Aufrufe mit Ajax

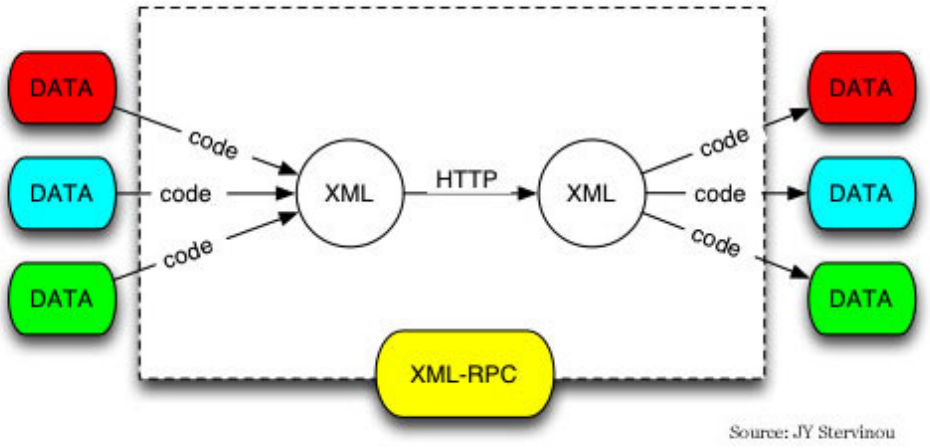


Abbildung 14: XML-RPC (Quelle: http://www.xmlrpc.com/)

2.3.6 XML-RPC

XML-RPC ist ein Protokoll für Interprozesskommunikation, basierend auf XML als Nachrichtenformat.
Durch Interprozesskommunikation können nebenläufig Prozesse, aus unterschiedlichen Adressbereichen und über die physikalischen Geräte Grenzen hinweg, kommunizieren. Bereits in den 1970er Jahren war die Kommunikation in verteilten System ein wichtiges Forschungs Thema (rfc707, 1976) und ist bis heute aktuell. Die wohl bekannteste RPC Implementierung ist die sehr schwergewichtige und komplexe Common Object Request Broker Architecture (CORBA). Eine neuere, auf Java basierende alternative ist JAVA Remote Methode Invocation (RMI).

Bei der Entwicklung der XML-RPC Spezifikation (xmlrpc, 1999) war das Ziel ein schlankes, offenes Protokoll zu schaffen. Bei XML-RPC wird ein entfernter Methodenaufruf durch einen HTTP-Request abgebildet, der als Payload ein XML Dokument mit dem eigentlichen Aufruf liefert. Abbildung 14 zeigt das Prinzip von XML-RPC: Anwendungen verpacken Daten in XML und versenden diese mit HTTP zu einem entfernten Prozess. Durch die Kombination von HTTP als Transportprotokoll und XML als Nachrichtenformat basiert XML-RPC auf offenen Standards.
Listing 1 veranschaulicht, wie simpel solch ein XML-RPC Aufruf ist. Auch ohne lesen der Spezifikation wird dem Leser sicher schnell klar, was für ein Ergebnis der anfragende Pro-

zess erwartet.

Besondere stärken hat XML-RPC vor allem bei interoperabler Kommunikation über das Internet. Aus diesem Grunde wundert es nicht, dass die meisten Web Anwendungen XML-RPC Schnittstellen anbieten. Viele dieser Anwendungen sind in Sprachen wie PHP, Ruby oder Python entwickelt worden. Da XML-RPC Bibliotheken für alle aktuellen Programmiersprachen existieren, können diese leichtgewichtigen Webanwendungen auch mit Anwendungen, die in Java, C# oder C++ entwickelt wurden, kommunizieren.

XML-RPC kommt im Rahmen der Beispielanwendung der Middleware (3.1), welche über XML-RPC mit einer PHP Anwendung kommuniziert, zum Einsatz.

```
1  POST /rpc/ HTTP/1.0
2  User-Agent: MyBrowser (Windows)
3  Host: example.org
4  Content-Type: text/xml
5  Content-length: 181
6
7  <?xml version="1.0"?>
8  <methodCall>
9      <methodName>math.sum</methodName>
10     <params>
11         <param>
12             <value>
13                 <int>40</int>
14             </value>
15         </param>
16         <param>
17             <value>
18                 <int>2</int>
19             </value>
20         </param>
21     </params>
22 </methodCall>
```

Listing 1: Eine XML-RPC Anfrage

2.3.7 Mobile Webanwendungen

The Web on the Move ist das Motto der W3C Mobile Web Initiative (W3cMobile). Diese Initiative soll die Entwicklung von Webseiten für mobile Anwendungen zu unterstützen. Neben diversen Anleitungen und Whitepapers bietet die Initiative auch einen *W3C mobileOK checker* zum Testen der eigenen Seiten an. Dieser Checker überprüft, ob die übermittelte Webseite den besonderen Anforderungen mobiler Webseiten gerecht wird. Beim Verfassen dieser Arbeit ist der Checker noch in einem frühen Beta Stadium und überprüft die Seite nur sehr rudimentär auf Fehler.

2.4 Gesicherte Webanwendungen

Der allgemeine Zugriff auf eine Webanwendung ist nur in den seltensten Fällen gewünscht. Üblicherweise ist eine Zugriffsbeschränkung auf bestimmte Anwender erforderlich. Der Umgang mit Zugriffsbeschränkungen muss auf offenen Standards basieren, damit die Anwendung trotz Zugriffsbeschränkung auf allen Plattformen verfügbar bleibt. Im Gegensatz zu den sehr komplexen Möglichkeiten der XML Webservices, bieten normale Webanwendungen nur eine sehr begrenzte Möglichkeit der Absicherung. Die beiden relevanten Technologien sind in RFC 2617 (rfc2617, 1999) definiert: *Basic* und *Digest Access Authentication*. Beide basieren darauf, Credentials als zusätzliche HTTP Header an den Server zu übermitteln.

2.4.1 Basic Access Authentication

Bei Basic Access Authentication handelt es sich um eine sehr rudimentäre Benutzername-Passwort Übertragung.
Für ein Beispiel soll eine gesicherte Webanwendung den Zugriff für den Benutzer *arthur* mit dem Passwort *marvin* erlauben. Für den Zugriff müssen zunächst Benutzername und Passwort mit einem Doppelpunkt konkateniert werden:

```
arthur:marvin
```

Im folgenden muss diese Darstellung mit base64 kodiert werden. Diese Kodierung bewirkt, dass sämtliche Sonderzeichen und vor allem Zeilenumbrüche keine Probleme bei der Übertragung verursachen, da die Darstellung auf 8 Bit reduziert wird. Die konkatenierten Zugriffsdaten aus dem Beispiel haben base64 kodiert folgende Darstellung:

```
YXJ0aHVyOm1hcnZpbg==
```

Die erzeugte base64 Darstellung wird dann in den HTTP Header des Aufrufs eingefügt und so dem Server übermittelt:

```
1 GET /secured/secret.html HTTP/1.0
2 Host: example.org
3 Authorization: Basic YXJ0aHVyOm1hcnZpbg==
```

Listing 2: Basic Access Authentication

Nachteil ist hier, dass base64 keine Verschlüsselung, sondern lediglich eine umkehrbare Kodierung ist. Dies ist auch das Problem von Basic Access Authentication, die Credentials werden im Klartext übertragen. Basic Access Authentication sollte folglich nur in Verbindung mit einer zusätzlichen Übertragungsverschlüsselung wie beispielsweise SSL eingesetzt werden.

2.4.2 Digest Access Authentication

Aufgrund der Unsicherheit der Basic Access Authentication wurde die Digest Access Authentication in RFC 2069 definiert (rfc2069, 1997) und später in RFC 2617 (rfc2617, 1999) erweitert. Wichtigstes Merkmal ist, dass die Credentials nicht im Klartext sondern gehasht übertragen werden und ein Nonce die Gefahr eines Wiedereinspielungs Angriffs verringert. Im RFC wird jedoch auch klargestellt, dass Digest Access Authentication keine wirkliche Sicherheit bringt und eine wirklich effektive Absicherung nur über etwas wie Kerberos oder eine PKI möglich wird; jedoch:

> Nevertheless it is better than nothing, better than what is commonly used with telnet and ftp, and better than Basic authentication.

Da Digest Access Authentication von nahezu allen Browsern und Webservern unterstützt wird, ist diese Technologie sehr verbreitet und deshalb im Besonderen Relevant für diese Arbeit. Eine Middleware, die verzögert Nachrichten an eine geschützte Webanwendung versendet, muss Digest Access Authentication unterstützen. Aus diesem Grunde wird folgend ein kompletter Ablauf dargestellt.

```
1 challenge        =  "Digest" digest-challenge
2
3       digest-challenge  = 1#( realm | [ domain ] | nonce |
4                           [ opaque ] |[ stale ] | [ algorithm ] |
5                           [ qop-options ] | [auth-param] )
6
7
```

```
8       domain              = "domain" "=" <"> URI ( 1*SP URI ) <">
9       URI                 = absoluteURI | abs_path
10      nonce               = "nonce" "=" nonce-value
11      nonce-value         = quoted-string
12      opaque              = "opaque" "=" quoted-string
13      stale               = "stale" "=" ( "true" | "false" )
14      algorithm           = "algorithm" "=" ( "MD5" | "MD5-sess" |
15                            token )
16      qop-options         = "qop" "=" <"> 1#qop-value <">
17      qop-value           = "auth" | "auth-int" | token
```

Listing 3: WWW-Authenticate Response Header Definition

Ist eine Webanwendung mit Digest Access Authentication gesichert, liefert der Server einen Response mit dem Status `401` (*Unauthorized*) und einem WW-Authenticate Header, der definiert, in welcher Form der anfragende Client die Credentials zu liefern hat. Listing 3 zeigt die Definition des WWW-Authenticate Headers in EBNF aus dem RFC 2069 (rfc2069, 1997).

Besonders relevant ist der `nonce` Wert. Hierbei handelt es sich um einen serverseitig generierten Pseudo-Zufallswert, der typischerweise base64 kodiert ist. Dieser Wert wird bei jedem Aufruf neu generiert und vom Client unverändert an den Server mit dem nächsten Response zurückgesendet. Der RFC ist an dieser Stelle sehr unverbindlich formuliert, es muss jedoch davon ausgegangen werden, dass der nonce Wert an eine IP gebunden und nur zeitlich begrenzt gültig ist.

Neben dem nonce Feld ist auch noch das `algorithm` Feld relevant. Der Server gibt so vor, welcher Algotithmus zum hashen verwendet werden soll. Im RFC wird als Standard MD5 (Message-Digest Algorithm 5) definiert (rfc1321, 1992), andere Algorithmen können jedoch auch verwendet werden.

Ein konkreter Header ist in Listing 4 dargestellt.

```
1  HTTP/1.0 401 Unauthorised
2  Server: MyHttp/0.1
3  Date: Sun, 18 Aug 2008 13:16:23 CET
4  WWW-Authenticate: Digest realm="user@example.org",
5                           nonce="c8235bcc6d0ef0ef14eee5660279b0bc",
6                           opaque="7ff4c5548549ea211a1c63d7a8fd1722"
7  Content-Type: text/html
8  Content-Length: 171
9
10 [..]
```

Listing 4: WWW-Authenticate Response Header

Der Client muss dann einen erneuten Aufruf starten, mit den Credentials in der geforderten Form. Typischerweise würde ein Webbrowser ein Fenster öffnen, in dem der Realm Wert ausgewiesen und der Anwender aufgefordert wird, Benutzername und Passwort einzugeben.
Das RFC sieht vor, dass der Client die Credentials in einem zusätzlichen `Authorization` Header mitliefert. Die Definition dieses Headers ist in Listing 5 gegeben.

```
1  Authorization    = "Authorization" ":" "Digest" digest-response
2  digest-response  = 1#( username | realm | nonce | digest-uri
3                  | response | [ algorithm ] | [cnonce] |
4                  [opaque] | [message-qop] |
5                      [nonce-count]  | [auth-param] )
6
7  username         = "username" "=" username-value
8  username-value   = quoted-string
9  digest-uri       = "uri" "=" digest-uri-value
10 digest-uri-value = request-uri   ; As specified by HTTP/1.1
11 message-qop      = "qop" "=" qop-value
12 cnonce           = "cnonce" "=" cnonce-value
13 cnonce-value     = nonce-value
14 nonce-count      = "nc" "=" nc-value
15 nc-value         = 8LHEX
16 response         = "response" "=" request-digest
17 request-digest = <"> 32LHEX <">
18 LHEX             =  "0" | "1" | "2" | "3" |
19                     "4" | "5" | "6" | "7" |
20                     "8" | "9" | "a" | "b" |
21                     "c" | "d" | "e" | "f"
```

Listing 5: Authorization Request Header Definition

Hier im Besonderen Relevant ist das Feld `request-digest`. In diesem überliefert der Client die Credentials dem Server. Die Definition von `response-digest` ist in Listing 5 nur syntaktisch. Wie dieser Wert berechnet wird, ist in Listing 6 beschrieben.

```
1  request-digest  = <"> < KD ( H(A1),     unq(nonce-value)
2                                          ":" nc-value
3                                          ":" unq(cnonce-value)
```

```
4                                          ":" unq(qop-value)
5                                          ":" H(A2)
6                                    ) <">
```

Listing 6: response-digest und entity-digest Definition

Lautet die *algorithm* Direktive „*MD5*“ oder ist diese nicht gesetzt, dann ist A1 wie folgt definiert:

```
A1 = unq(username-value) ':' unq(realm-value) ':' passwd
```

Ist algorithm als „MD5-sess“ deklariert, so wird A1 nur einmal pro Sitzung berechnet.

```
A1 = H( unq(username-value) ':' unq(realm-value)
':' passwd )
':' unq(nonce-value) ':' unq(cnonce-value)
```

Ist die `qop` Direktive *auth*, dann ist A2 wie folgt definiert:

```
A2 = Method ':' digest-uri-value
```

Ist `qop` hingegen *auth-int*, dann ist A2 wie folgt definiert:

```
A2 = Method ':' digest-uri-value ':' H(entity-body)
```

Der so berechnete Hash Wert wird in den Authorization Header eingebettet und an den Server übertragen, der bei einem validen Wert, die angeforderte Datei zurückliefert. Durch dieses Verfahren muss das Passwort nicht mehr übermittelt werden, lediglich der Hash Wert, der keinen Rückschluss auf das Passwort zulässt.
Listing 7 zeigt die konkrete Anfrage des Clients mit Authorization Header.

```
1  GET /secured/secret.html HTTP/1.0
2  Host: example.org
3  Authorization: Digest username="arthur",
4                        realm="user@example.com",
5                        nonce="c8235bcc6d0ef0ef14eee5660279b0bc",
6                        uri="/secured/secret.html",
7                        qop=auth,
8                        nc=00000001,
9                        cnonce="012c43fa",
10                       response="ec78cc6e88fdff5cd0eb948193f7a9ae",
11                       opaque="7ff4c5548549ea211a1c63d7a8fd1722"
```

Listing 7: Konkrete Anfrage eines Clients mit Authorization Header

2.5 Web Services

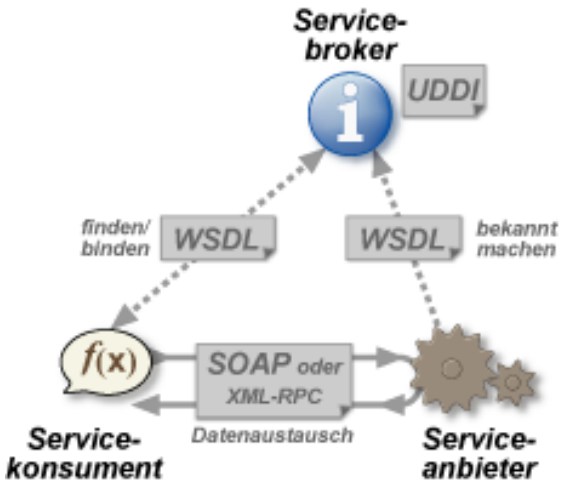


Abbildung 15: Übersicht der Infrastruktur für Web Services

Ein Web Service ist ein Dienst, der über eine standardisierte XML Schnittstelle und einen eindeutigen URI im Netzwerk verfügbar gemacht wird. Da Web Services XML als Nachrichtenformat verwenden, sind sie plattformunabhängig und interoperabel. Web Services werden primär für die Kommunikation von Anwendungen über ein Netzwerk eingesetzt.

Web Services basieren primär auf drei Standards:

- UDDI ist ein von der OASIS standardisierter (uddiv3, 2004) *Service Broker*. Dieser übernimmt die Rolle eines Verzeichnisdienstes. Web Services können ihren Dienst publizieren. Anwendungen können Dienste suchen und diese anhand der erhältlichen Informationen aufrufen. Durch den Service Broker wird es möglich, dynamisch Web Services aufzufinden und in eine Anwendung einzubinden.
- WSDL ist ein W3C Standard (wsdl2, 2007) für die Beschreibung eines Web Services. Der Web Services verwendet dieses Format, um sich bei dem Service Broker zu registrieren, der Client nutzt die Daten, um den Service aufzufinden.

- Als Protokoll für die Kommunikation zwischen einem Client und dem Web Service wird das vom W3C standardisierte SOAP (soap, 2007) oder XML-RPC (siehe Abschnitt 2.3.6) verwendet.

Auf diesen drei Standards aufbauend, existieren diverse erweiternde Standards, wie beispielsweise WS-* für Sicherheits Aspekte.

Web Services sind zum einen sehr mächtig, zum anderen aber durch die diversen Standards sehr komplex. Dieser Kritikpunkt wirkt sich besonders auf Plattformen mit begrenzten Ressourcen negativ aus und hat zur Entwicklung von Alternativen beigetragen. REST ist eine schlankere Alternative zu Web Services und wird im folgenden Kapitel vorgestellt.

2.6 Representational State Transfer

Als Representational State Transfer (REST) wird ein Software Architekturstil bezeichnet. REST wurde erstmals von Roy Fielding in seiner Dissertation *Architectural Styles and the Design of Network-based Software Architectures* (rest, 2000) beschrieben. Bei REST stehen Ressourcen, welche per HTTP über URIs adressiert werden, im Mittelpunkt. REST Services, welche Ressourcen anbieten, sollen zustandslos sein. REST unterstützt die HTTP Operationen, meist jedoch beschränkt sich dies auf `GET`, `POST` und `PUT`.
Wie Werte von REST Services zurückgeliefert werden, ist nicht definiert. Die Antwort eines REST Dienstes kann eine einfache Zeichenkette, eine Datei oder auch ein XML Fragment, wie er von einem Web Service zurückgeliefert wird. Weiterhin ist nicht definiert, wie der Client von den Anforderungen des Dienstes erfährt.
REST besticht durch seine Einfachheit und ist wesentlich schlanker als beispielsweise die im Kapitel zuvor beschriebenen Web Services, die mit WSDL und SOAP einen Overhead erfordern, der bei REST nicht notwendig ist. REST ist ein bewusst minimalistischer Ansatz, der bei weitem nicht den Umfang von XML-Web Services bieten will und kann (wie beispielsweise die Dienstebeschreibung oder die Sicherheitsstandards WS-*).

Eine per REST Angebotene Ressource wird über einen eindeutigen URI adressiert. Im Gegensatz zu Web Services erfolgt der komplette Adressierung über die URI. Bei einem Web Service Aufruf, muss zunächst der Web Service gebunden werden und anschließend ein SOAP Dokument für den Aufruf übergeben werden. Das folgende Beispiel soll den Unterschied zwischen den beiden Arten der Kommunikation verdeutlichen.

Beispiel 1: Web Service:

Zunächst benötigt der Client eine WSDL Datei, welche die detaillierten Informationen zum Aufruf des Web Services enthält. Diese lädt der Client vom Server herunter, wertet diese aus. Nun kann der Client auf Basis der Service Beschreibung in dem WSDL Dokument eine Nachricht in dem geforderten Format und den entsprechenden Parametern an den Web Service senden. Listing 8 zeigt die Nachricht an einen Währungs-Umrechnungs Service.

```
1  <?xml version='1.0' encoding='UTF-8'?>
2  <env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
3                xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/
                     addressing"
4                xmlns:ns="urn:currency-echange-example">
5      <env:Header>
6          <wsa:To>http://example.org/CurrencyExchange</wsa:To>
7          <wsa:Action>http://example.org/CurrencyExchange/Infos</wsa:Action
               >
8      </env:Header>
9      <env:Body>
10         <ns:getCurrencyExchange>
11             <ns:fromCurrency>EUR</ns:symbol>
12             <ns:toCurrenncy>USD</ns:symbol>
13             <ns:value>3.14</ns:symbol>
14         </ns:getCurrencyExchange>
15     </env:Body>
16 </env:Envelope>
```

Listing 8: SOAP Nachricht an einen Web Service

Anhand des Beispiels wird der Protokoll Overhead von Web Services deutlich. Selbst bei diesem minimalen Beispiel ist die Größe der Anfrage (ohne die vorausgegangene WSDL Abfrage und ohne die HTTP Abfrage selbst) über 2 Kilobyte.

Beispiel 2: REST

Bei REST existiert keine dynamische Bindung über ein beschreibendes Dokument wie WSDL bei Web Services. REST Dienste verzichten bewusst auf diese Flexibilität und vermeiden so den Overhead der durch das Abrufen und Auswerten einer Dienstbeschreibung verursacht wird.

Da die Abfrage und Auswertung eines beschreibenden Dokumentes entfällt, wird bei REST der Dienst mit allen Parametern direkt über die URI aufgerufen (restful, 2007). Da die Parameter in die URI kodiert werden, wird im Gegensatz zu einem Web Service Aufruf keine separate Nachricht benötigt. Dies reduziert vereinfacht nicht nur den Aufruf, es reduziert

auch die zu übertragenden Daten.

Listing 9 zeigt den Aufruf eines exemplarischen REST Services. Wie beim vorherigen Beispiel auch, rechnet dieser Währungen um.

```
1 GET /services/currency/us/eur/3.14
```

Listing 9: Aufruf eines REST Services

Anhand des Beispiels wird deutlich, dass die zu übertragenden Daten bei einem REST Service wesentlich geringer sind, als bei einem Web Service Aufruf.

Die Vorteile von REST erhalten in Umgebungen mit besonders beschränkten Ressourcen, wie es bei mobilen Plattformen der Fall ist, noch mehr an Bedeutung. Durch REST wird die Anzahl der zu übertragenden Daten reduziert. Weiterhin entfällt das Parsen und Auswerten eines Beschreibungsdokumentes für einen Services.

2.7 JavaScript Object Notation

JSON (sprich „Jason“) ist ein Datenaustausch Format. JSON bietet ähnlich wie XML die Möglichkeit, Werte zu definieren und diese zwischen Anwendungen oder Anwendungsteilen auszutauschen (intrJason, 2008). Im Gegensatz zu XML ist JSON kein allgemeingültiger Ansatz. JSON beschränkt sich darauf, Werte in simplen Strukturen abzubilden. In JSON ist kein Schema als Formatsdefinitionen erforderlich. Viel mehr existiert für JSON eine Formatsdefinition, die typische Datenstrukturen entspricht.

Syntaktische Grundlage aller JSON Objekte ist eine Menge kommaseparierter

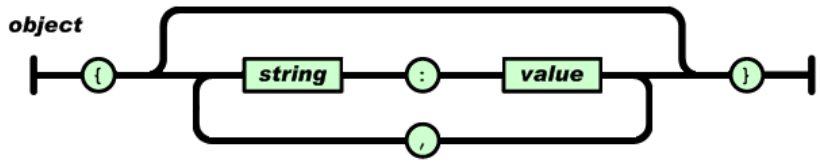


Abbildung 16: JSON - Object (Quelle: json.org

`Schlüssel:Wert` Paare in geschweiften Klammern, wie es in Abbildung 16 definiert ist. Dies entspricht einer Datenstruktur, die als Assoziatives Array bezeichnet wird und in den meisten Programmiersprachen als Hashmap realisiert ist. Das Assoziative Array grenzt

sich von einem Array dadurch ab, dass die Werte nicht über fortlaufend nummerierte Indizes angesprochen werden, sondern über Zeichenketten Schlüssel. Wie die Zielsprache dies konkret umsetzt, ist für JSON nicht relevant.

Weiterhin bietet JSON die Möglichkeit, als Wert eine Menge von indizierbaren Werten

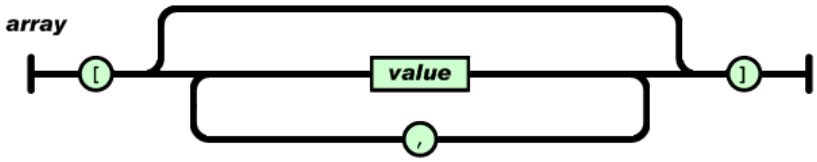


Abbildung 17: JSON - Object (Quelle: json.org

abzubilden. Dies entspricht einer Array Datenstruktur und wird auch in JSON als solche bezeichnet. Ein Array wird in JSON syntaktisch durch eine kommaseparierte Liste von Werten in eckigen Klammern dargestellt (siehe Abbildung 17). Ein Wert kann ein literaler Wert, ein boolescher Wert, ein JSON Objekt, wiederum ein Array oder ein Nullwert sein (siehe Abbildung 18)

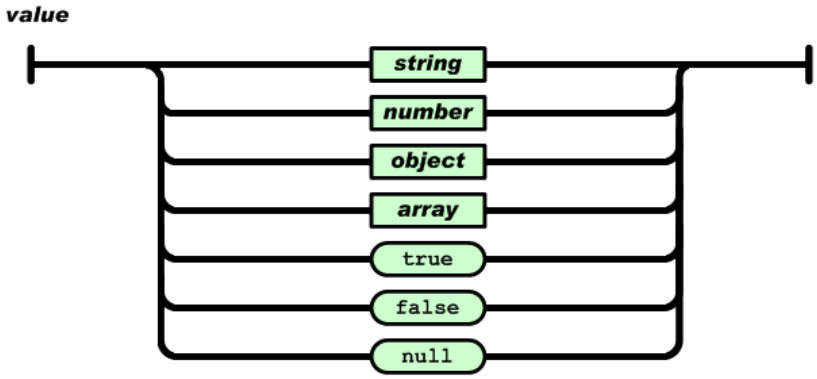


Abbildung 18: JSON - Value (Quelle: json.org

Listing 10 zeigt eine konkrete Darstellung eines JSON Objektes. Die ersten drei Wertepaare (Typ, Titel, Hochschule) sind einfache Wertepaare. Es folgt ein eingebettetes Objekt (Autor) und ein Array (Stichworte). Mit diesem Beispiel ist bereits der relevante Teil der JSON Syntax dargestellt.

Neben der Lesbarkeit beruht der Erfolg von JSON auf der einfachen und performanten Handhabe der Datenobjekte mit JavaScript.

Inzwischen bieten nahezu alle aktuellen Sprachen JSON Implementierungen, so wird mittels JSON eine interoperable Kommunikation möglich. Das JS in JSON (für JavaScript) ist folglich nicht präzise, sondern historisch bedingt.

Im Rahmen dieser Arbeit wird JSON für den Datenaustausch zwischen den Code on Demand Anwendungen und der Middleware verwendet und ist Basis des APIs der Middleware.

```
{
  "Typ"         : "Master Thesis",
  "Titel"       : "Entwicklung einer Messaging Middleware für Code on
                          Demand Anwendungen auf mobilen Plattformen",
  "Hochschule" : "Hochschule für Angewandte Wissenschaften Hamburg",
  "Autor"       : {
    "Name"        : "Falkenberg",
    "Vorname"     : "Eike",
  },
 "Stichworte"   : [
      "Mobile Computing",
      "Code on Demand",
      "Middleware"
    ]
  "Abgabe"      : "2008-09-20",
}
```

Listing 10: Lesbare Darstellung eines JSON Objekts

2.8 Existierende Lösungsansätze

Im Umfeld der im Rahmen dieser Arbeit entwickelten Middleware existieren bereits einige Lösungsansätze. Keine der Lösungsansätze deckt die in der Zielsetzung definierten Anforderungen.
Die Lösungsansätze sind grob in zwei Gruppen zu unterteilen. Ein Teil der Menge versucht, Code on Demand Anwendungen lokal auszuführen mit dem Fokus auf Desktop PCs. Die zweite Gruppe bilden Laufzeitumgebungen auf mobilen Endgeräten, die versuchen die Gerätevielfalt zu kapseln.
Einige der Lösungsansätze werden im folgenden Abschnitt, mit einer Bewertung hinsichtlich der Zieldefinition dieser Arbeit, vorgestellt.

2.8.1 (Google) Gears

Gears[12] bietet die Möglichkeit, speziell für Gears entwickelte JavaScript Anwendungen offline betreiben zu können. Gears wird primär von Google Inc. entwickelt, wurde jedoch unter die BSD-Lizenz gestellt und ist somit Open Source.
Gears besteht aus drei Kernkomponenten:

- Bedingte Ausführung von JavaScript Anwendungen auch offline
- Zugriff auf eine lokale SQLite Datenbank
- Datensynchronisation von einer entfernten Datenbank auf die lokale SQLite

Der Fokus von Gears liegt klar in der offline Funktionalität von JavaScript Anwendungen auf Desktop Computern. Andere Technologien wie beispielsweise Adobe Flash werden nicht unterstützt. Der mobile Kontext wird nicht berücksichtigt und ein erweiterter Hardware Zugriff wird nicht geboten.
Der Autor geht davon aus, dass Google im Zuge der Entwicklung seiner mobilen Plattform Android (2.1.2.2) auch vermehrt Gears für mobile Endgeräte entwickeln wird.

