Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Maik Weindorf

Ein Framework für Context-Awareness auf mobilen Geräten

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Maik Weindorf

Ein Framework für Context-Awareness auf mobilen Geräten

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. Olaf Zukunft
Zweitgutachter : Prof. Dr. rer. nat. Bettina Buth

Abgegeben am 09. April 2008

Maik Weindorf

Thema der Masterarbeit

Ein Framework für Context-Awareness auf mobilen Geräten

Stichworte

Context, Context-Awareness, Context Repräsentation, mobile Geräte, Framework, Rule Engine, XML, .Net

Kurzzusammenfassung

In dieser Arbeit wird ein Framework entworfen, mit dessen Hilfe die Entwicklung von Context-Sensitiven Applikationen für mobile Geräte erleichtert werden soll. Die Lösung ist dabei nicht auf spezielle Context Informationen beschränkt und somit für eine Vielzahl von Context-Sensitiven Applikationen geeignet.
Es werden zunächst die Begriffe *Context* und *Context-Awareness* näher betrachtet. Weiterhin werden Anforderungen an ein entsprechendes Framework definiert. Basierend auf diesen Anforderungen wird ein konzeptioneller Entwurf für das Framework entwickelt. Dieser Entwurf wird anschließend in einer prototypischen Implementierung erfolgreich auf seine Realisierbarkeit geprüft.

Maik Weindorf

Title of the paper

A Framework for Context-Awareness on mobile Devices

Keywords

context, context-awareness, context representation, mobile devices, framework, rule engine, XML, .Net

Abstract

In this thesis, a framework is designed that is meant to simplify the development of context-aware applications for mobile devices. The solution is not limited to specific context information and therefore suitable for many context-aware applications.
First of all, the terms *context* and *context-awareness* are introduced. In the following, the requirements for such a framework are analyzed and defined. Based on these requirements, a conceptual design for the framework is developed and its feasibility is evaluated successfully by means of a prototypical implementation.

Danksagung

An dieser Stelle möchte ich mich bei allen bedanken, die mich bei der Erstellung dieser Arbeit unterstützt haben.

Besonders danke ich Prof. Dr. Olaf Zukunft für die hervorragende Betreuung dieser Arbeit. Seine Anmerkungen und kritischen Fragen haben mir neue Anstöße gegeben und geholfen, meinen Blick auf das Wesentliche zu richten.
Ich danke auch Prof. Dr. rer. nat. Bettina Buth, die sich bereit erklärt hat, die Zweitkorrektur dieser Arbeit zu übernehmen.

Darüber hinaus möchte ich meinen Eltern danken, die mich während dieser Arbeit und des gesamten Studiums in jeglicher Hinsicht unterstützt haben. Sie haben mir eine hervorragende Ausbildung ermöglicht. Ohne sie wäre ich heute nicht da, wo ich bin.

Nicht zuletzt danke ich meiner Freundin. Danke für alles!

Maik Weindorf, April 2008

Inhaltsverzeichnis

Inhaltsverzeichnis **I**

Abbildungsverzeichnis **V**

1 Einleitung **1**
- 1.1 Motivation . . . 1
- 1.2 Abgrenzung und Ziel . . . 2
- 1.3 Szenario . . . 3
- 1.4 Aufbau der Arbeit . . . 4

2 Grundlagen **5**
- 2.1 Mobile Geräte . . . 5
- 2.2 Mobile Kommunikation . . . 7
  - 2.2.1 Client-Client . . . 8
  - 2.2.2 Client-Server . . . 8
  - 2.2.3 Synchron vs. Asynchron . . . 9
- 2.3 Context und Context-Awareness . . . 9
  - 2.3.1 Context Beispiele . . . 11
- 2.4 Context Repräsentation . . . 12
  - 2.4.1 RDF . . . 12
  - 2.4.2 CC/PP . . . 13
  - 2.4.3 OWL . . . 13
- 2.5 Frameworks . . . 14
- 2.6 Technologien . . . 16
- 2.7 Zusammenfassung . . . 16

3 Analyse **18**

3.1 Vision 18
3.1.1 Beispiele 19
3.2 Funktionale Anforderungen 20
3.2.1 Trennung von Framework- und Applikationslogik 20
3.2.2 Konfiguration des Frameworks zur Laufzeit 23
3.2.3 Verarbeitung von Context Informationen 25
3.2.4 Persistenz von Context Informationen 26
3.3 Nichtfunktionale Anforderungen 26
3.3.1 Zuverlässigkeit 26
3.3.2 Benutzbarkeit 27
3.3.3 Effizienz 27
3.3.4 Änderbarkeit 28
3.3.5 Übertragbarkeit 28
3.4 Verwandte Themen und Arbeiten 29
3.4.1 A Context Ontology for Pervasive Service Provision 29
3.4.2 Hydrogen Context-Framework 29
3.4.3 Sensor-based Context-Awareness 29
3.4.4 Pervasive Gaming Framework 30
3.4.5 Weitere interessante Arbeiten 30

4 Entwurf **31**
4.1 Architektur Konzept 31
4.1.1 Verteilung 33
4.2 Komponenten 35
4.2.1 Context Server 35
4.2.2 Context Adapter 38
4.2.3 Application Stub 39
4.2.4 Context Objekte 41
4.3 Interaktion 41
4.3.1 Nachrichtenformat 42
4.3.2 Rule Format 50
4.3.3 Context Server - Application Stub 59
4.3.4 Context Server - Context Adapter 60
4.4 Szenarien: Framework Erweiterung 61
4.4.1 Neue Applikation 63
4.4.2 Neue Context Quelle 64

4.4.3 Neue Context Domain 64
4.5 Abgleich mit Anforderungen 65
4.6 Testkonzept 67

5 Realisierung **69**
5.1 Entwicklungsumgebung 69
5.2 Funktionsumfang 70
5.2.1 Komponenten 71
5.2.2 Prototyp: Beispiel Anwendung 73

6 Fazit **74**
6.1 Ausblick 75

Literaturverzeichnis **76**

A Inhalt der CD **80**

B XML Schema: Context **81**

C XML Schema: Rules **85**

Abbildungsverzeichnis

2.1 Mobile Kommunikation 8
2.2 Context [Schmidt et al., 1998] 10
2.3 RDF Example [TALIS, 2005] 13
2.4 CC/PP Example [W3C 5, 2007] 14
2.5 OWL Example [Horridge et al., 2007] 15

4.1 Konzept 32
4.2 Szenarien: Komponenten Verteilung 34
4.3 Context Adapter Klassendiagramm 39
4.4 Application Stub Klassendiagramm 40
4.5 Context Type 43
4.6 Simple Context XML - subscribe Example 44
4.7 Simple Context XML - get Example 45
4.8 Position Context XML - put Example 46
4.9 User Profile Context XML - put Example 47
4.10 Calendar Context Type 48
4.11 Device Profile Context Type 49
4.12 Position Context Type 50
4.13 Time Context Type 50
4.14 User Profile Context Type 51
4.15 Simple Context Extension Type 52
4.16 Complex Context Type 53
4.17 Rule Schema 54
4.18 Rule XML - Simple Example 55
4.19 Rule Fact: Device Profile 56
4.20 Rule Implication 57
4.21 Randbedingungen: Regeln 59

4.22 Context Server - Application Stub: Interaktion 1 60
4.23 Context Server - Application Stub: Interaktion 2 61
4.24 Context Quelle - Context Server: Interaktion 1 62
4.25 Context Quelle - Context Server: Interaktion 2 62
4.26 Abgleich: funktionale Anforderungen 67

5.1 Microsoft Visual Studio 70
5.2 HP iPAQ [HP] 71

Kapitel 1

Einleitung

1.1 Motivation

Bereits 1991 entwarf Mark Weiser in seiner viel zitierten Arbeit "The Computer for the 21st Century" [Weiser, 1991] eine Vision, wie die Zukunft der Mensch-Computer Interaktion aussehen könnte. Er bemängelt darin, dass sich Anwender bisher zu sehr darauf konzentrieren müssen, WIE sie etwas mit einem Computer tun können, anstatt sich damit zu beschäftigen, WAS sie tun möchten. Er folgert daraus, dass ein Umdenken notwendig ist, um das volle Potential der Informationstechnologie ausschöpfen zu können und spricht von:

> "...a new way of thinking about computers in the world, one that takes into account the natural human environment and allows the computers themselfs to vanish into the background."

Weiterhin sagt er:

> "...only when things disappear in this way are we free to use them without thinking and so to focus beyond them on new goals."

Seit der Veröffentlichung dieser Arbeit hat die Informationstechnologie enorme Fortschritte gemacht. In vielen Punkten liegt sie jedoch noch weit hinter den damaligen Visionen zurück. Durch die zunehmende Verbreitung leistungsfähiger mobiler Geräte, ist es jedoch heute möglich, den Visionen sehr nahe zu kommen und in einigen Punkten darüber hinaus zu gehen.

Besonders interessant sind in diesem Zusammenhang sog. "ultra-mobile Geräte" - wie z.B. PDAs und Smartphones - da sich diese in vielen Aspekten deutlich von Laptops oder PCs unterscheiden (Kapitel 2.1). Der hohe Mobilitätsgrad dieser Geräte in Kombination mit den daraus resultierenden wechselnden Rahmenbedingungen, ermöglicht eine völlig neue Art von Applikationen, die ihr Verhalten und ihre Funktionalität dynamisch anhand dieser Rahmenbedingungen verändern.

Das Schlüsselwort in diesem Zusammenhang lautet "Context-Awareness" (bzw. "Context-Sensitivität").
Laut [Schmidt et al., 1998] beschreibt Context[1]:

> "...a situation and the environment a device or user is in."

Context-Awareness bedeutet somit das Bewusstsein über diese Situation und Rahmenbedingungen. In Kapitel 2.3 folgt eine nähere Betrachtung des Begriffs "Context".

Trotz der neuen und viel versprechenden Möglichkeiten, die diese mobilen Geräte bieten, darf man jedoch auch deren Einschränkungen nicht vergessen. Geräte dieser Kategorie haben aufgrund ihrer begrenzten Akku-Leistung eine begrenzte Laufzeit. Ihre Bedienbarkeit ist durch kleine Tastaturen oder Touch-Bildschirme eingeschränkt. Außerdem verfügen sie im Vergleich zu aktuellen PCs noch immer über recht begrenzte Rechenleistung und Arbeitsspeicher. Hinzu kommt eine eingeschränkte Konnektivität durch langsame oder instabile Netzwerkverbindungen. Diese Einschränkungen müssen bei der Entwicklung von Applikationen berücksichtigt werden. In Verbindung mit Context-Awareness bieten sich hier jedoch viele Möglichkeiten für eine Verbesserung der Usability.

Dies gilt sowohl für Applikationen, die speziell für diese Geräte entwickelt werden, als auch für Applikationen, die nicht auf eine spezielle Geräte Kategorie beschränkt sein sollen (z.B. Web-Anwendungen). [Hofer et al., 2003] [Yan und Sere, 2004] [Eckstein, 2004] [Kouadri et al., 2004] [Moran und Dourish, 2001]

1.2 Abgrenzung und Ziel

Context-Awareness kann potentiell eine große Bandbreite an Applikationen bereichern. Bisher ist dies jedoch meist mit einem erheblichen Mehraufwand verbunden. Ziel dieser

[1] In dieser Ausarbeitung wird durchgängig der Begriff "Context" verwendet. Von einer Verwendung des deutschen Begriffs "Kontext" wird aus Konsistenzgründen abgesehen.

Arbeit ist es, ein Framework, bzw. eine Plattform für Context-Awareness auf mobilen Geräten zu schaffen. D.h. mit dem Resultat dieser Arbeit soll die Entwicklung von Context-Sensitiven Applikationen vereinfacht werden (siehe auch Kapitel 3.1).

Diese Arbeit beschäftigt sich speziell mit "Context-Awareness auf mobilen Geräten". Das Gebiet der Context-Awareness ist sehr weit. Im Rahmen dieser Arbeit kann daher nicht auf alle Teilaspekte eingegangen werden. Speziell die Einsatzgebiete von "Context-Awareness" in nicht-mobilen Umgebungen spielen im Rahmen dieser Arbeit eine untergeordnete Rolle. Ebenso ist es nicht Ziel dieser Arbeit ein marktreifes Produkt zu entwickeln. Es geht vorrangig darum, grundlegende Ideen und Konzepte darzulegen und zu evaluieren. Daher werden die Aspekte "Sicherheit", "Datenschutz" und "Performance" im Rahmen dieser Arbeit nicht näher betrachtet.

1.3 Szenario

Das folgende Szenario soll eine Idee davon vermitteln, wozu die im Rahmen dieser Arbeit zu entwickelnde Lösung, in der Lage sein soll:

Angenommen ein Nutzer besitzt ein GPS fähiges mobiles Gerät (z.B. Smartphone oder PDA). Der Nutzer hat ein großes Interesse an Kunst und ein mittelmäßiges Interesse an Autos. Er befindet sich gerade zu Fuß in der Nähe des Dammtor Bahnhofs in Hamburg. In "Planten un Blomen" findet eine Open Air Kunstausstellung statt und beim CCH eine Automobilausstellung[2]. Die Ausstellungen sind dem Nutzer bisher nicht bekannt. Alternativen in Abhängigkeit verschiedener fiktiver Rahmenbedingungen:

- Das Gerät informiert den Nutzer aktiv[3] über die Kunstausstellung, da sein Interesse an Kunst größer ist, als das Interesse an Autos. (optional: zusätzlich hinterlässt es für den Nutzer eine passive[4] Notiz über die Automobilausstellung)
- Das Gerät informiert den Nutzer nicht aktiv über die Ausstellungen, da es anhand der überdurchschnittlich hohen Laufgeschwindigkeit des Nutzers davon ausgeht, dass der Nutzer es eilig hat. (optional: das Gerät hinterlässt passive Notizen über die

[2]Beides liegt in unmittelbarer Nähe des Dammtor Bahnhofs.

[3]Aktiv könnte z.B. ein akustisches Signal und/oder Vibration in Verbindung mit einer Text-basierten Nachricht bedeuten.

[4]Passiv könnte z.B. eine Text-basierten Nachricht ohne zusätzliches akustisches Signal oder Vibration bedeuten.

Ausstellungen)

- Das Gerät informiert den Nutzer aktiv über die Automobilausstellung (obwohl der Nutzer eigentlich ein größeres Interesse an Kunst hat), da die Wettervorhersage auf Regen schließen lässt und eine Aktivität an einem überdachten Ort daher zu bevorzugen ist.
- Das Gerät informiert den Nutzer über keine der Ausstellungen und hinterlässt auch keine passiven Notizen, da aus dem Kalender des Nutzers ersichtlich ist, dass er in 15 Minuten zu einer 1 wöchigen Geschäftsreise aufbricht und die Ausstellungen bei seiner Rückkehr bereits beendet sein werden.

Natürlich ist dies nur eines von sehr vielen denkbaren Szenarien. Es wird jedoch bereits an diesem einfachen Beispiel deutlich, dass potentiell verschiedenste Context Quellen kombiniert werden können um eine Applikation zu bereichern, bzw. neue Arten von Applikationen zu ermöglichen. Weitere Szenarien finden sich unter anderem in Kapitel 3.1.1.

1.4 Aufbau der Arbeit

In Kapitel 2 werden relevante Grundlagen zum Thema dieser Arbeit vorgestellt. Kapitel 3 beschäftigt sich vor allem mit den funktionalen und nichtfunktionalen Anforderungen an die zu entwickelnde Lösung. Zusätzlich werden verwandte Themen und Arbeiten vorgestellt. Kapitel 4 stellt den umfangreichsten und auch wichtigsten Teil dieser Arbeit dar. In diesem Kapitel wird basierend auf den in Kapitel 3 gestellten Anforderungen ein Entwurf für die zu entwickelnde Lösung vorgestellt. Kapitel 5 befasst sich mit der prototypischen Implementierung des in Kapitel 4 vorgestellten Entwurfs. Anschließend folgt ein Fazit und Ausblick in Kapitel 6.

Kapitel 2

Grundlagen

Wenn man sich dem Thema ”Context-Awareness auf mobilen Geräten” annähern möchte, muss man vorerst einige grundlegende Betrachtungen vornehmen, die in Verbindung mit diesem Thema stehen. Zuerst sollte man sich verdeutlichen, was unter dem Begriff ”mobile Geräte” zu verstehen ist und wodurch sich diese von ”herkömmlichen” Computern unterscheiden (Kapitel 2.1). Dies gilt für die Geräte selbst, sowie für die damit verbundenen, unterschiedlichen Möglichkeiten der Kommunikation (Kapitel 2.2). Weiterhin ist es essentiell, die Begriffe ”Context” und ”Context-Awareness” näher zu betrachten (Kapitel 2.3). Bei einer eingehenden Beschäftigung mit dem Thema, stellt sich darüber hinaus die Frage, in welcher Form Context repräsentiert werden kann. Auf diese Problematik wird in Kapitel 2.4 näher eingegangen. Da im Rahmen dieser Arbeit ein Framework entwickelt werden soll, ist es außerdem sinnvoll, den ”Framework” Begriff zu erläutern und abzugrenzen, sowie potentielle Technologien zu dessen Realisierung zu betrachten (Kapitel 2.6).