2.8.2 Adobe Integrated Runtime (AIR)

Mit der Adobe Integrated Runtime (AIR)[13] bezeichnet Adobe eine Laufzeitumgebung für RIAs auf dem Desktop. Es wird kein Browser benötigt und die RIAs laufen auf dem Desktop wie lokal installierte Anwendungen. Die Abkürzung hat somit auch einen Sinn: AIR rückwärts wird wieder RIA und AIR bringt die RIAs wieder zurück auf den Desktop.
AIR ist derzeit nur für Desktop Anwendungen verfügbar, es ist nicht bekannt, ob eine mobile Version geplant ist. AIR bietet keine offline Funktionalität und keinen Hardware Zugriff.

2.8.3 Mozilla Prism

Das Prism Projekt der Mozilla Foundation[14] fährt einen ähnlichen Ansatz wie Adobe mit AIR: Webanwendungen laufen als lokal installierte Anwendungen. Bei Prism wird für jede Anwendung eine eigene SQLite Datenbank angelegt, auf die diese zugreifen kann. Der mobile Kontext wird bei Prism nicht berücksichtigt, auch offline Funktionalität und erweiterter Hardware Zugriff fehlen.

[12] http://gears.google.com/ - Zugriffsdatum: 2008-08-17
[13] http://www.adobe.com/go/air - Zugriffsdatum: 2008-08-17
[14] http://labs.mozilla.com/2007/10/prism/ - Zugriffsdatum: 2008-08-17

2.8.4 Mojax

Mojax [15] bezeichnet sich als ein Ajax Anwendungs Framework für mobile Endgeräte. Mojax Anwendungen laufen nicht in einem Browser, sondern in einer speziellen Applikation und haben beschränkten Zugriff auf lokale Ressourcen und sind offline fähig. Auch wenn der Name Mojax auf mobile Ajax Applikationen schließen lässt, ist dies nicht exakt der Fall. Mojax weicht in nahezu allen Bereichen von Standards ab, Kritiker[16] definieren Mojax daher als

> client-side application runtime that uses „their own flavor of XML to describe UI“ + „their own XML to describe other application metadata“ + „their own flavor of ECMAScript“

Mojax bietet beschränkten Hardware Zugriff und offline Funktionalität ohne Nachrichten Replikation auf mobilen Endgeräten. Nachrichten Replikation ist bei Mojax nicht vorgesehen.

2.8.5 Yahoo Go!

Yahoo Go![17] ist eine Anwendungsplattform für mobile Endgeräte. Yahoo Go! Anwendungen, werden als Widgets bezeichnet. Widgets für Yahoo Go! werden in Blueprint, einem XML Dialekt deklariert. Ein lokal installierter Client rendert die Anwendung und führt sie aus. Yahoo Go! unterstützt derzeit weder Hardware Zugriff, noch offline Funktionalität.

2.8.6 HTML 5

Im aktuellen Entwurf der nächsten HTML Spezifikation (html5, 2008) sind Persistierungskonzepte zu finden, die eine lokale Datenspeicherung ermöglichen. Nach Ansicht des Autors ein richtiger Schritt, jedoch ist fraglich, ob sich die Browser an diesen Standard halten werden, bzw. das neue API implementieren. Die Fertigstellung der neuen HTML Spezifikation wird von der Web Hypertext Application Technology Working Group nicht vor 2012 erwartet (html5Finish, 2008).
Zugriffe auf lokale Ressourcen sind auch künftig nicht vorgesehen.

[15] http://mojax.mfoundry.com/display/mojax/Main+Page - Zugriffsdatum: 2008-08-17

[16] http://weblog.cenriqueortiz.com/mobility/2006/12/27/about-mobile-ajax-or-mojax-is-not-mobile-ajax/ - Zugriffsdatum: 2008-08-17

[17] http://de.mobile.yahoo.com/go - Zugriffsdatum: 2008-08-17

3 Analyse

Wie in Abschnitt zur Motivation (1.1) erläutert, können Code on Demand Anwendungen viele aktuelle Probleme auf mobilen Plattformen lösen. Die aus dem Einsatz von Code on Demand Anwendungen resultierenden Einschränkungen, wie der Mangel an offline Funktionalität, fehlender Hardwarezugriff und die Beschränkung auf transienten Speicher sind nach Ansicht des Autors nur durch eine zusätzliche, lokal installierte Middleware zu handhaben.
In diesem Kapitel wird herausgearbeitet, welche Anforderungen an solch eine Middleware gestellt werden und wie sich diese auf den Entwurf auswirken.

Im ersten Schritt wird eine Beispielanwendung definiert (3.1). Die Beispielanwendung hat zum einen den Nutzen, dass die praktischen Anforderungen solch einer Anwendung klarer herausgearbeitet werden können. Darüber hinaus dient sie dem Leser als Visualisierung der Möglichkeiten der Middleware.
Als nächstes folgt die Betrachtung der Meta-Architektur (3.2), in der auch das Umfeld der Middleware relevant ist. Nach der Betrachtung der Middleware „von Weitem“ ist die Betrachtung der Anwendungsfälle (3.3) der nächste Schritt der Annäherung. Diese stellen das erwartete Verhalten und die Akteure des Systems dar.
Anschließend beginnt die Analyse der Definition der Anforderungen. Diese werden separiert in funktionale Anforderungen (3.4), welche den Leistungsumfang der Software definieren und nichtfunktionale Anforderungen (3.5), die die Qualität der Software beschreiben.

3.1 Beispielanwendung

Um ein besseres Gefühl für den Umfang und die Leistungsfähigkeit der Middleware zu bekommen, wird nun eine Beispielanwendung definiert. Die Beispielanwendung soll exemplarisch die Verwendung der Middleware aus Code on Demand Anwendungen aufzeigen. Die hier vorgestellte Beispielanwendung wird im Verlauf der Prototyp Entwicklung (5) als Proof of Concept dienen.

Eines der bekanntesten Elemente des Web 2.0 Paradigmas sind virtuelle Tagebücher, sogenannte *Blogs*. Die Beispielanwendung soll die Möglichkeit bieten, über eine Weboberfläche Blogeinträge zu erfassen und an das Blog zu senden. Ohne den Einsatz der Middleware macht solch eine Anwendung nur beschränkt Sinn, da ein Blog für gewöhnlich bereits eine Webanwendung für eben diesen Zweck mitliefert. Wordpress[18] als einer der

[18] http://www.wordpress.org; Zugriffsdatum: 2008-07-20

größten Anbieter von blogging Software bietet eine umfangreiche Backend Anwendung zum verfassen von Blog Einträgen (siehe Abbildung 19). Durch den Einsatz der Middleware

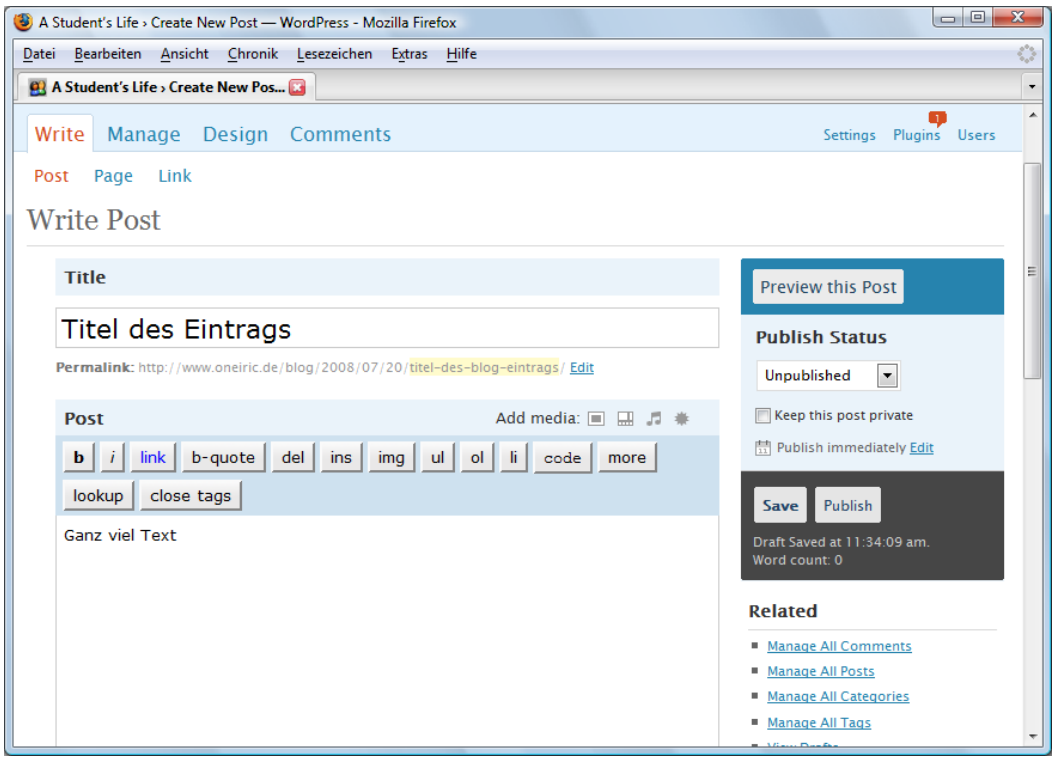


Abbildung 19: Wordpress Oberfläche zum verfassen eines Blog Eintrags

bekommt der Verfasser von Blog Einträgen im mobilen Kontext einen deutlichen Mehrwert. Der Anwender erhält die Möglichkeit Beiträge offline zu verfassen und Fotos der integrierten Kamera in seine Beiträge einzubeziehen. Auch das Ermitteln und Einbeziehen der aktuellen Position über ein GPS Modul wird durch die Middleware möglich. Das Resultat könnte wie in Abbildung 20 aussehen.

Damit die Beispielanwendung einen offline erfassten Beitrag an die Blog Software senden kann, wird eine Server Komponente oder Schnittstelle benötigt. Im Beispiel Wordpress ist dies mit Hilfe des Blogger APIs (bloggerApi, 2008) über XML-RPC (2.3.6) möglich. Die Anwendung muss den XML-RPC Aufruf vorbereiten, die Nachricht kann wie jede andere Nachricht später versendet werden. Listing 11 zeigt eine Nachricht.

Die Beispielanwendung zeigt die wichtigsten Möglichkeiten der Middleware: der Anwender kann offline Inhalt erfassen, auf lokale Ressourcen zugreifen (hier Kamera und GPS) und die Daten werden asynchron an den Server übertragen.

Abbildung 20: Blog Eintrag mit Middleware Beispielanwendung

Das Szenario ist typisch für mobile Anwendungen, die auch offline funktionieren müssen. Denkt man beispielsweise an eine Software für einen Schadens-Gutachter, so würde diese sehr ähnliche Funktionen erfordern. Ein Gutachter muss vor Ort Fotos von einem Schaden machen können, die Möglichkeit haben einen Kommentar zu verfassen, der Ort muss festgehalten werden und die gesammelten Daten müssen an den Firmenserver übertragen werden. Dies ist exakt dieselbe Funktionalität, wie ihn die Beispielanwendung bietet.

```
<entry xmlns='http://www.w3.org/2005/Atom'>
  <title type='text'>Hello World!</title>
  <content type='xhtml'>
    <div xmlns="http://www.w3.org/1999/xhtml">
      <p>Mobile Computing rocks!</p>
    </div>
  </content>
  <category scheme="http://www.blogger.com/atom/ns#" term="Mobile" />
  <category scheme="http://www.blogger.com/atom/ns#" term="Master" />
  <author>
    <name>Eike</name>
    <email>eike@domain.org</email>
    <uri>http://www.domain.org/eike</uri>
  </author>
</entry>
```

Listing 11: Blog Eintrag erstellen mittels XML-RPC Aufruf über Blogger API

3.2 Meta Architektur

Das Schaubild 21 zeigt das Umfeld der Middleware. Im linken Teil des Schaubildes ist ein mobiles Endgerät mit installierter Middleware, auf der rechten Seite ist ein mobiles Endegerät mit einer klassischen Code on Demand Anwendung zu sehen. Beiden gleich ist, dass die Code on Demand Anwendung in einem Browser ausgeführt wird, im linken Teil wird jedoch die Middleware einbezogen und die Code on Demand Anwendung erhält Zugriff auf lokale Ressourcen.
Anhand der Abbildung wird deutlich, dass die Einführung der Middleware Lösung im Online Kontext für den Webserver transparent ist.

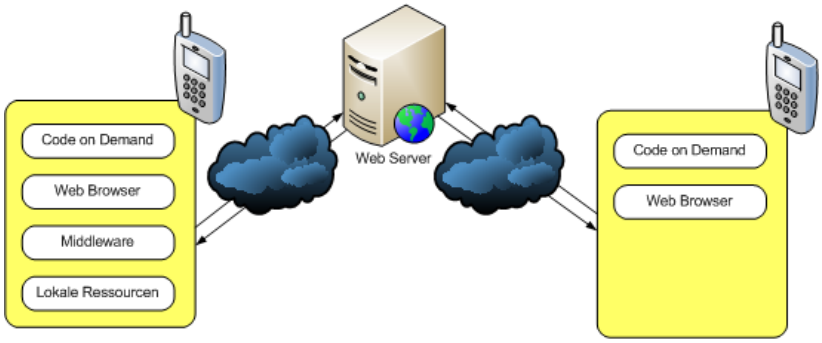


Abbildung 21: Umfeld der Middleware

Die Middleware selber wird ausschließlich auf dem mobilen System installiert, es existiert keine Server Komponente. Abbildung 22 zeigt die Integration der Middleware auf dem mobilen System. Die Middleware wird als zusätzliche logische Schicht zwischen den Browser und das Betriebssystem platziert. Die Code on Demand Anwendung läuft auf- beziehungsweise im Browser und spricht über diesen die Middleware an. Die Middleware greift direkt oder indirekt über Betriebssystemaufrufe auf die lokalen Ressourcen zu.

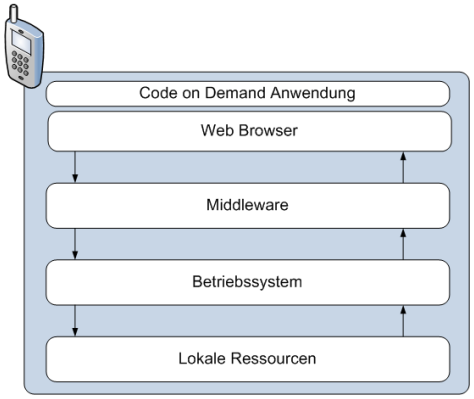


Abbildung 22: Integration der Middleware in das mobile System

3.3 Anwendungsfälle

Im folgenden werden die Anwendungsfälle für die Middleware definiert. Durch die Definition der Anwendungsfälle sollen die ermittelten Anforderungen aus dem Analyse Kapitel (3) in eine Architektur für die Middleware überführt werden. Die Anwendungsfälle beschreiben die beteiligten Akteure, die erwarteten Ablaufschritte und zu erwartende Ausnahmen.

3.3.1 Use Case 1: Installation der Middleware

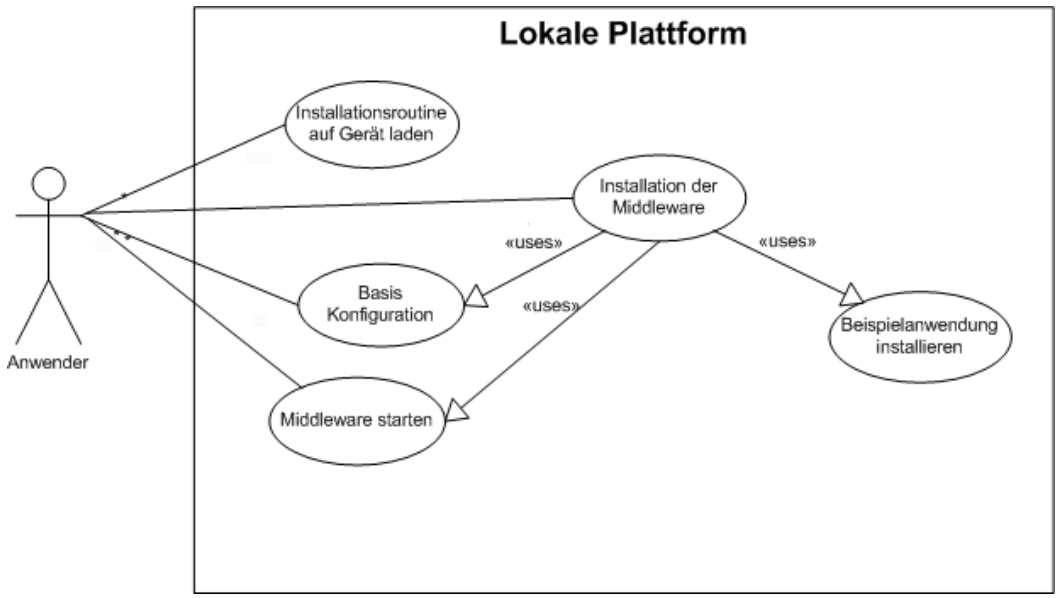


Abbildung 23: Use Case 1: Installation der Middleware

Für die Verbreitung der Middleware wäre es von Vorteil, wenn diese durch die Anbieter von mobilen Endgeräten vorinstalliert währe. Alternativ muss der Anwender die Middleware manuell auf seinem Gerät nachinstallieren.

Der Anwender lädt eine ausführbare Installationsroutine der Middleware auf sein mobiles Endgerät. Die Anwendung wird anschließend gestartet und der Anwender erhält die Möglichkeit, bereits während der Dialog-basierten Installation, grundlegende Einstellungen vorzunehmen. Zu den grundlegenden Einstellungen gehören der Speicherort der Middleware als auch Konfigurationsmöglichkeiten der Middleware selbst. Der Anwender kann optional Verknüpfungen für das starten und stoppen, das setzen des Status der Middleware, sowie für das Starten einer Synchronisation in den Menüs des mobilen Betriebssystems eintragen

lassen.
Im nächsten Schritt der Installationsroutine installiert diese mitgelieferte Beispielanwendungen in die Middleware.
Im letzten Dialog der Installation erhält der Anwender die Möglichkeit, die Middleware nach Abschluss der Installation zu starten, zusätzlich kann sich der Anwender die Dokumentation der Middleware anzeigen lassen.

3.3.2 Use Case 2: Starten der Middleware

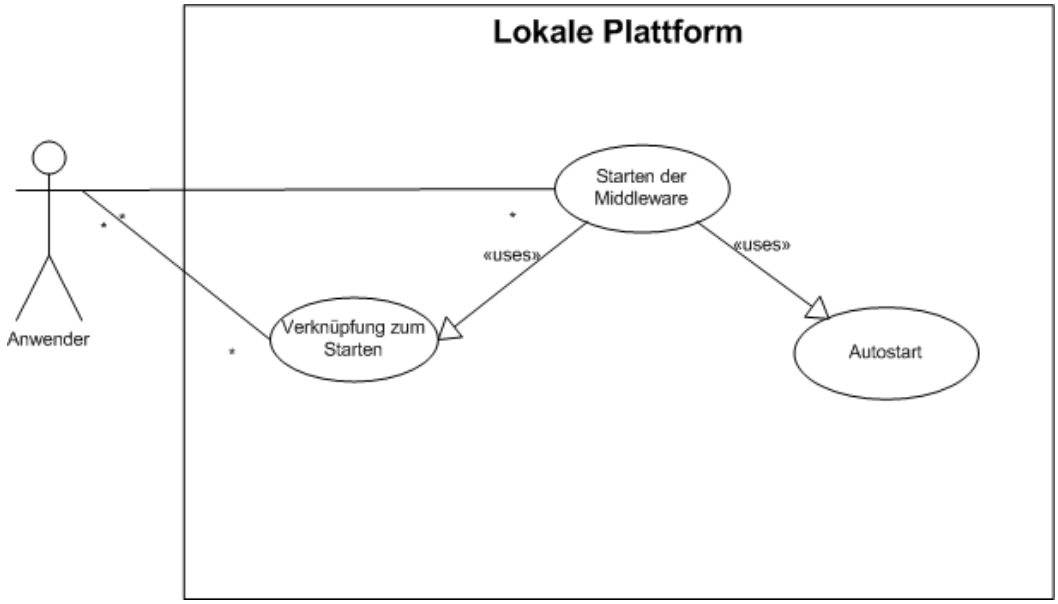


Abbildung 24: Use Case 2: Starten der Middleware

Die Middleware muss gestartet werden, bevor sie verwendet werden kann. Der Anwender kann die Middleware manuell durch die bei der Installation (3.3.1) angelegte Verknüpfung für das starten der Middleware starten.
Die meisten mobilen Betriebssysteme bieten die Möglichkeit, eine Anwendung als Hintergrunddienst einzurichten und diesen beim Start des Systems mit zu starten.

3.3.3 Use Case 3: Installation einer Anwendung

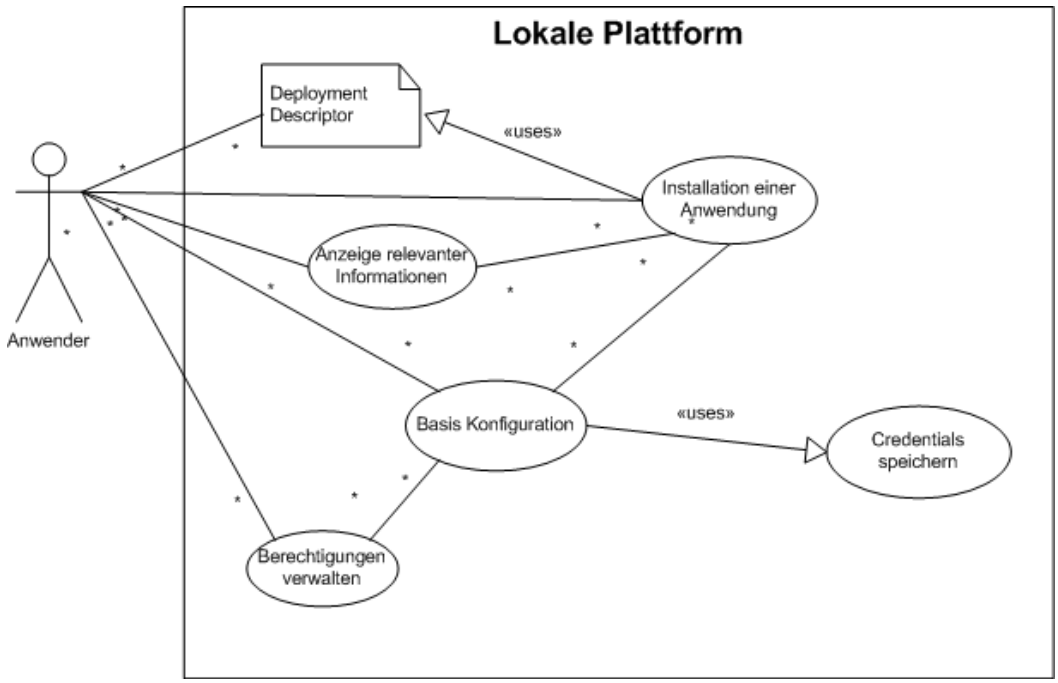


Abbildung 25: Use Case 3: Installation einer Anwendung

Die Installation einer Anwendung wird vom Anwender initiiert. Der Anwender lädt hierzu den vom Entwickler zur Verfügung gestellten Installations Deskriptor (3.4.2) auf das mobile Endgerät. Die Middleware bietet eine Oberfläche, um den Installations Deskriptor zu laden und die Installation durchzuführen. Der Anwender erhält zum einen Informationen über die zu installierende Anwendung, zum anderen die Möglichkeit, bereits während der Dialog-basierten Installation, grundlegende Einstellungen vorzunehmen. Zu den grundlegenden Einstellungen gehören beispielsweise die Credentials für den Zugriff auf einen gesicherten Web Service. Weiterhin muss der Anwender im Zuge der Installation, den im Installations Deskriptor geforderten Zugriffsberechtigungen einer Anwendung zustimmen.
Zum Ende der Installation einer Anwendung kann der Anwender auswählen, dass nach Abschluss der Installation eine Verknüpfung im Browser auf die neue Anwendung hinzugefügt werden soll. Abschließend bietet die Installation die Möglichkeit, die Anwendung direkt nach Abschluss der Installation auszuführen und im Browser anzuzeigen.

3.3.4 Use Case 4: Ausführen einer Anwendung

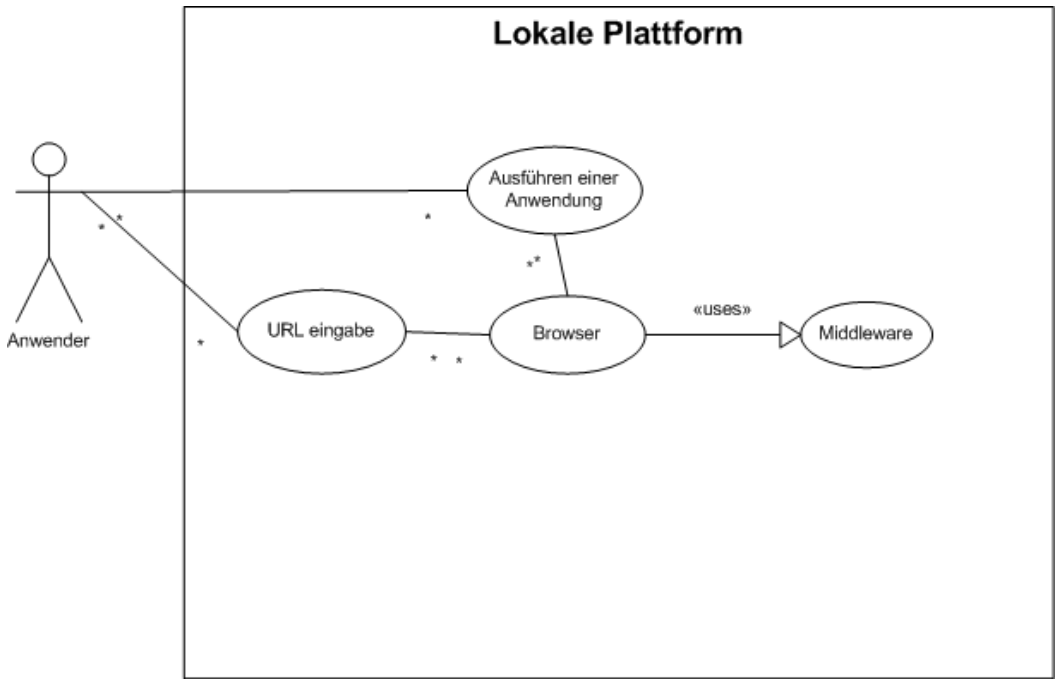


Abbildung 26: Use Case 4: Ausführen einer Anwendung

Der Anwender führt eine in der Middleware installierte Code on Demand Anwendung durch das Aufrufen einer definierten, lokalen URL auf. Die Anfrage wird über die HTTP-Schnittstelle der Middleware entgegen genommen und anschließend über diese die Antwort an den Browser gesendet. Die eigentliche Ausführung der Anwendung findet im Browser statt.

3.3.5 Use Case 5: Konfiguration der Middleware

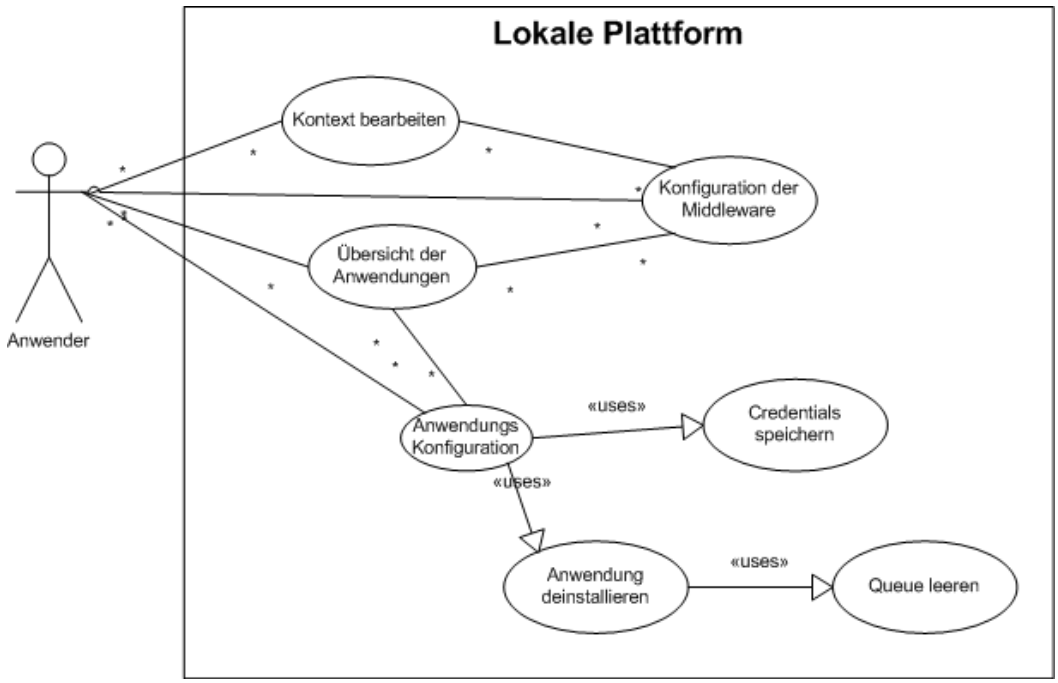


Abbildung 27: Use Case 5: Konfiguration der Middleware

Die Middleware liefert eine eigene, nicht deinstallierbare Code on Demand Anwendung mit. Diese eigene Anwendung bietet dem Anwender die Möglichkeit, die Middleware zu konfigurieren. Die Anwendung zur Konfiguration der Middleware wird wie alle anderen Anwendungen der Middleware über eine definierte, lokale URL aufgerufen (3.3.4)
Der Anwender erhält in der Anwendung zur Konfiguration der Middleware eine Übersicht aller installierten Anwendungen. Der Anwender kann eine Anwendung selektieren und dann auswählen, dass er diese bearbeiten oder deinstallieren möchte. Möchte der Anwender eine Anwendung bearbeiten, so stellt die Middleware ihm relevante Informationen, ähnlich den Dialogen der Installation von Anwendungen (3.3.3) aufbereitet dar. Der Anwender kann in dem Dialog beispielweise die Credentials ändern.
Fordert der Anwender die Deinstallation einer Anwendung, so prüft die Middleware, ob sich noch nicht übertragene Daten im Puffer des HTTP-XML Messaging Proxy (3.4.4) befinden. Ist dies der Fall, weist die Middleware den Anwender auf die noch nicht übertragenen Daten hin und löscht nur bei Bestätigung des Anwenders die Daten aus dem Puffer und deinstalliert die Anwendung.
Die Anwendung zur Konfiguration der Middleware bietet neben der Bearbeitung der installier-

ten Anwendungen die Möglichkeit, das kontextuelle Verhalten der Middleware zu definieren (3.4.1).