2.1 Mobile Geräte

Es gibt eine Vielzahl sog. ”mobiler” Geräte. Die im Rahmen dieser Arbeit zu entwickelnde Lösung soll für ein möglichst breites Spektrum an mobilen Geräten einsetzbar sein. Es ist daher notwendig, sich die Unterschiede zwischen den verschiedenen Kategorien mobiler Geräte zu verdeutlichen.

Sowohl Notebooks, Subnotebooks, Ultra-Portables, PDAs, Smartphones, als auch Geräte mit klar definiertem Funktionsumfang (sog. Appliences), wie z.B. Handys, Navigationsge-

räte und MP3-Player können als mobile Geräte aufgefasst werden.

Um für diese Arbeit von Bedeutung zu sein, müssen die mobilen Geräte gewisse Mindestanforderungen erfüllen. Vor allem müssen sie (ohne Hacks) programmierbar sein und über standardisierte Kommunikationsschnittstellen verfügen. Die Kategorie der Appliences ist daher nicht weiter relevant, solange sie diese Anforderungen nicht erfüllen. Näher betrachtet werden "PC-ähnliche" Geräte.

- Notebooks
  - Rechenleistung und Speicher unterscheiden sich kaum von denen eines PCs
  - Gewicht und Größe schränken die Mobilität ein
  - Abgesehen vom Preis gibt es praktisch keine Nachteile gegenüber einem PC
- Subnotebooks
  - Rechenleistung und Speicher sind geringer als bei PCs
  - Einfach zu transportieren, jedoch nicht überall einsetzbar (z.B. im Gehen)
  - Kleinere Displays und Tastaturen (als bei Notebooks) schränken die Bedienbarkeit geringfügig ein
  - Die Unterstützung von Betriebssystemen und Programmen, sowie die Ausstattung an Schnittstellen unterscheidet sich nur unwesentlich von denen eines PCs
- Ultra-Portables (z.B. Samsung Q1[1])
  - Rechenleistung und Speicher sind deutlich geringer als bei PCs
  - Potentiell für mobilen Einsatz geeignet, da sie mit einer Hand gehalten werden können
  - Kleine Displays (ca. 7") und Touchbedienung schränken die Bedienbarkeit merkbar ein
  - PC Betriebssysteme und Programme werden mit Einschränkungen unterstützt (Ressourcen oft ausgelastet)
- PDAs und Smartphones (können hier gemeinsam betrachtet werden)

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

- Rechenleistung und Speicher dramatisch geringer als bei PCs und deutlich geringer als bei Ultra-Portables
- Speziell für mobilen Einsatz geeignet
- Sehr kleine Displays, sehr kleine Tastaturen und/oder Touchbedienung schränken die Bedienbarkeit deutlich ein
- PC Betriebssysteme und Programme werden nicht unterstützt (stattdessen spezielle Betriebssysteme und Programme)

Besonders wichtig ist die Abgrenzung der PDAs und Smartphones von den anderen aufgeführten Geräte-Kategorien. Bis hin zu Ultra-Portables ist der Einsatz von PC Betriebssystemen möglich, wodurch sich praktisch keine Einschränkung bei der Wahl von Entwicklungs- und Laufzeitumgebungen, sowie Programmiersprachen und Standardsoftware (wie z.B. Servern) ergeben. PDAs und Smartphones hingegen erfordern derzeit den Einsatz von speziellen Entwicklungs- und Laufzeitumgebungen (z.B. .Net Compact [Microsoft, 2004] oder Java ME[2] [Sun, 2008], siehe Kapitel 2.6). Speziell für den Einsatz von Serverkomponenten auf PDAs und Smartphones gibt es bisher kaum Standardlösungen, wodurch sich der Realisierungsaufwand solcher Komponenten dramatisch erhöhen kann. Allerdings verwischen diese Unterschiede und Einschränkungen zusehends. In ein paar Jahren wird eine solche Abgrenzung vermutlich nicht mehr notwendig sein.

2.2 Mobile Kommunikation

In Kapitel 2.1 wurde bereits auf einige Probleme von mobilen Geräten hingewiesen. Ein weiteres Problem, bzw. Herausforderung stellt die Kommunikation von mobilen Geräten untereinander und mit evtl. vorhandenen Servern dar. Für diese Kommunikation stehen potentiell diverse Technologien zur Verfügung, die je nach Situation besser oder schlechter geeignet sind [Salvatori, 2006] (siehe Tabelle: Abbildung 2.1).

Da all diese Technologien diverse Vor- und Nachteile bieten, muss je nach zu realisierendem Szenario abgewogen werden, welche Technologie angemessen ist. Die Wahl einer oder mehrerer Technologien, hat entscheidenden Einfluss auf das Spektrum der realisierbarer Szenarien.

[2] Java ME soll in den nächsten Jahren durch Java Standard Edition abgelöst werden.

	Reichweite	Geschw.	P2P	Internet	Kosten
WLAN	+	++	ad hoc	ja	0-?
UMTS	++	++	nein	ja	hoch
GPRS	++	∘	nein	ja	hoch
Bluetooth	∘	∘	ja	nein	0
Infrarot	–	–	ja	nein	0

++ = sehr gut, + = gut, ∘ = mäßig, - = schlecht

Abbildung 2.1: Mobile Kommunikation

2.2.1 Client-Client

Für die Kommunikation von mobilem Client zu mobilem Client kommen die in Tabelle 2.1 aufgelisteten Technologien infrage.

Dabei muss unterschieden werden, ob die Kommunikation ”direkt” oder ”indirekt” erfolgt. Also ob die Clients direkt miteinander kommunizieren (z.B. über Infrarot, Bluetooth oder eine ad-hoc WLAN Verbindung), oder über einen Umweg wie z.B. einen gemeinsamen Access Point, oder das Internet. Hier spielen z.B. Überlegungen eine Rolle, ob die Kommunikation kostenlos sein soll, ob die Clients vermutlich nah genug für direkte Kommunikation sein werden und vieles mehr.

2.2.2 Client-Server

Bei der Kommunikation zwischen mobilen Clients und Servern gibt es 2 grundlegende Ansätze:

- kabellos (mobil)
- kabelgebunden (stationär, z.B. Docking-Station)

Für die kabellose Client-Server Kommunikation stehen die meisten der in Tabelle 2.1 aufgelisteten Technologien zur Verfügung. Lediglich Bluetooth und Infrarot sind in diesem Umfeld nur sehr eingeschränkt nutzbar.

Die zweite Variante stellt einen Sonderfall dar und ist mit der Kommunikation zwischen stationären Clients (z.B. PCs) und Servern zu vergleichen. Diese Form der Kommunikation ist im Vergleich zur kabellosen Kommunikation sehr beschränkt. Kabellose Kommunikation ermöglicht die Kommunikation zwischen Clients und Servern ”unterwegs”, d.h. es können

wesentlich dynamischere Szenarien realisiert werden als mit kabelgebundener Kommunikation.

2.2.3 Synchron vs. Asynchron

Unabhängig von der zu Grunde liegenden Technologie, kann die Kommunikation über synchrone oder asynchrone Protokolle erfolgen. Synchrone Protokolle bieten sich an, wenn Nachrichten ohne nennenswerte Verzögerungen ausgetauscht werden sollen. Asynchrone Kommunikation bietet sich vor allem bei instabilen Verbindungen an um die Robustheit der Nachrichtenübertragung zu erhöhen. Diese Art der Kommunikation lässt sich z.B. durch Messaging realisieren [Pashtan, 2005] [Hohpe et al., 2004].

2.3 Context und Context-Awareness

In diesem Abschnitt wird zunächst der Begriff "Context" näher betrachtet. Des Weiteren werden Einsatzmöglichkeiten von Context-Awareness in mobilen Umgebungen aufgezeigt (vgl. [Winograd, 2001] [Kouadri et al., 2004] [Moran und Dourish, 2001] [Pashtan, 2005]).

"There is more to Context than Location" [Schmidt et al., 1998]. Context-Awareness in mobilen Umgebungen wird häufig gleichgesetzt mit Location-Awareness. Es gibt jedoch wesentlich mehr verwertbare Context-Informationen über die reine Location Information hinaus. In [Schmidt et al., 1998] wird eine Unterteilung in menschliche Faktoren und physikalische Umgebung vorgenommen. Menschliche Faktoren beschreiben dabei z.B. Benutzereinstellungen, soziales Umfeld (z.B. wer sind Freunde und Kollegen? wo befinden sich diese?) und die konkrete Aufgabe eines Nutzers (Task). Die physikalische Umgebung hingegen beschreibt unter anderem Umgebungsbedingungen, wie z.B. Licht, Temperatur und Beschleunigung, sowie Infrastruktur Informationen und natürlich auch die Location. All diese Faktoren lassen sich messen, bzw. ermitteln und unterschiedlich granular betrachten. So kann man beispielsweise Intensität und Wellenlänge als messbare Faktoren von Licht auffassen (siehe Abbildung 2.2). Die Information über die aktuelle Licht Intensität könnte dann z.B. zur automatischen Anpassung der Display-Helligkeit eines Gerätes verwendet werden.

Hofer [Hofer et al., 2003] hingegen nimmt eine Unterteilung in physikalischen und logischen Context vor. Dabei beschreibt der physikalische Context "low level" Informationen, wie z.B.

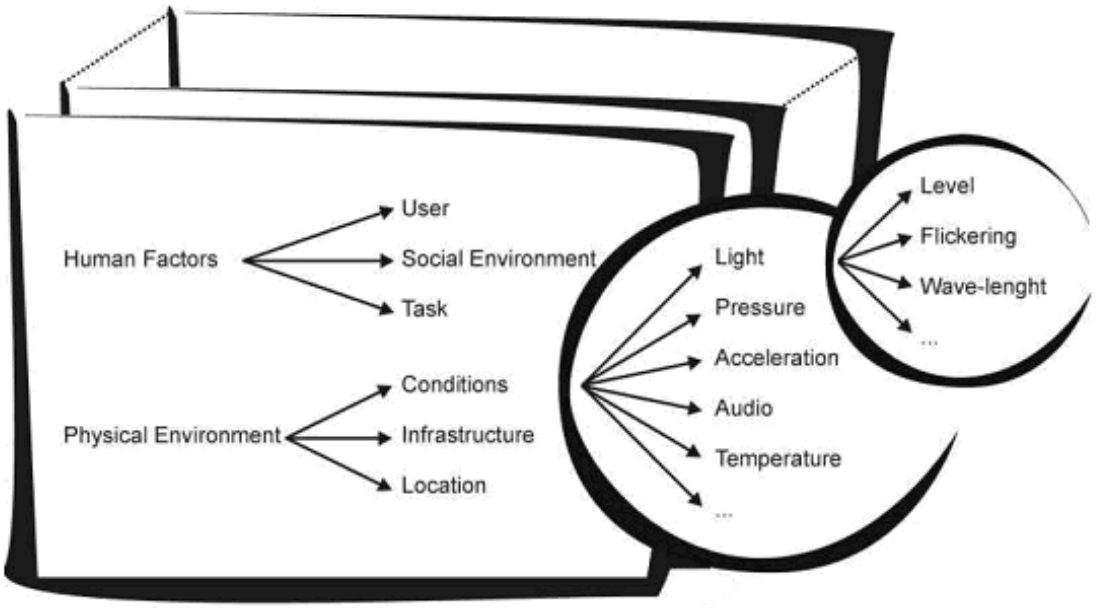

Abbildung 2.2: Context [Schmidt et al., 1998]

eine GPS-Position oder eine Temperatur. Der logische Context hingegen beschreibt ”high level” Informationen, wie beispielsweise Straßennamen oder Temperatur-Umschreibungen, wie ”heiß” und ”kalt”. Somit wird schnell klar, dass die Definitionen von Context weder einheitlich noch eindeutig sind.

Es drängt sich die Frage auf, welches Ziel mit der Einbeziehung von Context in die Logik von Applikationen erreicht werden soll. Eine aktuelle Studie des BSI [BSI, 2006] nennt als Folge von Context-Awareness einen erhöhten Komfort für die Nutzer (Usability) und auf längere Sicht einen weiteren Schritt in Richtung Autonomie
(vgl. Autonomic Computing [Ganek und Corbi, 2003] [IBM, 2001] [Barkhuus und Dey, 2003]). Wie am Szenario in Kapitel 1.3 deutlich wird, ermöglicht die Einbeziehung von Context neue Arten von Applikationen, die sonst nicht realisierbar wären. Siehe dazu auch die Beispiele in Kapitel 2.3.1.

2.3.1 Context Beispiele

Im Folgenden werden einige Anwendungsbeispiele für Context-Awareness in mobilen und nicht-mobilen Umgebungen aufgezeigt. Obwohl nicht-mobile Umgebungen für diese Arbeit nicht weiter relevant sind, sind sie an dieser Stelle doch geeignet, die vielseitigen Einsatzgebiete von Context-Awareness zu verdeutlichen.

GPS Navigationsgeräte Navigationsgeräte sind typische, wenn auch recht einfache Anwendungsbeispiele von Context-Awareness. Sie beziehen hauptsächlich die gegenwärtige Position in ihre Applikationslogik ein und verbinden diese mit einem logischen Context (z.B. einer Landkarte). Abgesehen davon verwenden moderne Navigationsgeräte jedoch auch noch weniger offensichtliche Context-Informationen. Vor allem die aktuelle Geschwindigkeit wird meist als zusätzliche Information genutzt. So ist es beispielsweise möglich, dem Benutzer Ansagen bei hoher Geschwindigkeit früher mitzuteilen, als bei geringer Geschwindigkeit. Außerdem kann die Lautstärke der Sprachausgabe an die gegenwärtige Geschwindigkeit angepasst werden. Dabei wird davon ausgegangen, dass die Umgebungsgeräusche mit zunehmender Geschwindigkeit ebenfalls zunehmen. Zusätzlich verwenden einige Navigationsgeräte Verkehrsinformationen, die sie z.B. über TMC[3] Funk erhalten, um Umleitungen für Staus, Sperrungen usw. ermitteln zu können.

Spamfilter Das Thema Spam, bzw. Spambekämpfung wird zunehmend wichtiger. Einfache regelbasierte Spamfilter sind inzwischen kaum noch in der Lage, Spam Mails wirkungsvoll zu unterdrücken. Moderne Verfahren setzen unter anderem auf Context-Awareness. Dabei wird beispielsweise berücksichtigt, ob die Mail von einem Bekannten kommt. Wenn ja, wird die Mail positiv bewertet und es bedarf weit mehr einschlägiger Schlüsselwörter und anderer Merkmale, bis die Mail als Spam eingestuft wird. In diesem Beispiel wird also u.a. der soziale Context mit einbezogen.

personalisierte Suche Diese Art von Suche, wie man sie z.B. aus Onlineshops wie Amazon[4] kennt, ist eines der bekanntesten Anwendungsgebiete von Context-Awareness. Im Gegensatz zu anderen Anwendungsgebieten von Context-Awareness (wie z.B. Spamfilter) ist es in diesem Fall ganz offensichtlich für die Anwender, dass Context-Informationen genutzt werden.

[3] http://de.wikipedia.org/wiki/Traffic_Message_Channel
[4] http://www.amazon.de

Loadbalancer Ein weniger offensichtliches Anwendungsgebiet von Context-Awareness sind Loadbalancer für Web- oder DB-Server. Loadbalancer gibt es schon deutlich länger als den Begriff der Context-Awareness und man findet die beiden Begriffe selten in direkter Verbindung. Jedoch nutzen Loadbalancer verschiedene Context-Informationen, wie z.B. die aktuelle Last im System, die aktuelle Anzahl an Clients und darüber hinaus "Erfahrungswerte" aus der Vergangenheit.

2.4 Context Repräsentation

Bei der Entwicklung von Applikationen mit Context-Awareness stellt sich stets die Frage, wie die relevanten Context Informationen repräsentiert werden können. Natürlich besteht die Möglichkeit von proprietären Lösungen. Allerdings etablieren sich seit einigen Jahren zunehmend Standards, die für diese Aufgabe geeignet sind. Im Folgenden werden einige dieser Standards vorgestellt.

2.4.1 RDF

Das "Resource Description Framework" (RDF) ist eine W3C Recommendation [W3C 1, 2004] [W3C 4, 2004] für die Repräsentation von strukturierten (Meta-)Daten. Unter anderem unterstützt RDF eine Repräsentation in Form von XML (sog. RDF/XML [W3C 2, 2004]). RDF/XML kann z.B. genutzt werden um Meta-Informationen über (Web)Ressourcen auszutauschen. RDF ist Teil der W3C Vision für ein "Semantic Web" [W3C 3, 2001]. Bisher findet RDF, bzw. RDF/XML u.a. Verwendung im Mozilla Framework[5], bei Creative Commons[6], bei MIT's DSpace[7] und bei FOAF[8] ("Friend-of-a-Friend") [Powers, 2003].