3.3.6 Use Case 6: Setzen des Middleware Kontext Status

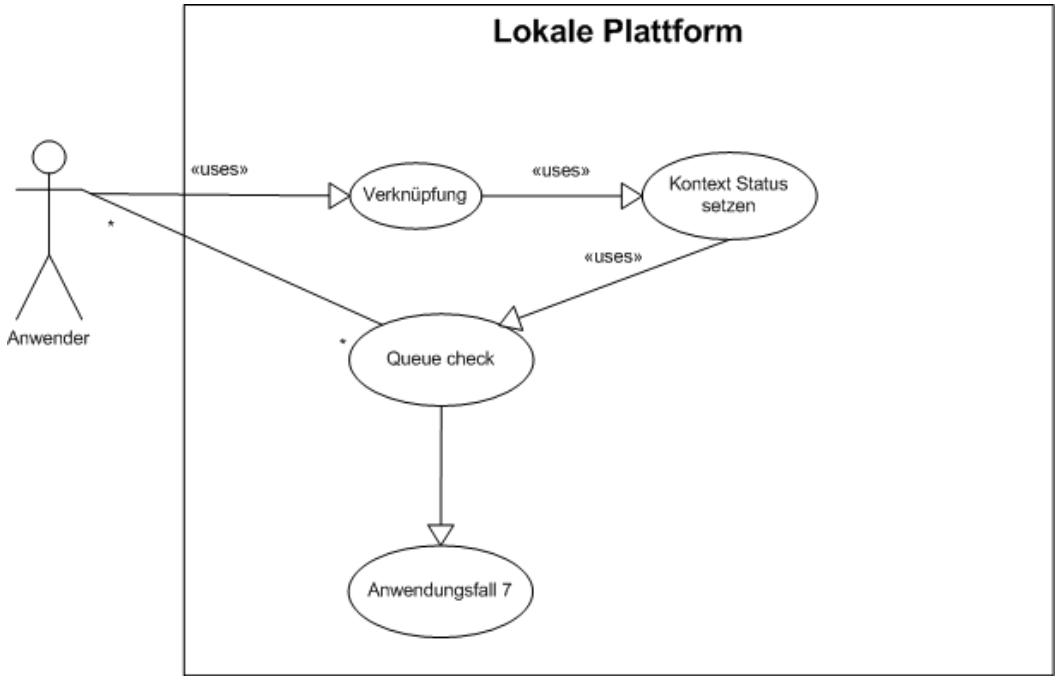


Abbildung 28: Use Case 6: Setzen des Middleware Kontext Status

Der Anwender kann den Status der Middleware manuell durch die im Use Case 1 (3.3.1) angelegte Verknüpfung für das Setzen des Status der Middleware bestimmen. Das Setzen des Status der Middleware in den Online Modus bedeutet nicht zwingend, dass das Gerät online ist und vice versa. Der Status der Anwendung bezieht sich lediglich auf das Verhalten dieser.
Für den Fall, dass die Queue nicht geleert ist, wird bei einem Kontextwechsel eine Rückfrage an den Anwender gestellt, ob eine Übergang in den Anwendungsfall 7 (3.3.7) gewünscht ist. Ist in der Konfiguration der Middleware ein transparenter Kontextwechsel definiert, steht dieser Anwendungsfall nicht zur Verfügung (3.3.5).

3.3.7 Use Case 7: Starten der Synchronisation

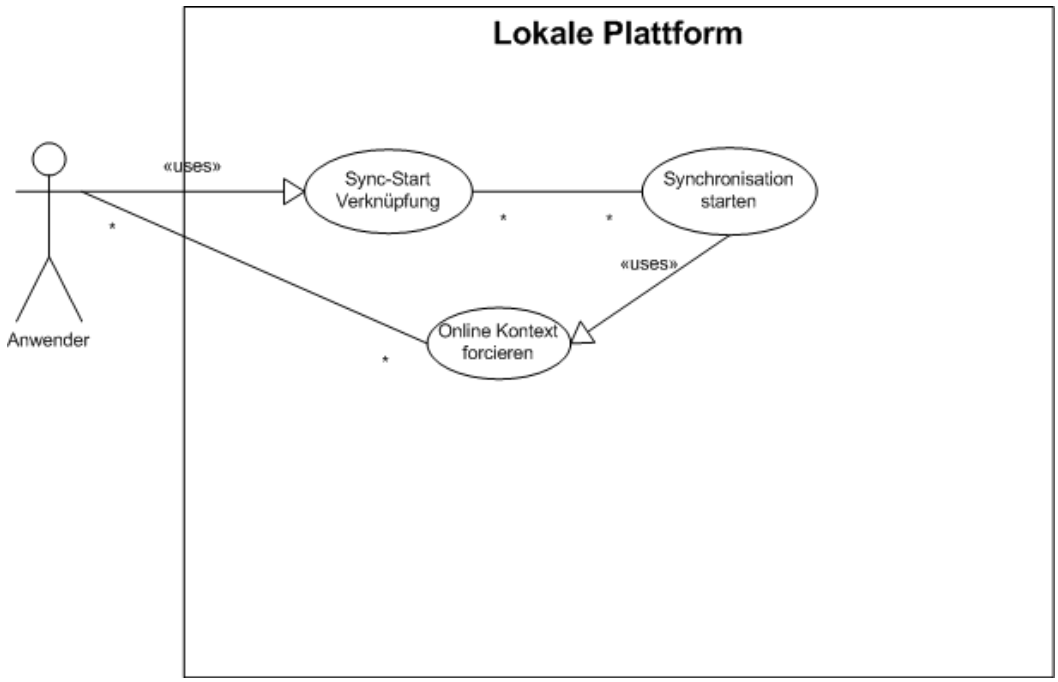


Abbildung 29: Use Case 7: Starten der Synchronisation

Der Anwender kann die Synchronisation der Middleware manuell durch die bei der Installation (3.3.1) angelegte Verknüpfung starten. Befindet sich die Anwendung im offline Kontext, wird dem Anwender die Option geboten, in den Online Kontext zu wechseln, um die Synchronisation durchzuführen.

3.3.8 Use Case 8: Stoppen der Middleware

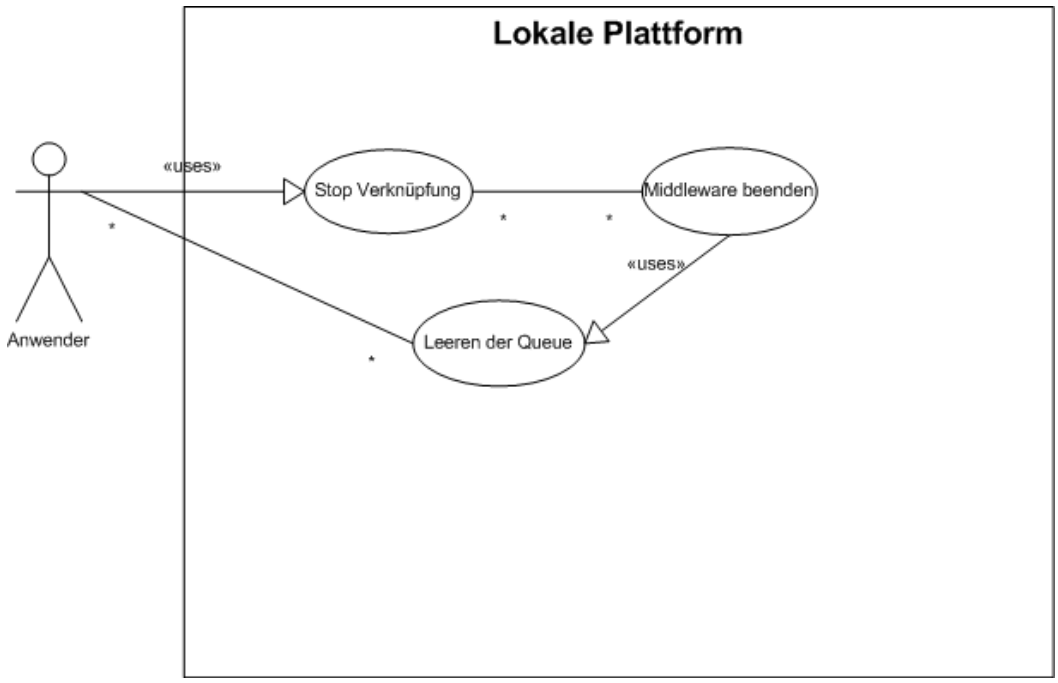


Abbildung 30: Use Case 8: Stoppen der Middleware

Der Anwender kann die Middleware manuell durch die bei der Installation (3.3.1) angelegte Verknüpfung für das stoppen der Middleware stoppen. Sind noch nicht gesendete Daten in der Queue, wird der Anwender darauf hingewiesen und erhält die Möglichkeit, in den Use Case 7 zu wechseln (3.3.7), danach wird die Middleware beendet.
Nach dem stoppen der Middleware sind die in der Middleware installierten Anwendungen nicht mehr erreichbar.

3.4 Funktionale Anforderungen

Im Abschnitt der funktionalen Anforderungen wird definiert, welche Funktionen die Middleware bieten muss. Die Reihenfolge der Anforderungen orientiert sich an dem Lebenszyklus einer Code on Demand Anwendung im Zusammenspiel mit der Middleware.

Unabhängig von der Code on Demand Anwendung muss die Middleware Funktionalitäten bieten, den Kontextwechsel zwischen Online und offline zu handhaben (3.4.1).
Der für diese Arbeit relevante Lebenszyklus einer Code on Demand Anwendung beginnt mit der Installation und Konfiguration in der Middleware (3.4.2). Durch die lokale Installation in der Middleware liegen relevante Codeteile der Anwendung lokal vor und können auch ohne Verbindung zum entfernten Webserver ausgeführt werden.
Damit die installierte Anwendung offline ausgeführt werden kann, muss die Middleware den Kommunikationsfluss kontrollieren. Um dies zu ermöglichen bietet die Middleware einen HTTP Proxy Dienst an (3.4.3). Der HTTP Proxy kann im Online Kontext den Request an den Server weiterleiten und den resultierenden Response an die Anwendung zurück geben. Befindet sich die Middleware im offline Kontext, muss sie mit Hilfe der lokal installierten Anwendung selbst einen Response erzeugen.
Typischerweise kommunizieren Code on Demand Anwendungen per asynchronem Nachrichten Austausch mit einem Server. Im offline Kontext muss die Middleware diese Nachrichten puffern können. Für diese Funktionalität muss die Middleware einen HTTP Messaging Proxy anbieten (3.4.4). Der Messaging Proxy arbeitet mit Gegensatz zu dem HTTP Proxy auf der Nachrichten Ebene der Applikation.
Um entfernte Dienste offline verfügbar zu machen, müssen relevante Teile des Anwendungscodes lokal vorgehalten werden. Die Middleware deckt diese Anforderung durch einen Applikations Cache ab (3.4.5). Zur Laufzeit im offline Modus ruft der Browser den Code der Anwendung von der Middleware ab und führt ihn aus.
Da die Zugriffe auf Webanwendungen meist Benutzer bezogen erfolgen, benötigt die Messaging Middleware einen Speicher für Credentials (3.4.6). Durch das Vorhalten der Credentials wird das zeitversetzte Versenden von zwischengespeicherten Nachrichten an gesicherte Webanwendungen ermöglicht.
Damit Code on Demand Anwendungen auf lokale Ressourcen zugreifen können, müssen diese über die Middleware verfügbar gemacht werden (3.4.7). Als Erweiterung zu dem Zugriff auf lokale Ressourcen benötigen die Anwendungen die Möglichkeit der Datenspeicherung (3.4.8).
Die im XML-HTTP Proxy zwischengespeicherten Nachrichten, ebenso wie die Änderungen

an den lokalen Persistierungsmechanismen, müssen an den Server repliziert werden, dies wird im Abschnitt Replikation (3.4.9) behandelt.

3.4.1 Kontextwechsel

Der Verbindungskontext des Gerätes hat Einfluss auf das Verhalten der Middleware und der installierten Code on Demand Anwendungen. Die definierten Zustände sind:

- Das Gerät verfügt über eine Verbindung zum Internet
- Das Gerät verfügt nicht über eine Verbindung zum Internet

Die Ermittlung des Verbindungskontextes ist grundsätzlich trivial.
Es gilt jedoch zusätzlich zu der technischen Machbarkeit zu berücksichtigen, dass dieser Automatismus vom Anwender nicht unbedingt gewünscht ist. Es sind Situationen denkbar, in denen der Anwender eine Internetverbindung herstellt und nicht möchte, dass die Middleware eine Nachrichtensynchronisation durchführt. Beispiele hierfür sind ein nahezu leerer Akku oder auch eine schlechte Verbindung, dessen knappe Kapazitäten für andere Zwecke benötigt werden.
Die beiden denkbaren Lösungsansätze sind folglich:

- **Transparenter Kontextwechel**, der Benutzer muss keine Aktion vornehmen, um den Synchronisationsprozess zu starten. Ist eine Synchronisation möglich, erkennt die Middleware dies und startet den Vorgang.
- **Benutzerinitiierter Kontextwechsel**, der Benutzer muss die Synchronisation manuell starten.

Der transparente Kontextwechsel ist für den Anwender komfortabler, der benutzerinitiierte Kontextwechsel gibt dem Anwender mehr Kontrolle.

3.4.2 Installation und Konfiguration von Anwendungen

Da die Messaging Middleware einige Sicherheitskonzepte bewusst aushebelt, muss der Zugriff auf die Middleware restriktiv beschränkt werden. Wäre der Zugriff auf die Middleware aus einer beliebigen Code on Demand Anwendung möglich, würde dies die Sicherheit des gesamten Systems gefährden. Ohne eine Beschränkung des Zugriffs könnte jede aufgerufene Web Anwendung auf lokale Ressourcen zugreifen.
Aufgrund der Sicherheitsbedenken folgt, dass eine Code on Demand Anwendung in der Middleware einen Installationsprozess durchlaufen muss, um den Zugriff auf die Messaging

Middleware zu erlangen. Die Installation muss benutzergesteuert sein und mit entsprechenden Abfragen dafür sorgen, dass der Anwender weiß, welche Anwendung, welche Rechte durch die Installation erhält. Wie feingranular diese Bestätigung erfolgt, muss abgewogen werden. Wird lediglich zwischen Vollzugriff und Zugriffsverweigerung unterschieden, bekommen alle Anwendungen mehr Rechte, als sie eigentlich benötigen und es entstehen Sicherheitsrisiken. Ist die Abfrage zu feingranular wird der Anwender mit Bestätigungsanforderungen überflutet, was unter dem Gesichtspunkt der User Experience negativ empfunden wird und die Gefahr birgt, dass der Anwender immer bestätigt.
Die Abbildung und Prüfung der Zugriffsbeschränkung wird in (3.5.4) im Detail behandelt.

Für die Installation benötigt die Middleware diverse Daten von der zu installierenden Anwendung. Dabei handelt es sich um Daten über die Anwendung selbst, die Anforderungen der Anwendung an die Middleware und statische Zusatzdaten.

Um Plattformunabhängigkeit zu gewährleisten, sollte die Übergabe dieser Parameter anhand eines Installations Deskriptors erfolgen. Dieses in XML formulierte Dokument muss der Entwickler der Code on Demand Anwendung zur Verfügung stellen. Der Deskriptor wird vom Anwender im Online Modus heruntergeladen und der Middleware übergeben. Die Daten müssen später konfigurierbar sein, im ersten Schritt einer Entwicklung genügt jedoch die Möglichkeit der Deinstallation einer Anwendung.

3.4.3 HTTP-Proxy

Code on Demand Anwendungen wie beispielsweise das auf Java Script basierende AJAX, werden von einem Browser beim Aufruf der URL heruntergeladen und anschließend lokal ausgeführt. Der Browser nutzt die Netzwerkschnittstellen des Betriebssystems, um die Daten von dem entfernten Webserver herunterzuladen. Ohne Verbindung zu einem Webserver ist das starten solch einer Anwendung auf diesem Wege nicht möglich. Die Middleware muss daher zwischen den Browser und die Netzwerkschnittstelle des Betriebssystems integriert werden, damit eine offline Funktionalität geboten werden kann. Ziel ist, dass die Messaging Middleware im offline Modus die HTTP Anfragen selbst beantwortet und im Online Modus die Anfragen weiterleitet und die Antwort des Webservers zurückgibt. Für die Kontrolle des HTTP Kommunikationsflusses bietet sich eine Proxy Lösung an.
Sucht man im Duden nach einem Synonym für den Begriff Proxy, so wird *Bevollmächtigter* ausgewiesen. In diesem Fall wird ein Stück Software (der Proxy) dazu bevollmächtigt, den externen HTTP-Verkehr an Stelle des Browsers zu übernehmen. Der Browser sendet seine

HTTP-Aufrufe nicht direkt über das Internet an den Webserver, sondern indirekt über die Proxy Anwendung. Abbildung 31 veranschaulicht die Arbeitsweise solch eines Proxy und den daraus resultierenden, indirekten Internetzugriff.

Diese Proxy Funktionalität ist ein standardisierter Teil der HTTP Protokolls und ist im HTT-

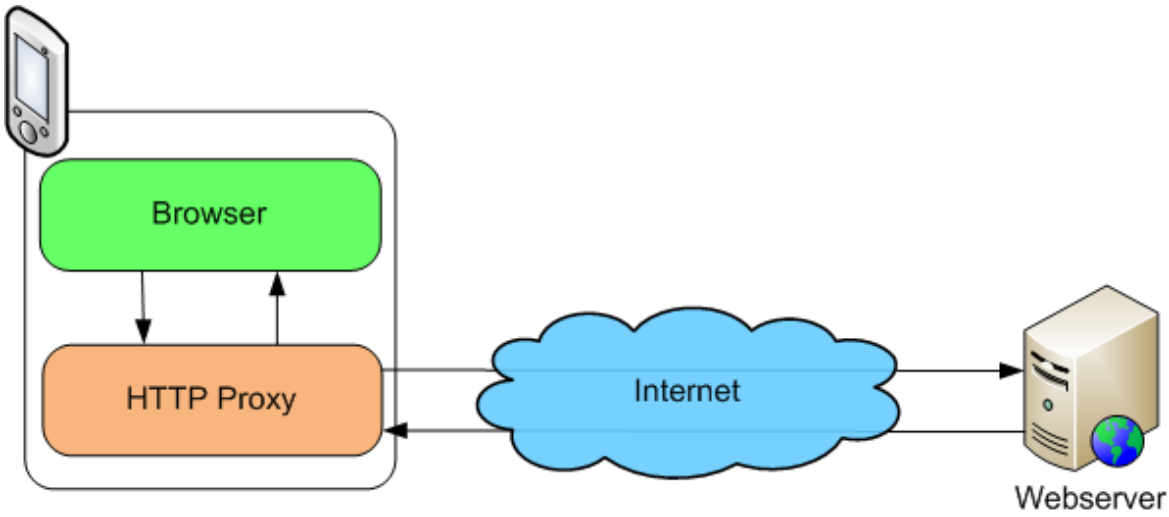


Abbildung 31: Kommunikation mit HTTP-Proxy

P/1.1 RCF (HttpRFC, 1999) als HTTP-Proxy definiert. Ein Proxy bietet neben der reinen Weiterleitung von Nachrichten ebenfalls die Möglichkeit Daten zu cachen und so die externe Netzwerklast zu reduzieren.

Browser bieten meist die Möglichkeit, einen Proxy fest zu hinterlegen. Wird dies getan, leitet der Browser den kompletten HTTP-Verkehr über den Proxy. Alternativ kann der Proxy explizit aufgerufen werden, wenn HTTP-Verkehr über ihn geleitet werden soll. Für den fest eingestellten Proxy spricht, dass so garantiert der HTTP-Verkehr über diesen geht. Das unnötige routen über einen lokalen Proxy im Online Modus kann aber auch dazu führen, dass der Internetverkehr verlangsamt wird. Aus diesem Grunde sollte die Middleware nicht als statischer Proxy hinterlegt werden. Anstelle dessen, sollte die Middleware explizit über eine lokale URI angesprochen werden.

Für die Middleware ist es zunächst entscheidend, dass ein Proxy die Möglichkeit bietet, programmatisch Kontrolle über den HTTP-Verkehr zu erhalten. Der HTTP-Verkehr kann im Proxy verändert, gepuffert oder auch umgelenkt werden und bietet so die Möglichkeit, dem Verbindungskontext entsprechend zu reagieren. Der HTTP-Proxy wird als Teil der Middleware durch einen kompakten Webserver realisiert.

3.4.4 XML-HTTP Messaging Proxy

Wie in Abschnitt (2.3.5) erläutert, ist die Kernkomponente von Ajax der asynchrone Nachrichtenaustausch im Hintergrund.
Der in (3.4.3) eingeführte HTTP-Proxy kann die Ajax Nachrichten empfangen und weiterleiten. So lange das mobile Gerät mit dem Internet verbunden ist und der HTTP-Proxy den HTTP-Verkehr lediglich weiterleitet, hat der HTTP-Proxy keine Auswirkungen auf den Ajax Datenverkehr und ist für alle Teilnehmer transparent. Problematisch wird es erst, wenn die Middleware im offline Modus arbeitet. Die Anwendung sendet einen Request, die Middleware ist jedoch nicht in der Lage einen Response zu generieren und der Proxy könnte maximal eine alte Antwort aus dem Cache liefern. Für die Handhabe von Nachrichten im offline Kontext langt folglich ein HTTP-Proxy nicht, weil dieser nicht beantwortbare Requests (im besten Fall) mit einem Fehler quittiert und die Anfrage verwirft.

Damit Code on Demand Anwendungen im offline Kontext funktionieren können, müssen deren Nachrichten in einer Warteschlange zwischengespeichert werden. Die Middleware benötigt folglich eine Komponente, die XML-HTTP Nachrichten der Ajax Anwendungen zwischenspeichert und zu einem späteren Zeitpunkt an einen externen Server überträgt. Diese Komponente wird als Teil der Middleware hier *XML-HTTP Messaging Proxy* oder kurz XHMP bezeichnet. Der XHMP nimmt die Nachrichten im offline Kontext vom HTTP-Proxy entgegen und leitet sie zu einem späteren Zeitpunkt an den externen Server weiter. Das Schaubild 32 zeigt den veränderten Kommunikationsfluss durch Einführung eines XHMP. Besondere Beachtung gilt hier der zeitlichen Differenz zwischen dem Absenden der Nachricht an den Proxy und dem Moment in dem die Nachricht zum eigentlichen Server weitergeleitet wird. Die Antwort des Servers (Response) wird über den HTTP Proxy an die Anwendung zurückgegeben.

Anwendungen die asynchrone Kommunikation nutzen, verwendet meist ein Callback-on-Response, also das reflexive aufrufen eines Programmteils, wenn die Antwort des Servers eintrifft. Anwendungen die eine asynchrone Nachricht über den XHMP versenden, dürfen ihr Verhalten nicht von dem Response abhängig machen. Durch den XHMP kann es Sekunden, Minuten, Stunden oder gar Tage dauern, bis ein Response kommt, wenn der Server nicht mehr existiert, wird niemals ein Response kommen.

Die in der als Queue bezeichneten Liste gespeicherten Nachrichten werden sukzessive übertragen, wenn die Middleware wieder im Online Kontext ist. Die Liste wird im offline Kontext aufgebaut und Online Kontext geleert. Da die Queue irgendwann geleert werden

muss, sollte eine maximale Anzahl an Nachrichten definiert werden. Ist die Queue voll, muss sich der Anwender mit dem Internet verbinden und die Queue leeren. Der XHMP muss ein Fehlerprotokoll, insbesondere für Timeouts anbieten. Der Anwender muss darüber informiert werden, wenn nicht alle Nachrichten weitergeleitet werden konnten und die Gefahr des Datenverlusts besteht. Der Anwender muss dann die Möglichkeit haben, Nachrichten zu löschen, zu bearbeiten oder ein erneutes Übertragen zu initiieren. Der Replikations Dienst 3.4.9 übernimmt diese Aufgabe.
Eine Ajax Anwendung muss die Möglichkeit haben, den Status ihrer Nachrichten vom XHMP zu erfragen. Beim Beenden einer Anwendung kann der Hinweis, dass noch nicht alle Nachrichten übermittelt werden konnten, relevant für den Anwender sein.
Nachrichten werden auch weitergeleitet, wenn die Ajax Anwendung bereits beendet wurde, jedoch kann dann der Response nicht mehr von der Anwendung verarbeitet werden.

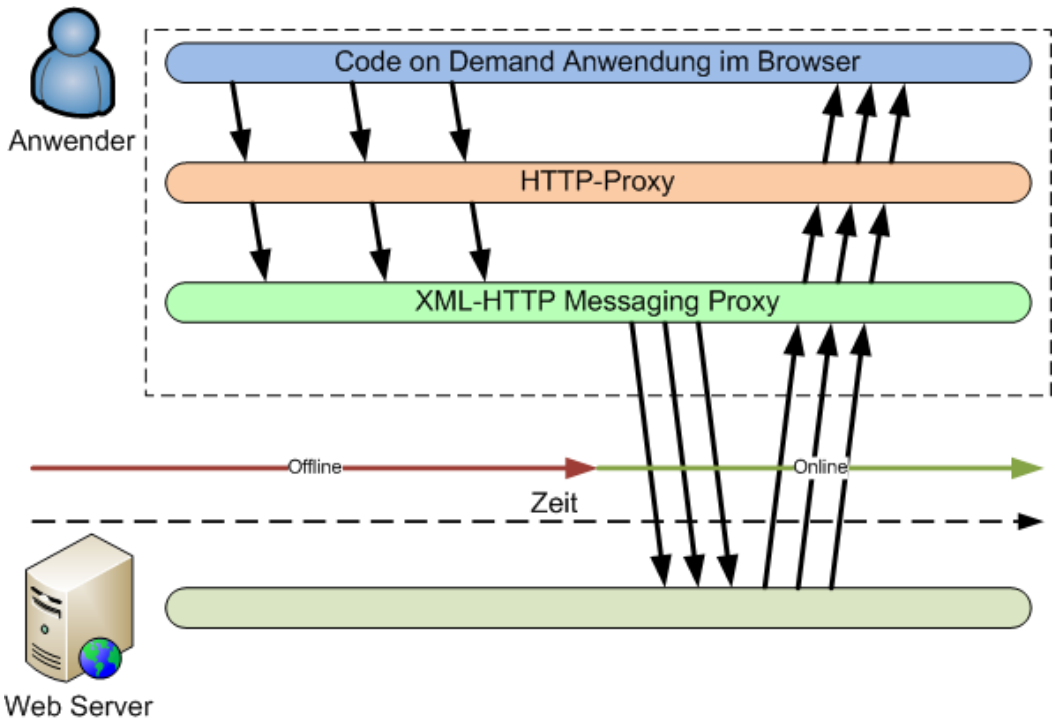


Abbildung 32: Kommunikation mit XML-HTTP Messaging Proxy

3.4.5 Applikations Cache

Die Middleware führt keinen Code selber aus, ist somit keine Laufzeitumgebung. Die eigentliche Laufzeitumgebung, im Beispiel Ajax der Browser beziehungsweise dessen JavaScript

Runtime, fordert von der Middleware den relevanten Code und führt diesen aus.
Die Middleware speichert den relevanten und applikationsbezogenen Code in einem Modul, das hier als Applikations Cache bezeichnet wird. Die Bezeichnung Cache, da der in der Middleware vorgehaltene Code nur der relativ Aktuellste sein kann. Bei der Installation der Anwendung (wie es in Abschnitt (3.4.2) beschrieben wurde) wird eine Version des relevanten Codes in der Middleware gespeichert. Ob es sich hierbei um die aktuellste Version handelt, kann zu dem Zeitpunkt der Installation nicht gewährleistet werden, da der Installations Deskriptor auch im offline Kontext auf das Gerät kopiert worden sein kann. Es ist denkbar und wünschenswert, dass die Middleware die Möglichkeit der Aktualisierung bietet.