Abbildung 2.3 zeigt ein einfaches RDF Beispiel, in dem Meta-Informationen zu einer "Buch" Ressource dargestellt werden.

Aufgrund seiner Eigenschaften kann RDF, bzw. RDFS zur Beschreibung von Context Informationen eingesetzt werden [Pashtan, 2005].

[5] http://www.mozilla.org/
[6] http://creativecommons.org/
[7] http://dspace.mit.edu/
[8] http://www.foaf-project.org/

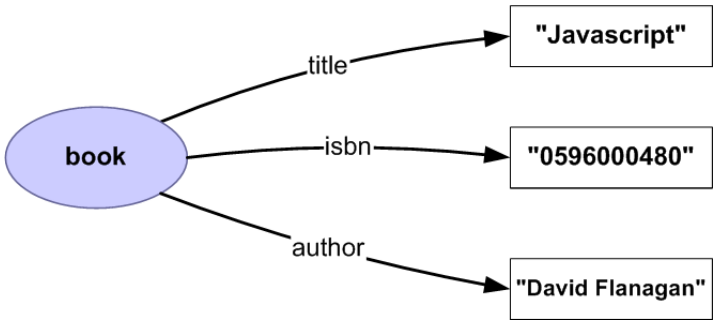

Abbildung 2.3: RDF Example [TALIS, 2005]

2.4.2 CC/PP

"Composite Capability/Preference Profiles" (CC/PP) basiert auf RDF (Kapitel 2.4.1) und stammt ebenfalls vom W3C [W3C 5, 2007]. Allerdings hat CC/PP bisher lediglich den Status "Draft". Es handelt sich bei CC/PP um eine strukturierte Sprache für die Beschreibung der individuellen Eigenschaften von Clients und darüber hinaus der Beschreibung von Präferenzen und Nutzer Profilen. Die drei wichtigsten Eigenschaften, die mit Hilfe von CC/PP beschrieben werden können, sind Client-Hardware, Client-Software und Client-Browser (siehe Abbildung 2.4). Dies ermöglicht z.B. das Senden von angepassten/reduzierten Inhalten für Clients [Eckstein, 2004].

CC/PP ist also nicht für jede beliebige Art von Context Information geeignet.

2.4.3 OWL

Bei der "Web Ontology Language" handelt es sich, wie auch bei RDF (Kapitel 2.4.1), um eine W3C Recommendation [W3C 6, 2004] im Rahmen von "Semantic Web". OWL ist eine Erweiterung von RDF-Schema [W3C 4, 2004] und unterstützt die Verwendung von Prädikatenlogik zur Definition von Ontologien [Gruber, 1992]. OWL ist der offizielle Nachfolger von DAML+OIL [DAML, 2000].

Ontologien dienen der Repräsentation von Wissen. Mit ihrer Hilfe lassen sich Begriffe formal spezifizieren [Gruber, 1992]. Im Zusammenhang mit Context-Awareness können Ontologien verwendet werden, um Context Informationen zu vereinheitlichen. Abbildung 2.5 zeigt ein OWL Beispiel, in dem sog. "Individuals" in Klassen gruppiert sind. Einige der "Individuals" sind dabei durch sog. "Properties" verbunden.

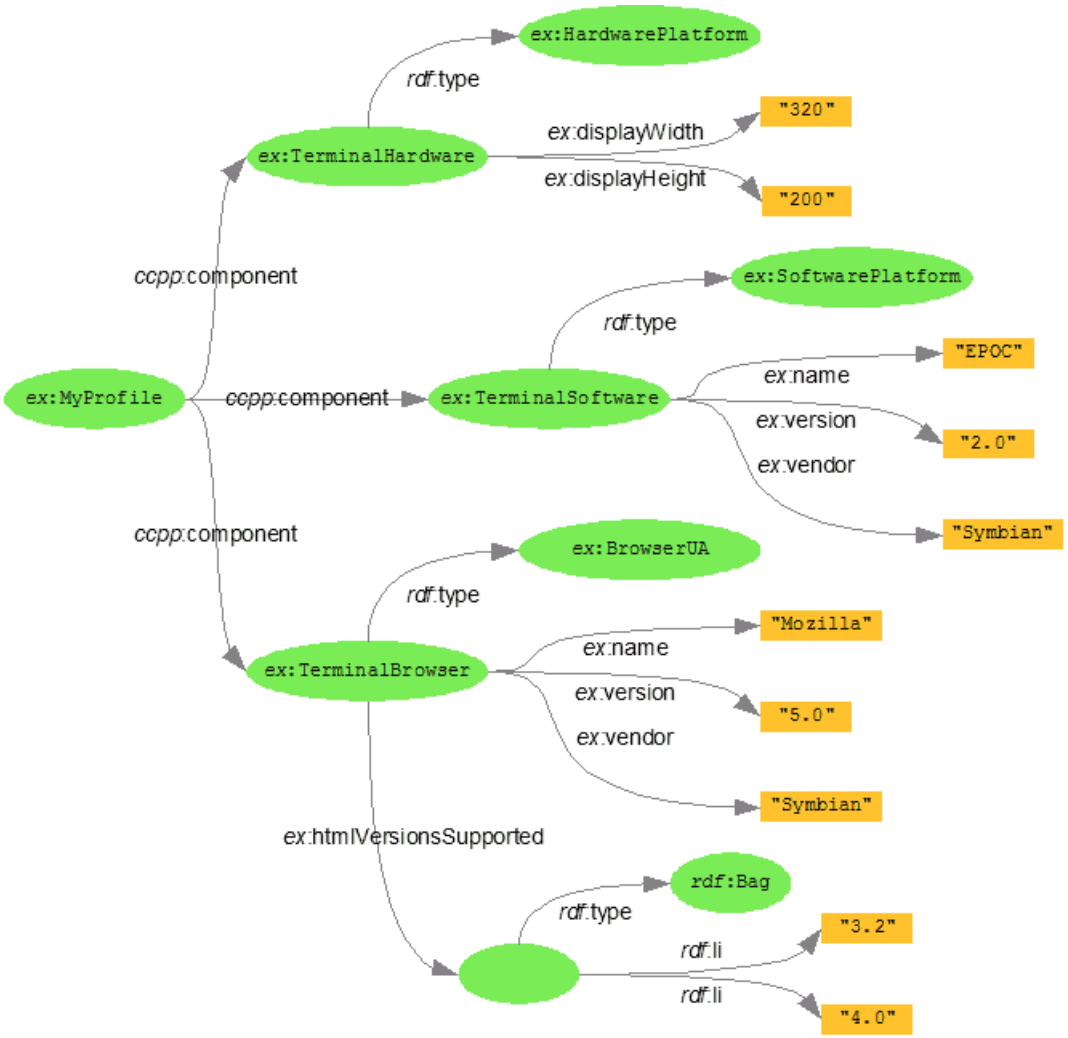

Abbildung 2.4: CC/PP Example [W3C 5, 2007]

2.5 Frameworks

Bei der Betrachtung verschiedener Softwaresysteme lassen sich häufig Gemeinsamkeiten, bzw. Redundanzen in ihrer Funktionalität und ihrem Design ausmachen. Dies ist insbesondere der Fall, wenn die betrachteten Softwaresysteme zu einer vergleichbaren Kategorie gehören. Mit Hilfe sog. Frameworks [Gamma et al., 1995] [Szyperski, 2002] lassen sich unnötige Redundanzen vermeiden.

Ein Framework gibt einen Kontrollfluss und eine Architektur vor. Ein Framework kann konkrete Funktionalität beinhalten, ist dabei aber keine eigenständige Applikation. Auf dem Framework aufbauende Applikationen, instanziieren das Framework und ergänzen

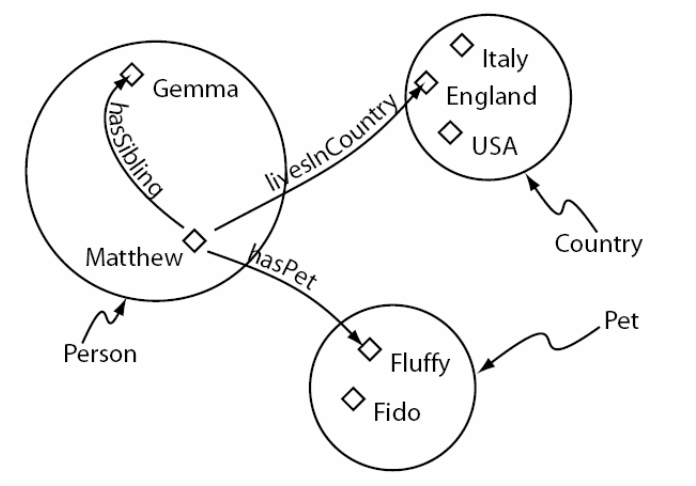

Abbildung 2.5: OWL Example [Horridge et al., 2007]

es um ihre eigene Funktionalität. Da das Framework für den Kontrollfluss zuständig ist, werden Komponenten des Frameworks nicht durch die konkreten Applikationen aufgerufen, sondern das Framework ruft Komponenten der Applikationen auf. Dieses Prinzip wird auch *Inversion of Control* [Gamma et al., 1995] genannt.

Weiterhin lassen sich sog. Black-Box und White-Box Frameworks unterscheiden [Reussner et al., 2006] [Szyperski, 2002]. Bei Black-Box Frameworks muss ein Anwendungsentwickler, der das Framework nutzen möchte, lediglich die externen Schnittstellen der Framework Komponenten kennen. Eine Kenntnis der internen Architektur und Abläufe des Frameworks ist nicht erforderlich. Beim White-Box Framework hingegen, wird vorausgesetzt, dass ein Anwendungsentwickler den internen Aufbau des Frameworks kennt. Dies bietet einerseits mehr Freiheiten, erfordert aber auch mehr Aufwand für die Einarbeitung. Natürlich sind auch Mischformen möglich.

Abgrenzung: Der Ansatz des Frameworks unterscheidet sich stark von dem einer Klassenbibliothek. Die Aufgabe einer Klassenbibliothek ist hauptsächlich die Kapselung von Funktionalität. Applikationen, die eine Klassenbibliothek verwenden müssen sich nicht mit den Implementierungsdetails der von der Klassenbibliothek angebotenen Funktionalität auseinandersetzen. Die Verantwortung für den Kontrollfluss und die Architektur liegt jedoch bei der konkreten Applikation.

2.6 Technologien

Für die Entwicklung von Softwaresystemen auf mobilen Geräten, existieren eine Vielzahl von Technologien. Im Rahmen dieser Arbeit spielt die Wahl von konkreten Technologien eine untergeordnete Rolle, da der konzeptionelle Aspekt im Vordergrund steht. Der Vollständigkeit halber werden an dieser Stelle jedoch die zwei wichtigsten Technologien vorgestellt. Für eine nähere Betrachtung sei auf die angegebenen Quellen verwiesen.

.Net Compact Das .Net Compact Framework [Microsoft, 2004] stellt eine Untermenge des .Net Frameworks dar. Es ist speziell für die Entwicklung von Anwendungen für die Windows Mobile Betriebssysteme von Microsoft ausgerichtet [Salvatori, 2006]
[Dröge et al., 2006].

Java ME Wie auch das .Net Compact Framework, bietet Java ME eine Untermenge der vollständigen Java Distribution [Sun, 2008]. Allerdings können mit Java ME Anwendungen für diverse mobile Betriebssysteme erstellt werden, die Java ME unterstützen. Dies stellt einen Vorteil gegenüber dem .Net Compact Framework dar. Allerdings ist der Funktionsumfang von Java ME stärker eingeschränkt, als der des .Net Compact Framework. Dies kann zu erhöhtem Implementierungsaufwand führen.

2.7 Zusammenfassung

Der Begriff Context-Awareness gewinnt zunehmend an Bedeutung. Wie bereits beschrieben (Kapitel 2.3), ist seine Definition nicht eindeutig. Es entsteht leicht der Eindruck, dass es sich bei ”Context-Awareness” um ein völlig neues Thema handelt. Der Begriff wurde jedoch schon Mitte der Neunziger Jahre geprägt. Einige der unter Kapitel 2.3.1 beschriebenen Beispiele für Context-Awareness sind sogar bedeutend älter als der Begriff selbst. Es stellt sich also die Frage, warum dieses Thema momentan so viel Beachtung findet. Die Beispiele unter Kapitel 2.3.1 zeichnen sich alle dadurch aus, dass jeweils nur sehr begrenzte Context-Informationen genutzt werden. Wenn man die aktuellen Entwicklungen betrachtet (Kapitel 3.4), wird deutlich, dass der Trend im Bereich der Context-Awareness dahin geht, viele verschiedene Context-Informationen zu nutzen. Dazu werden die verschiedenen Informationen meist akkumuliert und in abstraktere Informationen überführt. Aufschwung

bekommt das Thema zusätzlich durch die zunehmende Verbreitung leistungsfähiger mobiler Geräte.

Kapitel 3

Analyse

Wie bereits in Kapitel 1.2 erwähnt, soll im Rahmen dieser Arbeit ein Framework für Context-Awareness auf mobilen Geräten entwickelt werden. Im Folgenden wird nun die Vision, die dieser Arbeit zu Grunde liegt, konkretisiert (Kapitel 3.1). Weiterhin werden, basierend auf der Vision, funktionale und nichtfunktionale Anforderungen [Ebert, 2005] [Rupp et al., 2007] an das Framework definiert (Kapitel 3.2 und 3.3). Die funktionalen und nichtfunktionalen Anforderungen dienen in Kapitel 4 als Randbedingungen für einen Architekturentwurf[1]. Die Analyse in diesem Kapitel umfasst außerdem die Auseinandersetzung mit verwandten Themen und Arbeiten (Kapitel 3.4).

3.1 Vision

Die Vision[2] für das zu entwickelnde "Framework für Context-Awareness auf mobilen Geräten" muss aus zwei unterschiedlichen Gesichtspunkten dargestellt werden. Aus Entwicklersicht soll das Framework den Einsatz von Context-Awareness in Applikationen erleichtern, indem den Entwicklern die Akkumulation und Analyse von Context Informationen so weit wie möglich abgenommen wird. Entwickler sollen durch das Framework in die Lage versetzt werden, schnell und effizient Applikationen zu entwickeln, die Context Informationen für ihre Programmlogik verwenden. Aus Benutzersicht soll der Nutzen eines mobilen Gerätes erhöht werden, indem verschiedene Context Informationen kombiniert und für ein schein-

[1] Um auf die jeweiligen Anforderungen referenzieren zu können, sind diese hierarchisch nummeriert.

[2] In [Ebert, 2005] wird erläutert, was unter dem Begriff "Vision" im Rahmen eines IT Projekts zu verstehen ist.

bar ”intelligentes” Verhalten von Applikationen verwendet werden (siehe Szenario: Kapitel 1.3). Dieses zweite Ziel kann nicht direkt durch das Framework erreicht werden. Da mit dem Framework jedoch die Komplexität der Entwicklung von Context-Sensitiven Applikationen erheblich verringert wird, kann das Framework indirekt zur Erreichung dieses Zieles beitragen.

Die Vision für das Framework umfasst weiterhin, dass einerseits ein hoher Abstraktionsgrad erreicht werden soll (d.h. es sollen möglichst beliebige Arten von Context Informationen verarbeitet werden können), aber andererseits diese Abstraktion nicht dazu führen darf, dass das Framework zu viele Verantwortlichkeiten auf die konkreten Applikationen verlagert. Das Framework muss sich also möglichst generisch verhalten.

Die Neuentwicklung eines entsprechenden Frameworks ist notwendig, da es auf diesem Gebiet bisher praktisch keine Standardlösungen gibt. Es finden sich zwar einige Konzepte in diversen Publikationen. Die meisten sind jedoch auf konkrete Context Informationen spezialisiert (z.B. Location Awareness), verfolgen andere Aspekte von Context Awareness als der Ansatz in dieser Arbeit und/oder sind eher abstrakter, bzw. theoretischer Natur (bieten also keine Implementierungen, die für andere Arbeiten weiterverwendet/erweitert werden könnten). Siehe dazu auch Kapitel 3.4.

3.1.1 Beispiele

Um die abstrakte Vision zu konkretisieren, werden im Folgenden zwei Beispiele aufgezeigt.

Location Awareness Positionsinformationen sollen vom Framework verwaltet werden. Dabei soll von der Lokalisierungs-Technologie wie GPS, IMAPS oder Cricket abstrahiert werden. Applikationen, die mit dem Framework realisiert werden, müssen sich dann nicht mit den technischen Eigenheiten der verschiedenen Technologien beschäftigen und können außerdem durch Eventhandling über Positionsänderungen, Signalqualität usw. informiert werden. Somit müssen Applikationen nur reaktiv auf Änderungen eingehen. Darüber hinaus soll das Framework auch komplexere Context Informationen zur Verfügung stellen. Im Fall der Location Awareness bedeutet dies, dass z.B. Positionsänderungen durch das Framework in abstrakte Beschreibungen, wie ”Benutzer geht” oder ”Benutzer rennt”, überführt werden.

Client-Server Wie in Kapitel 1 beschrieben, ist die Konnektivität von mobilen Geräten nicht durchgehend und nicht mit konstanter Qualität gegeben. Aus Sicht von Context-

Awareness bedeutet dies, dass Informationen über Dienstgüte, Stabilität von Verbindungen, aber auch weniger offensichtliche Informationen wie die verbleibende Laufzeit, Rechenleistung und Display-Eigenschaften von mobilen Geräten für eine Beeinflussung von Client-Server Kommunikation genutzt werden können. Es ist beispielsweise denkbar, dass ein Client bei der Kommunikation mit einem Server mitteilt, wie "schnell" seine aktuelle Verbindung ist, wie viele Farben sein Display darstellen kann und welche Auflösung sein Display hat. Der Server könnte anhand dieser Informationen entscheiden, ob er dem Client zu seiner angefragten Information über reinen Text hinaus noch Bilder oder Videos schicken kann. Dies ist nur eines von vielen denkbaren Szenarien.

3.2 Funktionale Anforderungen

Bei "normalen" Applikationen/Systemen haben die funktionalen Anforderungen die Aufgabe, zu beschreiben, was das System tun soll. Dies betrifft sowohl interne Abläufe, als auch die Interaktion des Systems mit seiner Umgebung. Diese Anforderungen leiten sich üblicherweise aus Use Cases ab. Dabei spielen oft Geschäftsprozesse eine wichtige Rolle. Bei einem Framework ist die Ermittlung von funktionalen Anforderungen etwas schwieriger, da ein Framework üblicherweise nicht der Bewältigung konkreter Use Cases, bzw. Geschäftsprozesse dient. Die funktionalen Anforderungen an ein Framework sind daher eher abstrakter Natur.

3.2.1 Trennung von Framework- und Applikationslogik

Bei der Entwicklung eines Frameworks ist es wichtig, zu spezifizieren, welche Aufgaben durch die Programmlogik der Framework Komponenten erfüllt werden sollen. Dies muss von den Aufgaben der auf dem Framework basierenden Applikationen abgegrenzt werden. Ziel dabei muss sein, dem Framework soviel Verantwortung wie möglich zu geben, um einen maximalen Nutzen für Applikationen, die auf dem Framework basieren, zu erzielen. Gleichzeitig darf das Framework nicht zu restriktiv und unflexibel gestaltet werden, um ein möglichst breites Spektrum an Applikationen[3] zu ermöglichen [Parsons et al., 1999].

[3]Im Folgenden werden die auf dem Framework basierenden Applikationen nur noch als "Applikationen" bezeichnet.

A.1 Anbindung von Context Quellen

Die Verantwortung für die Anbindung von Context Quellen liegt bei Komponenten des Framework. Applikationen greifen nie direkt auf Context Quellen zu.

Das Framework soll

1. fähig sein, Context Informationen von unterschiedlichen Context Quellen zu beziehen und zu verarbeiten. Welche Context Quellen benötigt werden, hängt von den Anforderungen der konkreten Applikationen ab. Um für möglichst viele Applikationen geeignet zu sein, darf das Framework nicht auf bestimmte Context Quellen beschränkt sein.

2. erweiterbar sein, um die Anbindungen weiterer Context Quellen zu ermöglichen. Dies sollte mit möglichst geringem Aufwand möglich sein. D.h. die Anbindung einer neuen Context Quelle darf nicht zur Folge haben, dass große Teile des Frameworks geändert werden müssen. Es muss ein Standard Verfahren für die Erweiterung des Frameworks um neue Context Quellen geben.

3. die Kommunikation mit den Context Quellen kapseln. Applikationen sollen nie direkt auf die Context Quellen zugreifen.

4. die Anbindung von Context Quellen über externe Schnittstellen ermöglichen. Somit können bei Bedarf verschiedene Programmiersprachen verwendet werden um Context Quellen an das Framework anzubinden.

5. in der Lage sein, Applikationen, die auf dem Framework basieren als Context Quellen zu verwenden. Diese Rückkopplung ermöglicht es, Context Informationen einer spezifischen Applikation für andere Applikationen bereit zu stellen, ohne dafür einen speziellen Mechanismus einführen zu müssen.

Context Quellen müssen

6. standardisierte Schnittstellen anbieten und Protokolle einhalten, um mit Komponenten des Frameworks kommunizieren zu können. Dies ist eine wichtige Voraussetzung, um beliebige Context Quellen an das Framework anbinden zu können. Um den Implementierungsaufwand pro Context Quelle so gering wie möglich zu halten, sollte das Framework für diese Aufgabe Standard Komponenten anbieten, die dann für die jeweiligen Context Quellen erweitert werden.

7. Context Informationen

   - aktiv an das Framework senden.
   - passiv bereitstellen. (Diese können dann über Schnittstellen angefragt werden.)

   Welche der beiden Alternativen zu bevorzugen ist, hängt von den konkreten Context Quellen ab. Eine Lokalisations-Komponente sollte beispielsweise die Änderung der aktuellen Position aktiv an das Framework übermitteln, wohingegen eine Kalender-Komponente bei Bedarf vom Framework angesprochen werden sollte. Die Grenzen sind dabei jedoch fließend, da eine Kalender-Komponente durchaus auch aktive Elemente haben kann (z.B. Erinnerungen).

8. eine Schnittstelle anbieten, über die das Framework Context Informationen "abonnieren", bzw. "abbestellen" kann (subscribe/unsubscribe). So kann die Kommunikation auf die Context Quellen und Informationen beschränkt werden, die für die aktuell laufenden Applikationen relevant sind.

9. Context Informationen in ein einheitliches Format überführen. (z.B. XML oder RDF/XML) Dies ist ebenfalls eine wichtige Voraussetzung, um beliebige Context Quellen an das Framework anbinden zu können.

A.2 Anbindung von Applikationen

Die Verantwortung für die Anbindung von Applikationen liegt sowohl bei Komponenten des Frameworks, als auch bei den Applikationen selbst.

Das Framework soll

1. die Kommunikation mit den Applikationen kapseln, so dass Applikationen nicht direkt mit Komponenten auf verschiedenen Ebenen des Frameworks kommunizieren.

2. definierte externe Schnittstellen bieten, über die sich Applikationen beim Framework registrieren können. Somit können bei Bedarf verschiedene Programmiersprachen verwendet werden um Applikationen an das Framework anzubinden.

3. fähig sein, Context Informationen an mehrere Applikationen (parallel) zu verteilen. Also die Anbindung mehrerer Applikationen zur gleichen Zeit ermöglichen.

4. den Applikationen die Context Informationen sowohl aktiv, als auch passiv zur Verfügung stellen können. Abhängig von den Anforderungen einer konkreten Applikation

können dann Context Informationen entweder durch die Applikation "abonniert" oder bei Bedarf direkt angefragt werden.

Applikationen werden

5. sich beim Framework registrieren, indem sie
   - von einer Klasse des Frameworks erben, welche die Grundfunktionen für die Anbindung an das Framework bereitstellt.
   - sich direkt über eine externe Schnittstelle des Frameworks registrieren. (ermöglicht bei Bedarf die Verwendung verschiedener Programmiersprachen)
6. abstrahierte Context Informationen (in Abhängigkeit von konkreten Anforderungen)
   - aktiv anfragen.
   - "abonnieren" und dann durch das Framework erhalten.

A.3 Verarbeitung von applikationsspezifischer Logik

Das Framework soll

1. keine applikationsspezifische Logik in hart codierter Form enthalten, jedoch um Regeln erweiterbar sein, die für konkrete Applikationen oder Applikationsklassen festlegen, wie Context Informationen von verschiedenen Context Quellen zu interpretieren und weiterzuleiten sind (siehe Szenario: Kapitel 1.3).

Applikationen werden

2. **nicht** für die Abstraktion und Akkumulation von Context Informationen verantwortlich sein.
3. nur über definierte Schnittstellen mit den Framework Komponenten kommunizieren.
4. die Verantwortung für alle weitere applikationsspezifische Logik tragen.

3.2.2 Konfiguration des Frameworks zur Laufzeit

Es wird davon ausgegangen, dass zentrale Komponenten des Frameworks in Betrieb sein müssen, solange Applikationen an das Framework angebunden sind. Da verschiedene Applikationen nicht zwangsläufig zur gleichen Zeit benötigt werden, sollte die Anbindung von

Applikationen zur Laufzeit möglich sein. Da in Abhängigkeit der angebundenen Applikationen unterschiedliche Context Informationen benötigt werden, muss dynamisch entschieden werden können, von welchen Context Quellen die Informationen zu beziehen sind. Aus Performance-Gründen sollten nicht ständig alle Context Informationen sämtlicher verfügbarer Context Quellen vom Framework verarbeitet werden müssen.

Auf die Anforderungen an die Konfigurierbarkeit des Frameworks zur Laufzeit, wird im Folgenden näher eingegangen.

B.1 Registrierung von Context Quellen

Um die Komplexität der Gesamtlösung in einem angemessenen Rahmen zu halten, wird vorausgesetzt, dass potentielle Context Quellen bekannt sein müssen. Diese müssen jedoch nicht ständig verfügbar, bzw. an das Framework angebunden sein. Bekannte Context Quellen können sich dann zur Laufzeit registrieren. Siehe dazu auch Kapitel 3.2.1.

B.2 Registrierung von Applikationen

Im Gegensatz zu Context Quellen, kann bei Applikationen nicht davon ausgegangen werden, dass sie den Framework-Komponenten bekannt sind. Dies ist auch nicht notwendig, da Context Informationen von den Applikationen nur konsumiert werden. Eine Applikation muss dem Framework daher bei der Registrierung zu Laufzeit nur mitteilen, an welchen Context Informationen sie interessiert ist. Siehe dazu auch Kapitel 3.2.1.

B.3 Applikationsspezifische Regeln

Abgesehen von einem Standard Set an Regeln soll das Framework um applikationsspezifische Regeln erweiterbar sein. Dies ist notwendig, da je nach konkreter Applikation die Context Informationen anders aufbereitet und weitergeleitet, bzw. zur Verfügung gestellt werden müssen. Ohne solche Regeln, könnte nicht entschieden werden, welche Informationen für eine konkrete Applikation relevant sind. Dafür muss das Framework:

1. Regeln in einem standardisierten Format annehmen können. (z.B. XML oder RDF/XML)
2. diese Regeln zu den bereits bestehenden Regeln hinzufügen. (die bestehenden Regeln also nicht überschreiben)

3. die Regeln einer Applikation zuordnen können. Konflikte zwischen den Regeln für verschiedene Applikationen müssen vermieden werden.
4. Fehlertolerant in Bezug auf die Regeln sein. (Widersprüchlich, unvollständig oder anderweitig fehlerhaft formulierte Regeln dürfen die Funktionalität des Frameworks nicht beeinträchtigen.)

3.2.3 Verarbeitung von Context Informationen

Die Verarbeitung von Context Informationen ist die Hauptaufgabe des Frameworks. Sie gliedert sich in drei wesentliche Aufgabengebiete: Context Akkumulation, Context Abstraktion, Context Distribution. Auf die Anforderungen bezüglich dieser drei Aufgabengebiete wird im Folgenden eingegangen.

C.1 Context Akkumulation

Das Framework muss Informationen verschiedenster Context Quellen entgegennehmen und zusammenführen können. Da voraussichtlich nicht alle Context Quellen ihre Context Informationen in einem einheitlichen Datenformat bereitstellen, ist es Aufgabe des Frameworks, eine Transformation der Context Informationen in ein einheitliches Datenformat vorzunehmen. Weiterhin fällt unter die Akkumulation von Context Informationen auch deren Verwaltung. Daher muss das Framework u.a. in der Lage sein, neue Context Informationen aufzunehmen, vorhandene Context Informationen zu aktualisieren und diese bei Bedarf auch zu verwerfen (z.B. wenn diese veraltet oder inkonsistent sind).

C.2 Context Abstraktion

Unter der Abstraktion von Context Informationen, versteht man die Überführung von ”low level” Context Informationen in ”high level” Context Informationen (Kapitel 2.3). Dies soll vom Framework anhand von Regeln vollzogen werden. Diese Regeln sollen ein standardisierten Format (z.B. XML oder RDF/XML) haben. Änderungen, bzw. Erweiterungen dieser Regeln sollen keine Codeänderungen an den Framework Komponenten erfordern.

C.3 Context Distribution

Die Distribution von (abstrahierten) Context Informationen an die Applikationen soll entsprechend den applikationsspezifischen Regeln erfolgen. Diese Regeln müssen dazu beschreiben, welche Context Informationen von der jeweiligen Applikation benötigt werden. Das Framework muss diese Regeln den konkreten Applikationen zuordnen können.

3.2.4 Persistenz von Context Informationen

Es ist nicht Aufgabe des Frameworks, Context Informationen persistent zu speichern. Das Framework dient lediglich der Akkumulation, Abstraktion und Distribution von Context Informationen. Jedoch sollen zur Laufzeit, aus Performance Gründen, durchaus Context Informationen von Komponenten des Frameworks vorgehalten werden.

3.3 Nichtfunktionale Anforderungen

Nichtfunktionale Anforderungen beziehen sich hauptsächlich auf Qualitätsmerkmale eines Systems. Da eine Realisierung/Implementierung des in Kapitel 4 vorgestellten Entwurfs, im Rahmen dieser Arbeit in Form eines Prototyps geplant ist, spielen die nichtfunktionalen Anforderungen eine untergeordnete Rolle. Einige nichtfunktionale Anforderungen haben jedoch durchaus Einfluss auf grundlegende Architekturentscheidungen, daher ist es wichtig sich trotzdem mit diesen auseinander zu setzten. Im Folgenden wird auf die nichtfunktionalen Anforderungen an die zu entwickelnde Lösung eingegangen.

Nach [DIN 66272], bzw. [ISO/IEC 9126] sind mindestens die folgenden Aspekte bei der Analyse von nichtfunktionalen Anforderungen zu beachten.

3.3.1 Zuverlässigkeit

Die Zuverlässigkeit bewertet die Verfügbarkeit und die Korrektheit der Software. Im vorliegenden Fall bedeutet dies u.A, dass Ausfälle/Fehlverhalten einer Context Quelle vom Framework kompensiert werden müssen. Dies gilt ebenso für Ausfälle/Fehlverhalten von Applikationen, die auf dem Framework basieren. Die verschiedenen Context Quellen und Applikationen dürfen sich also nicht gegenseitig negativ beeinflussen.

Im Hinblick auf die Verfügbarkeit muss beachtet werden, dass beim Betrieb mehrerer Applikationen keine der Applikationen benachteiligt wird. Die Context Informationen müssen für alle Applikationen gleichermaßen zur Verfügung stehen. D.h. auch wenn eine Applikation das Framework z.B. durch ständige Anfragen zu blockieren droht, müssen für die anderen Applikationen ausreichend Ressourcen reserviert werden (siehe auch Kapitel 3.3.3).

Da das Framework Informationen sammelt, verarbeitet und weiterleitet, muss außerdem sichergestellt werden, dass diese Informationen nicht durch das Framework verfälscht werden. Korrektheit bedeutet hier, dass Informationen auf dem Weg von den Context Quellen zu den Applikationen nicht in ihrem Informationsgehalt verändert werden dürfen. Außerdem hat dieses Qualitätsmerkmal eine zeitliche Komponente. Veraltete Context Informationen müssen vom Framework erkannt und verworfen, bzw. ersetzt werden.

3.3.2 Benutzbarkeit

Die Benutzbarkeit (engl. usability) steht für die Benutzerfreundlichkeit. Dieses Qualitätsmerkmal befasst sich u.a. damit, wie komfortabel sich ein Programm bedienen lässt und wie hoch der Aufwand ist, den Umgang mit dem Programm zu erlernen. Da im Rahmen dieser Arbeit ein Framework und keine konkrete Applikation, bzw. GUI entwickelt werden soll, lässt sich dieses Qualitätsmerkmal nur bedingt anwenden.

Aus Entwicklersicht, die in gewissem Sinne die ”Anwender” des Frameworks darstellen, lässt sich festhalten, dass eine gute Lesbarkeit und Struktur des Quellcodes anzustreben ist. Änderungen und Erweiterungen am Framework sollten stets auf so wenige logische Komponenten wie möglich beschränkt bleiben. Noch wichtiger ist jedoch, dass der Einsatz des Frameworks in Standard Fällen keine tiefgreifenden Kenntnisse über die innere Struktur der Framework Komponenten erfordern sollte. So sollte z.B. die Anbindung einer neuen Applikation (welche nur bestehende Funktionalitäten des Frameworks nutzen soll) lediglich Kenntnisse über die externen Schnittstellen des Frameworks erfordern.

3.3.3 Effizienz

Die Effizienz beschreibt das zeitliche Verhalten bei Anfragen, sowie den Ressourcenverbrauch. In der angestrebten Lösung soll es möglich sein, mehrere Applikationen parallel mit den jeweils benötigten Context Informationen zu versorgen. Die Anbindung einer zusätzlichen Applikation sollte die Performance des Gesamtsystems nur marginal beeinflussen.