Der bei der Installation relevante Installations Deskriptor wie exemplarisch in Listing 17 enthält zwei für den Applikations Cache relevante Bereiche. Dies ist zum einen *code* und zum Anderen *style*.Zwischen dem öffnendem und dem schließendem *code*-Tag befindet sich der für den offline Kontext relevante Code der Code on Demand Anwendung, der lokal vorgehalten werden soll. Der zweite Relevante Bereich, *style* bietet dem Entwickler die Möglichkeit, Layoutangaben mitzuliefern, die ebenfalls im offline Kontext verfügbar sind. In dem Beispiel (siehe Listing 12) wird lediglich ein simpler Text (das obligatorische „Hello World“) in blau auf das Display geschrieben.

```
1     <!-- Offline Anwendung & Layout-->
2     <offline>
3         <code language="JavaScript">
4 <![CDATA[
5 function init() {
6     document.getElementById("main").innerHTML = '<b>Hello World</b>';
7 }
8 ]]>
9         </code>
10        <style>
11 <![CDATA[
12    body { color: #0000FF;}
13 ]]>
14        </style>
15    </offline>
```

Listing 12: code und style Elemente des Installations Deskriptors

Da die Messaging Middleware den Code nicht ausführt sondern der Browser, muss sie lediglich eine Transformation durchführen. Der Applikations Cache liefert auf Basis der installierten Code Fragmente und Layout Definitionen ein XHTML Dokument, welches von einem Browser interpretiert werden kann. Das Ergebnis ist ein XHTML Dokument, mit den eingebetteten Daten (siehe Listing 13)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
  <head>
  <script language="JavaScript" type="text/javascript">
function init()
{
    document.getElementById("main").innerHTML = '<b>Hello World</b>';
}
  </script>
  <style type='text/css'>
body { color: #0000FF;}
  </style>
  </head>
  <body onLoad="init();" id="main">
  </body>
</html>
```

Listing 13: XHTML Dokument aus den transformierten Daten

3.4.6 Credential Store

Basis vieler Ajax Anwendungen ist die Darstellung benutzerspezifischen Inhalts, aus diesem Grunde benötigen diese eine Benutzerverwaltung und die Möglichkeit, Aufrufe zu authentifizieren. Bei normalen Ajax Anwendungen muss sich der Anwender bei dem Server anmelden. Sind die Credentials (Authentifizierungsmerkmale) valide, so sendet der Server einen nahezu eindeutigen Wert an den Client zurück und die Anwendung sendet diesen Wert bei späteren Anfragen mit an den Server. Dieser Wert ist meist nur für einen bestimmten Zeitraum gültig und wird deshalb auch als *Session Identifier* oder kurz *Session-Id* bezeichnet.
Viele browserbasierte Anwendungen speichern zusätzlich die Credentials in HTTP-Cookies. Durch den Einsatz von HTTP-Cookies kann der Anwender bei der nächsten Nutzung der browserbasierten Anwendung automatisch angemeldet werden. Diese Funktionalität ist

nicht ohne Risiko, da die Daten auf einem lokalen Speichermedium abgelegt werden. Dieses Risiko wird jedoch von den meisten Anwendern zugunsten des Komforts in Kauf genommen.
Das Prinzip der Session-IDs ist weiterhin grundsätzlich nicht sicher. Da die Authentifizierung lediglich auf der Übertragung eines nahezu eindeutigen Wertes basiert, langt ein ausspähen dieses Wertes, um einen legitimen Zugriff vorzutäuschen. Einige Systeme prüfen daher zusätzlich die Netzwerkadresse des aufrufenden Clients, doch auch dies ist kein wirklicher Schutz gegen solch einen Angriff. Dies ist jedoch kein Problem der Middleware, sondern ein allgemeines Problem von sitzungsbezogenen Webanwendungen und wird daher auch nicht im Rahmen dieser Arbeit gelöst werden können.

Mobile Endgeräte sind im Gegensatz zu stationären Computern nicht auf Mehrbenutzer Einsatz ausgelegt. Ein mobiles Gerät hat nur einen Anwender, dies wirkt sich auch auf die Middleware aus. Der Anwender kann bei der Installation einer Anwendung (3.4.2) Credentials hinterlegen, welche in der Middleware gespeichert werden. Dieses Modul der Middleware wird als Credential Store bezeichnet. Der Credential Store hält die Credentials persistent auf dem Gerät vor. Die Middleware nutzt dann bei der Synchronisation die Credentials aus dem Credential Store, um auf gesicherte Serverdienste zuzugreifen. Das Prinzip der Credential Stores entspricht dem lokalen HTTP-Cookie Store, in dem applikationsbezogene Daten zwischengespeichert werden.

Durch die Speicherung der Credentials in der Middleware entstehen neue Herausforderungen bezüglich der Sicherheit. Mobile Geräte können verloren gehen oder gestohlen werden. Es entsteht eine unangenehme Situation, wenn der vielleicht gar nicht so ehrliche Finder Zugriff auf Credentials erhält. Die Credentials müssen verschlüsselt auf dem Gerät abgelegt werden. Denkbar ist, dass der Benutzer beim Start der Middleware und in bestimmten Intervallen ein Passwort eingeben muss, welches zur Entschlüsselung verwendet wird.
Ein gewisses Restrisiko bleibt, da der Schlüssel für die Zeit zwischen den Intervallen im Arbeitsspeicher gehalten wird und die Verschlüsselung auf dem Gerät mit manueller Passworteingabe auch nur eine beschränkte Verschlüsselungsstärke zulässt. Das Problem der Verschlüsselungsstärke ließe sich mit hardwarebasierter Verschlüsselung lösen, jedoch verfügt so gut wie kein mobiles Gerät über ein Trusted Computing Module (TCM). Daher ist dieser Ansatz heute kaum praktikabel. Eine höhere Verschlüsselung könnte beispielsweise durch biometrische Erkennungsverfahren erzielt werden, jedoch wird auch dies nur von den wenigsten Geräten unterstützt. Das Restrisiko muss also derzeit in Kauf genommen

werden. Eine Alternative hierzu wäre lediglich, die Credentials nicht zu speichern und bei der Synchronisierung vom Benutzer zu erfragen

3.4.7 Zugriff auf lokale Ressourcen

Ein grundsätzliches Problem aller browserbasierten Applikationen ist der mangelnde Zugriff auf lokale Ressourcen. Mit lokalen Ressourcen ist insbesondere der Zugriff auf das Dateisystem (lesend und schreibend) gemeint. Weiterhin ist kein Zugriff auf die internen und externen Peripheriegeräte möglich. Eine browserbasierte Applikation auf einem mobilen Endgerät kann also weder die Position durch das GPS Modul ermitteln, noch ein Foto mit der internen Kamera aufnehmen oder auf das Dateisystem zugreifen.
Die Beschränkung des Zugriffs auf lokale Ressourcen erhöht die Sicherheit des Systems dramatisch. Wären beliebige Webanwendungen in der Lage, auf die lokalen Ressourcen des Computers zuzugreifen, wären die Folgen kaum abzusehen. Daher ist diese Abschottung zum Rest des Systems, die auch als *Sandboxing* bezeichnet wird grundsätzlich positiv zu bewerten, beschränkt jedoch die Möglichkeit in der Entwicklung von browserbasierten Anwendungen erheblich.

Blickt man auf die Desktop Systeme, so sieht man, dass auch dort mit den Zugriffsbeschränkungen gekämpft wird. Typischerweise bieten die Browserhersteller Plugin-Schnittstellen für die Erweiterung ihrer Browser an. Besonders das Firefox Projekt[19] der Mozilla Foundation hat hier große Stärken.
Eines der bekanntesten Erweiterung ist das Flash-Player Plugin von Adobe Systems Incorporated, das auf nahezu allen PCs im Netz installiert ist Eine von Adobe selbst herausgegeben Statistik zeigt die breite Verteilung des Flash Players (FlashPenetration, 2007).
Der Flash Player bietet neben seinen Visualisierungsfähigkeiten auch einen beschränkten Zugriff auf lokale Ressourcen. So bietet Flash die Möglichkeit, eine lokal installierte Webcam freizugeben und so Flash Programmen den Zugriff auf diese zu ermöglichen.
Weiterhin bietet der Flash Player die lokale Speicherung von sogenannten *Local Shared Objects* (LSO). LSO bieten ähnlich einem Browser-Cookie die Möglichkeit, Werte auf dem lokalen Rechner zu speichern. Die Speicherung findet in einem speziell isolierten und nicht ausführbaren Bereich statt (*Isolated Storage*), um eine größtmögliche Sicherheit zu gewährleisten.
Adobe bietet auch eine Lite Version[20] ihres Flash Players für mobile Plattformen an. Diese

[19] http://www.mozilla.com/firefox/; Zugriffsdatum: 2008-03-21
[20] Adobe Flash Lite - http://www.adobe.com/products/flashlite; Zugriffsdatum: 2008-03-21

Laufzeitumgebung bietet die bekannten Animationsmöglichkeiten des Desktop-Pendants und auch die LSO zum Speichern kleinerer Mengen an Werten. Jedoch ist auch hier die Möglichkeit des lokalen Zugriffs sehr beschränkt, minimale Persistierungstechnologien und beschränkter Zugriff auf gerätespezifische Funktionen.
Aufgrund der Beschränkungen ist die Flash Plattform für lokale Anwendungen im nichtverbundenen Verbindungskontext nicht ausreichend, außerdem ist der Flash Player nur auf das Ausführen von Flash Filmen beschränkt.

Da die bestehenden Lösungen für das beschriebene Szenario nicht weit genug gehen, wird folgend definiert, welchen Anforderungen die Middleware bei dem Zugriff auf lokale Ressourcen genügen muss:

- Definierter Zugriff auf das Dateisystem
- Bereitstellung eines API für den Zugriff auf gerätespezifische Funktionen (z.B. die Helligkeit des Displays)
- Bereitstellung eines API für den Zugriff auf interne und externe Peripheriegeräte (z.B. GPS Empfänger, interne Kamera)
- Bereitstellung eines API für den Zugriff auf Dienste des Gerätes (z.B. Telefonfunktion, Kalender, Kontakte)
- Sicherheitsbeschränkungen, die eine applikationsspezifische Rechtezuweisung erlauben

Der letzte Punkt ist von besonderer Bedeutung. Die zu entwickelnde Middleware soll nicht allgemein sämtliche Sicherheitsrichtlinien aufheben. Die besonderen Anforderungen bezüglich der Sicherheit werden in (3.4.2) behandelt.
Die Bereitstellung eines API ist für die Messaging Middleware möglich, da diese als lokale Anwendung auf der mobilen Plattform läuft. Sie unterliegt daher nicht den Beschränkungen einer Webbrowser-basierten Anwendung. Jedoch kann auch das Betriebssystem den Zugriff für lokale Anwendungen beschränken, dies gilt es bei dem Entwurf einer Lösung zu berücksichtigen.
Mit Hilfe des in (3.4.4) beschriebenen Messaging Proxys kann die Messaging Middleware ein API bereit stellen. Dieses API wird auf Basis des REST Paradigmas realisiert (siehe 2.6), somit kann eine Browseranwendung entsprechende Nachrichten an das API senden und auf die Dienste der Middleware zugreifen.

Der definierte Zugriff auf das Dateisystem wird ebenfalls über einen nach dem REST Paradigma realisierten Dienst auf den Messaging Proxy angeboten werden. Auch hier gilt jedoch die Beschränkung, dass die Middleware nur dort Zugriff gewähren kann, wo sie selbst Zugriff vom Betriebssystem erhält. Sinnvoll ist es hier, den Zugriff auf das Dateisystem auf relative Pfade zu beschränken um zu verhindern, dass auf jede beliebige Datei zugegriffen werden kann. Jede installierte Anwendung erhält einen Ordner, welcher dann lesend oder schreibend verfügbar gemacht wird.

Die Grafik 33 zeigt, wie durch die Middleware als lokale Anwendung der Zugriff auf die lokalen Ressourcen möglich wird.

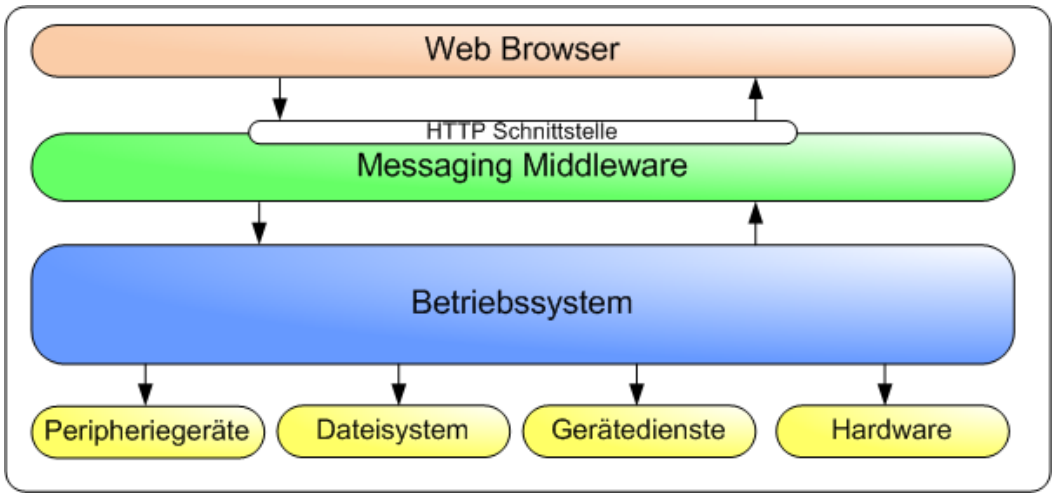


Abbildung 33: Zugriff auf lokale Ressourcen mit einer Middleware Lösung

3.4.8 Persistierungsmechanismen

Die Möglichkeit, Daten persistent zu speichern, ist für die Funktionalität vieler Anwendungen zwingend erforderlich. Code on Demand Anwendungen, im konkreten Fall Ajax-basierte Webanwendungen persistieren über den Webserver. Abfragen der Daten werden in Echtzeit an den Server gesendet und das Resultat asynchron an den Client zurückgeliefert. Besteht jedoch keine Internetverbindung, funktioniert dieser Ansatz zur Persistierung nicht.
Aktuelle, mobile Plattformen beinhalten leichtgewichtige, relationale Datenbankmanagementsysteme. Windows Mobile beispielsweise bringt in der aktuellen Version eine SQL Server Compact Installation mit, Google's Android bietet beispielsweise SQLite. Es ist somit möglich, auf einer lokalen Datenbank Daten zu speichern und mit der Abfragesprache SQL

auf diesen Daten lokal zu arbeiten. Webanwendungen haben jedoch keinen Zugriff auf lokale Ressourcen und somit auch nicht auf diese Datenbanken.
Die Middleware kapselt den Zugriff auf lokale Persistierungstechnologien und macht diese über definierte URLs verfügbar. Ajax Anwendungen erhalten so die Möglichkeit, Daten lokal zu speichern und auf diese zuzugreifen. Um die Komplexität an dieser Stelle beherrschbar zu machen, sind nur Auswahlen nach Primärschlüssel, Löschen nach Primärschlüssel, Suche und Einfügen von Datensätzen möglich. Die Definition der Datenstrukturen erfolgt bei der Installation über den Installations Deskriptor.
Das vorhalten der Daten im Speicher der Ajax Anwendung selbst genügt nicht, da diese sonst beim beenden der Anwendung verloren gingen. Auch das einfache cachen der Daten als Nachrichten an den Server genügt nicht, da diese dann nicht für die Anwendung abfragbar wären.

3.4.9 Daten Replikation

Nachdem im XML-HTTP Messaging Proxy die Nachrichten an den Server im offline Kontext zwischengespeichert wurden, ist es die Aufgabe des Replikations Dienstes, die Daten im Online Kontext an den Server zu transferieren. Die in der Queue gespeicherten Nachrichten werden durch programmatisch erzeugte HTTP Requests sukzessive an die entsprechenden Server weitergeleitet. Werden bei der Übermittlung einer Nachricht Credentials benötigt, so wird im Credential Cache (3.4.6) geprüft, ob die entsprechenden Credentials vorhanden sind, ansonsten wird der Anwender aufgefordert, diese einzugeben. Treten bei der Übermittlung Probleme auf, so erhält der Anwender die Möglichkeit auf diese durch löschen, bearbeiten oder erneutes senden zu reagieren.

Die in 3.4.8 beschriebene Nutzung lokaler Datenbanken führt zu der Fragestellung, wie Daten vorgehalten und repliziert werden sollen. Das aus der lokalen Datenbank resultierende Konzept entspricht einer verteilten Datenbank. In (DbSysteme, 2002) werden vier mögliche Strategien zur Platzierung der Daten in verteilten Datenbanksystemen vorgestellt:

- **Zentralisiert**, die Daten werden zentral in einer einzelnen Datenbank vorgehalten, auf die alle Benutzer direkt zugreifen. Fällt diese zentrale Datenbank aus oder ist sie aufgrund von Problemen im Netzwerk nicht verfügbar, können die Benutzer auf keinerlei Daten zugreifen. Dieser Ansatz macht heutzutage in den wenigsten Fällen Sinn und ist vor allem in dem Kontext dieser Arbeit nicht praktikabel.
- **Partitioniert oder fragmentiert**, die Datenbank wird in disjunkte Fragmente aufgeteilt.

Um den lokalen Zugriff zu optimieren, werden die Fragmente auf die Knoten verteilt. Die Knoten arbeiten so primär auf Ihren lokalen Daten und greifen bei Bedarf auf die entfernten Fragmente zu. Durch den dezentralen Ansatz existiert kein Single-Point-of-Failure mehr, jedoch ist ein Fragment weiterhin nicht verfügbar, wenn der Knoten ausfällt oder nicht verfügbar ist. Auch dieser Ansatz ist für eine mögliche Lösung nicht praktikabel, da so die auf dem mobilen Knoten befindliche Fragmente für den Rest der verteilten Datenbank oft nicht erreichbar wären.

- **Vollständige Replikation**, die Knoten halten in dieser Strategie ein komplettes Replikat der zentralen Datenbank vor. Die Knoten arbeiten ausschließlich auf ihren lokalen Kopien, wobei Änderungen an alle anderen Knoten repliziert werden. Die Kosten der Replizierung sind in dieser Strategie sehr hoch, da die komplette Datenbank auf allen Knoten repliziert wird. Um dem entgegenzuwirken werden sogenannte Snapshots, also Momentaufnahmen der Datenbank, zu bestimmten Zeitpunkten repliziert. So hat nicht jeder Knoten vollständig aktuelle Daten, dies ist aber in vielen Fällen akzeptabel. Diese Strategie ist grundsätzlich für eine Lösung praktikabel, jedoch ist eine komplette Replizierung nicht sehr effizient.

- **Ausgewählte Replikation**, ist eine Kombination der zuvor dargestellten Strategien. Daten, die nur auf einem Knoten benötigt werden, werden partitioniert, auf mehreren Knoten benötigte Daten werden repliziert und alles andere zentral vorgehalten. Diese Lösung vereint die Vorteile aller zuvor vorgestellten Strategien und minimiert deren Nachteile. Auch für eine Messaging Middleware wie sie in dieser Arbeit beschrieben wird, ist dies der optimale Ansatz. Mit dieser Strategie können nur lokal relevante Daten vorgehalten werden, gemeinsame Daten werden repliziert. Auf die zentral vorgehaltenen Daten kann im nichtverbundenen Verbindungskontext nicht zugegriffen werden, diese Beschränkung erscheint akzeptabel.

Für die Replikation von Fragmenten existieren zwei Ansätze: die synchrone und die asynchrone Replikation.
Eine synchrone Replikation würde bedeuten, dass Änderungen an einer Datenbank sofort auf alle anderen Knoten repliziert würden. Hierzu wird ein *Zwei-Phasen-Commit* (2PC) verwendet. Bei einem 2PC sendet die Quelldatenbank eine Commit-Anfrage an die anderen Knoten und nur wenn alle diese Anfrage positiv bestätigen, wird das Commit global durchgeführt. Antwortet ein Knoten nicht, wird bei einem 2PC kein Commit durchgeführt. Der Umgang mit nicht verbundenen Knoten ist für diese Arbeit essentiell, deshalb ist ein 2PC nicht praktikabel und folglich auch keine synchrone Replikation.

Das Gegenstück zur synchronen Replikation ist die asynchrone Replikation. Bei der asynchronen Replikation wird nicht schon während der lokalen Transaktion die globale Replikation durchgeführt, sondern Sekunden, Minuten, Tage später. Die asynchrone Replikation birgt das Problem der Sicherstellung der Datenintegrität, dies wird aber meist akzeptiert und in der Literatur (DistributedDb, 1999) finden sich Lösungsansätze, wie mit Datenintegritätsverletzung umgegangen werden kann.

Für die Replikation muss die Frage nach dem Datenbesitz geklärt werden, also wer, welche Daten ändern darf. Ein typischer Ansatz ist der *Master/Slave-Besitz*. Hierbei hat ein Master-Knoten die alleinigen Schreibrechte auf ein Fragment und informiert die anderen Knoten, bei Änderungen an dem Fragment. Alle anderen Knoten können nur lesend auf dieses Fragment zugreifen. Dieser Ansatz erscheint auch logisch für die zu entwickelnde Messaging Middleware und sichert die Verfügbarkeit und Datenintegrität. Der *Update-Anywhere-Besitz*, wie er bei symmetrischer Replikation verwendet wird, führt bei der beschriebenen Aufgabenstellung zu möglichen Dateninkonsistenzen durch das Arbeiten auf Schattenkopien und bewirkt Replikationskonflikte.

Zusammenfassend sind folgende Verteilungs und Replikationsstrategien denkbar:

- die lokale Datenbank dient nur der offline Auswertung der Daten und wird nach der nächsten Synchronisation verworfen. Durch diese Strategie kann auf die Replikation gänzlich verzichtet werden. Jedoch sind auch keinerlei Stammdaten verfügbar.
- die lokale Datenbank wird mit einer serverseitigen Datenbank über Datenbankspezifische Dienste vollständig oder ausgewählt repliziert.
- die lokale Datenbank wird mit einer serverseitigen Datenbank über einen Dienst der Middleware vollständig oder ausgewählt repliziert.

Im Entwurf gilt es abzuwägen, ob der Einsatz eigener Replikationstechnologien Sinn macht. Datenbanken bieten für gewöhnlich meist bereits erprobte und robuste Technologien, um Datenbanken zu replizieren. Für eine Eigenentwicklung spricht die Effizienz, die Kontrolle und die Möglichkeit einer Datenbank-Anbieter unabhängigen Lösung.

3.5 Nichtfunktionale Anforderungen

Die Analyse der nichtfunktionalen Anforderungen mündet im Gegensatz zu der Analyse der funktionalen Anforderungen (3.4) nicht in abgrenzbare Funktionalitäten, sondern dient der

Qualität des Systems. Die Auswahl der relevanten Aspekte ist an dem Volere Requirements Techniques (volere, 1999) angelehnt.

3.5.1 Performance

Die Middleware als zusätzlicher Dienst auf dem Gerät, darf das Verhalten des Gesamtsystems nicht wesentlich negativ beeinflussen.

Bei der Entwicklung der Middleware ist eine effiziente Programmierung zwingend erforderlich. Den Speicherverbrauch zu berücksichtigen, effiziente Algorithmen zu nutzen und den Netzwerkverkehr gering zu halten sind drei der wichtigsten Punkte, um effizienten Code für mobile Plattformen zu entwickeln. Dies muss bereits bei der Entwicklung des Prototyps berücksichtigt werden.
Auf den Punkt bringt die Einleitung des Software Development Kit von Android (androidSdk, 2008) die beiden primären Paradigmen bei der Entwicklung auf beschränkten Systemen:

- Don't do work that you don't need to do.
- Don't allocate memory if you can avoid it.

Diese beiden Regeln müssen bei der Entwicklung für beschränkte Systeme stets beachtet werden.

Ein wichtiger Faktor für die Performance des Systems liegt im Programmcode. Neben der Vermeidung unnötigen Ressourcenverbrauchs und der Nutzung passender Algorithmen, gibt es einige Regeln, die vor allem in beschränkten Systeme für die Performance entscheidend sein können. In (JavaPerform, 2003) von Jack Shirazi findet man viele Hinweise, um performanten Code zu entwickeln. Die Quelle bezieht sich primär auf die Programmiersprache Java, die Strategien sind jedoch für objektorientierte Sprachen allgemeingültig. Shirazis Hinweise basieren meist darauf, unnötigen oder aufwändigen Code zu vermeiden. Viele der aktuellen Hochsprachen bieten ein sehr abstraktes API, das zu unperformanter Programmierung verleitet, da der Entwickler kaum abschätzen kann, welchen Aufwand ein Aufruf verursacht.

Ein typisches Beispiel ist die Objektorientierte Entwicklung selbst. Die Abbildung in Klassen und Objekte bietet viele Vorteile, birgt jedoch auch die Gefahr riesiger Aufrufketten und Speicherreservierungen. Aus diesem Grund empfiehlt Shirazi die Notwendigkeit der Abbildung auf Objekte im Einzelfall genau abzuwägen. Gerade Werteobjekte können oftmals durch

die hohe Anzahl der Instanzen einen hohen Speicherverbrauch verursachen. Temporäre Objekte in Methoden können oftmals durch Referenzen vermieden werden, dies ist bei der Implementierung zu prüfen. Weiterhin kritisch ist die Verwendung von Interfaces zu betrachten. Bei nach Außen verfügbaren Schnittstellen sollten wegen der Austauschbarkeit Interfaces genutzt werden, interne Implementierungen sollten aus Gründen der Effizienz Interfaces nicht verwendet.
Ein weiteres Problem stellt das automatische Bereinigen nicht mehr benötigten Speichers dar, das *Garbage Collection*. Dieser zusätzliche Dienst führt in meist nicht beeinflussbaren Intervallen Bereinigungen des Heaps durch und bremst so das System aus. Je mehr Objekte existieren, desto mehr Bereinigungen führt das Garbage Collection durch. Selbst wenn kein Speicher zu bereinigen ist, benötigt das Garbage Collection Ressourcen, da die Prüfung ob Objekte aus dem Speicher zu Räumen sind, vorgenommen werden muss. Die hohen Kosten der Objekt Erzeugung und Zerstörung müssen bei der Entwicklung der Middleware stets abgewogen werden.

Strings werden in den meisten modernen Programmiersprachen als ein, in einem Objekt gekapseltes, Zeichen Array abgebildet. Da Arrays eine feste Größe haben, wird bei der Stringverarbeitung fast immer ein neues Objekt erzeugt. Ein typisches Problem ist das Aneinanderketten von Strings, wie es in Listing 14 gezeigt wird. In der ersten Zeile wird ein Objekt erzeugt. Durch die erneute Zuweisung in der zweiten Zeile, wird ein weiteres Objekt erzeugt, das erste wird verworfen. In der dritten Zeile ist das Problem der Aneinanderkettung dargestellt. Es werden zunächst zwei String Objekte erzeugt („Hallo“ und „ „), dann wird ein neues String Objekt aus der Konkatenation der beiden erzeugt („Hallo „). Anschließend wird das dritte String Objekt erzeugt („Welt“), dass dann wiederum für die Erzeugung des finalen String Objektes für die Zuweisung verwendet wird („Hallo Welt“). In der Summe bewirkt die dritte Zeile die Erzeugung von fünf Objekten!