Dies gilt ebenfalls für die angebundenen Context Quellen. Was die maximale Anzahl an Applikationen und Context Quellen ist, die gemeinsam "flüssig" betrieben werden kann, hängt von der Leistungsfähigkeit der eingesetzten Hardwarekomponenten ab und muss jeweils experimentell bestimmt werden.

Für das Antwortverhalten gilt allgemein, dass die Verarbeitung von Context Information und die Anwendung von Regeln in "angemessener" Zeit erfolgen muss. Z.B. sollte die Verarbeitung einer GPS Positionsänderung erfolgt sein, bevor die nächste Positionsänderung eintrifft. Um eine Akzeptanz bei den Endanwendern zu gewährleisten, sollten Verzögerungen im Antwortverhalten durch den Overhead der Framework Komponenten möglichst nicht (oder nur gering) über die menschliche Wahrnehmungsschwelle[4] hinausgehen.

3.3.4 Änderbarkeit

Dieses Qualitätsmerkmal beschreibt den Aufwand für Änderungen am System. Die Änderungen können u.a. der Erweiterung, Fehlerbehebung oder Anpassung dienen. Änderbarkeit/Erweiterbarkeit sind primäre Anforderungen an das Framework, da das Hinzufügen neuer Context Quellen voraussichtlich häufig erfolgen wird. Dies gilt ebenfalls für die Regeln des Frameworks, die beschreiben, wie Context Informationen verarbeitet werden sollen. Darüber hinaus sollte es mit möglichst wenig Aufwand verbunden sein, das Framework auf verschiedenen Zielplattformen einzusetzen (siehe auch Kapitel 3.3.5). Weiterhin ist ein möglichst modularer Aufbau des Frameworks anzustreben, damit Änderungen/Erweiterungen einzelner Komponenten unabhängig von den restlichen Komponenten erfolgen können.

3.3.5 Übertragbarkeit

Die Übertragbarkeit beschreibt, ob und mit welchem Aufwand eine Software auf anderen Plattformen/Systemen eingesetzt werden kann. Als Anforderung an das Framework bedeutet dies, dass es möglichst auf verschiedenen mobilen und evtl. auch auf nicht mobilen Geräten einsetzbar sein sollte. Außerdem sollten Context Quellen, Kernkomponenten des Frameworks, sowie Applikationen, die auf dem Framework basieren nicht zwangsläufig auf dem selben physikalischen Gerät betrieben werden müssen.

[4] http://de.wikipedia.org/wiki/Zeitwahrnehmung#Schwellen

3.4 Verwandte Themen und Arbeiten

Das Thema "Context-Awareness" im Allgemeinen, sowie im Speziellen in Verbindung mit mobilen Geräten ist ein aktives Forschungsfeld. Es gibt diverse Arbeiten, die sich mit verschiedenen Teilaspekten dieses Themas beschäftigen. Die Bandbreite ist dabei enorm. Um einen Überblick zu geben und gleichzeitig eine Abgrenzung gegenüber dieser Arbeit vorzunehmen, werden im Folgenden einige interessante Arbeiten vorgestellt.

3.4.1 A Context Ontology for Pervasive Service Provision

In dieser Arbeit wird ein Context Management System vorgestellt, welches Ontologien für die Verarbeitung von Context Informationen nutzt [Strimpakou et al., 2006]. Ziel dabei ist, Applikationen und die Ermittlung von Context Informationen voneinander zu trennen. Zu diesem Zweck wird eine Middleware eingesetzt, die die Aufgaben der Context Akkumulation, Speicherung, Verwaltung und Verteilung übernimmt. Nach Angaben der Autoren wurde das System in Form eines Prototyps umgesetzt. Aufgrund der sehr beschränkten Informationen zu dieser Arbeit, ist ein detaillierter Vergleich mit der vorliegenden Arbeit nicht möglich.

3.4.2 Hydrogen Context-Framework

Bei dem Hydrogen Context-Framework [Hofer et al., 2003] handelt es sich um ein Java basiertes Framework. Es ist auf eine modulare Anbindung verschiedenster Sensoren (bzw. Informationsquellen) und Applikationen ausgelegt. Zentraler Punkt des Frameworks ist ein Context Server, der die verschiedenen Informationen der Sensoren akkumuliert und den Applikationen in abstrahierter Form zur Verfügung stellt. Der Ansatz und die Zielsetzung des Hydrogen Context-Frameworks sind denen der vorliegenden Arbeit sehr ähnlich. Allerdings sind die verfügbaren Informationen sehr begrenzt und abstrakt, wodurch ein detaillierter Vergleich verhindert wird.

3.4.3 Sensor-based Context-Awareness

...for Adaptive PDA User Interfaces [Schmidt et al., 1998]. Diese Arbeit beschäftigt sich mit den Einsatzmöglichkeiten von Sensoren zur automatisierten Anpassung von PDA Displays,

bzw. den darauf dargestellten Benutzerschnittstellen. Als Beispiele werden Lichtsensoren genannt, anhand derer eine automatische Anpassung der Display Helligkeit an das Umgebungslicht ermöglicht wird. Zusätzlich werden Lagesensoren betrachtet, mit deren Hilfe sich z.B. die Darstellung auf einem Display stets "richtig" herum drehen lässt. D.h. wenn ein Benutzer einen PDA dreht, muss er die Darstellung nicht manuell anpassen. Die Beschränkung auf die Anpassung von Displays unterscheidet sich von der Ausrichtung auf beliebige Context-Sensitive Systeme, die in der vorliegenden Arbeit im Mittelpunkt steht.

3.4.4 Pervasive Gaming Framework

Das Pervasive Gaming Framework wurde im Rahmen eines Projekts an der HAW Hamburg entwickelt [Weindorf, 2007]. Es handelt sich dabei um ein Framework für ortsbezogene Spiele auf mobilen Geräten. Als Context Informationen wird in diesem Framework hauptsächlich die aktuelle Position des mobilen Clients verwendet. Zur Bestimmung der Position wird ein GPS Empfänger eingesetzt. Das Framework ist spezialisiert auf Spiele, bzw. Applikationen, die Routen Punkte (Sequenz von Orten) für ihre Programmlogik benötigen. Es können nicht beliebige Context Informationen verwenden werden, wie es durch den Ansatz der vorliegenden Arbeit ermöglicht wird.

3.4.5 Weitere interessante Arbeiten

- Design and Implementation of a Large-Scale Context Fusion Network [Chen et al., 2004]
- Call Forwarding: Active Badge location System ACM, 1992. Beschrieben durch [Chen und Kotz, 2000]
- Context-aware Service Protocol [Tan et al., 2003]
- A Formalism for Context-Aware Mobile Computing [Yan und Sere, 2004]
- Plattform für Smartphonebasierte ortsabhängige Interaktionen [UbiComp]

Kapitel 4

Entwurf

Unter Berücksichtigung der Anforderungen, die in Kapitel 3 beschrieben wurden, wird in diesem Kapitel ein Entwurf für das Framework entwickelt. Zu diesem Zweck wird in Kapitel 4.1 vorerst ein grobes Architektur Konzept für das Framework vorgestellt. Dieses Konzept wird in Kapitel 4.2 verfeinert, indem die einzelnen Komponenten des Frameworks und ihre Interaktionen näher beschrieben werden. Abschließend erfolgt ein Abgleich mit den Anforderungen aus Kapitel 3.

4.1 Architektur Konzept

Abbildung 4.1 zeigt das abstrakte Design des Frameworks. Zentrale Komponente ist ein ***Context Server***, der für die Akkumulation, Verwaltung und Abstraktion von Context Informationen zuständig ist. Wichtigste Bestandteile des *Context Servers* sind die ***Rule Engine*** und der ***Context Store***. Der *Context Store* dient der Verwaltung von Context Informationen, die von verschiedenen Context Quellen bezogen werden. Die *Rule Engine* dient der Verarbeitung dieser Context Informationen, basierend auf Regeln, die in Form eines standardisierten Formats bereitgestellt werden. Es wird dabei zwischen allgemeinen und applikationsspezifischen Regeln unterschieden. Die applikationsspezifischen Regeln werden über eine externe Schnittstelle zur *Rule Engine* hinzugefügt. Dies ermöglicht eine Konfiguration der *Rule Engine* zu Laufzeit. Die allgemeinen Regeln müssen hingegen nicht zur Laufzeit veränderbar sein. Um sie jedoch trotzdem flexibel zu halten, werden sie ebenfalls über eine externe Schnittstelle beim Starten des *Context Servers* eingelesen. Es kommen diverse Formate, wie z.B. XML oder RDF/XML (siehe Kapitel 2.4), für die Regel Re-

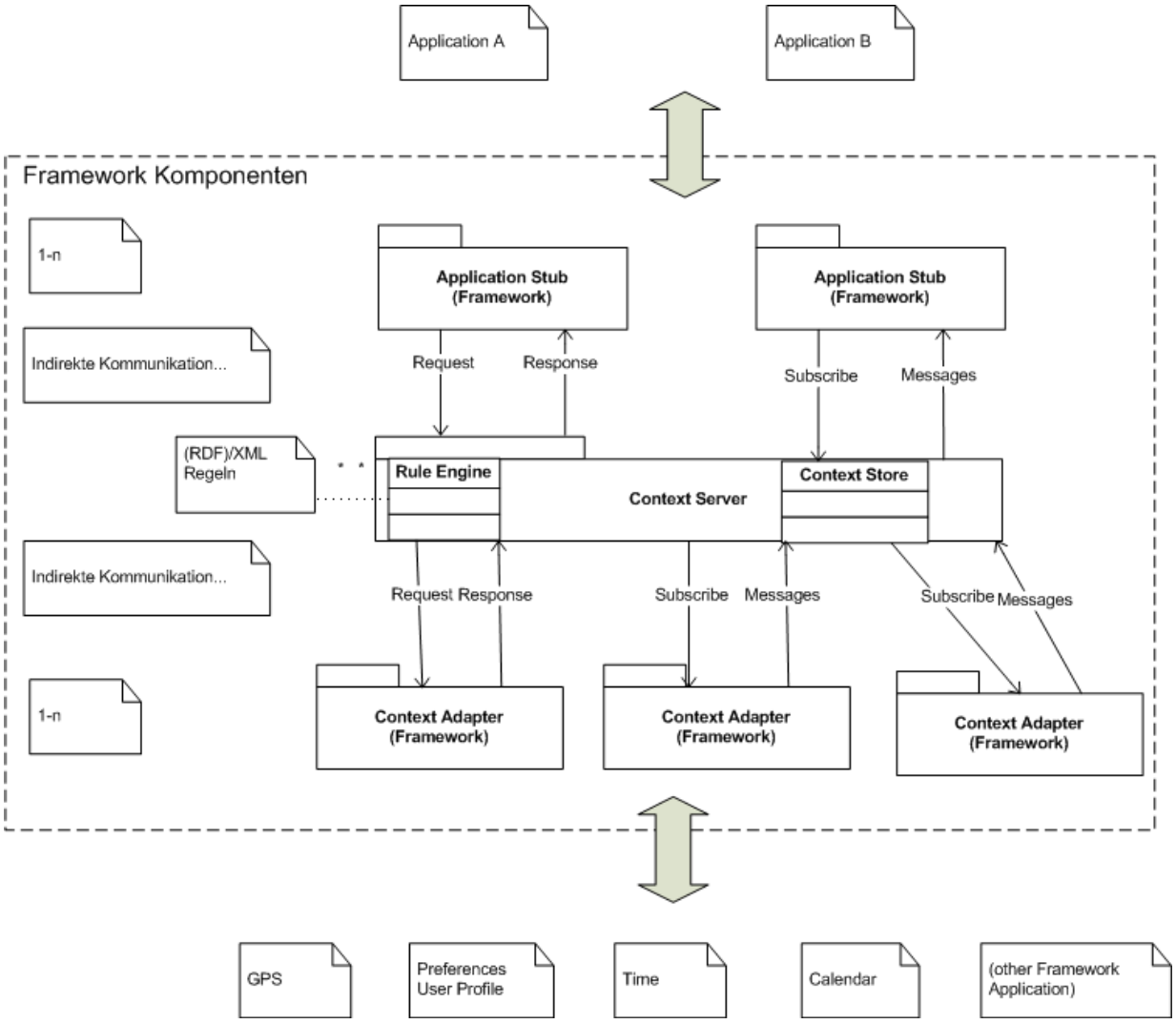

Abbildung 4.1: Konzept

präsentation infrage. Für die Erfüllung der an die Lösung gestellten Anforderungen, ist eine XML Repräsentation vorerst ausreichend. Im Folgenden werden daher alle Beispiele in einem XML Format dargestellt.

Weiterhin bietet das Framework **Context Adapter**, welche die Kommunikation mit dem *Context Server* kapseln und außerdem dafür verantwortlich sind, Context Informationen der Context Quellen in ein standardisiertes Format zu überführen.

Für die Anbindung von Applikationen an das Framework sind die **Application Stubs** (bzw. Adapter) zuständig. Sie kapseln, wie auch die *Context Adapter*, die Kommunikation mit dem *Context Server*. Außerdem bieten sie definierte Schnittstellen, über die Context

Informationen für die konkreten Applikationen leicht zugänglich sind.

Anmerkung: Die Kommunikation zwischen *Context Server* und *Context Adaptern*, bzw. *Application Stubs* erfolgt über Standard Technologien (siehe Kapitel 4.3). Daher können Applikationen und Context Quellen theoretisch auch direkt mit dem *Context Server* kommunizieren. Voraussetzung dafür ist, dass sie das gleiche Protokoll wie die Komponenten des Frameworks nutzen. Außerdem ermöglicht dies die Anbindungen von *Context Adaptern* und *Application Stubs*, die in einer anderen Programmiersprache geschrieben wurden, als die Komponenten das Frameworks.

4.1.1 Verteilung

Abbildung 4.2 zeigt einige Szenarien, die sich potentiell mit dem vorgestellten Konzept realisieren lassen. Der Fokus liegt dabei auf der flexiblen Verteilung der einzelnen Framework Komponenten auf verschiedene physikalische Geräte. Abgesehen von den vorgestellten Szenarien, sind noch viele weitere Variationen/Mischformen denkbar.

Verteilungs-Szenario 1: Die in der ersten Abbildung gezeigte "Verteilung" stellt die einfachste Variante dar. Hier werden alle Komponenten des Frameworks auf einem mobilen Gerät betrieben. Für die Realisierung eines Szenarios, wie es in Kapitel 1.3 beschrieben wird, wäre diese Variante voraussichtlich hinreichend. Generell kann diese einfache Variante nur eingesetzt werden, wenn alle benötigten Context Quellen direkt auf dem Gerät verfügbar sind. Weiterhin muss die Performance des Gerätes ausreichend sein, um alle Komponenten lokal betreiben zu können. Diese Variante bietet den Vorteil, dass sie einfach zu konfigurieren und robust gegen instabile Netzwerkverbindungen ist.

Verteilungs-Szenario 2: Die zweite Variante stellt eine Erweiterung der ersten dar. Zu den *Context Adaptern* auf dem mobilen Gerät werden zusätzlich *Context Adapter* auf einem entfernten Gerät (in diesem Fall einem Server) genutzt. Im Hinblick auf das in Kapitel 1.3 beschriebene Szenario, könnte eine solche (entfernte) Context Quelle z.B. zur dynamischen Ergänzung von Veranstaltungs-Informationen dienen. Diese Variante bietet sich an, wenn nicht alle benötigten Context Quellen auf dem Gerät vorhanden sind (bzw. nicht dort betrieben werden können) und trotzdem eine einfache und robuste Lösung angestrebt wird.

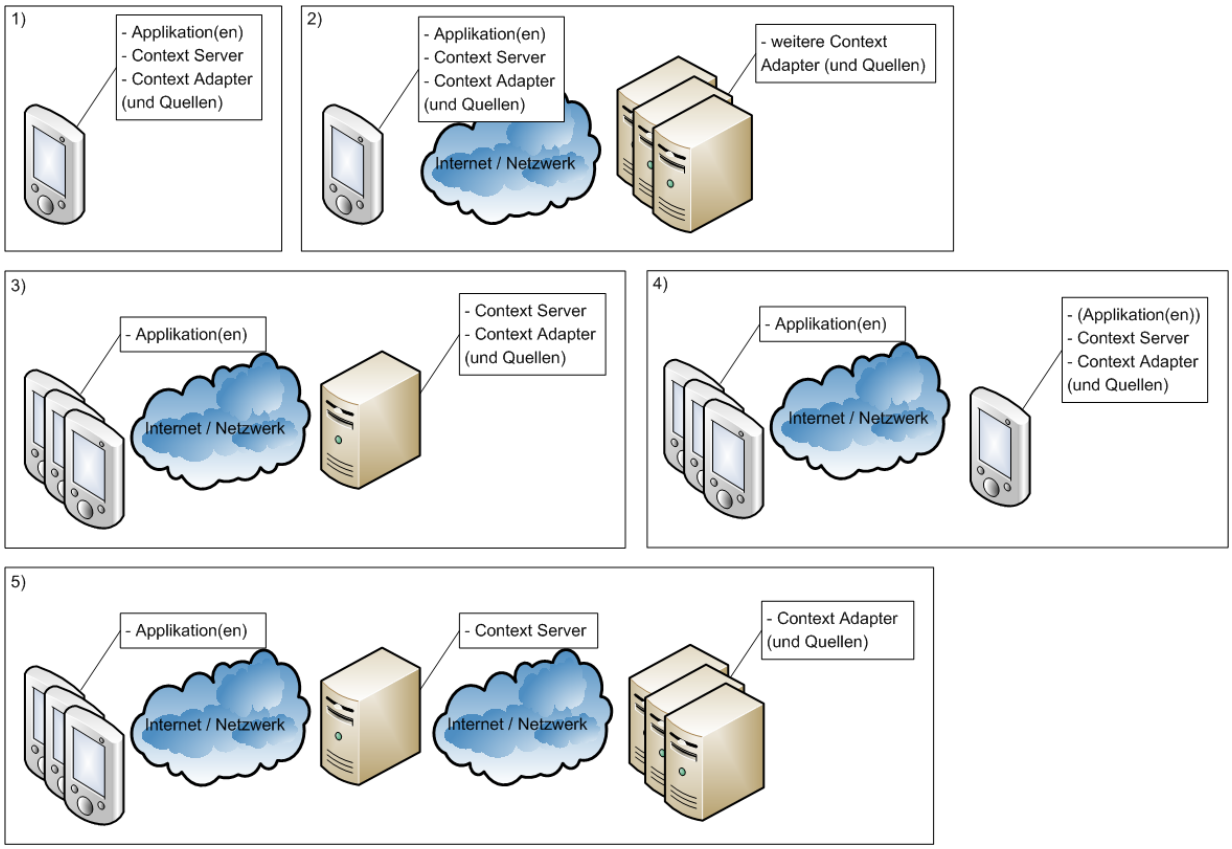

Abbildung 4.2: Szenarien: Komponenten Verteilung

Verteilungs-Szenario 3: Die dritte Variante hingegen zeigt einen grundsätzlich anderen Ansatz. Hier werden lediglich die auf dem Framework basierenden Applikationen auf dem mobilen Gerät betrieben. Der *Context Server*, sowie die Context Adapter befinden sich auf einem entfernten Gerät (Server). Diese Variante ist nur dann praktikabel, wenn entweder nur sehr sporadisch Context Informationen benötigt werden, oder von einer dauerhaften/zuverlässigen Verbindung ausgegangen werden kann. Diese Variante bietet sich vor allem an, wenn mehrere Geräte (bzw. Applikationen) von einem gemeinsamen *Context Server* versorgt werden sollen und keine gerätespezifischer Context (z.B. GPS) benötigt wird.

Verteilungs-Szenario 4: Diese Variante ist eine Abwandlung von Variante 3. Die Server Rolle wird hier von einem mobilen Gerät übernommen, welches als Master für die anderen Geräte fungiert. Dieses Szenario könnte z.B. in einem mobilen Ad-Hoc Netzwerk Anwendung finden und ermöglicht sehr dynamische Szenarien. Es ist dabei zu beachten, dass die

Last auf dem Master sehr groß werden kann. Daher muss sichergestellt werden, dass die Performance des Masters für konkrete Szenarien ausreichend ist.

Verteilungs-Szenario 5: Die fünfte Variante unterscheidet sich von den vorhergehenden insofern, dass der *Context Server* hier unabhängig von Context Quellen und Applikationen betrieben wird. Diese Variante bietet sich vor allem an, wenn viel Last auf dem Server zu erwarten ist und die Context Quellen aus infrastrukturellen- und/oder Performance-Gründen nicht auf dem Server betrieben werden können.

4.2 Komponenten

Im Folgenden werden die in Kapitel 4.1 vorgestellten Komponenten ausführlich beschrieben (Kapitel 4.2.1 - 4.2.3). Neben der Beschreibung der einzelnen Komponenten erfolgt außerdem eine Betrachtung der Interaktion zwischen den Komponenten (Kapitel 4.3.4 u. 4.3.3).

4.2.1 Context Server

Der *Context Server* stellt die zentrale Komponente des Frameworks dar. Seine wichtigsten Sub-Komponenten sind die *Rule Engine* und der *Context Store*. Von diesen Komponenten darf es innerhalb des *Context Servers* stets nur eine Instanz geben (vgl. Singleton Pattern [Gamma et al., 1995]). Auf die Details der beiden Komponenten wird weiter unten gesondert eingegangen.

Abgesehen von den genannten Komponenten, enthält der *Context Server* noch zwei Kommunikationskomponenten, die für die Kommunikation mit den *Context Adaptern*, bzw. *Application Stubs* zuständig sind. Der *Context Server* kapselt die *Rule Engine* und den *Context Store* vollständig. Die Kommunikation mit Komponenten außerhalb des *Context Servers* erfolgt ausschließlich über die Kommunikationskomponenten.

Auf die Interaktion des *Context Servers* mit den *Context Adaptern* und den *Application Stubs* wird in den Kapiteln 4.3.3 und 4.3.4 eingegangen.

Rule Engine

Wie bereits in Kapitel 4.1 erwähnt, dient die *Rule Engine* der Verarbeitung von Context Informationen. Dabei ist zu beachten, dass die *Rule Engine* die Context Informationen nicht selbst verwaltet, sondern bei Bedarf vom *Context Store* erfragt (nie direkt von den Context Adaptern). Die *Rule Engine* verwaltet lediglich die Regeln, die vorgeben, wie bestimmte Context Informationen zu interpretieren, abstrahieren und weiterzuleiten sind. Zu diesem Zweck kann die *Rule Engine* zwei Arten von Regeln verarbeiten (die Syntax und Semantik der Regeln wird in Kapitel 4.3.2 vorgestellt):

1. Framework interne Regeln, die festlegen, wie Context Informationen - der dem Framework bekannten Context Quellen - zu abstrahieren sind (siehe Kapitel 2.3). Diese Regeln müssen beim Start des *Context Servers* bekannt sein und können zur Laufzeit nicht verändert werden. Eine Änderbarkeit/Erweiterbarkeit dieser Regeln wäre zwar potentiell realisierbar, würde die Komplexität der Lösung jedoch signifikant erhöhen.

2. Applikationsspezifische Regeln, die für die jeweilige Applikation festlegen, wie Context Informationen zu abstrahieren sind. Diese Regeln müssen zur Laufzeit erweiterbar sein, da beim Start des *Context Servers* nicht bekannt ist, welche Applikationen sich zur Laufzeit mit dem *Context Server* verbinden werden und welche Context Informationen sie benötigen. Es sind durchaus Lösungen vorstellbar, bei denen vor dem Start des *Context Servers* bekannt ist, welche Applikationen angebunden werden. Dies würde jedoch eine zu große Einschränkung bedeuten.

Die Verarbeitung der Regeln erfolgt generisch. D.h. die *Rule Engine* ist nicht von konkreten Context Informationen abhängig und somit robust gegen Änderungen, bzw. Erweiterungen des Frameworks um neue Context Quellen (siehe Kapitel 4.4).

Da die *Rule Engine* nicht direkt mit Komponenten außerhalb des *Context Servers* kommunizieren kann, muss das Einpflegen neuer Regeln über eine Schnittstelle des *Context Servers* erfolgen. Dieser reicht die Regeln dann an die *Rule Engine* weiter.

Weiterhin muss die *Rule Engine* fehlertolerant in Bezug auf die Regeln sein. D.h. widersprüchlich, unvollständig oder anderweitig fehlerhafte formulierte Regeln dürfen die Funktionalität der *Rule Engine* und somit des *Context Servers* nicht beeinträchtigen. Dies spielt besonders für die applikationsspezifischen Regeln eine Rolle, da diese zu Laufzeit zur *Rule Engine* hinzugefügt werden können. Fehlerhafte Regeln einer einzigen Applikation könnten somit potentiell Einfluss auf alle weiteren angebundenen Applikationen haben. Um zu gewährleisten, dass sich die Regeln verschiedener Applikationen nicht beeinflussen, müssen

sie von der *Rule Engine* unabhängig voneinander verwaltet werden. Es muss stets gewährleistet sein, dass sich Regeln einer spezifischen Applikation zuordnen lassen und diese keine Änderungen am applikationsübergreifenden Context vornehmen. Auf konkrete Bedingungen, die in Bezug auf die Regeln zu prüfen sind, wird in Kapitel 4.3.2 näher eingegangen.

Context Store

Im Gegensatz zur *Rule Engine* dient der *Context Store* hauptsächlich der Speicherung, bzw. Verwaltung von Context Informationen. Eine Verarbeitung der Informationen findet nicht statt. Der wesentliche Vorteil des *Context Store* ist, dass mit seiner Hilfe die Kommunikation mit den Context Adaptern minimiert werden kann (vgl. Chain of Responsibility Pattern [Gamma et al., 1995]). Ohne den *Context Store* müsste jede Anfrage der *Rule Engine* direkt an die Context Adapter weitergeleitet werden. Mit dem *Context Store* können Informationen mehrfach verwendet werden. Dieser Vorteil wird in Szenarien mit mehreren angebundenen Applikationen besonders offensichtlich.

Da der *Context Store* - abgesehen von den reinen Context Informationen der *Context Adapter* - zusätzlich die abstrahierten Context Informationen verwaltet, welche von der *Rule Engine* ermittelt werden, reduziert sich außerdem der Arbeitsaufwand der *Rule Engine*, da die abstrahierten Context Informationen ebenfalls mehrfach verwendet werden können.

Abbildung 4.5 (Kapitel 4.3) zeigt ein vereinfachtes XML Schema für die Context Informationen. Die XML Strukturen in Abbildung 4.8 und 4.9 (Kapitel 4.3) zeigen Beispiele für Context Informationen (Rohdaten).

In Kapitel 4.3 werden die Details des XML Schemas näher betrachtet.

Klassen

Neben den bereits beschriebenen, wichtigsten Klassen (*RuleEngine* und *ContextStore*), verfügt der *Context Server* über einen *AbstractApplicationCommunicationService*, sowie einen *AbstractContextCommunicationService*. Aufgabe dieser beiden Klassen ist die Interaktion mit den *Application Stubs*, bzw. *Context Adaptern* (siehe auch Kapitel 4.3).

Der *AbstractApplicationCommunicationService* nimmt Anfragen von Applikationen entgegen und leitet diese an die *RuleEngine* weiter. Außerdem dient sie der Übertragung von Context Informationen an die Applikationen.

Der *AbstractContextCommunicationService* leitet Anfragen der *RuleEngine* an die konkreten *Context Adapter* weiter und nimmt außerdem Context Informationen, die von den *Context Adaptern* gesendet werden, entgegen.

Da der *Context Server* sowohl mit mehreren *Context Adapter*, als auch mit mehreren *Application Stubs* verbunden sein kann, sind *AbstractApplicationCommunicationService* und *AbstractContextCommunicationService* Multithreading fähig.

4.2.2 Context Adapter

Die *Context Adapter* bestehen aus einem allgemeingültigen Framework Teil, sowie einem Teil, der spezifisch für jede Context Quelle ist (siehe Klassendiagramm: Abbildung 4.3). Sie implementieren das Adapter Pattern [Gamma et al., 1995].

Der Framework Teil der *Context Adapter* ist für die Kommunikation mit dem *Context Server* zuständig. Er bietet Schnittstellen an, über den der *Context Server* Anfragen stellen kann. Außerdem kann er Context Informationen in einem standardisierten Format an den *Context Server* senden. Die Überführung von Context Informationen in ein standardisiertes Format kann nicht vom Framework Teil des *Context Adapters* erfüllt werden. Diese Aufgabe wird daher vom spezifischen Teil des *Context Adapters* übernommen. Da dies individuell für jede Context Quelle gelöst werden muss, kann in der Architekturbeschreibung nicht näher auf diesen Aspekt eingegangen werden. Der generelle Ablauf wird jedoch in den abstrakten *Context Adaptern* gemäß des Factory Method Pattern [Gamma et al., 1995] festgelegt.

Um den Anwendungsentwicklern so viel Arbeit wie möglich abzunehmen, beinhaltet das Framework nicht nur einen Standard *Context Adapter*, sondern diverse *Domain Context Adapter* für typische Klassen von Context Quellen (siehe Klassendiagramm: Abbildung 4.3 und XML Schema: Abbildung 4.5).

Die Details der Interaktion zwischen *Context Adapter* und *Context Server* werden in Kapitel 4.3.4 näher betrachtet.

Klassen

Die Oberklasse für alle *Context Adapter* ist *AbstractContextAdapter*. In dieser Klasse werden grundlegende Funktionalitäten für die Interaktion mit dem *Context Server* bereitgestellt (siehe Kapitel 4.3.4). Von *AbstractContextAdapter* erbt die Klasse *AbstractPassi-*

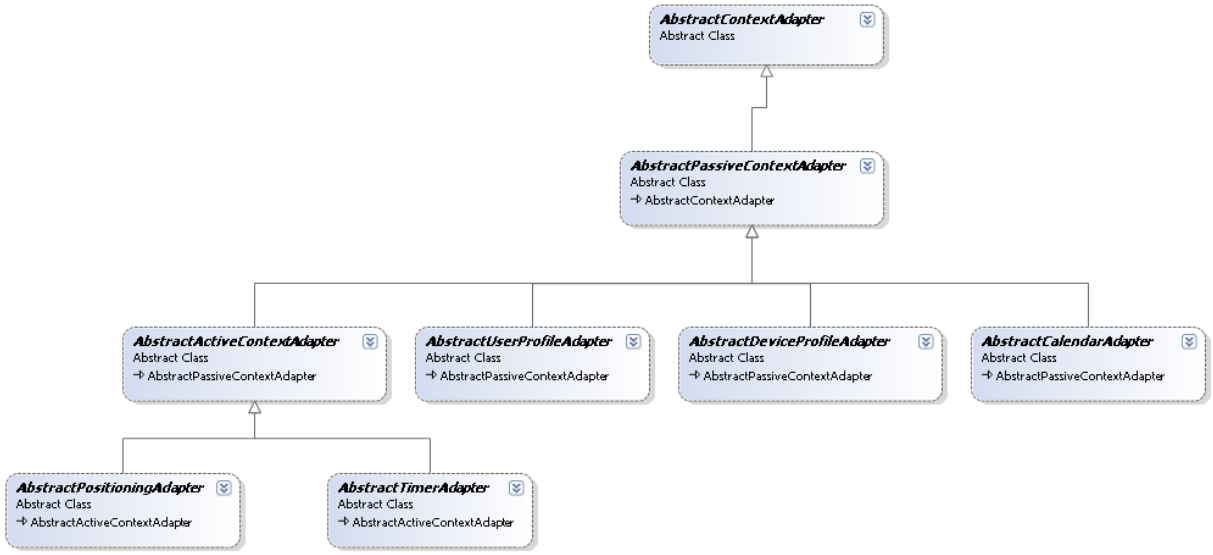

Abbildung 4.3: Context Adapter Klassendiagramm

veContextAdapter. Sie bietet für den *Context Server* die Möglichkeit, bei Bedarf Context Informationen "ad-hoc" zu erfragen. Bei Änderungen des Contexts kann der *Context Server* jedoch nicht aktiv benachrichtigt werden. Sollen Context Informationen zusätzlich aktiv an den *Context Server* gesendet werden, wird die Klasse *AbstractActiveContextAdapter* benötigt. Sie erbt von *AbstractPassiveContextAdapter* und bietet somit auch die passiven Funktionalitäten. Theoretisch sind diese drei Klassen ausreichend, um alle wesentlichen Aufgaben der *Context Adapter* zu erfüllen. Um die Anbindung von Context Quellen jedoch noch weiter zu vereinfachen, gibt es zusätzlich domainspezifische *Context Adapter*. Vorgesehene *Domain Context Adapter* sind: *AbstractPositioningAdapter*, *AbstractTimerAdapter*, *AbstractDeviceProfileAdapter*, *AbstractUserProfileAdapter* und *AbstractCalendarAdapter*. Diese erben entsprechend ihrer vorgesehenen Funktionalität entweder von *AbstractPassiveContextAdapter* oder von *AbstractActiveContextAdapter*.

Wenn eine Context Quelle an das Framework angebunden werden soll, die sich keinem *Domain Context Adapter* zuordnen lässt, sollte das Framework um einen entsprechenden *Domain Context Adapter* erweitert werden (siehe Kapitel 4.4.2).

4.2.3 Application Stub

Das Konzept der *Application Stubs* ist vergleichbar mit dem der *Context Adapter*. Sie bestehen ebenfalls aus einem Framework Teil, der die Kommunikation mit dem *Context Server*

übernimmt und einem applikationsspezifischen Teil (siehe Klassendiagramm: Abbildung 4.4). Sie implementieren ebenfalls das Adapter Pattern [Gamma et al., 1995]. Allerdings gibt es keine domainspezifischen *Application Stubs*.