```
1 string s = "Hallo";
2 s = "Hallo Welt";
3 s = "Hallo" + " " + "Welt";
```

Listing 14: Unnötige Objekterzeugung durch Stringkonkatenation

Das Problem ist offensichtlich: die einfache Syntax der Stringkonkatenation führt zu Anwendungscode mit hohem Speicherverbrauch. Die Stringkonkatenation ist ein typisches Beispiel für ein komfortables API, das zu schlechtem Code führt. Jede Programmiersprache bietet die Möglichkeit eines Puffers für solche Stringoperationen, um so eine Operation Speichereffizienter zu Programmieren.

Moderne Programmiersprachen bieten die Möglichkeit, Nachrichten basierte, definierte Ausnahmen zu nutzen (Exceptions). Diese Technologie ist zwar sehr elegant, sollte jedoch nach Shirazi mit Bedacht eingesetzt werden, da auch sie sich negativ auf die Performance auswirken können. Eine Möglichkeit, die Nutzung von Exceptions zu reduzieren, bieten Fehlercodes als Rückgabewerte von Funktionen. Parametrisiert bieten viele Laufzeitumgebungen die Möglichkeit, das Exception Handling komplett zu deaktivieren. In einigen Situationen ist die Nutzung von Exceptions jedoch durchaus gerechtfertigt, daher wird diese Möglichkeit bei der Entwicklung des Prototyps für die Middleware nicht genutzt.

Bei der Entwicklung der Middleware sollte stets bedacht werden, dass auf einem mobilen Gerät nur beschränkte Ressourcen zur Verfügung stehen. Die von Shirazis formulierten Anregungen werden bei der Entwicklung des Prototyps berücksichtigt werden.

3.5.2 Energieeffizienz

Ein wichtiger Aspekt auf mobilen Endgeräten ist die Energieeffizienz. Je sparsamer ein System mit seinen Energieressourcen umgeht, desto länger ist dessen Verfügbarkeit. Bei der Entwicklung der Middleware müssen dies deshalb berücksichtigt werden.
Um die möglichen Auswirkungen auf die Akkulaufzeit abschätzen zu können, muss zunächst die ist-Situation betrachtet werden. Es sind grundsätzlich die beiden Situationen denkbar: es existieren lokale Anwendungen auf den mobilen Geräten oder es wurden bisher klassische Webanwendungen verwendet.
Die erste Betrachtung zielt auf die Anzahl der notwendigen Prozesse. Viele Prozesse bedeuten einen höheren Aufwand für die Verwaltung durch das Betriebssystem. Eine geringe Anzahl von Prozessen führt unabhängig betrachtet, zu einem geringeren Energieverbrauch. Zum Einen besteht die Möglichkeit, dass bisher lokale Anwendungen auf dem Gerät laufen. Eine lokale Anwendung bedeutet immer mindestens einen Betriebssystem Prozess. Werden beispielsweise n Anwendungen verwendet, erzeugt das Betriebssystem n Prozesse, die einzeln verwaltet werden müssen. In dieser Situation kann die Middleware eine effizientere Lösung darstellen, da im Optimalfall nur ein Prozess für die Middleware und ein Prozess für den Browser benötigt wird. Der Vorteil der Middleware ist hier, dass sie unabhängig von der Anzahl der Anwendungen eine konstante Anzahl Prozesse verursacht. Kamen bisher nur Webanwendungen zum Einsatz, benötigt die Middleware immer mindestens einen Prozess mehr als die bisherige Lösung. Da die Middleware den Browser als Rendering Plattform verwendet, ist dies nicht veränderbar.

Weiterhin relevant ist die Lebensdauer der Anwendung. Der Lebenszyklus einer lokalen Anwendung endet mit dem Schließen des Fensters. Das Betriebssystem räumt den Speicher auf, beendet den Prozess und die Anwendung nimmt keine Energie mehr in Anspruch. Für eine Webanwendung gilt dasselbe, wird der Browser geschlossen, räumt das Betriebssystem den Speicher auf, beendet den Browserprozess und die Webanwendung nimmt somit keine Energie mehr in Anspruch. Bei der Middleware ist hingegen ein separates beenden dieser notwendig. Durch das Schließen des Browser alleine wird die Middleware zunächst nicht beendet. Es müssen separate Maßnahmen getroffen werden, um dies zu bewirken. Es bleibt die Frage, ob dies im Sinne der Nutzbarkeit gewünscht ist. Grundsätzlich ist die Middleware auch nur eine lokale Anwendung und kann ebenso wie andere lokale Anwendungen und der Browser beendet werden. Im Entwurf gilt es diesen Punkt zu berücksichtigen. Grundsätzlich stellt die Lebensdauer der Middleware keinen Nachteil aus Energieeffizienz Gesichtspunkten dar.

Viel relevanter als die Anzahl der Prozesse und die Lebensdauer einer Anwendung ist, ob weitere Hardwarekomponenten für ein Szenario benötigt werden. Im Falle einer lokalen Anwendung gibt es grundsätzlich keinen Unterschied zu einer Middleware Lösung. Wird eine bestimmt Komponente benötigt, wie beispielsweise ein GPS Modul, so wird dies aktiviert, verwendet und so früh wie möglich wieder deaktiviert. Webanwendungen verhalten sich in diesem Punkt grundlegend anders. Zum einen haben sie auf die meisten Hardwarekomponenten keinen Zugriff, beispielsweise ist ein Zugriff auf ein GPS Modul nicht möglich, zum anderen erfordern sie zusätzliche Hardwarekomponenten zwingend. Eine Webanwendung kann nicht ohne eine Netzwerkverbindung funktionieren. Aus diesem Grunde ist eine dauerhafte, mit hohen Energiekosten behaftete Netzwerkverbindung erforderlich. Dies ist ein klarer Nachteil von Webanwendungen. Die lokalen Anwendungen und vor allem die Middleware, benötigen nicht zwingend eine dauerhafte Netzwerkverbindung und können deshalb wesentlich energieeffizienter ihren Dienst verrichten.
Neben weiteren Hardwarekomponenten müssen auch zusätzliche Softwarekomponenten betrachtet werden. Eine Webanwendung schneidet bei diesem Vergleich am besten ab, theoretisch werden keine weiteren Softwarekomponenten benötigt. Lokale Anwendungen und ebenso die Middleware können auf lokale Softwarekomponenten zugreifen, hier muss bei jeder zusätzlichen Komponente der Mehrwert gegen den Energieverbrauch abgewogen werden. Exemplarisch sei hier eine lokale Datenbank genannt.

Der vergleich hat gezeigt, dass die Middleware, als Ersatz für lokal installierte Anwen-

dungen oder Webanwendungen, Einfluss auf die Energieeffizienz des Gerätes haben kann. Die Gegenüberstellung mit den anderen Szenarien hat gezeigt, dass die zusätzlichen Möglichkeiten der Middleware als positiven Seiteffekt auch die Möglichkeit der Energieersparnis bieten können. Diese Vorteile bewusst zu nutzen, muss ein Ziel des Entwurfs sein.

3.5.3 Skalierbarkeit

Für die Middleware relevante mobile Endgeräte wie PDAs und Smartphones weisen gegenüber klassischen Desktop PCs Beschränkungen auf, die eine beliebige Skalierung der Middleware verhindern. Zum einen werden in solchen Geräten wesentlich langsamere, weil kleiner und stromsparender, Prozessoren verbaut. Zum anderen stehen auf den Geräten nur sehr begrenzte Speicher Kapazitäten zur Verfügung, dies gilt im Besonderen auch für den Arbeitsspeicher. Die Leistung der mobilen Endgeräte entsprechen bei der Erstellung dieser Arbeit den Desktop Computern von vor 10-15 Jahren.

Aus den hardwareseitigen Beschränkungen folgen Einschränkungen des Betriebssystems. Die mobilen Betriebssysteme sind primär auf die effiziente Nutzung, nicht auf Skalierung ausgelegt. So ermöglichte beispielsweise *Microsoft Windows Mobile* bis zu der Version 6 lediglich 32 parallele Prozesse im laufenden Betriebssystem. Auch die Netzwerk Nutzung ist eher auf Effizienz, als auf Skalierung ausgelegt.
Zusätzlich zu den Beschränkungen der Hardware und der Software unterliegen mobile Endgeräte auch noch den Einschränkungen der verfügbaren Übertragungsmedien. Da sich die Middleware transparent in die bestehende Netzwerktopologie integriert, ist dieser Aspekt nicht für den Entwurf der Middleware relevant. Es liegt in der Verantwortung der Anwendungsentwickler, die Anwendungen gemäß den besonderen Anforderungen im mobilen Kontext zu optimieren.

Da die Middleware Nachrichten an den Server sequentiell im XML-HTTP Messaging Proxy (3.4.4) puffert, skaliert die Middleware bezüglich der Anzahl der Anwendungen. Relevant ist potentiell nur die Anzahl der Anfragen an die Middleware, aus wie vielen unterschiedlichen Anwendungen die Anfragen kommen, hat keinen Einfluss auf die Middleware. Die Betriebssystembeschränkungen sind bei der Entwicklung der Middleware relevant, dies ist jedoch bereits in den Abschnitten Performance (3.5.1) und Energieeffizienz (3.5.2) behandelt worden.
Die Skalierung, im Besonderen das Limit der Queue und die lokale Persistierung von Daten, wird lediglich durch die auf dem Gerät verfügbaren Ressourcen beschränkt.

3.5.4 Sicherheit

Um die Integrität des Systems zu gewährleisten, muss der Zugriff auf die Middleware beschränkt sein. Nur installierte Anwendungen mit den entsprechenden Rechten dürfen die Dienste der Middleware aufrufen.

Abgeleitet von der Theorie der Betriebssysteme (OperatingSys, 2003), werden die Berechtigungen der installierten Anwendungen in einer Zugriffsmatrix abgelegt. Entgegen einer klassischen Unix Zugriffsmatrix, sind nur boolesche Werte erlaubt; entweder eine installierte Anwendung hat Zugriff auf einen definierten Service oder nicht. Abbildung 34 zeigt eine exemplarische Zugriffsmatrix der Middleware mit drei installierten Applikationen.

Die Middleware muss im Moment eines Serviceaufrufes sicherstellen, dass nur eine auto-

Domain-id	Offline	Kamera	GPS	Telefon	File IO
email1	1	0	0	1	0
blog-privat	1	1	1	0	0
geocacher	1	1	1	0	0

Abbildung 34: Zugriffsmatrix der Middleware

risierte Applikation Zugriff erhält. Das primäre Problem hierbei ist die Authentifizierung der Anwendungen. Die Middleware kann nicht darauf vertrauen, welche Anwendung sie aufruft, da die angegebenen Werte manipulierbar sind. Die aufrufende Plattform, also der Browser als Laufzeitumgebung für die Anwendung, müsste kontrollieren und sicherstellen, welche Anwendung Anfragen an die Middleware sendet. Durch Signaturen könnte zusätzlich auch sichergestellt werden, dass die Antwort der Middleware wirklich von *der* Middleware stammt.

Um die erforderliche Infrastruktur zu schaffen, müsste der Browser erweitert, beziehungsweise angepasst werden. Aktuelle Browser auf mobilen Plattformen bieten noch nicht die Möglichkeit, diese durch eigene Erweiterungen anzupassen. Somit muss mit den bestehenden Mitteln eine maximale Sicherheit erzielt werden, ein Restrisiko ist aufgrund der Beschränkungen unvermeidlich.

Dankbarerweise existiert in allen aktuellen Browsern (auch auf Desktop Systemen) eine Sicherheitsrichtlinie welche verhindert, dass aus einer Code on Demand Anwendung Nachrichten asynchron an einen anderen Server geschickt werden. Abbildung 35 veranschaulicht dieses Sicherheitsfeature: eine von server1.org geladene Code on Demand Anwendung

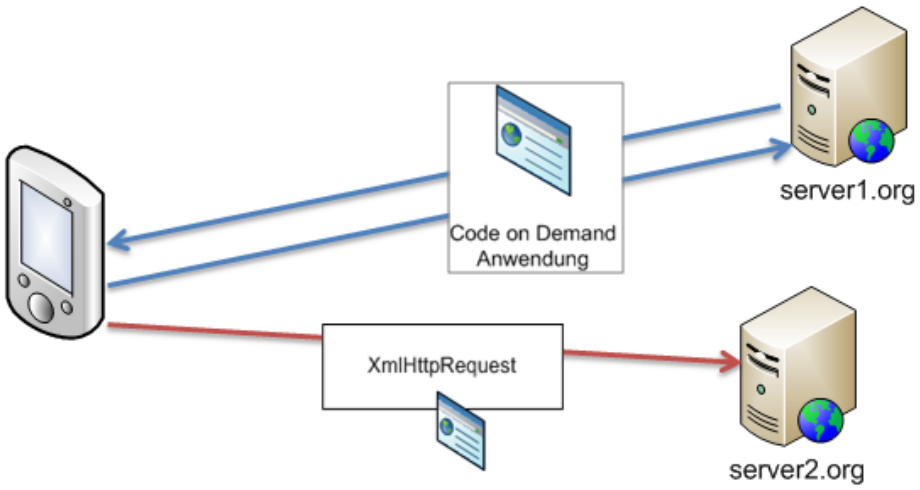


Abbildung 35: Keine Asynchronen Nachrichten an externe Server

kann nicht asynchrone Nachrichten an server2.org senden. Durch diese Beschränkungen können nicht installierte Code on Demand Anwendungen nicht auf die Middleware zugreifen. Dieses Feature allein reicht jedoch nicht aus, da auch installierte Anwendungen bösartig sein können. So könnte eine installierte Anwendung versuchen, sich als eine andere installierte Anwendung auszugeben um so erweiterte Zugriffsrechte zu erlangen. Um diese Angriffstechnik, die in der IT Sicherheit als *Spoofing* bezeichnet wird, abwehren zu können, muss die Middleware zuverlässig erkennen können, durch welche Code on Demand Anwendung sie aufgerufen wird. Eine typische Strategie besteht darin, dass Anwendung und Middleware ein Geheimnis teilen, welches die Anwendung nutzt, um sich beim Aufruf zu authentifizieren. Da beide Anwendungen auf demselben Rechner laufen und zudem die Middleware selbst den Code ausliefert wird die Schlüsselverteilung trivial: der Server schreibt den Schlüssel in den Quellcode der Anwendung.

Durch die vorgestellten Sicherheitskonzepte wird eine grundlegende Sicherheitsarchitektur geboten. Zugriffe auf die Services der Middleware sind beschränkt und die Middleware kann sicherstellen, durch welche Anwendung ein Aufruf erfolgt ist. Eine gesonderte Sicherung der Übertragung ist nicht notwendig, da sich die Middleware transparent in den HTTP Transport integriert und bei Bedarf die zur Verfügung stehenden Techniken nutzen kann.

3.5.5 Portierbarkeit

Das Konzept der Middleware ist nicht Plattform spezifisch und kann daher auf beliebige Plattformen portiert werden. Die Abbildung 36 veranschaulicht, wie vielseitig das Umfeld der Middleware ist. Unter der Middleware stehen mehrere Betriebssysteme und entsprechende Programmiersprachen beziehungsweise Laufzeitumgebungen zur Auswahl. Da die Betriebssysteme und Programmiersprachen stark abweichen, muss die Middleware in großen Teilen für die jeweilige Plattform angepasst werden. Auch bei dem zu verwendenden Browser gibt es eine Vielfalt an Möglichkeiten, die denkbar ist. Da die Middleware aber für den Browser transparent ist, stellt dies kein Problem dar. Entscheidend ist nur, dass der Browser mit dem Ziel Code on Demand Format kompatibel ist. Ist das Ziel Code on Demand Format Ajax, so muss der Browser Ajax handhaben können. Ein mobiler Browser mit Ajax Fähigkeit ist beispielsweise Opera Mini[21] ab Version 3.
Der Prototyp, der im Rahmen dieser Arbeit entwickelt wird, basiert auf dem Microsoft .NET Compact Framework und kann daher auf allen Plattformen ausgeführt werden, die vom .NET Compact Framework unterstützt werden (eine Übersicht der unterstützten Plattformen ist unter (netCfPlat, 2008) aufgeführt).

3.5.6 Benutzbarkeit

Ergonomie und Usability spielt im Besonderen auf mobilen Geräten eine herausragende Rolle, was die Akzeptanz des Anwenders angeht. Da die im Rahmen dieser Arbeit entwickelte Software als Middleware für den Anwender nicht sichtbar ihren Dienst verrichten soll, ist die Benutzbarkeit nicht mit klassischen Fragen der Oberflächenbenutzbarkeit abzudecken. Die notwendige Konfigurations Anwendung der Middleware wird dem Konzept der Arbeit folgend, als Code on Demand Anwendung auf der Middleware selber laufen.
Die Middleware wird für den Anwender im besten Fall nur selten in den Fokus gelangen. Anwenderinteraktion muss auf ein Minimum reduziert und auf präzise, kurze Dialoge beschränkt sein.

3.5.7 Datenintegrität

Eine für diese Arbeit im speziellen relevante, nichtfunktionale Anforderung ist die Datenintegrität. Die Datenintegrität bei asynchroner Speicherung im mobilen Kontext ist eine spezielle Herausforderung, da kein Synchronisationsintervall garantiert werden kann. Dies könnte

[21] http://www.operamini.com/; abgerufen am 2008-07-08

Abbildung 36: Vielseitiges Umfeld der Middleware

nur durch erzwungene Synchronisationen erreicht werden, was jedoch nicht im Sinne der Benutzbarkeit ist (3.5.6).

Die möglichen Replikationsstrategien für die lokalen Persistierungsdienste , sowie für die Nachrichten in der Queue des XML-HTTP Messaging Proxy und die daraus resultierenden Herausforderungen wurden bereits in (3.4.9) erläutert.
Um die Datenintegrität gewährleisten zu können, muss auch der Fehlerfall behandelt werden. Bezüglich der Queue des XHMP ist das Worst Case Szenario, dass der Server dauerhaft nicht erreichbar ist und dadurch die Nachrichten nicht vom lokalen System an den Server übertragen werden können. Auch wenn der Benutzer normalerweise die Middleware nicht wahrnehmen soll (3.5.6), so ist in diesem Fall eine Benutzerinteraktion erforderlich. Dem Anwender muss die Möglichkeit gegeben werden, Einträge der Queue zu löschen oder zu bearbeiten. Der Anwender sollte auch die Option erhalten, erst nach einer bestimmten Anzahl an gescheiterten Replikationsversuchen einbezogen zu werden.

Der Entwickler einer Code on Demand Anwendung für die Middleware muss die Schwierigkeiten der verteilten Datenhaltung und asynchronen Nachrichtenübermittlung bei der Anwendungsentwicklung berücksichtigen. Eine Nachrichtenübermittlung kann zu keinem Zeitpunkt garantiert werden.

4 Entwurf

Ziel dieses Kapitels ist die Entwicklung einer Architektur für die Middleware.
Die Erarbeitung der Middleware Architektur ist Top-Down, die Herangehensweise zielt folglich vom Abstrakten zum Konkreten. Das Definieren von Teilaufgaben des Systems ermöglicht die Bildung von Modulen. Die Modul Definitionen werden in Abschnitt (4.1) vorgenommen.
Ein besseres Verständnis für das Verhalten der Middleware bieten Zustands- und Aktivitätsdiagramme, die im Abschnitt Sichten (4.2) zu finden sind. Im selben Abschnitt befindet sich auch die Daten Sicht, die die persistente Daten Abbildung definiert.
Den Abschluss des Kapitels bildet die Schnittstellen Definition (4.3), das sogenannte API (Application Programming Interface).

4.1 Module der Middleware

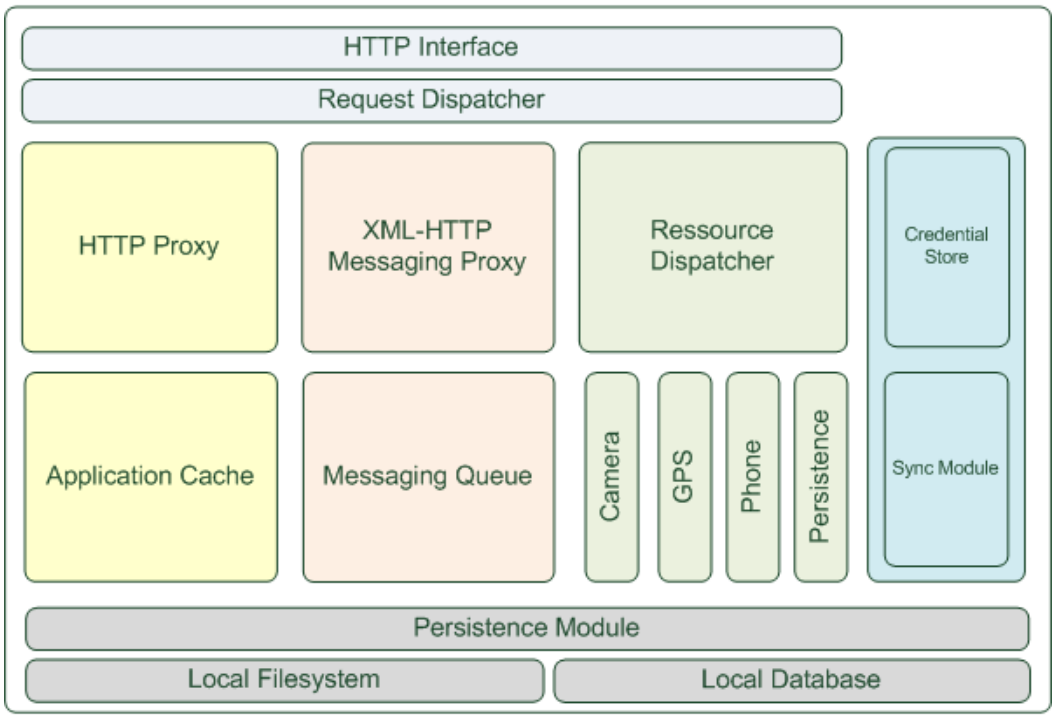


Abbildung 37: Sicht auf die Module der Middleware

Die aus den funktionalen Anforderungen (3.4) und den Use Case Definitionen (3.3) erlangten Kenntnisse, ebenso wie die sich aus der Definition der Middleware Beispielan-

wendung (3.1) resultierenden praktischen Anforderungen, sollen nun für die Definition von Teilaufgaben der Middleware genutzt werden. Durch die Identifizierung und klare Abgrenzung von Teilaufgaben wird die Bildung von Modulen ermöglicht. Die definierten Module werden bei der Realisierung (5) als gekapselte Software Komponenten abgebildet.
Die Module werden dem Aufruf folgend, von oben nach unten definiert, eine grafische Darstellung der Module wird auf dem Schaubild 37 gegeben.

4.1.1 HTTP Interface

Das HTTP Interface bildet die Schnittstelle für den Zugriff auf die Middleware durch den Browser. Die Schnittstellen Definition für das HTTP Interface wird später in diesem Kapitel beschrieben (4.3). Da die Middleware auf dem gleichen Gerät ausgeführt wird wie der Browser, kann dieser das HTTP Interface der Middleware über `localhost`, die lokale IPv4 Adresse `127.0.0.1` oder die lokale IPv6 Adresse `::1` ansprechen.
Technisch entspricht die Umsetzung des HTTP Interface einem kompakten Webserver, der über das HTTP Protokoll angesprochen werden kann und auf demselben Weg eine Antwort liefert. Dieses Interface ermöglicht eine nahtlose Integration in die bestehende Nachrichten und Protokollstruktur, wie es in der Zielsetzung definiert wurde (1.2). Aus Sicht des Browsers ist ein Zugriff auf die Middleware nur ein beliebiger Request an einen Webserver.

Aus Sicht der Middleware sind grundsätzlich zwei Arten von Anfragen zu unterscheiden: Code Anfragen und Nachrichten Anfragen.
Code Anfragen werden aktiv vom Anwender initiiert. Der Anwender ruft im Browser eine lokale URL auf (beispielsweise `http://localhost/blogTool`) und der Browser leitet die Anfrage an die Middleware weiter. Die Middleware generiert den Response und sendet ihn zurück an den Browser. Der Browser rendert die Seite und zeigt sie dem Anwender an.
Nachrichten Anfragen werden nicht direkt vom Anwender, sondern aus dem Code einer Code on Demand Anwendung an die Middleware geschickt. Die asynchronen Nachrichten werden nicht wie sonst an einen entfernten Webserver, sondern an eine lokale, definierte URL gesendet (beispielsweise `http://localhost/ressources/gps`). Die Middleware behandelt die Anfrage und sendet einen Response zurück an den Browser, der die Information an die Code on Demand Anwendung weiterreicht.

Weitere Schnittstellen über das HTTP Interface hinaus sind nicht vorgesehen. Weitere Schnittstellen würden der Zielsetzung, dass sich die Middleware transparent in die vorhan-

denen Nachrichten- und Protokollstrukturen einfügen soll, widersprechen (1.2). Denkbar jedoch ist eine weitere Abstraktion der Programmierschnittstelle, beispielsweise in Java Script, um das Ansprechen der Middleware noch zu Vereinfachen.

4.1.2 Request Dispatcher

Der Request Dispatcher nimmt die Anfrage des HTTP Interface entgegen und ermittelt, um was für eine Art von Anfrage es sich handelt. Alle für den Dispatcher relevanten Informationen ermittelt dieser anhand der URL, auf Basis der API (4.3) Spezifikation. Der Protkoll-Teil der URL, ebenso wie die lokale Adresse (`http://localhost`) wird nicht berücksichtigt, da sie für die Middleware keine Rolle spielt und nur der Adressierung der Middleware dient.

Nachdem der Request Dispatcher ermittelt hat, um was für eine Art Anfrage es sich handelt, wird die Berechtigung geprüft. Der Dispatcher stellt auf Basis der Berechtigungen sicher, dass nur vom Anwender autorisierte Zugriffe (3.3.1; 3.3.5) erfolgen können.
Ist die Anwendung autorisiert den aufgerufenen Service anzusprechen, leitet der Request Dispatcher die Anfrage an das entsprechende Modul weiter. Handelt es sich um eine Code Anfrage, wird diese an den HTTP Proxy weitergeleitet (4.1.3). Eine Nachrichten Anfrage wird entweder an den XML-HTTP Messaging Proxy (4.1.5) weitergeleitet oder bei Anfragen nach einer lokalen Ressource an den Resource Dispatcher delegiert (4.1.6).

4.1.3 HTTP Proxy

Der HTTP Proxy dient der Rückgabe von lokal gespeicherten Code und Layout Daten, macht also die installierten Anwendungen (3.3.3) offline verfügbar. Der HTTP Proxy holt hierfür die transformierten Code und Layout Daten aus dem Applikations Cache (4.1.4) und fügt bei Bedarf noch zusätzliche Meta-Daten ein, wie beispielsweise Schlüssel Daten (3.5.4).

Abbildung 38 zeigt das Verhalten des HTTP Proxy. Im Browser wird eine, in der Middleware installierte, Anwendung aufgerufen. Der HTTP Proxy bettet einen temporär eindeutigen Sitzungsschlüssel in den Anwendungscode ein und liefert das Resultat an den Browser zurück. Das Einbetten des Sitzungsschlüssels erfolgt über die selbst definierte HTTP Header Erweiterung `X-Middleware-Key` und zusätzlich als HTML-Head Element `middleware-key`. Die Anwendung muss diesen Sitzungsschlüssel auslesen und bei folgenden Aufrufen an die Middleware übergeben.

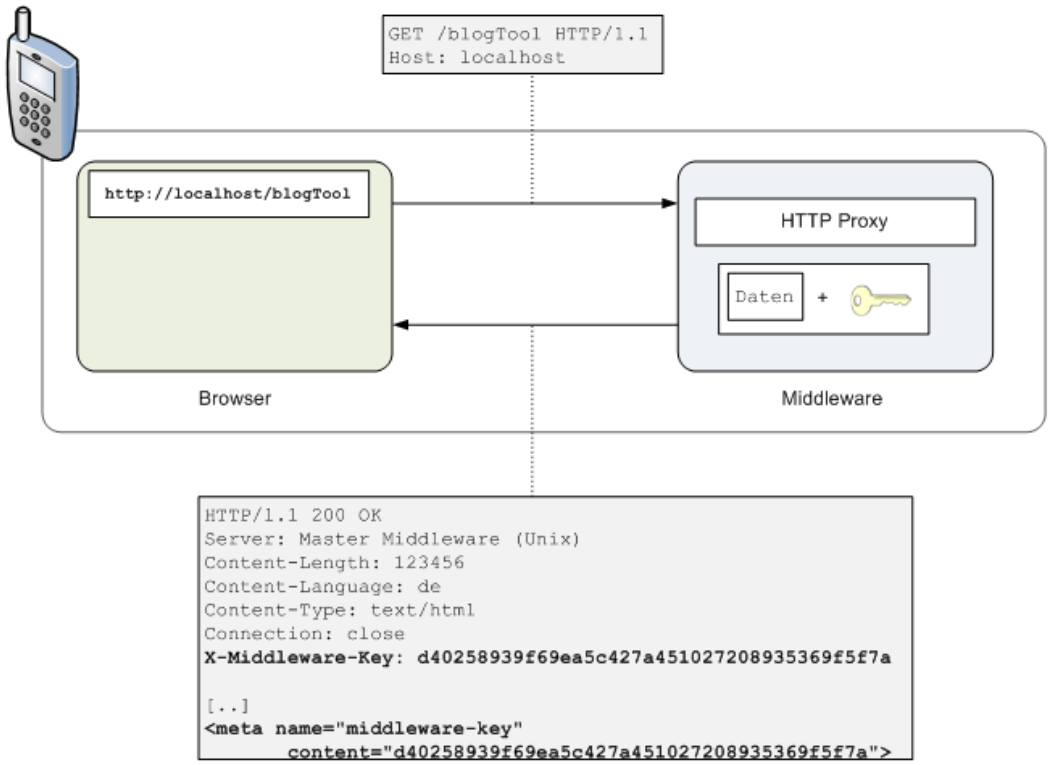


Abbildung 38: Einbettung des Sitzungsschlüssels durch den HTTP Proxy

Das HTTP Proxy Modul ist kein HTTP Proxy im Sinne des HTTP-RFC (HttpRFC, 1999), sondern die Schnittstelle der Middleware zu den offline Anwendungen.

4.1.4 Applikations Cache

Der Applikations Cache wird vom HTTP Proxy (4.1.3) aufgerufen, wenn dieser eine Anfrage nach einer offline Anwendung erhält. Es ist die Aufgabe des Applikations Cache, die installierten Anwendungen so aufzubereiten, dass sie vom HTTP Proxy an den Browser zurückgegeben werden können. Das Zielformat ist XHTML, mit eingebetteten Code und Layout Daten der Anwendung.

Für die Einbettung der Daten holt der Applikations Cache die bei der Installation der Anwendung übergebenen Code und Layout Daten (3.3.3) aus seinem lokalen Speicher. Anschließend transformiert der Applikations Cache die Daten in XHTML. Die Transformation basiert auf Templates. Listing 15 zeigt beispielhaft ein Template der Transformation für die Ajax Anwendungen. In dem Templates sind zum einen die Positionen für die Code (`#cod-code#`)

und Layout (`#cod-style#`) Daten definiert, zum anderen spezielle Deklarationen, die für die jeweilige Technologie relevant sind. So ist in Zeile 12 des Listings 15 der Einstiegspunkt für die Ausführung der Ajax Anwendung definiert (`onLoad="init();"`).

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE html
3       PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
4      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
5  <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
6    <head>
7        <script language="JavaScript" type="text/javascript">
8          <!-- #cod-code# -->
9        </script>
10       <style type='text/css'>
11         <!-- #cod-style# -->
12       </style>
13   </head>
14   <body onLoad="init();" id="main">
15   </body>
16 </html>
```

Listing 15: Ajax Template des Applikation Cache

Durch das Template basierte Transformieren ist der Applikations Cache unabhängig von der Technologie. Diese Abstraktion ermöglicht, anstelle von Ajax eine andere Technologie wie Adobe Flash oder Microsoft Silverlight einzubinden. Eine Flash Anwendung benötigt eine andere Einbettung und einen anderen Einstiegspunkt, hierfür müsste lediglich das Template angepasst werden (siehe Listing 16).