Die Details der Interaktion zwischen *Application Stubs* und *Context Server* werden in Kapitel 4.3.3 näher betrachtet.

Klassen

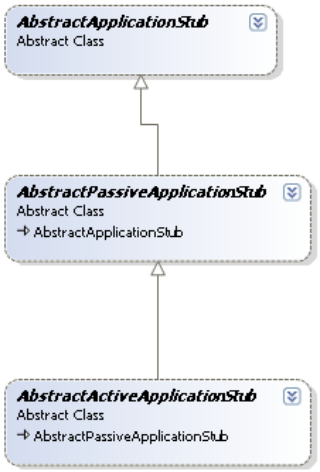

Abbildung 4.4: Application Stub Klassendiagramm

Die Oberklasse für alle *Application Stubs* ist *AbstractApplicationStub*. In dieser Klasse werden grundlegende Funktionalitäten für die Interaktion mit dem *Context Server* bereitgestellt (siehe Kapitel 4.3.3). Von *AbstractApplicationStub* erbt die Klasse *AbstractPassiveApplicationStub*. Applikationen, die nur passiv mit dem *Context Server* kommunizieren (also keine Context Informationen aktiv anfragen, sondern nur entgegennehmen), können von dieser Klasse erben. Sie bietet die Möglichkeit, empfangene Context Informationen in ein Context Objekt zu überführen (siehe Kapitel 4.2.4). Applikationen müssen sich daher nicht mit den Details des Nachrichtenformats zwischen *Application Stub* und *Context Server* auseinander setzen. Von *AbstractPassiveApplicationStub* erbt die Klasse *AbstractActiveApplicationStub*. Soll eine Applikation auch aktiv mit dem *Context Server* kommunizieren, kann sie von dieser Klasse erben. Sie stellt die Funktionalität zur Verfügung, um konkrete Context Informationen vom *Context Server* "ad-hoc" zu erfragen. Da diese Klasse nicht direkt von *AbstractApplicationStub* erbt, kann außerdem die volle Funktionalität aus

AbstractPassiveApplicationStub genutzt werden. Wie bei den *Context Adaptern* wird auch hier der generelle Ablauf in den abstrakten *Context Adaptern* gemäß des Factory Method Pattern [Gamma et al., 1995] festgelegt.

4.2.4 Context Objekte

Innerhalb des Frameworks werden alle Context Informationen in vereinheitlichten Context Objekten verwaltet. Für die Kommunikation/Interaktion zwischen *Context Adapter* und *Context Server*, bzw. *Context Server* und *Application Stub*, werden diese Context Objekte in ein ebenfalls vereinheitlichtes XML Format überführt (siehe Kapitel 4.3).

Die beiden wichtigsten Context Objekte, bzw. Klassen sind *SimpleContext* und *ComplexContext*. Wobei *ComplexContext* von *SimpleContext* erbt. *SimpleContext* beinhaltet domainspezifische Context Objekte, welche die Context Informationen der *Domain Context Adapter* abbilden. *ComplexContext* beinhaltet darüber hinaus abstrahierte Context Informationen, die anhand von Regeln der *Rule Engine* ermittelt werden.

Aufgrund dieser Struktur, wird der *SimpleContext* von *Context Adaptern* und dem *Context Server* genutzt, wohingegen der *ComplexContext* von Applikationen und dem *Context Server* genutzt wird.

Diese Objekte, bzw. Klassen dienen jedoch nicht nur als Behälter für Context Informationen, sondern können auch als Nachrichten verwendet werden. Auf diesen Aspekt wird im Kapitel 4.3 näher eingegangen.

4.3 Interaktion

Im Folgenden wird das Nachrichten- und Rule Format für die Interaktion zwischen *Context Adaptern*, *Context Server* und *Application Stubs* betrachtet. Außerdem wird anhand von Sequenzdiagrammen der Ablauf der Interaktion zwischen diesen Framework Komponenten gezeigt.

Für die Interaktion könnten theoretisch Web-Services in Verbindung mit SOAP Nachrichten verwendet werden. Da die Framework Komponenten hauptsächlich auf mobilen Geräten eingesetzt werden sollen, gilt es jedoch, unnötigen Overhead zu vermeiden. Für die Interaktion wird daher im Folgenden ein möglichst schlankes Protokoll vorgestellt, welches jedoch

an die Struktur von SOAP Nachrichten angelehnt ist.

In Kapitel 2.4 wurden verschiedene Alternativen für die Repräsentation von Context Informationen vorgestellt. Insbesondere RDF/XML ist im Rahmen dieser Arbeit interessant. Für die Erfüllung der in Kapitel 3 definierten Anforderungen, ist jedoch ein einfaches XML Schema, bzw. Protokoll ausreichend. Um die Komplexität des Entwurfs nicht unnötig zu erhöhen, wird daher auf den Einsatz von RDF/XML verzichtet.

4.3.1 Nachrichtenformat

Das Nachrichtenformat für die Interaktion der verschiedenen Framework Komponenten, wird durch das in Abbildung 4.5 dargestellte XML Schema beschrieben und entspricht den Context Objekten, die in Kapitel 4.2.4 vorgestellt wurden. Ein vollständiges Listing des Schemas im XML Format findet sich in Anhang B. Obwohl es gewisse Überschneidungen gibt, darf dieses Nachrichtenformat nicht mit dem Schema für die Definition von Regeln verwechselt werden. Auf dieses wird in Kapitel 4.3.2 gesondert eingegangen.

Wie dem XML Schema zu entnehmen ist, wird zwischen *simpleContext* und *complexContext* unterschieden (entsprechend Kapitel 4.2.4). Neben den beiden Context Arten, sieht das Schema einen *senderIdentifier* und einen *messageType* vor. Der *senderIdentifier* gibt an, von wem die Nachricht gesendet wurde (also vom *Context Server*, einem *Application Stub* oder einem *Context Adapter*). Dabei muss beachtet werden, dass eindeutige Identifier gewählt werden. Der *messageType* hat die wichtige Aufgabe, festzulegen, um welche Art von Nachricht es sich handelt. Dies ist essentiell, um entscheiden zu können, wie der Inhalt der Nachricht zu interpretieren ist. Es gibt 4 Arten von Nachrichten: *subscribe*, *unsubscribe*, *get*, und *put*. Im Folgenden werden diese Nachrichten-Arten näher betrachtet. Im Anschluss daran erfolgt zusätzlich eine detailliertere Betrachtung der Unterelemente von *simpleContext* und *complexContext*.

subscribe Die *subscribe* Nachricht kann von Applikationen genutzt werden, um Context Informationen vom *Context Server* zu "abonnieren". Dafür muss die Nachricht diejenigen XML Elemente und Attribute enthalten, an denen die Applikation interessiert ist. Dabei kann sehr feingranular bestimmt werden, welche Context Informationen "abonniert" werden sollen. Die Semantik sieht dabei vor, dass Elemente, die leer angegeben werden, vollständig "abonniert" werden. Wenn also z.B. nur *simpleContext*, ohne weitere Unterelemente angegeben wird, so entspricht das dem vollständigen *simpleContext*. In Abbildung

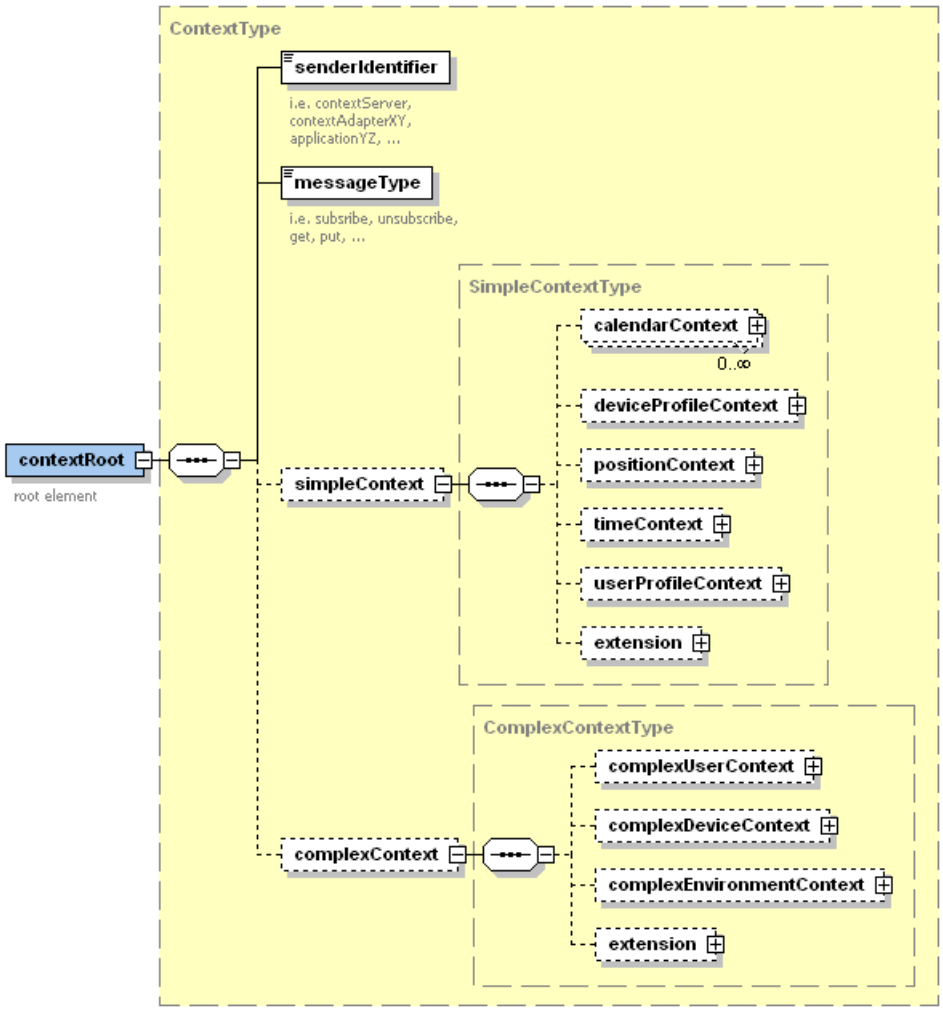

Abbildung 4.5: Context Type

4.6 wird hingegen gezeigt, wie eine Applikation lediglich *positionDeviceType*, *precision* und *currentPosition* von *positionContext* innerhalb des *simpleContext* "abonniert". Dieser Mechanismus kann auch genutzt werden, um *complexContext* zu "abonnieren".

Da die Überführung von Context Objekten in das XML Nachrichtenformat von Komponenten des Frameworks übernommen werden, entsteht kein zusätzlicher Implementierungsaufwand für die Anwendungsentwickler.

```
<?xml version="1.0" encoding="UTF-8"?>
<contextRoot xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="..\ContextSchema.xsd">
   <senderIdentifier>Application0815</senderIdentifier>
   <messageType>subscribe</messageType>
   <simpleContext>
      <positionContext>
         <positionDeviceType/>
         <precision/>
         <currentPosition/>
      </positionContext>
   </simpleContext>
   <complexContext/>
</contextRoot>
```

Abbildung 4.6: Simple Context XML - subscribe Example

unsubscribe Die Nachrichten für *subscribe* und *unsubscribe* unterscheiden sich kaum. Lediglich ihre Wirkung ist genau entgegengesetzt. D.h. die Angabe von XML Elementen bewirkt hier, dass sie und alle ihre Unterelemente "abbestellt" werden.

Theoretisch könnte man auch auf *unsubscribe* verzichten, und stattdessen einfach eine neue *subscribe* Nachricht senden, welche die vorherigen Abonnements überschreibt. Dieser Ansatz wäre jedoch weniger elegant.

get Die *get* Nachrichten haben keinen Einfluss auf *subscribe* oder *unsubscribe*. Sie dienen der "ad-hoc" Beschaffung von Context Informationen. Welche Context Informationen bezogen werden sollen, wird in gleicher Weise festgelegt, wie in den *subscribe* und *unsubscribe* Nachrichten. Die durch *get* angeforderten Context Informationen werden dann nur einmalig übertragen.

Im Gegensatz zu *subscribe* und *unsubscribe* Nachrichten, werden *get* Nachrichten sowohl zwischen *Context Adapter* (Empfänger) und *Context Server* (Sender), als auch zwischen *Context Server* (Empfänger) und *Application Stub* (Sender) verwendet. Die *complexContext* Elemente können allerdings nicht zwischen *Context Adapter* und *Context Server* verwendet werden.

Abbildung 4.7 zeigt ein Beispiel für eine *get* Nachricht. In diesem Beispiel wird der vollständige *deviceProfileContext* des *simpleContext* angefordert.

```
<?xml version="1.0" encoding="UTF-8"?>
<contextRoot xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="..\ContextSchema.xsd">
   <senderIdentifier>Application0815</senderIdentifier>
   <messageType>get</messageType>
   <simpleContext>
      <deviceProfileContext/>
   </simpleContext>
</contextRoot>
```

Abbildung 4.7: Simple Context XML - get Example

put Im Gegensatz zu den anderen Nachrichten, werden innerhalb einer *put* Nachricht Nutzdaten transportiert. D.h. die Attribute der einzelnen XML Elemente können Werte enthalten. Die *put* Nachrichten, werden sowohl für die Übertragung "abonnierter" Context Informationen genutzt, als auch als "Antwort" auf *get* Nachrichten. Eine Unterscheidung macht hier wenig Sinn.

Die Abbildungen 4.8 und 4.9 zeigen zwei Beispiele für *put* Nachrichten. Es werden *positionContext* und *userProfileContext* Informationen übertragen.

Simple Context

Die Grobe Struktur des *simpleContext* wurde bereits vorgestellt. Im Folgenden werden die einzelnen Unterelemente, bzw. Domain Context Elemente betrachtet, um ein besseres Verständnis für das vorgestellte Konzept zu erzielen. Selbstverständlich stellt dies nur eine kleine Auswahl aller denkbaren Domain Context Elemente dar und dient im Rahmen dieser Arbeit als Standard Set für den Domain Context (bzw. *simpleContext*). Erweiterungen dieses Standard Sets sind nicht nur möglich, sondern wahrscheinlich und gewollt. Daher

```
<?xml version="1.0" encoding="UTF-8"?>
<contextRoot xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="..\ContextSchema.xsd">
   <senderIdentifier>PositionContextAdapter4711</senderIdentifier>
   <messageType>put</messageType>
   <simpleContext>
      <positionContext>
         <positionDeviceType>GPS</positionDeviceType>
         <precision>5</precision>
         <currentPosition>
            <xPosition>53.5607046750403</xPosition>
            <yPosition>10.018501281738281</yPosition>
         </currentPosition>
         <currentSpeed>1.3</currentSpeed>
         <currentDirection>NW</currentDirection>
      </positionContext>
   </simpleContext>
</contextRoot>
```

Abbildung 4.8: Position Context XML - put Example

wird im XML Schema eine Extension für zusätzlichen Domain Context bereit gestellt (siehe Abbildung 4.15). Domain Context, der häufig benötigt wird, kann bei Bedarf auch durch eine Erweiterung des XML Schemas in das Standard Set aufgenommen werden.

Abbildung 4.10 zeigt die Details des *calendarContext*. Anders als bei den folgenden Context Arten, kann es mehr als einen *calendarContext* gleichzeitig geben. Somit lassen sich für einen Zeitraum verschiedene Typen von *calendarContext* beschreiben (z.B. Geburtstage, Meetings, Reminder). Zusätzlich kann ein Ort für die für jeden Eintrag angegeben werden, wodurch sich in Kombination mit anderen Context Informationen ortsabhängiges Verhalten realisieren lässt (vgl. Szenario: Kapitel 1.3).

Abbildung 4.11 zeigt die Details des *deviceProfileContext*. Unter Verwendung dieses Domain Contextes, lassen sich sowohl statische, als auch dynamische Informationen über ein Gerät darstellen. Statische Informationen sind dabei z.B. die Displaygröße und die Art des Gerätes. Eine dynamische Informationen hingegen ist u.a. die aktuelle Geschwindigkeit der genutzten Datenverbindung. All diese Informationen können beispielsweise genutzt werden, um Multimedia Inhalte individuell für ein Gerät anzupassen (Auflösung und Qualität von Bildern, etc.).