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE html
3       PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
4      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
5  <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
6
7      <title><!-- #cod-title# --></title>
8    </head>
9    <body>
10     <object width="<!-- #cod-width# -->" height="<!-- #cod-height# -->">
11       <param name="movie" value="#swf_file#">
```

```
12        <embed src="<!-- #cod-swf-file# -->" width="<!-- #cod-width# -->"
              height="<!-- #cod-height# -->"></embed>
13      </object>
14    </body>
15  </html>
```

Listing 16: Flash Template des Applikations Cache

4.1.5 XML-HTTP Messaging Proxy

Der XML-HTTP Messaging Proxy (XHMP) ist die Schnittstelle der asynchronen Kommunikation zu entfernten Webanwendungen und ist Basis für die Ausführung von Anwendungen im offline Kontext.

Der XHMP erhält die Nachrichten Anfrage vom Resource Dispatcher (4.1.2) zur Bearbeitung weitergeleitet. Zunächst parst der XHMP die Anfrage und stellt so die syntaktische Korrektheit bezüglich der bei der Installation definierten Endpunkte und Parameter sicher. Sind keine Fehler aufgetreten, versieht der XHMP die Nachricht mit einer eindeutigen ID und legt die Nachricht in die Queue, welche nebenläufig vom Synchronisierungs Modul (4.1.9) abgearbeitet wird.
Weiterhin bietet der XHMP die Möglichkeit, den Status der Queue für eine Anwendung abzufragen. Durch diese Funktionalität erhalten Anwendungen die Möglichkeit, Anwender über noch nicht übertragene Nachrichten zu informieren (z.B. beim Beenden einer Anwendung).

4.1.6 Resource Dispatcher

Der Resource Dispatcher stellt eine einheitliche Schnittstelle für alle Zugriff auf die lokalen Ressourcen dar. Die Umsetzung entspricht dem Fassade Pattern (DesignPatterns, 2003).
Der Resource Dispatcher ist in seinem Verhalten dem Request Dispatcher (4.1.2) sehr ähnlich. Eine an den Resource Dispatcher weitergeleitete Anfrage wird von diesem zunächst gegen die Berechtigungen geprüft. Ist der Aufruf valide, wird die Anfrage an die entsprechende Ressource weitergeleitet, welche den Zugriff auf die lokale Ressource implementiert.

4.1.7 Credential Store

In der Anforderungs Analyse wurde die Notwendigkeit beschrieben, Credentials für den Zugriff auf gesicherte Webanwendungen in der Middleware vorzuhalten (3.4.6). Der Credential Store hält die der Middleware übergebenen Credentials (3.3.1; 3.3.5) vor und bietet

eine Schnittstelle für das Synchronisierungs Modul (4.1.9), damit dieses auf gesicherte Webanwendungen zugreifen kann.

Bereits in der Analyse wurde das Speicherproblem behandelt. Die Credentials im Credential Store müssen in irgendeiner Art und Weise auf dem Gerät gespeichert werden.
Denkbar wäre der Einsatz eines asymmetrischen Kryptosystems um mit diesem Problem umzugehen. Die Passworte würden mit dem öffentlichen Schlüssel eines Servers verschlüsselt abgelegt und wären so nicht einsehbar. Das Problem ist so jedoch nicht gelöst; denn die verschlüsselten Credentials können vom Gerät kopiert und wie Klartext Schlüssel verwendet werden. Somit bietet dieser Ansatz nicht mehr Sicherheit.
Ein weiteres Argument gegen eine asymmetrische Verschlüsselung ist die daraus resultierende Notwendigkeit einer Server Komponente für die Entschlüsselung. Dies widerspricht der definierten Zielsetzung, dass sich die Middleware transparent in die bestehende Nachrichten und Protokollstruktur einfügen soll (1.2).
Ein symmetrisches Kryptosystems ver- und entschlüsselt mit ein und demselben Schlüssel. Mit solche einem Verfahren kann ein Credential vor der Speicherung verschlüsselt und später auch wieder lokal entschlüsselt werden. Als Schlüssel wird ein vom Anwender einzugebendes Passwort verwendet. Problem ist hierbei, dass entweder der Anwender bei jeder Übertragung einer Nachricht das Passwort erneut eingeben muss oder das Passwort im Speicher vorgehalten wird. Die erste Option ist aus Sicht der Benutzbarkeit nicht akzeptabel. Eine praktikable und aus Sicherheits Standpunkten vertretbare Lösung liegt darin, den Anwender in bestimmten Intervallen nach dem Passwort zu fragen und in der Zwischenzeit das Passwort im Speicher zu halten.

Die Middleware legt die im Credential Store gespeicherten Passworte symmetrisch verschlüsselt ab. Als symmetrisches Kryptosystem wird der derzeit als sehr sicher geltende und als *Advanced Encryption Standard (AES)* bekannte *Rijndael-Algorithmus* (aes, 2001) verwendet. Der Anwender sollte konfigurieren können, ob und in welchen zeitlichen Intervallen der Schlüssel (das Passwort) neu eingegeben werden muss.

4.1.8 Persistierungs Modul

Um von der eigentlichen Daten Speicherung zu abstrahieren, bietet das Persistierungs Modul eine einheitliche Schnittstelle um auf das lokale Dateisystem, sowie auf das lokale relationale Datenbank Management System (rDBMS) zuzugreifen.

Der Zugriff auf das Dateisystem ist eingeschränkt, lesend und schreibend möglich. Das Ausführen von Dateien wird aus Gründen der Sicherheit nicht unterstützt, hinzugefügte Dateien werden, soweit es das Betriebssystem unterstützt, als *nicht ausführbar* deklariert. Das Konfigurations Modul (4.1.10) legt bei der Installation einer Anwendung einen lokalen Ordner für die Anwendung an. Der Zugriff einer Anwendung ist auf ihren eigenen Ordner beschränkt.
Das Modul bietet weiterhin die Möglichkeit, lokale Ordner anzulegen. Dadurch entsteht die Möglichkeit, eine relative Ordnerstruktur zu bilden. Eine Schnittstelle bietet die Möglichkeit, eine Liste der enthaltenen Dateien und Ordner zu einem Pfad abzufragen (*Directory Listing*).

Wie in der funktionalen Analyse (3.4.8) definiert, bietet die Middleware eingeschränkten Lese- und Schreibzugriff auf das lokale rDBMS. Das Persistierungs Modul implementiert diese Operationen rDBMS spezifisch. Durch diese Abstraktion kann die Middleware mit unterschiedlichen rDBMSs betrieben werden.
Die folgenden Zugriffe sind definiert:

- Einfügen von Datensätzen (`INSERT`)
- Auswählen von Datensätzen nach Primärschlüssel (`SELECT`)
- Auswählen von Datensätzen durch Suche (`SELECT LIKE`)
- Löschen von Datensätzen nach Primärschlüssel (`DELETE`)

Bei einer Auswahl Abfrage nach Primärschlüssel gibt das Persistierungs Modul ein JSON Objekt zurück (2.7). Bei der Auswahl der Suche nach Datensätzen liefert das Persistierungs Modul eine Liste von JSON Objekten zurück. Einfügen und Löschen liefern einen Rückgabewert, ob die Operation erfolgreich war. Die genaue Schnittstellendefinition ist im API Unterabschnitt (4.3) gegeben.
Das vorgestellte Prinzip entspricht einem Object-Relational Mapping (ORM) für JSON.

4.1.9 Synchronisierungs Modul

Das Synchronisierungs Modul erfüllt die Anforderungen der Daten Replikation (3.4.9). Es handelt sich um einen nebenläufigen Dienst, der im Online Kontext die zwischengespeicherten Nachrichten sukzessive an die entfernten Webanwendungen überträgt. Das Synchronisierungs Modul nimmt seine Arbeit auf, wenn eine Synchronisierung entsprechend (3.3.7) begonnen werden soll.

Die vom Synchronisierungs Modul verwendete Queue entstammt dem XML-HTTP Messaging Proxy (4.1.5) und ist eine nicht priorisierte FIFO (First In First Out) Warteschlange. Das Synchronisierungs Modul greift auf den Credential Store (4.1.7) zu, um die benötigte Credentials für eine Nachricht zu erhalten. erzeugt einen HTTP Request und versendet die zwischengespeicherte Nachricht an einen entfernten Web Service.
Tritt bei der Übertragung ein Fehler auf, so wird der Anwender aufgefordert zu entscheiden, ob die Nachricht erneut in die Warteschlange eingefügt oder verworfen werden soll.

Die Synchronisierung der lokalen Datenbank ist nicht als integrierter Teil der Middleware vorgesehen. Das lokale Datenbank Management System wird Initial bei der Installation mit Daten gefüllt, auf die die Anwendung zugreifen kann. Weiterhin besteht für eine lokal installierte Anwendung die Möglichkeit, Daten zu lesen, ändern und zu löschen. Die Replikation der Daten soll nicht als Teil der Middleware geboten werden, da das Verhalten sehr Datenbankspezifisch ist und viele Datenbankhersteller Tools bieten, die diese Aufgabe sehr gut erledigen. Daher sollte eine zusätzliche Softwarekomponente eines Anbieters verwendet werden, sofern eine Datenbankreplikation erforderlich ist. Zu diesem Zweck bietet die Middleware eine entsprechende Schnittstelle für Erweiterungen.

4.1.10 Konfigurations Modul

Dem Paradigma der Middleware folgend, ist das Konfigurations Modul kein integrierter Teil der Middleware, sondern eine Anwendung auf der Middleware, mit besonderen Rechten. Die Anwendung hat eine Art Startseite, die den Eintritt in die unterschiedlichen Abläufe anbietet.

Entsprechend Anwendungsfall 3 (3.3.3) bietet das Konfigurations Modul die Möglichkeit, weitere Anwendungen auf der Middleware zu installieren. Das Konfigurations Modul stellt hierfür eine Oberfläche zum Laden des Installations Deskriptors bereit, wie es in den funktionalen Anforderungen definiert ist (3.4.2).
Aus der bisherigen Analyse folgt, dass folgende Daten für die Installation benötigt werden:

- **Anwendungsbezeichnung** Als identifizierenden Namen der Anwendung für den Anwender
- **Eindeutiger Anwendungs Identifizierer** Nicht änderbarer, eindeutiger Bezeichner der Anwendung
- **Server Adresse** Soll die Anwendung asynchrone Nachrichten an einen entfernten Webserver schicken können, muss der Middleware die URI des Servers bekannt sein.

- **Server Port** Ist eine Server Adresse bekannt, aber kein Server Port, wird Port 80 als Standard verwendet.
- **Anwendungs Endpunkte und Parameter** Die validen Endpunkte und Parameter eines entfernten Services müssen der Middleware bekannt sein.
- **Policies für den Serverzugriff** Die Middleware benötigt Informationen darüber, welche Anforderungen ein entfernter Service an einen Aufrufer stellt.
- **Credentials** Sind für einen entfernten Server Credentials für einen Aufruf erforderlich, müssen diese der Middleware bekannt sein.
- **Erforderliche Berechtigungen für Zugriffe auf Gerätedienste** Anwendungen müssen deklarieren, auf welche Ressourcen sie Zugriff benötigen. Diese Berechtigungen müssen vom Anwender im Zuge der Installation bestätigt werden.
- **(offline) Code der Anwendung** Die Middleware benötigt den Code der Anwendung, die offline ausgeführt werden soll.
- **Daten der Anwendung** Daten, welche lokale vorgehalten und für die Anwendung abrufbar sein sollen.

Listing 17 zeigt solch einen Deskriptor für die Beispielanwendung.
Im ersten Abschnitt des Beispiel Deskriptors werden die relevanten Server Daten definiert. Eine Code on Demand Anwendung auf einer offline Plattform benötigt theoretisch keinen korespondierenden Server, daher ist dieser Punkt optional. Für gewöhnlich wird hier aber immer ein Eintrag gesetzt sein, da die asynchrone Kommunikation mit einem Server integraler Bestandteil der meisten Code on Demand Anwendungen ist. Im *Policies* Abschnitt wird in diesem Beispiel definiert, dass der Server Passwort geschützt ist. Während der Installation wird die Middleware Benutzername und Passwort vom Anwender erfragen.
Die Beispielanwendung deklariert im Abschnitt der Berechtigungen, dass sie offline Funktionalität und Zugriff auf ein Positionierungs Modul (z.B. GPS) benötigt. Die Anwendung präsentiert diese Berechtigungsanforderung bei einer Installation dem Benutzer, der die entscheidende Instanz bei der Bewilligung der Rechte ist.
Der `offline` Bereich beinhaltet die relevanten Daten für die Ausführung der Anwendung im offline Kontext. Dies ist zum einen der Code der Anwendung (in diesem Falle JavaScript), welcher in `code` deklariert wird. Die Darstellung beeinflussende Daten sind in einer Stylesheet Definitionen im Abschnitt `style` zu finden.

```
<?xml version="1.0" encoding="UTF-8"?>
<application>
    <info>
        <id>blogomat</id>
        <title>blog Tool 2000 - Gold Edition</title>
        <uri>http://www.example.org/blog123/</uri>
        <port>8080</port>
        <endpoints>
            <endpoint id="add" path="rpc.php">
                <endpoint-param id="do" value="addEntry" />
                <endpoint-param id="text" />
            </endpoint>
        </endpoints>
    </info>
    <policies>
        <policy type="usernamePassword" />
    </cod-policies>
    <rights>
        <right type="offline" />
        <right type="position" />
    <rights>
    <offline>
        <code language="JavaScript">
<![CDATA[
function init() {
    document.getElementById("main").innerHTML = '<b>Hello World</b>';
}
]]>
        </code>
        <style>
<![CDATA[
    body { color: #0000FF;}
]]>
        </style>
    </offline>
</application>
```

Listing 17: Installations Deskriptor einer Anwendung

Nach dem ersten auswerten des Installations Deskriptors zeigt das Konfigurations Modul einige relevante Daten an, die dem Benutzer verdeutlichen, welche Anwendung installiert wird. Der Anwender hat an dieser Stelle die Möglichkeit die Installation abzubrechen oder fortzufahren. Wählt der Benutzer, dass die Anwendung installiert werden soll, so werden zunächst die im Installations Deskriptor definierten Berechtigungen geprüft. Sind nicht alle Berechtigungen erfüllbar, weil beispielsweise eine Gerätekomponente fehlt, wird der Anwender über dies informiert und die Installation beendet. Ansonsten wird dem Anwender die erste Berechtigung angezeigt, die von der Anwendung gefordert wird. Der Anwender hat dann die Möglichkeit, diese eine Berechtigung zu vergeben, ungesehen alle zu bestätigen oder aber die Berechtigung zu verweigern, dann wird die Installation beendet.
Sind alle Berechtigungen geprüft, folgt optional im nächsten Schritt das Hinterlegen von Credentials. Der Anwender kann für die Anwendung relevante Credentials lokale speichern, die dann beim Übertragen der Daten an den Server verwendet werden. Der Anwender sollte an dieser Stelle über das offensichtliche Risiko informiert werden und hat die Wahl ob er die Credentials hinzufügen, keine Credentials eintragen oder die Installation abbrechen möchte. Ist auch dieser Schritt erfolgreich abgeschlossen, installiert das Konfigurations Modul die neue Anwendung auf der Middleware, indem es die relevanten Daten in die lokale Datenbank der Middleware einfügt (4.2.4).
Nach erfolgreicher Installation wird dem Anwender die Möglichkeit geboten, die neu installierte Anwendung zu starten. Wählt der Anwender diese Option, wird der Browser zu der URL der neu installierten Anwendung weitergeleitet, ansonsten kehrt das Konfigurations Modul zu seiner Startseite zurück.

Wie in Anwendungsfall 5 (3.3.5) spezifiziert, erhält der Anwender durch das Konfigurations Modul die Möglichkeit, die Middleware selbst zu konfigurieren.
Der Anwender kann sich in einer Übersicht alle aktuell installierten Anwendung anzeigen lassen. Ihm wird die Möglichkeit geboten, eine Anwendung zu selektieren um sich dann Details zu der Anwendung anzeigen zu lassen oder diese zu deinstallieren. Möchte der Anwender die Details der Anwendung ansehen, wird ihm in einer weiteren Ansicht einige Details zu der Anwendung geboten. Wählt der Anwender die Deinstallation und bestätigt die folgende Nachfrage („sind sie sicher...?“), so wird zunächst geprüft, ob noch Daten für diese Anwendung in der Queue des XML-HTTP Messaging Proxys (4.1.5) sind. Ist dies der Fall, erhält der Anwender einen Hinweis und die Option, alle Einträge in der Queue nun zu senden, diese zu löschen oder die Deinstallation abzubrechen. Ist dieser Schritt erfolgreich abgeschlossen, so löscht die Konfigurations Anwendung die in der lokalen Datenbank der Middleware (4.2.4) erfassten Daten für die zu deinstallierende Anwendung und kehrt zur

Startseite des Konfigurations Moduls zurück.
Neben der Deinstallation hat der Anwender auch die Möglichkeit, die hinterlegten Credentials für eine Anwendung zu setzen. Es sollte aus Gründen der Sicherheit nicht möglich sein, die Credentials einzusehen, es sollte nur möglich sein sie zu überschreiben. Wählt der Anwender diese Option für eine Anwendung, so bietet ihm die Anwendung eine Ansicht wie auch bei der Installation der Anwendung und speichert die neuen Credentials in der lokalen Datenbank der Middleware.

Das Konfigurations Modul deckt auch Anwendungsfall 6 (3.3.6) ab, indem es die Möglichkeit bietet, den Kontext der Middleware zu setzen.
In den funktionalen Anforderungen wurden die aus den unterschiedlichen Arten des Kontextwechsels resultierenden Folgen beschrieben 3.4.1. Auch wenn der transparente Kontextwechsel technisch machbar ist und für den Anwender einen gewissen Komfort bieten kann, bietet die Middleware nur einen manuellen Kontextwechsel. Ein transparenter Kontextwechsel würde die Gefahr eines höheren Energieverbrauchs, sowie ungewollte Verbindungs-Kosten mit sich bringen. Durch den manuellen Kontextwechsel behält der Anwender die volle Kontrolle.
Ändert der Anwender den Status der Middleware, wird die Queue auf noch nicht übertragene Elemente untersucht. Fällt die Suche positiv aus, so wird der Anwender darauf hingewiesen und auf Wunsch eine Synchronisation durchgeführt.

Das in Anwendungsfall 7 (3.3.7) beschriebene Starten der Synchronisierung wird ebenfalls durch das Konfigurations Modul abgedeckt. Befindet sich die Anwendung im offline Kontext, wird der Anwender informiert, dass er zunächst in den Online Kontext wechseln muss. Im online Kontext wird das Synchronisierungs Modul 4.1.9 benachrichtigt eine Synchronisierung durchzuführen.
Weiterhin bietet dieser Teil der Konfigurations Anwendung die Möglichkeit zu definieren, ob und in welchen Abständen, eine automatische Synchronisierung im online Kontext durchgeführt werden soll.

4.2 Sichten

Eine Software Architektur lässt sich nicht mit nur einem Diagramm darstellen. Verschiedene Arten von Diagrammen, sogenannte Sichten auf eine Architektur, ermöglichen die Darstellung und das Herausarbeiten unterschiedlicher Schwerpunkte. Die Sichten mit ihren einzelnen Ausprägungen repräsentieren als Ganzes die Systemarchitektur der Middleware.

Die Details der internen Abläufe der Middleware sind Teil der Realisierung (5) und unabhängig vom Konzept der Middleware. In diesem Abschnitt werden die allgemeingültigen und relevanten Aspekte der Middleware Architektur veranschaulicht.

Der folgende Abschnitt bietet ein Aktivitätsdiagramm des Ablaufs einer Anfrage an die Middleware. Dieses veranschaulicht, wie die Middleware intern eine Anfrage bearbeitet (4.2.2).
Die erste Sicht (Abbildung 4.2.1) zeigt die logische Anordnung der in (4.1) definierten Module der Middleware.
Im nächsten Abschnitt wird das korrespondierende Sequenzdiagramm beschrieben. Es zeigt das von außen wahrnehmbare Verhalten der Middleware in einer zeitlichen Abfolge (4.2.3).
Den Abschluss bildet die Datensicht (4.2.4), in welcher das lokale Domänen Modell der Middleware aufgezeigt wird.

4.2.1 Module

Durch die Bildung von Modulen erhält eine Software Architektur klar voneinander abgegrenzte Software Komponenten. Weiterhin bieten Module die Möglichkeit, einer bewusst unscharfen Blickweise auf Details einer Implementierung. Module lassen die im Detail komplexe Verhaltensweise einer Software Komponente simpel erscheinen und helfen somit, die gesamt Architektur leichter zu verstehen.
In diesem Abschnitt sollen die zuvor (4.1) definierten Module der Middleware grafisch aufbereitet und logisch angeordnet werden.
In Schaubild 37 ist das System als Ganzes zu sehen. Logisch oberhalb des Systems sind die Code on Demand Anwendung, respektive der Browser. Unterhalb des Systems befindet sich das lokale Dateisystem, bzw. das mobile Betriebssystem.

Im oberen Bereich der Grafik sind hellblau eingefärbt die Schnittstelle zu den Anwendungen zu sehen. Die Anfrage geht über das HTTP Interface an den Request Dispatcher, welcher die Anfrage an das entsprechende Modul weiterleitet. Gelb, rot und grün sind die drei Request Module mit ihren Sub-Modulen dargestellt, sie werden vom Request Dispatcher aufgerufen. Unterhalb und in grau ist die Persistierungs Schicht dargestellt, auf die aus den Request Modulen heraus zugegriffen werden kann. Im rechten Bereich des Bildes ist der Credential Store und das Sync Module abgebildet. Auf beide kann nicht über das HTTP

Interface zugegriffen werden.

4.2.2 Ablauf einer Anfrage (Aktivität)

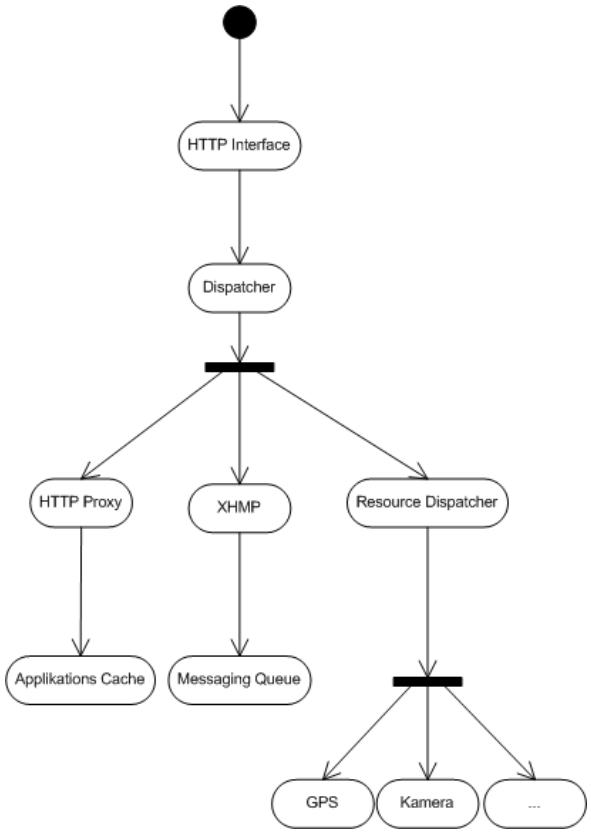


Abbildung 39: Aktivitätsdiagramm für eine Anfrage an die Middleware

Basierend auf der Definition der Module der Middleware (4.1) kann nun der Ablauf einer Anfrage als Aktivitätsdiagramm dargestellt werden (siehe Abbildung 39).
Der Startzustand einer Anfrage beginnt immer mit dem Eingang einer Anfrage über das HTTP Interface. Das HTTP Interface reicht diese an den Request Dispatcher weiter.
Der Dispatcher prüft zunächst, ob die aufrufende Anwendung für den Zugriff autorisiert ist. Als nächstes entscheidet der Dispatcher, für welches Modul die Anfrage bestimmt ist und leitet diese weiter. Es bestehen drei Möglichkeiten: es ist eine Code und Layout Anfrage für den HTTP-Proxy, eine Nachricht für den XML-HTTP Messaging Proxy oder es handelt sich um eine Anfrage nach einer lokalen Ressource, die dann an den Resource Dispatcher weitergeleitet wird.

Handelt es sich um eine Anfrage für den HTTP Proxy, so lädt dieser zunächst den Code und die Layout Daten aus dem Applikations Cache. Im Folgenden transformiert der Applikations Cache die Daten und bereitet sie für die Rückgabe an den Aufrufer vor.
Erkennt der Dispatcher, dass die Anfrage für den XHMP bestimmt ist, so wird diese in die lokale Messaging Queue eingefügt und später asynchron, unabhängig von dem Aufruf übertragen.
Anfragen nach lokalen Ressourcen gelangen zunächst zum Resource Dispatcher, welcher dann ähnliche dem Request Dispatcher die Anfragen an Sub-Module weiterleitet. Im konkreten Fall stehen im Diagramm GPS und Kamera zur Verfügung, es sind jedoch noch weitere Sub-Module denkbar.

4.2.3 Ablauf einer Anfrage (Sequenz)

Das Verhalten einer Anwendung zur Laufzeit lässt sich am besten durch ein Sequenzdiagramm darstellen. Das Sequenzdiagramm auf der Abbildung 40 zeigt zwei Aufrufe aus einer Code on Demand Anwendung im Browser an die Middleware.

Der erste Aufruf stellt die Anfrage einer Code on Demand Anwendung bezüglich einer lokalen Ressource, hier das aktuelle Bild der integrierten Kamera, dar.
Bei Eintritt in die Sequenz ist der Browser geöffnet und die Middleware ist aktiv (3.3.2) und im offline Kontext, es besteht keine Verbindung zum Internet. Der Ablauf beginnt damit, dass der Browser den Code und die Layout Daten der anzuzeigenden Code on Demand Anwendung über eine lokale URL von der Middleware anfordert. Die Middleware ermittelt und transformiert die Daten aus dem Applikations Cache und sendet die generierten Daten an den Browser zurück. Der Browser rendert die Seite und startet die eingebettete Anwendung. Es folgt eine Benutzerinteraktion (`click()`), die in der Anwendung eine Anfrage an die Middleware auslöst. Beispielhaft wird im Diagramm das aktuelle Bild der integrierten Kamera angefordert, die Middleware liefert das Bild zurück.
Der zweite Aufruf der Middleware läuft in die andere Richtung: die Anwendung respektive der Browser, sendet eine Bild Datei an die Middleware, welche asynchron an den Server übertragen werden soll. Die Middleware speichert die Bild Datei zwischen, bis sie sich wieder im Online Kontext befindet und überträgt diese dann zeitversetzt. Die Anwendung ist in der Zwischenzeit vom Anwender beendet worden.

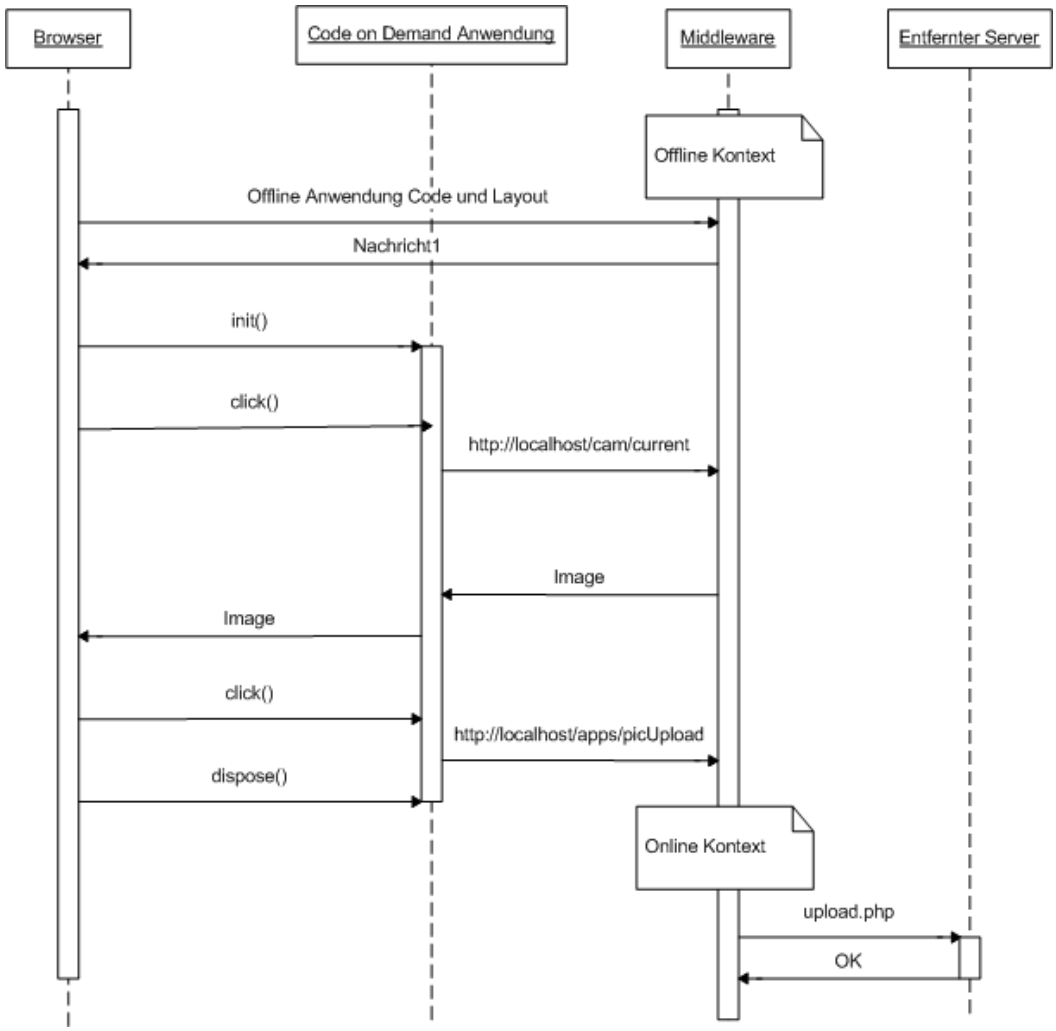


Abbildung 40: Sequenzdiagramm der Middleware im offline Kontext

4.2.4 Daten Sicht

Die Middleware hält die Daten lokal in einer relationalen Datenbank vor, auf die mit der Datenbanksprache Structured Query Language (kurz: SQL) zugegriffen werden kann. Die grafische Darstellung des Daten Modells wird in diesem Abschnitt mittels Entity-Relationship-Diagrammen (ERD) vorgenommen.

Die ERD auf Abbildung 41 zeigt die Relationen für die in der Middleware installierten Anwendungen. Im Zentrum steht hier die Anwendung selber, die in der Relation `application` modelliert ist. Die zu der Anwendung gehörigen Daten, im Detail die Credentials, Endpunkte,

Berechtigungen, Code- und Layout Daten, wurden in separaten Relationen modelliert.
Im Uhrzeigersinn, oben beginnend, ist die erste separierte Relation `application_credential`. Diese Mapping Relation verbindet `application` und `credentials`, es können mehrere Credentials für eine Anwendung gespeichert werden. Die Art des Credentials wurde nach `credential_types` ausgelagert und wird von `credential` referenziert.
Rechts im ERD sind die Endpunkte einer entfernten Webanwendung modelliert. Auch hier wird eine Mapping Tabelle (`application_endpoint`) für die Abbildung der Verknüpfung verwendet; verknüpft werden hier `applications` und `endpoints`. Die Endpunkte können mehrere Parameter haben, daher wurden diese sind ebenfalls ausgelagert und in `endpoint_param` modelliert.
Auf dem ERD unterhalb von `applications` wurden die Berechtigungen in `rights` modelliert, die über `application_right` verknüpft werden. Die Einträge in `rights` sind Singletons, ein Eintrag in `application_right` stellt dar, dass die verknüpfte Anwendung das Recht hat, welches im korrespondierenden `rights` abgebildet ist.
Die Links im ERD positionierten Relationen `application_code` und `application_layout` korespondieren mit den `cod-code` und `cod-layout` Elementen des bei der Installation verwendeten Installations Deskriptors. Zu jeder Anwendung existiert genau ein Eintrag in der Code und in der Layout Relation.

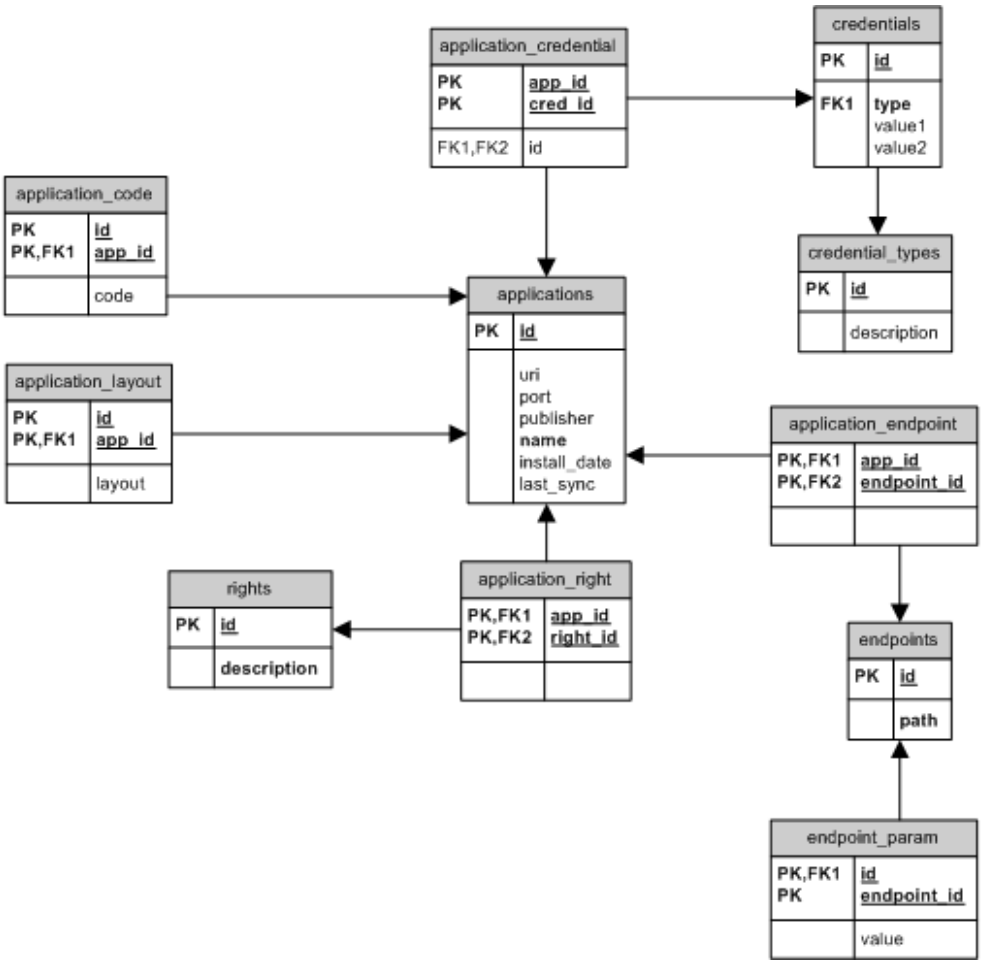


Abbildung 41: Entity-Relationship-Diagramm Applikation

4.3 Application Programming Interface

Damit Code on Demand Anwendungen auf die Middleware zugreifen können, muss diese eine programmierbare Schnittstelle, ein sogenanntes Application Programming Interface (API), anbieten.

In der Zielsetzung zu Beginn dieser Arbeit wurde festgelegt, dass die Middleware auf offenen Standards aufbauen und sich transparent in die vorhandenen Nachrichten- und Protokollstrukturen einfügen soll. Daher wurden bereits in dem Grundlagen Kapitel die beiden in Frage kommenden Standards für die Kommunikation zwischen Middleware und Code on Demand Anwendungen vorgestellt: Web Services (siehe Abschnitt 2.5) und REST (siehe Abschnitt 2.6). Bei der Gegenüberstellung dieser beiden Architekturstile wurde schnell klar, dass REST als schlanke und effiziente Lösung auf beschränkten Systemen von Vorteil ist.

Aus diesem Grund bietet die Middleware eine REST Schnittstelle.
Für den Datenaustausch zwischen einer Code on Demand Anwendung und der Middleware wird JSON verwendet. JSON wurde ebenfalls bereits im Grundlagen Kapitel vorgestellt (siehe Abschnitt 2.7). JSON ist eine schlanke und performante Lösung und daher auf einem mobilen System mit begrenzten Ressourcen einer schwergewichtigen XML Struktur vorzuziehen.

Code on Demand Anwendungen können das auf REST basierende API über lokale URIs zugreifen. Eine Code on Demand Anwendung kann dieses API nutzen, um asynchrone Daten zu speichern und auf lokale Ressourcen zuzugreifen. Die Code on Demand Anwendungen selbst werden ebenfalls über dieses API verfügbar gemacht.
Das API ist in zwei Abschnitte unterteilt, das Resource API und das Messaging API. Das Resource API stellt Inhalt zur Verfügung, wobei auch eine Anwendung selbst als Inhalt verstanden wird. Das Messaging API steht für die asynchrone Server Kommunikation und bietet installierten Anwendungen so die Möglichkeit auch im offline Kontext zu funktionieren.

Das API ist über das Hypertext Transfer Protocol (HTTP) auf Port 8080 der lokalen Netzwerkadresse erreichbar. Aus dieser Definition folgt folgender Rumpf, auf dem das API aufbaut:

```
1 http://localhost:8080/
```

Listing 18: Basis Adresse des API

Gefolgt wird der Rumpf von einem Schlüssel, welcher der Anwendung beim Laden von der Middleware übergeben wurde. Dieser Schlüssel wird verwendet, um die Anwendung beim Aufruf gegenüber der Middleware zu authentifizieren. Einzige Ausnahme ist das Application API, da bei diesem Aufruf noch kein Schlüssel vorhanden ist, lautet hier der Schlüssel immer `apps`.

```
1 <!-- Standard Zugriff auf die Middleware -->
2 http://localhost:8080/<key>/
3
4 <!-- Zugriff auf Application API -->
5 http://localhost:8080/apps/
```

Listing 19: Sitzungsschlüssel als API Parameter

4.3.1 Application API

Jede Anwendung besitzt einen eindeutigen Identifizierer, den sie selbst im Installations Deskriptor definiert (siehe `id` in Listing 17). Anhand dieses Identifzierers kann eine in der Middleware installierte Anwendung in den Browser geladen werden. Hierfür muss lediglich die entsprechende Adresse in der Adressleiste des Browsers eingegeben oder über ein Lesezeichen aufgerufen werden.

```
1 <!-- Laden einer Anwendung über den eindeutigen Identifizierer -->
2 http://localhost:8080/apps/<cod-id>
```

Listing 20: Laden einer Anwendung von der Middleware

4.3.2 Resource API

Über das Resource API stellt die Middleware lokale Ressourcen der Wirtsplattform zur Verfügung. Für den Zugriff wird ein gültiger Sitzungsschlüssel benötigt und die Anwendung muss für den Zugriff auf die Anwendung autorisiert sein.
Der Rumpf des Resource API hat die Form:
`http://localhost:8080/<key>/resource/<resource-id>`
Die `resource-id` ergibt sich aus den folgenden Definitionen.

4.3.2.1 Kamera

`/resource/cam/image`
Dieser Aufruf bewirkt das Öffnen des Kameradialogs und bietet so die Möglichkeit, ein Foto aufzunehmen. Das aufgenommene Foto wird anschließend zurückgeliefert.

`/resource/cam/imagepath`
Dieser Aufruf bewirkt das Öffnen des Kameradialogs und bietet so die Möglichkeit, ein Foto aufzunehmen. Das aufgenommene Foto wird anschließend im persistenten Ordner der aufrufenden Anwendung gespeichert und der relevante Pfad zu dem Bild zurückgeliefert.

4.3.2.2 PDA

Die Ressource PDA ermöglicht den Zugriff auf den Kalender des Gerätes, ebenso wie die lokale Kontakt Datenbank. Für den Austausch von Termin und Kontakt Daten werden JSON Objekt verwendet. Das Kontakt Objekt ist in Listing 21 und das Termin Objekt in Listing 22 exemplarisch dargestellt.

```
{
    "firstName"     : "Max",
    "lastName"      : "Mustermann",
    "address": {
        "streetAddress"     : "Rathausmarkt 1",
        "postalCode"        : "D-20000",
        "city"              : "Hamburg"
    },
    "email"         : "max@noreply.com",
    "phoneNumbers": [
        "+49-172-123456",
        "+49-40-123456"
    ]
}
```

Listing 21: Ein Kontakt als JSON Objekt

```
{
    "title"     : "Abgabe Master Thesis",
    "location" : "HAW Hamburg bei Frau Mertens",
    "date"      : "2008-09-19",
    "begin"     : "10:00 am",
    "end"       : "10:30 am"
}
```

Listing 22: Ein Termin als JSON Objekt

Alle Datumsangaben werden im erweiterten Datumsformat nach DIN ISO 8601:2006-09 dargestellt (YYYY-MM-DD).

`/resource/pda/day`

Liefert die Termine des aktuellen Tages als Liste von Termin Objekten

`/resource/pda/day/<date>`

Bekommt als Parameter ein Datum übergeben und liefert eine Liste mit Termin Objekten des definierten Datums zurück.

`/resource/pda/create/<dateObject>`
Bekommt als Parameter ein Termin Objekt (siehe Listing 22) übergeben und trägt diesen in den lokalen Kalender ein.

`/resource/pda/contacts/name/[name]`
Liefert eine Liste von Kontakt Objekten (siehe Listing 21) entsprechend des übergebenen Namens zurück.

`/resource/pda/contacts/namfirstchar/[char]`
Liefert eine Liste von Kontakt Objekten (siehe Listing 21) entsprechend des übergebenen Anfangsbuchstabens zurück.

`/resource/pda/contacts/add/<personObject>`
Bekommt als Parameter ein Kontakt Objekt (siehe Listing 21) übergeben und trägt diesen in die lokale Kontakt Datenbank ein.

4.3.2.3 System

Die Ressource System liefert Informationen zu dem lokalen System zurück.

`/resource/system/operatingSystem`
Liefert den Namen des lokalen Bertriebssystems mit der Versionsangabe.

`/resource/system/ownerName`
Liefert den hinterlegten Benutzernamen des Gerätes zurück.

`/resource/system/bateryLevel`
Liefert den prozentuellen Status des Geräte Akkus.

4.3.2.4 Telefon

Ist das mobile Gerät ein Mobiltelefon, so bietet die Ressource Telefon den Zugriff auf Telefonfunktionen.

`/resource/fon/phoneOperatorName`
Ist das Telefon in einem Mobilfunknetz eingebucht, wird der Namen des Mobilfunkanbieters zurückgeliefert.

`/resource/fon/phoneAvailable`
Liefert den Wert `1` zurück, wenn das Telefon in einem Mobilfunknetz eingebucht ist, sonst `0`.

`/resource/fon/call/[number]`
Bietet die Möglichkeit, einen Anruf zu der übergebenen Nummer zu initiieren.

`/resource/fon/call/list/[incoming|outgoing]`
Liefert ein String Array im JSON Format der letzten 10 eingehenden oder ausgehenden Anrufe.

`/resource/fon/sms/send/[number]/[message]`
Legt eine Kurzmitteilung in den Kurzmitteilungs Ausgangsordner.

4.3.2.5 GPS

Besitzt das Gerät ein GPS Modul, so stellt die Ressource GPS einen Zugriff auf dieses bereit

`/resource/gps/activated`
Liefert den Wert `1` zurück, wenn das GPS Modul aktiviert ist, sonst `0`.

`/resource/gps/status/[activate|deactivate]`
Bietet die Möglichkeit, das GPS Modul zu aktivieren und zu deaktivieren.

`/resource/gps/visibleSat`
Liefert die Anzahl der für das Gerät sichtbaren Satteliten zurück.

`/resource/gps/currentPosition/json`
Liefert ein nach JSON transformiertes GPS Exchange Format Dokument (gpxschema, 2004) der aktuellen Position zurück (siehe Listing 23).

```
{
    "lat"       : "53.550556",
    "lon"       : "9.993333",
    "ele"       : "3.423751",
    "name"      : "Hamburg",
    "sym"       : "City"
}
```

Listing 23: Ein in JSON transformiertes GPS Exchange Format Dokument

4.3.3 Messaging API

Das Messaging API bietet installierten Anwendungen die Möglichkeit, Nachrichten an entfernte Webserver zu übertragen, auch wenn diese gerade nicht verfügbar sind.

`/messaging/<messageObject>`
Speichert das übergebene JSON Nachrichten Object (siehe Listing 24) in der lokalen Messaging Queue.

```
{
    "endpoint-id" : "rpc-add",
    "params"      : [
            {
                "id" : "title",
                "value" : "Hallo Welt"
            }
            {
                "id" : "text",
                "value" : "Dies ist ein Text!"
            }
            {
                "id" : "tags",
                "value" : "Middleware, Test, Foo"
            }
```

```
16        ]
17    }
```

Listing 24: Ein JSON Nachrichten Objekt

Es ist zu beachten, dass der im Nachrichten Objekt deklarierte Endpunkt einem Endpunkt der aufrufenden Anwendung gemäß ihren im Installations Deskriptor (17) definierten Endpunkten entsprechen muss.

`/messaging/lastsync`
Liefert den Zeitpunkt der letzten Synchronisation zurück.

`/messaging/issyncing`
Liefert den Wert `1` zurück, wenn eine Synchronisation gerade durchgeführt wird, sonst `0`.

`/messaging/isonline`
Liefert den Wert `1` zurück, wenn sich die Middleware aktuell im online Kontext befindet, sonst `0`.

`/messaging/queuesize`
Liefert die aktuelle Anzahl der Nachrichten der aufrufenden Anwendungen in der Queue zurück.

5 Realisierung

Im Rahmen dieser Masterarbeit wurde ein Prototyp der beschriebenen Middleware entwickelt. Dieser Prototyp dient als Proof of Concept der entwickelten Lösungsansätze und kann als Basis für künftige Entwicklungen verwendet werden.

Der Prototyp besteht aus zwei Komponenten:

- Middleware Software für ein mobiles Endgerät
- Beispielanwendungen

Die Middleware für ein mobiles Endgerät ist eine protypische Umsetzung des im Rahmen dieser Arbeit spezifizierten Konzeptes für eine Middleware. Der Prototyp deckt nicht den kompletten Funktionsumfang ab, zeigt jedoch die Umsetzbarkeit der entwickelten Lösung. Der Prototyp ist in der Lage Anwendungen offline auszuführen, Nachrichten zeitversetzt an einen entfernten Server zu übertragen und bietet Zugriff auf ausgewählte, lokale Ressourcen.
Im Prototyp enthalten sind zwei Beispielanwendungen, UbiFlickr (siehe Abschnitt 5.3.1) und UbiBlog (siehe Abschnitt 5.3.2).

Dieses Kapitel wird zunächst einen Überblick der bei der Entwicklung des Prototypen verwendeten Technologien geben (Abschnitt 5.1). Im Anschluss folgt eine Einführung in die Bedienung des Prototypen (Abschnitt 5.2). Die Bereits kurz vorgestellten Beispielanwendungen werden in Abschnitt 5.3 ausführlicher vorgestellt. Abschließend wird in Abschnitt 5.4 die Integration der Middleware in Lösungen von Drittanbietern beschrieben.

5.1 Technologien

Der Prototyp wurde basierend auf dem *Microsoft .NET Compact Framework 3.5*[22] mit der Programmiersprache C# entwickelt. Für die Entwicklung wurde die Entwicklungsumgebung Microsoft Visual Studio 2008 verwendet.
Für die Transformation der .NET Objekte nach JSON wurde die Bibliothek *Json.NET*[23] eingesetzt. Für den Umgang mit XML-RPC Aufrufen wurde die Bibliothek XML-RPC.NET[24] genutzt.

[22] http://msdn.microsoft.com/de-de/library/f44bbwa1.aspx - Zugriffsdatum 2008-09-10
[23] http://www.codeplex.com/Json - Zugriffsdatum: 2008-09-10
[24] http://www.xml-rpc.net/

Die Beispielanwendung UbiFlickr verwendet das *Flickr XML-RPC API*[25], um die Bilder auf die Online-Fotoplattform zu übertragen.
Die Beispielanwendung UbiBlog verwendet das *Yahoo! Maps Web Services - AJAX API* [26] für die Anzeige der Position und das Wordpress[27] XML-API um die Daten an die Blog Software zu übergeben.
Alle anderen Komponenten sind Teile des Betriebssystems oder selbst entwickelt.

Die Middleware ist grundsätzlich auf allen Smartphones und PDAs lauffähig, die vom .NET Compact Framework in der Version 3.5 unterstützt werden. Getestet und entwickelt wurde die Middleware auf einem PDA mit Windows Mobile 6.1 Professional bzw. den entsprechenden Emulatoren.

5.2 Verwendung des Prototyps

Im Folgenden wird ein grundlegender Überblick über den Prototypen gegeben. Weitere Informationen, der Quellcode, sowie eine Installationsanleitung befinden sich im Unterverzeichnis `/prototyp/` des Datenträger zu dieser Arbeit.

5.2.1 Starten der Middleware

Die Middleware kann über eine entsprechende Verknüpfung im Menü oder durch ausführen der `master.exe` Datei im Installationsverzeichnis gestartet werden.
Es öffnet sich zunächst ein Dialog, in dem der Anwender die Middleware explizit starten kann. Erst wenn in dem Dialog die Middleware gestartet wurde, ist diese aktiv und die installierten Anwendungen erreichbar. Ist die Middleware erfolgreich gestartet, öffnet diese den Browser und leitet ihn auf die lokale URL `http://localhost:8080/apps/welcome`, dies ist die Startseiten-Anwendung der Middleware (5.2.2).

5.2.2 Startseite

Die Startseite der Middleware ist selbst eine Anwendung, welche in der Middleware installiert ist.

[25] http://www.flickr.com/services/api/request.xmlrpc.html - Zugriffsdatum 2008-09-10
[26] http://developer.yahoo.com/maps/ajax/ - Zugriffsdatum 2008-10-09
[27] http://www.wordpress.org - Zugriffsdatum 2008-09-14

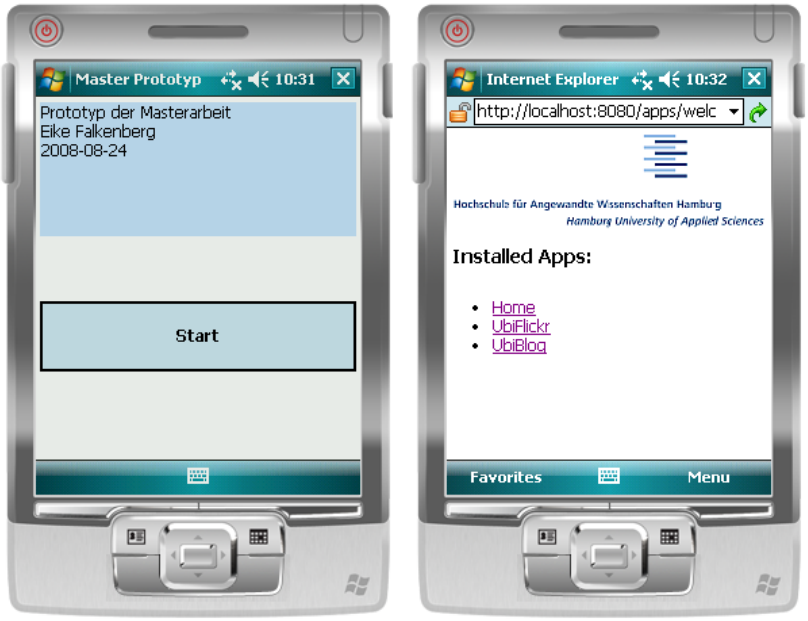


Abbildung 42: Prototyp - Staren der Middleware

Abbildung 42 zeigt die Startseiten Anwendung der Middleware, welche die installierten Anwendungen anzeigt. Klickt der Anwender auf eine der angezeigten Anwendungen, wird diese im Browser aufgerufen, aus der Middleware geladen und im Browser ausgeführt.

5.3 Beispielanwendungen

Zum Prototyp der Middleware gehören zwei Beispielanwendungen, UbiFlickr (siehe 5.3.1) und UbiBlog (siehe 5.3.2). Die beiden Beispielanwendungen werden in diesem Unterkapitel vorgestellt.

5.3.1 UbiFlickr

Die Startseite der Middleware (5.2.2) ist als Anwendung der Middleware vorinstalliert. Eine weitere vorinstallierte Anwendung ist *UbiFlickr*. Über die lokale Adresse `http://localhost:8080/apps/ubiflickr` lässt sich die Anwendung *UbiFlickr* starten. Diese Anwendung ermöglicht es, Fotos mit der integrierten Kamera des Gerätes zu machen und diese anschließend auf die Online-Fotoplattform Flickr[28] zu laden.

[28] http://www.flickr.com - Zugriffsdatum: 2008-09-10

Abbildung 43: Prototyp - Beispielanwendung UbiFlickr

Abbildung 43 zeigt die Oberfläche von UbiFlickr. Der Screenshot auf der linken Seite zeigt die Anwendung nach dem Start. Klickt der Anwender auf das Kamerasymbol, wird eine asynchrone Nachricht an die Middleware gesendet und der Windows Mobile Kameramodus gestartet. Der Kameramodus ist auf dem mittleren Screenshot zu sehen. Hat der Anwender ein Foto gemacht, speichert die Middleware das Foto und liefert einen eindeutigen Bezeichner für das Foto über das HTTP Interface an die Anwendung zurück. Die Anwendung ruft das neue Foto durch einen zusätzlichen Aufruf ab.
Der Screenshot rechts im Bild entspricht dem Erscheinungsbild von UbiFlickr, nach dem zurückkehren aus dem Kameramodus. Die Anwendung bietet die Möglichkeit, dem neuen Foto beschreibende Attribute zuzuordnen (sogenannte *Tags*) und dieses anschließend an Flickr zu übertragen.

Die Übertragung des Bildes an Flickr geschieht über die Replikation der Middleware an das Flickr REST-API[29].

[29] http://www.flickr.com/services/api/request.rest.html - Zugriffsdatum 2008-09-13

5.3.2 UbiBlog

UbiBlog ist eine Weiterentwicklung von UbiFlickr. Die Anwendung ermöglicht eine umfangreichere Texteingabe, außerdem kann die Anwendung die aktuelle Position durch das GPS Modul abfragen. Die Positionsdaten werden verwendet, um daraus eine Kartenansicht zu generieren.

UbiBlog funktioniert grundsätzlich ähnlich wie UbiFlickr. Besonders ist hier, dass durch einen

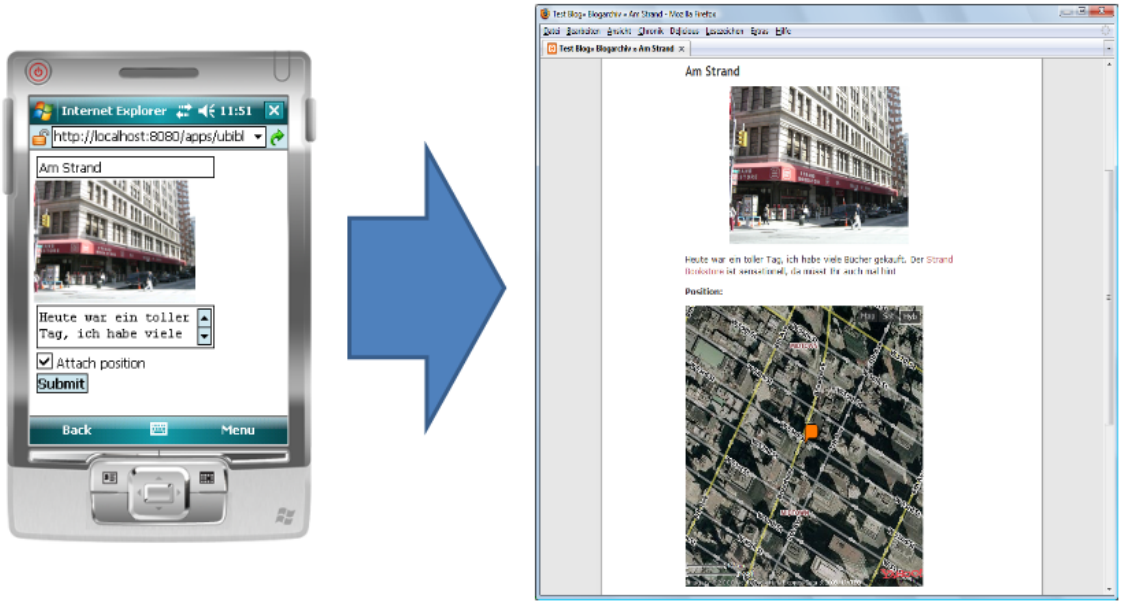


Abbildung 44: Prototyp - Beispielanwendung UbiBlog

zusätzlichen Aufruf die Positionsdaten eingebunden werden. Das Ergebnis ist rechts in Abbildung 44 zu sehen: ein Blog Eintrag mit dem Foto und einem Satellitenbild, das auf die Position zentriert ist.

5.4 Integration der Middleware

Am Beispiel des Prototypen soll an dieser Stelle ein Überblick über die mögliche Integration der Middleware in eigene Lösungen gegeben werden.

Damit eine Code on Demand Anwendung die Middleware nutzen kann, muss sie in der Midddleware als Anwendung installiert sein. Voraussetzung ist ein Installations Deskriptor und die Installation durch den Anwender (siehe Anwendungsfall *Installation einer Anwendung* in Abschnitt 3.3.3). Die Anwendung kann dann die Middleware über das API (wie in

Abschnitt 4.3 definiert) ansprechen.
Solch eine Anwendung kann auch hybrid sein, also nur für Teile der Anwendung auf die Dienste der Middleware zugreifen. Am Beispiel des Prototypen könnte eine solch hybride Anwendung eine AJAX Messenger Anwendung sein, die auch das Versenden von Bildern ermöglicht. Als reine online Anwendung würde die Kommunikation von den AJAX eigenen Technologien, vorbei an der Middleware, gehändelt werden. Die Anwendung nutzt jedoch die Middleware, um auf lokale Bilddateien zuzugreifen und diese an Nachrichten anzuhängen.

Damit eine Anwendung der Middleware auf einem mobilen Endgerät ausgeführt werden kann, muss auf diesem die Middleware installiert sein, respektive installiert werden. Ein Anbieter einer Anwendung der Middleware sollte neben seiner Anwendung auch immer die Middleware mitliefern, so dass diese optional mit installiert werden kann. Dies entspricht dem Vorgehen, wie es bei Laufzeitumgebungen üblich und dem Anwender bekannt ist. Für eine Java Anwendung wird eine Java Laufzeitumgebung benötigt, welche meist vom Anbieter der entsprechenden Anwendung mit ausgeliefert wird.
Eine weitere Möglichkeit besteht darin, die Anwendung integriert in der Middleware auszuliefern. Der zweite Ansatz bietet Sicherheit, dass die Middleware in einer kompatiblen Version vorhanden ist. Dieses Vorgehen birgt aber das Problem, dass nach n-Anwendungsinstallationen n-Instanzen der Middleware auf dem Gerät sind, dies kann unter anderem zu Speicherplatz Problemen führen.
Grundsätzlich empfiehlt der Autor die erste Lösung, die zweite sollte nur gewählt werden, wenn lediglich eine einzelne Middleware Anwendung auf dem Gerät sein wird.

6 Fazit

Zum Abschluss der Arbeit wird im ersten Teil dieses Kapitels eine rückblickende Zusammenfassung gegeben (siehe 6.1). Daran folgt die Bewertung der definierten Ziele der Arbeit (Abschnitt Ergebnisse 6.2).
Auf die Bewertung der Ergebnisse folgt der Ausblick (Abschnitt Ausblick 6.3) auf künftige Arbeiten.

6.1 Zusammenfassung

Im Rahmen dieser Arbeit wurde eine Middleware entwickelt, um reichhaltige Code on Demand Anwendungen auf mobilen Plattformen als Ersatz für lokal installierte Anwendungen zu ermöglichen.
Durch die entwickelte Middleware wird es möglich, mobile Code on Demand Anwendungen in einem Browser auszuführen, auch wenn keine Verbindung zum Internet verfügbar ist. Durch die angebotene Nachrichtenreplikation können installierte Anwendungen Nachrichten an die Middleware senden, die bei Wiederverfügbarkeit einer Internetverbindung an einen entfernten Webserver übertragen werden. Zusätzlich stellt die Middleware lokale Ressourcen über eine Schnittstelle zur Verfügung und ermöglicht es installierten Code on Demand Anwendungen so, auf diese zuzugreifen. Der Zugriff auf lokale Ressourcen, wie beispielsweise das Dateisystem oder eine integrierte Kamera sind sonst nur lokal installierten Anwendungen vorbehalten.

Beginnend mit einem Überblick über die relevanten mobilen Hardware und Software-Plattformen wurden in Kapitel 2 die Grundlagen vermittelt, die für das Verständnis der Arbeit relevant sind. Es folgte eine genauere Beschreibung von Windows Mobile, da dies Grundlage des Prototypen ist. Ein Überblick über die Evolution der Webanwendungen schließt mit mobilen Webanwendungen und zeigt dessen Probleme, die im Besonderen Relevant für diese Arbeit sind. Nachdem die Technologien REST und JSON beschrieben wurden, schließt eine Übersicht existierender Lösungsansätze das Kapitel ab.

Im Analyse Kapitel 3 wurde zunächst eine Beispielanwendung der zu entwickelnden Lösung definiert. Durch die Beispielanwendung bekommt der Leser ein besseres Gefühl für den praktischen Nutzen der Middleware, weiterhin dient diese bei der darauf folgenden Anforderungsanalyse.
Es folgt eine Meta Architektur, also eine Sicht auf das System und dessen Umfeld. Als

nächsten Annäherungsschritt wurden die Anwendungsfälle definiert.
Die bis dahin gewonnenen Erkentnisse wurden genutzt, um die funktionalen und nichtfunktionalen Anforderungen an die Middleware zu spezifizieren.

Auf den Abschluss des Analyse Kapitels folgt das Entwurfs Kapitel 4. In diesem Kapitel werden zunächst die einzelnen Module der Middleware definiert. Die anschließenden Sichten führen zu einer Architektur der Middleware. Abschließend wird das API definiert, über das Anwendungen auf die Middleware zugreifen können.

Der im Rahmen dieser Arbeit entstandene Prototyp wird in Kapitel 5 vorgestellt. Zunächst werden die verwendeten Technologien beschrieben, anschließend die Verwendung des Prototypen und dessen Beispielanwendungen erläutert. Den Abschluss bildet eine Beschreibung, wie die Middleware am Beispiel des Prototypen in eigene Lösungen integriert werden kann.

Den Abschluss der Arbeit bildet das Fazit Kapitel 6. Nach dieser Zusammenfassung folgt eine Gegenüberstellung der definierten Ziele mit dem Erreichten. Den Abschluss des Kapitels und somit auch dieser Arbeit bildet der Ausblick auf künftige Arbeiten und Themefelder.

6.2 Ergebnisse

Im ersten Kapitel wurden im Abschnitt Zielsetzung (siehe 1.2) folgende Ziele für die Entwicklung einer Middleware als lokalen Anwendungscache für Code on Demand Anwendungen definiert:

- Offline Fähigkeit und Zugriff auf relevanten lokale Ressourcen müssen angeboten werden
- Plattformunabhängigkeit sowohl von der Wirts-Plattform als auch von der Code on Demand Technologie
- Transparente Einbettung in die vorhandenen Nachrichten- und Protokollstrukturen durch Verwendung offener Standards
- Entwicklung eines Prototyps als Proof of Concept

Im folgenden wird auf die einzelnen Ziele eingegangen.

6.2.1 Offline Fähigkeit und Zugriff auf relevante Ressourcen

Im Abschnitt Motivation (siehe 1.1) wurde herausgearbeitet, warum die fehlende offline Fähigkeit für Code on Demand Anwendungen ein Problem ist. Diese Herausforderung zu bewältigen war eine der Hauptmotivationen für diese Arbeit. Daher ist die offline Fähigkeit für Code on Demand Anwendungen eine elementare Aufgabe der in dieser Arbeit entwickelten Middleware.
Entsprechend des Anwendungsfalles *Ausführen einer Anwendung* (wie in 3.3.4 beschrieben) ist die Middleware in der Lage, Code on Demand Anwendungen lokal in einem Applikations Cache vorzuhalten (siehe 3.4.5) und diese einem Browser über eine HTTP Schnittstelle anzubieten. Durch diese Schnittstelle wird es möglich, auf einem mobilen Gerät wie beispielsweise einem PDA eine Code on Demand Anwendung auszuführen, obwohl der entfernte Webserver nicht erreichbar ist.
Das Bereitstellen solch einer Anwendung alleine genügt jedoch nicht. Fundamentaler Bestandteil von Code on Demand Anwendungen ist in den meisten Fällen das asynchrone Senden von Nachrichten an einen entfernten Server. Die Middleware bewältigt diese Aufgabe durch einen XML-HTTP Messaging Proxy (siehe 3.4.4), welcher die Nachrichten in einer interenen Messaging Queue zwischenspeichert und zeitversetzt an den Server überträgt.
Durch den Applikations Cache und den XML-HTTP Messaging Proxy können mit der Middleware Code on Demand Anwendungen offline ausgeführt werden.

In der Motivation wurde auch das Problem beschrieben, dass Code on Demand Anwendungen keinen oder nur sehr beschränkten Zugriff auf lokale Ressourcen haben. Die Beispielanwendung (siehe 3.1) zeigt auf, wieso Code on Demand Anwendungen auf mobilen Geräten auch Zugriff auf deren Ressourcen benötigen.
Die Middleware bietet definierte Ressourcen der Plattform über lokale URIs an. Da so bewusst relevante Sicherheitsbeschränkungen umgangen werden, ist der Zugriff auf diese lokalen URIs Anwendungen vorbehalten, denen die Berechtigung vom Anwender zugewiesen wurde. Das Zuweisen der Berechtigungen erfolgt im Rahmen der Installation einer Anwendung, wie es im Anwendungsfall *Installation einer Anwendung* (siehe 3.3.3) beschrieben ist.
Die von der Middleware bereitgestellten Services für den Zugriff auf lokale Ressourcen ermöglichen installierten Code on Demand Anwendungen mit den entsprechenden Berechtigungen, auf lokale Ressourcen zuzugreifen, wie es sonst nur lokal installierten Anwendungen vorbehalten ist.

Anhand der Beispielanwendung wurde deutlich, dass auch eine Synchronisation von Daten benötigt wird, die über das reine zeitversetzte Senden von Nachrichten hinaus geht. In den funktionalen Anforderungen im Abschnitt Daten Replikation (siehe 3.4.9) wird definiert, dass zwischen Nachrichten Replikation und Daten Replikation zu unterscheiden ist.
Die Nachrichten Replikation wird durch die interne Queue ermöglicht, die von einem nebenläufigen Dienst bei der Synchronisation abgearbeitet wird.
Die Daten Replikation, die sich auf die Synchronisierung einer lokalen Datenbank mit einer entfernten Datenbank bezieht, wurde im Rahmen dieser Arbeit nicht umgesetzt. Die definierten Ziele, transparente Einbettung in die bestehenden Protokollstrukturen und Plattformunabhängigkeit waren mit einer Lösung im Rahmen dieser Arbeit nicht vereinbar. Diese Funktionalität könnte Basis für eine spätere Arbeit sein (siehe *Ausblick* Abschnitt 6.3).

6.2.2 Plattformunabhängigkeit

Bei der im Rahmen dieser Masterarbeit entwickelten Middleware handelt es sich um ein plattformunabhängiges Konzept. Die Middleware kann prinzipiell auf beliebige Plattformen migriert werden. Einzige Anforderung an die Zielplattform ist eine entsprechende Laufzeitumgebung für die jeweilige Code on Demand Technologie.
Soll die Middleware beispielsweise mit Ajax Anwendungen wirken, wird ein Browser mit Ajax Unterstützung benötigt, für eine Flash Anwendung bedingt es den Flash Player usw.

6.2.3 Transparente Einbettung in vorhandene Strukturen

Die im Rahmen dieser Arbeit entwickelte Middleware ist eine zusätzliche Anwendung auf dem mobilen Gerät. Damit Code on Demand Anwendung mit der Middleware lauffähig sind, müssen diese auf die Middleware angepasst sein. In manchen Situationen kann es weiterhin von Vorteil sein, eine Server Komponente für die Kommunikation mit der Middleware anzupassen.

Im Gegensatz zu der Anwendung müssen vorhandene Strukturen nicht verändert werden. Weder die Ausführungsumgebung (beispielsweise ein Browser), noch die verwendeten Protokolle müssen angepasst werden.
Im Besonderen Relevant jedoch ist, dass die Middleware für den Anwender transparent sein kann. Ob eine Anwendung Ad-Hoc aus dem Internet geladen oder lokal zwischengespeichert wurde ist für den Anwender kaum wahrnehmbar. Im besten Fall wird ein Anwender

mehrheitlich online sein und die Middleware kann nahezu alle Nachrichten direkt replizieren.

6.2.4 Prototyp

Der entwickelte Prototyp zeigt exemplarisch und in ausgewählten Umfang, die Umsetzung des Konzeptes der in dieser Arbeit entwickelten Middleware. Der Prototyp beweist, dass das Konzept der Middleware grundsätzlich umsetzbar ist, dies war die primäre Motivation für die Prototyp Entwicklung.
Anhand der enthaltenen Beispielanwendungen wird verdeutlicht, dass Anwendungen für reale Anforderungen mit der Middleware umsetzbar sind und das Konzept der Middleware für diese tragfähig ist.

Als problematisch hat sich die zum Teil noch sehr unausgereifte AJAX Unterstützung des Windows Mobile eigenen Browsers IEMobile erwiesen. AJAX wird unterstützt, jedoch fehlen viele Möglichkeiten aus dem JavaScript Sprachumfang, die auf Desktop Browsern zur Verfügung stehen. Ein weiteres Problem ist das Caching Verhalten des IEMobile. Manche AJAX Aufrufe werden zwischengespeichert und kein weiteres mal ausgeführt, auch wenn dies notwendig wäre. Dies sind jedoch allgemeine Probleme des Browsers, unabhängig von der Middleware.

6.3 Ausblick

Kurz vor Fertigstellung dieser Arbeit hat das Fraunhofer Institut für Offene Kommunikationssysteme (FOKUS[30]) bekannt gegeben, dass sie eine Mobile Widget Runtime entwickelt. Diese Runtime soll kleine Programme, welche als Widget bezeichnet werden, auf mobilen Geräten ausführen. Fraunhofer setzt hierfür auf einen speziellen Browser, der als Laufzeitumgebung für die Widgets dient.
Der Einsatz von Fraunhofer und vielen anderen (siehe 2.8) in dem thematischen Umfeld dieser Arbeit zeigt, wie dynamisch und relevant das Thema derzeit ist.
Nach Ansicht des Autors werden allgemeingültige Lösungen, wie sie in dieser Arbeit vorgestellt wurde, die größten Chancen haben, sich durchzusetzen. Diese Middleware ist offen für Drittentwickler und auf vielen Plattformen bereits verfügbar. Anwender können selber Programme erstellen, die dann auf allen Geräten lauffähig sind.

[30] http://www.fokus.fraunhofer.de; abgerufen am 2008-08-30

Am Ende dieser Arbeit stehen noch Themen, die nicht behandelt wurden und sich für künftige Arbeiten anbieten. Es ist denkbar und vom Autor ausdrücklich gewünscht, dass auf die im Rahmen dieser Arbeit entwickelte Middleware künftige Arbeiten aufbauen.
Ein deutlicher Mehrwert für Anwendungen wäre die Möglichkeit, mit einer Script Sprache Callbacks definieren zu können. So könnte eine Code on Demand Anwendung auf asynchrone Server Antworten reagieren, selbst wenn die Anwendung gerade nicht läuft.
Im Rahmen dieser Arbeit wurde definiert, dass die Code on Demand Anwendungen für die Middleware neu geschrieben werden müssen. Hier könnte eine automatisierte Code Transformation für AJAX Anwendungen ansetzen, um Anwendungen kompatibel zu der Middleware zu machen.
Die in dieser Arbeit nicht umgesetzte, generische Replikation der Daten würde Anwendungen ermöglichen, die lokal Daten vorhalten, diese manipulieren und das Resultat zurückliefern.

Glossar

Glossar

AJAX	Asynchronous JavaScript and XML; erweitert JavaScript um asynchronen Nachrichtenaustausch
API	Application Programming Interface; Programmierbare Schnittstelle eines Software Systems
Code on Demand	Prinzip, Anwendungscode nicht lokal auf einem Rechner vorzuhalten, sondern diesen bei Bedarf aus einem Netzwerk herunterzuladen.
ECMAScript	siehe JavaScript
GPS	Global Positioning System; Satellitengestütztes, weltweites System zur Positionsbestimmung
HTML	Hypertext Markup Language; eine XML ähnliche Auszeichnungs Sprache für die Erstellung von Dokumenten. Wird primär für die Erstellung von Webseiten verwendet
JavaScript	Primär in Browsern angebotene und in Webseiten verwendete Scriptsprache
JSON	JavaScript Object Notation
OASIS	Organization for the Advancement of Structured Information Standards; Eine auf Webtechnologien spezialisierte Standardisierungs Organisation
PDA	Personal Digital Assistant; kompakter, portabler Computer
Pocket PC	PDA ohne Tastatur zur Eingabe, mit Touchscreen
REST	Representational State Transfer

RIA	Rich Internet Application; Reichhaltige Internetanwendungen bieten Eigenschaften, die denen lokaler Anwendungen ähnlich sind, wie beispielsweise Ziehen und Fallenlassen (Drag and Drop).
Smartphone	PDA mit Tastatur zur Eingabe, ohne Touchscreen
SOAP	Simple Object Access Protocol; Vom W3C standardisiertes Netzwerkprotokoll auf der Anwendungsschicht. Wird hauptsächlich als Protokoll für Web Services verwendet
UDDI	Universal Description, Discovery and Integration; Standard für einen Verzeichnisdienst
URI	Uniform Resource Identifier; eindeutiger Bezeichner einer Ressource in einem Netzwerk
W3C	World Wide Web Consortium
WSDL	Web Service Description Language; ein Standard für die Beschreibung eines Web Services
XHTML	Auf XML basierender Standard als Nachfolger von HTML
XML	Extensible Markup Language; Auszeichnungssprache

Literaturverzeichnis

Literatur

[7rules 2005] DRUCKER, Elliott: *The 7 Rules of Mobile Data User Experience (Zugriffsdatum:2008-07-13)*. 2005. – URL http://www.drucker-associates.com/MUE.htm

[aes 2001] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY (NIST): *Announcing the Advanced Encryption Standard (AES)*. 2001. – URL http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[AjaxApproach 2005] GARRET, J.: *Ajax: A New Approach to Web Applications (Zugriffsdatum:2008-07-13)*. 2005. – URL http://www.adaptivepath.com/publications/essays/archives/000385.php

[androidSdk 2008] OPEN HANDSET ALLIANCE: *Android SDK (Zugriffsdatum:2008-07-13)*. 2008. – URL http://code.google.com/android/documentation.html

[ArchPractice 2003] BASS, L. ; CLEMENTS, P. ; KAZMAN, R.: *Sofwtare Architecture in Practice*. Second. Addison-Wesley, 2003

[ArchStyleAjax 2007] MESBAH, A. ; DEURSEN, A. van: An Architectural Style for Ajax. In: *The Working IEEE/IFIP Conference on Software Architecture*, IEEE, jan 2007, S. 9

[bloggerApi 2008] GOOGLE: *Blogger API (Zugriffsdatum:2008-07-30)*. 2008. – URL http://code.google.com/apis/blogger/

[CeDbNetCF 2003] TIFFANY, R.: *SQL Server CE Database Development with the .NET Compact Framework*. Apress, 2003

[CgiW3c 1993] W3C: *CGI: The Common Gateway Interface (Zugriffsdatum:2008-07-13)*. 1993. – URL http://www.w3.org/CGI/

[Compnetze 2002] KUROSE, J. ; ROSS, K.: *Computernetze*. Pearson Studium, 2002

[DbSysteme 2002] CONNOLLY, T. ; BEGG, C. ; STRACHAN, A.: *Datenbanksysteme*. Addison-Wesley, 2002

[DesignPatterns 2003] GAMMA, E. ; HELM, R. ; JOHNSON, R. ; VLISSIDES, J.: *Design Patterns*. Addison-Wesley, 2003

[diffWeb2 2006] TREESE, Win: *Web 2.0: is it really different?* ACM. 2006

[DistributedDb 1999] OZSU, M. T. ; VALDURIEZ, P.: *Principles of Distributed Database Systems*. 2nd. Prentice Hall, 1999

[DublicateProxy 2007] GAN, Yifu ; YANG, Huirong: Ajax and Web Services Integrated Framework Based on Duplicate Proxy Pattern. In: *First IEEE International Symposium on Information Technologies and Applications in Education*, IEEE, nov 2007, S. 297–302

[EcmaScriptSpec 2002] : *ISO-genormte ECMAScript-Spezifikation (ISO/IEC 16262:2002) (Zugriffsdatum:2008-07-13)*. 2002. – URL http://standards.iso.org/ittf/PubliclyAvailableStandards/c033835_ISO_IEC_16262_2002(E).zip

[expWeb 2007] EMARKETER: *The Expanding European Web (Zugriffsdatum:2008-07-13)*. 2007. – URL http://www.emarketer.com/Article.aspx?id=1005458&src=article1_newsltr

[FlashPenetration 2007] ADOBE SYSTEMS INCORPORATED: *Adobe Flash Player Version Penetration (Zugriffsdatum:2008-07-13)*. 2007. – URL http://www.adobe.com/products/player_census/flashplayer/version_penetration.html

[ForresterMobWeb 2008] FORRESTER RESEARCH: *Mobile Internet Usage In Europe To Surge Over The Next Five Years (Zugriffsdatum:2008-07-13)*. 2008. – URL http://www.forrester.com/ER/Press/Release/0,1769,1203,00.html

[gpxschema 2004] TOPOGRAFIX: *GPS Exchange Format 1.1 Schmema (Zugriffsdatum:2008-08-10)*. 2004. – URL http://www.topografix.com/GPX/1/1/gpx.xsd

[html5 2008] W3C: *HTML 5 - A vocabulary and associated APIs for HTML and XHTML (Zugriffsdatum:2008-07-13)*. 2008. – URL http://www.w3.org/TR/2008/WD-html5-20080610/

[html5Finish 2008] WEB HYPERTEXT APPLICATION TECHNOLOGY WORKING GROUP: *When will HTML 5 be finished? (Zugriffsdatum:2008-07-13)*. 2008. – URL http://wiki.whatwg.org/wiki/FAQ#When_will_HTML_5_be_finished.3F

[HttpRFC 1999] FIELDING et al: *Hypertext Transfer Protocol – HTTP/1.1 (Zugriffsdatum:2008-07-13)*. 1999. – URL http://www.w3.org/Protocols/rfc2616/rfc2616.html

[intrJason 2008] JSON.ORG: *Introducing JSON (Zugriffsdatum:2008-08-02)*. 2008. – URL http://www.json.org/

[JavaPerform 2003] SHIRAZI, Jack: *Java Performance Tuning*. 2. O'Reilly Media Inc., 2003

[LeeW3 1989] BERNERS-LEE, Tim: *World Wide Web (Zugriffsdatum:2008-07-13)*. 1989. – URL http://www.w3.org/History/19921103-hypertext/hypertext/WWW/TheProject.html

[MiddleAjax 2006] STAMEY, J. ; RICHARDSON, T.: *Middleware development with AJAX*. Journal of Computing Sciences in Colleges archive, Volume 22, Issue 2. dec 2006

[MiddlewareIssues 2004] NAKAJIMA, T. ; FUJINAMI, K. ; ISHIKAWA, E. Tokunagaand H.: Middleware design issues for ubiquitous computing. In: *Proceedings of the 3rd international conference on Mobile and ubiquitous multimedia*, ACM New York, NY, USA, 2004, S. 55 – 62. – URL http://doi.acm.org/10.1145/1052380.1052389

[MiddlewareMuddle 1998] RITTER, David: *The middleware muddle*. ACM SIGMOD Record archive, Volume 27, Issue 4. dec 1998. – URL http://doi.acm.org/10.1145/306101.306141

[MigCod 2008] KABALKIN, Mykhaylo: *Migration einer Rich Client Applikation auf Code on Demand Technologie*. 2008. – URL http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/master/kabalkin.pdf

[MobileInternet 2007] MARSDEN, Gary: *What is the mobile internet?* interactions, Volume 14 , Issue 6. dec 2007. – URL http://doi.acm.org/10.1145/1300655.1300672

[mobOpenAjax 2007] OPENAJAX ALLIANCE: *Introduction to Mobile Ajax for Developers (Zugriffsdatum:2008-07-13)*. 2007. – URL http://www.openajax.org/whitepapers/Introduction%20to%20Mobile%20Ajax%20for%20Developers.php

[MobSecurity 2006] EREN, E. ; DETKEN, K.: *Mobile Security*. Carl Hanser Verlag, 2006

[mobWeaver 2008] HAMDI, Louenas ; WU, Huaigu ; DAGTAS, Serhan ; BENHARREF, Abdel: Ajax for mobility: mobileweaver ajax framework. In: *Proceeding of the 17th international conference on World Wide Web*, 2008, S. 1077–1078

[mobWeb 2006] MOBILE WEB BEST PRACTICES WORKING GROUP: *Mobile Web Best Practices 1.0 (Zugriffsdatum:2008-07-13)*. 2006. – URL http://www.w3.org/TR/mobile-bp/

[MythsMobWeb 2006] TOMSOFT: *Myths of mobile Web2.0 (and mobile Ajax) (Zugriffsdatum:2008-07-13)*. 2006. – URL http://blog.landspurg.net/myths-of-mobile-web20-and-mobile-ajax/

[Net2Grundlagen 2006] NORTHRUP, T. ; WILDERMUTH, S. ; RYAN, B.: *Grundlagen der Anwendungsentwicklung mit .NET Framework 2.0*. Microsoft Press, 2006

[netCfPlat 2008] MICROSOFT CORPORATION: *Von .NET Compact Framework unterstützte Geräte und Plattformen (Zugriffsdatum:2008-07-13)*. 2008. – URL http://msdn.microsoft.com/de-de/library/ms172550.aspx

[OpenAjax] OPENAJAX ALLIANCE: *OpenAjax Alliance Overview (Zugriffsdatum:2008-07-13)*. – URL http://www.openajax.org/overview.php

[OpenAjaxMobile] SALETTA, R. ; FERRAIOLO, J.: *Mobile Ajax White Paper (Zugriffsdatum:2008-07-13)*. – URL http://www.openajax.org/member/wiki/WP3_-_Mobile_Ajax

[OperatingSys 2003] SILBERSCHATZ, A. ; GALVIN, P. ; GAGNE, G.: *Operating System Concepts*. Sixth. John Wiley & Sons, 2003

[ProgCFW 2006] DRÖGE, R. ; NOWAK, P. ; WEBER, T.: *Programmieren mit dem .NET Compact Framework*. Microsoft Press, 2006

[PushAjax 2007] BOZDAG, E. ; MESBAH, A. ; DEURSEN, A. van: A Comparison of Push and Pull Techniques for AJAX. In: *9th IEEE International Workshop on Web Site Evolution 2007*, IEEE, oct 2007, S. 15 – 22

[ReillyWeb2 2005] O'REILLY, Tim: *What is Web 2.0 (Zugriffsdatum:2008-07-13)*. 2005. – URL http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html

[rest 2000] FIELDING, Roy: *Architectural Styles and the Design of Network-based Software Architectures*. 2000. – URL http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[restful 2007] RICHARDSON, Leonard ; RUBY, Sam: *RESTful Web Services*. O'Reilly Media, 2007

[rfc1321 1992] RIVEST, R.: *RFC 1321 - The MD5 Message-Digest Algorithm*. 1992. – URL http://tools.ietf.org/html/rfc1321

[rfc2069 1997] J. FRANKS ET AL.: *RFC 2069 - An Extension to HTTP : Digest Access Authentication*. 1997. – URL http://tools.ietf.org/html/rfc2069

[rfc2617 1999] J. FRANKS ET AL.: *RFC 2617 - HTTP Authentication: Basic and Digest Access Authentication*. 1999. – URL http://tools.ietf.org/html/rfc2617

[rfc707 1976] WHITE, James E.: *A High-Level Framework for Network-Based Resource Sharing*. 1976. – URL http://tools.ietf.org/html/rfc707

[SecCode 2003] HOWARD, M. ; LEBLANC, D: *Writing Secure Code*. 2nd. Microsoft Press, 2003

[ServiceCompMobile 2005] CHAKRABORTY, D. ; JOSHI, A. ; FININ, T. ; YESHA, Y.: *Service composition for mobile environments*. Mobile Networks and Applications 10-2005. 2005. – URL http://doi.acm.org/10.1145/1160162.1160168

[soap 2007] W3C: *SOAP Specification (Zugriffsdatum:2008-09-07)*. 2007. – URL http://www.w3.org/TR/soap/

[TowardsAjax 2007] ZEPEDA, J. ; .SERGIO, C ; SERGIO, V.: From Desktop Applications Towards Ajax Web Applications. In: *4th International Conference on Electrical and Electronics Engineering 2007*, IEEE, sep 2007, S. 193 – 196

[uddiv3 2004] OPEN, OASIS: *UDDI Specification (Zugriffsdatum:2008-09-07)*. 2004. – URL http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm#uddiv3

[volere 1999] ROBERTSON, James ; ROBERTSON, Suzanne: *Mastering the Requirements Process*. Addison-Wesley Professional, 1999

[W3cMobile] : *The Web on the Move (Zugriffsdatum:2008-07-13)*. – URL http://www.w3.org/Mobile/

[whatsweb2 2006] LEWIS, Daniel: What is web 2.0? In: *Crossroads*, sep 2006, S. 3

[winMobArch 2008] MICROSOFT CORP.: *Architectural Overview of Windows Mobile Infrastructure Components (Zugriffsdatum:2008-08-12)*. 2008. – URL http://download.microsoft.com/download/c/b/d/cbdc18d1-1a01-4736-a557-08474ec73443/Windows_Mobile-Architectural-Overview-of-Infrastructure-Components.doc

[WinMobEntw 2007] BODDENBERG, U.: *Windows Mobile Integrations- und Entwicklungsbuch*. Microsoft Press, 2007

[winMobKernel 2008] MICROSOFT CORP.: *Windows Embedded Developer Center Windows Mobile Kernel Overview (Zugriffsdatum:2008-08-12)*. 2008. – URL http://msdn.microsoft.com/en-us/library/aa909237.aspx

[WinNetwork 2002] JONES, A. ; OHLUND, J.: *Network Programming for Microsoft Windows*. 2nd. Microsoft Press, 2002

[WMSec 2007] : *Security Model For Windows Mobile 5.0 and Windows Mobile 6 (Zugriffsdatum:2008-07-13)*. mar 2007. – URL http://download.microsoft.com/download/c/b/d/cbdc18d1-1a01-4736-a557-08474ec73443/Windows_Mobile_Security_Model.doc

[WsAjax 2007] DIEHL, Mike: *Writing web applications with web services and Ajax*. Linux Journal archive, Volume 2007, Issue 157. may 2007

[wsdl2 2007] W3C: *WSDL Specification (Zugriffsdatum:2008-09-07)*. 2007. – URL http://www.w3.org/TR/wsdl20/

[Xmiddle 2002] MASCOLO, C. ; CAPRA, L. ; ZACHARIADIS, S. ; EMMERICH, Wolfgang: *XMIDDLE: A Data-Sharing Middleware for Mobile Computing*. Wireless Personal Communications: An International Journal archive, Volume 21, Issue 1. apr 2002. – URL http://www.springerlink.com/content/hj7knrkaxy9bfk2r/

[XMLHttpRequest 2007] W3C: *The XMLHttpRequest Object (Zugriffsdatum:2008-07-13)*. 2007. – URL http://www.w3.org/TR/2007/WD-XMLHttpRequest-20071026/

[xmlrpc 1999] WINER, Dave: *XML-RPC Specification (Zugriffsdatum:2008-08-02)*. 1999. – URL http://www.xmlrpc.com/spec

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §22(4) bzw.§24(4) ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 15. September 2008 Eike Christian Falkenberg