Abbildung 4.12 zeigt die Details des *positionContext*. Über den *positionContext* lässt sich

```
<?xml version="1.0" encoding="UTF-8"?>
<contextRoot xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="..\ContextSchema.xsd">
   <senderIdentifier>UserProfileContextAdapter4712</senderIdentifier>
   <messageType>put</messageType>
   <simpleContext>
      <userProfileContext>
         <userName>Maik Weindorf</userName>
         <userMobilityPreset>walk</userMobilityPreset>
         <userLocation>
            <locationIdentifier>home</locationIdentifier>
            <location>
               <radius>20</radius>
               <xPosition>53.5607046750403</xPosition>
               <yPosition>10.018501281738281</yPosition>
            </location>
         </userLocation>
         <userLocation>
            <locationIdentifier>work</locationIdentifier>
            <location>
               <radius>50</radius>
               <xPosition>53.55661325836385</xPosition>
               <yPosition>10.023415088653564</yPosition>
            </location>
         </userLocation>
         <userInterest>
            <type>entertainment</type>
            <description>Cars</description>
            <intensity>8</intensity>
         </userInterest>
         <userInterest>
            <type>entertainment</type>
            <description>Art</description>
            <intensity>4</intensity>
         </userInterest>
      </userProfileContext>
   </simpleContext>
</contextRoot>
```

Abbildung 4.9: User Profile Context XML - put Example

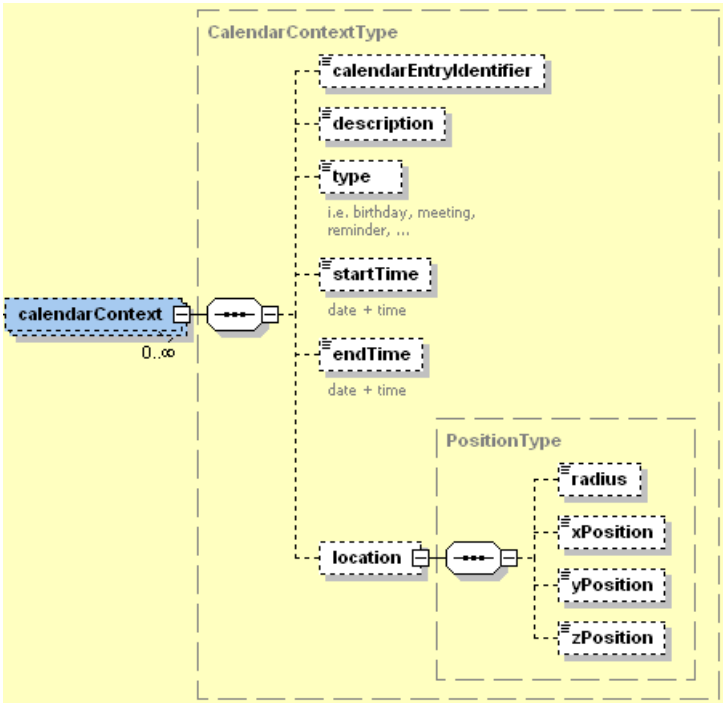

Abbildung 4.10: Calendar Context Type

vor allem die aktuelle Position eines Gerätes beschreiben. Zusätzlich können wichtige Zusatzinformationen mitgeteilt werden, wie z.B. die aktuelle Geschwindigkeit, Richtung, sowie die Genauigkeit der Positionsinformation.

Abbildung 4.13 zeigt die Details des *timeContext*. Dieser beschränkt sich momentan auf die aktuelle Zeit. Erweiterungen sind natürlich denkbar.

Abbildung 4.14 zeigt die Details des *userProfileContext*. Durch diesen Context lassen sich Informationen über einen Benutzer darstellen. Im Hinblick auf das in Kapitel 1.3 beschriebene Szenario umfasst dieser Context den aktuellen Mobilitätsgrad des Benutzers (”walk”, ”car”, etc.). Zusätzlich lassen sich mehrere Orte mit bestimmter Bedeutung für den Benutzer beschreiben (”office”, ”home”, etc.). Ebenfalls können mehrere Interessen des Benutzers beschrieben werden.

Abbildung 4.15 zeigt die *extension* für den *simpleContext*. Wie dem vollständigen Listing des XML Schemas in Anhang B zu entnehmen ist, wird der *simpleContextExtensionType* außerhalb des Hauptschemas in einer importierten Datei beschrieben. Erweiterungen lassen sich somit einbinden, ohne das Hauptschema verändern zu müssen. Dies gilt so auch für alle anderen *extension* Typen der in dieser Arbeit vorgestellten XML Schemata.

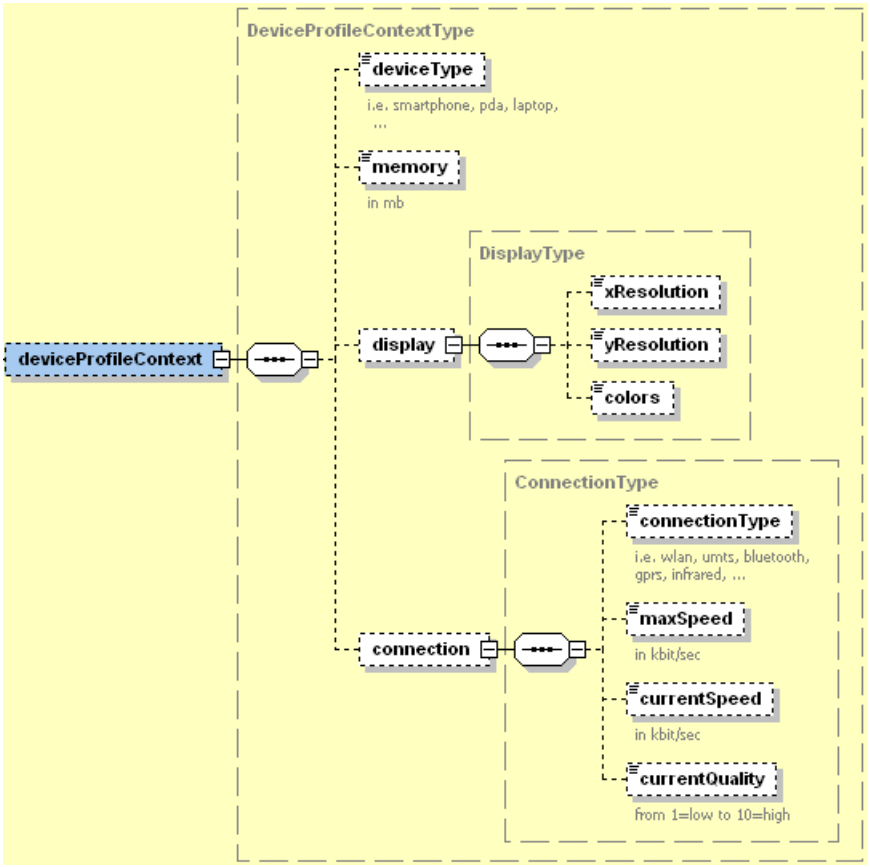

Abbildung 4.11: Device Profile Context Type

Complex Context

Der *complexContext* gliedert sich in *complexUserContext*, *complexDeviceContext* und *complexEnvironmentContext*. Weitere Unterelemente sind selbstverständlich denkbar. Wie beim *simpleContext* können diese zusätzlichen Elemente entweder durch die angebotene Extension, oder durch eine Erweiterung des XML Schemas hinzugefügt werden.

Abbildung 4.16 zeigt die Details der *complexContext* Unterelemente. Im Gegensatz zum *simpleContext* fällt auf, das der *complexContext* keine direkt messbaren Werte, sondern abstrahierte Context Informationen beschreibt. Diese Informationen lassen sich nur durch Kombination verschiedener Werte des *simpleContext* ermitteln. Die dazu nötige Überführung wird mit Hilfe von Regeln beschrieben. Auf dieses Thema wird im folgenden Kapitel 4.3.2 näher eingegangen.

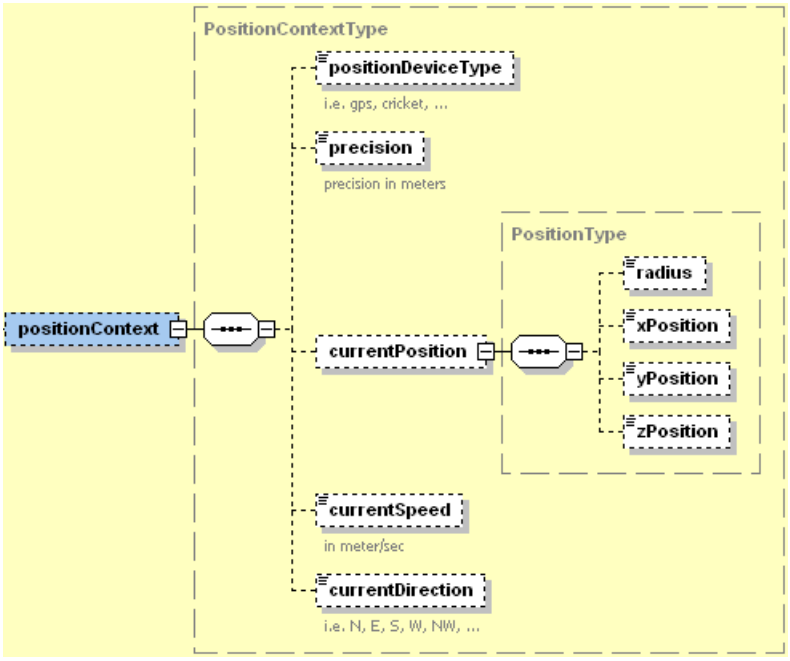

Abbildung 4.12: Position Context Type

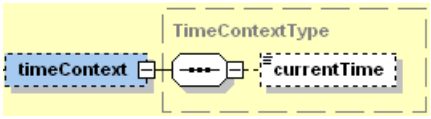

Abbildung 4.13: Time Context Type

4.3.2 Rule Format

Innerhalb des Frameworks wird domainspezifischer Context verarbeitet, daher bietet sich für die Definition der Regeln eine spezialisierte Regelsprache an. Eine allgemeingültige, (Turing-)vollständige Sprache ist zur Definition der benötigten Regeln nicht notwendig und würde die Komplexität der Lösung unnötig erhöhen. Da bereits für das Nachrichtenformat (Kapitel 4.3.1) ein XML Schema verwendet wurde, bietet sich für die Definition der Regelsprache ebenfalls ein XML Schema an.

Das XML Schema für die Definition von Regeln, ähnelt auf den ersten Blick dem vorgestellten Schema für das Nachrichtenformat. Jedoch gibt es entscheidende Unterschiede in der Syntax und vor allem in der Semantik der beiden Schemata. Abbildung 4.17 zeigt das XML Schema für die Definition von Regeln. Ein vollständiges Listing des Schemas im XML Format findet sich in Anhang C.

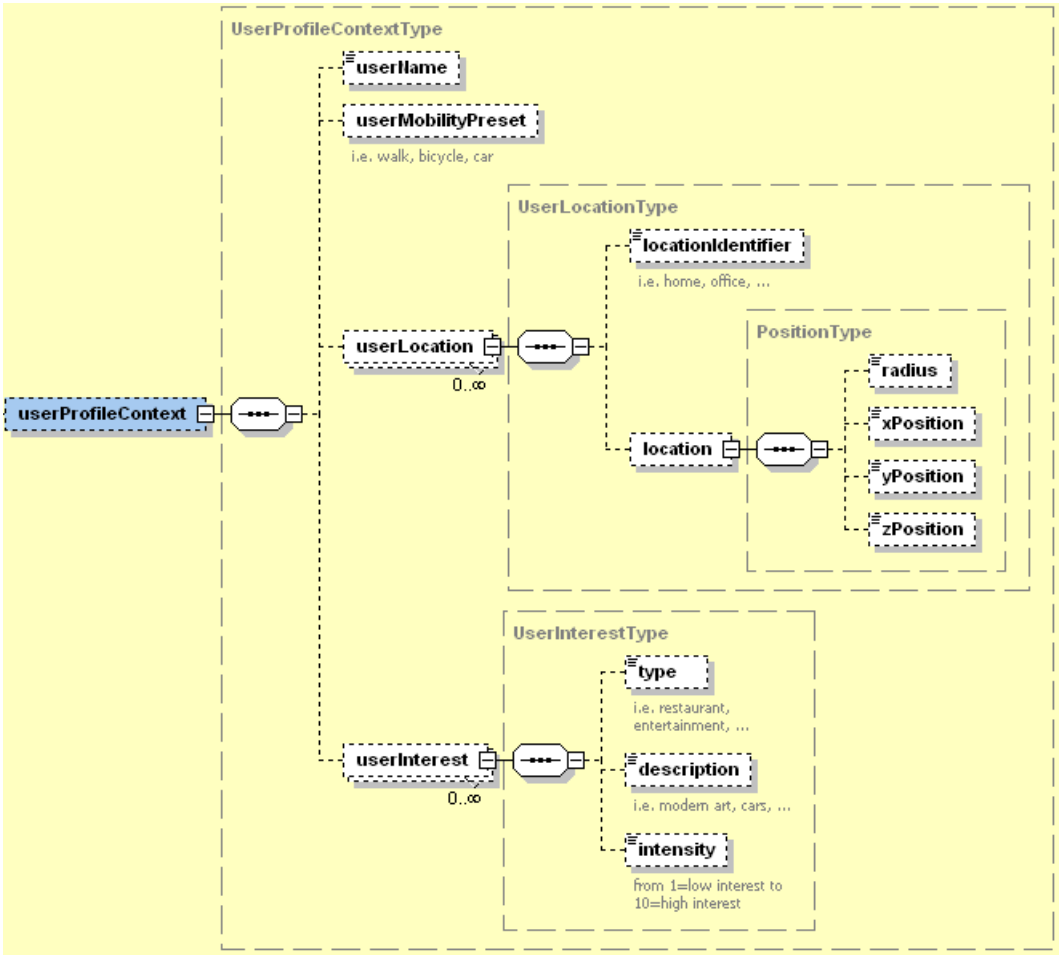

Abbildung 4.14: User Profile Context Type

Gegenüber einer (Turing-)vollständigen Sprache, können basierend auf diesem Schema ausschließlich Regeln endlicher Größe und Komplexität erzeugt werden. Dies liegt vor allem daran, dass innerhalb der Regeln keine Schleifen möglich sind und lediglich fest vorgegebene Attribute mit Werten belegt werden können.

Das XML Schema setzt sich aus einem *senderIdentifier*, einem *messageType* und beliebig vielen *contextRule* Elementen zusammen. Der *senderIdentifier* gibt an, von wem die Nachricht, bzw. die Regeln stammen. Server interne Regeln haben als *senderIdentifier* "server", unterscheiden sich aber strukturell nicht von anderen Regeln. Der *senderIdentifier* ist wichtig, um die Regeln einer konkreten Applikation zuordnen zu können. Der *messageType* beschreibt, um welche Art von Nachricht es sich handelt. Als *messageType* sind "set", "add", "alter", "delete" vorgesehen. Weitere Typen sind denkbar. Die Semantik der verschiedenen Typen ist:

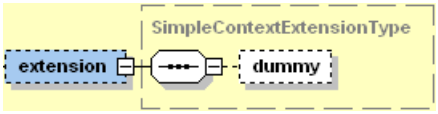

Abbildung 4.15: Simple Context Extension Type

- **set:** Legt einen neuen Satz von Regeln für eine Applikation an, bzw. überschreibt alle bestehenden Regeln einer Applikation. Dies gilt analog für die Server Regeln.
- **add:** Fügt Regeln zu einem bestehenden Satz von Regeln hinzu, ohne die bestehenden Regeln zu beeinflussen.
- **alter:** Ändert bestehende Regeln. Die zu ändernden Regeln werden anhand des eindeutigen *ruleIdentifier* bestimmt.
- **delete:** Löscht bestehende Regeln. Die zu löschenden Regeln werden anhand des eindeutigen *ruleIdentifier* bestimmt.

Die *contextRule* Elemente gliedern sich in *ruleIdentifier*, *contextFact* und *implication*. Der *ruleIdentifier* dient der eindeutigen Identifizierung einer Regel und muss innerhalb der Regeln einer Applikation, bzw. des Servers eindeutig sein. Auf die Details von *contextFact* und *implication* wird weiter unten eingegangen.

Abbildung 4.18 zeigt ein einfaches Beispiel, wie sich mit Hilfe des vorgestellten XML Schemas Regeln für den aktuellen Bewegungsstatus (*currentUserMovementStatus*) eines Benutzers beschreiben lassen. Es handelt sich dabei um zwei Regeln. Die erste Regel legt fest, dass bis zu einer Geschwindigkeit von 1,9 m/s und der Vorbedingung, dass der Benutzer zu Fuß geht, der *currentUserMovementStatus* auf "walking" gesetzt wird. Die zweite Regel legt unter der gleichen Vorbedingung und für Geschwindigkeiten über 1,9 m/s fest, dass der *currentUserMovementStatus* auf "running" gesetzt wird.

Context Fact

In *contextFact* sind alle Domain Context Elemente enthalten, die auch im *simpleContext* beschrieben werden (siehe Kapitel 4.3.1). Dies gilt auch für die *extension*. Jedoch unterscheiden sich die Elemente insofern, dass überall, wo im *simpleContext* konkrete (Zahlen)Werte vorgesehen sind, in *contextFact* dafür Wertebereiche genutzt werden. Eine Ausnahme stellen Positionsinformationen dar, da diese durch Angabe eines *radius* wie ein Wertebereich betrachtet werden können. Identifier und textuelle Beschreibungen hingegen werden als

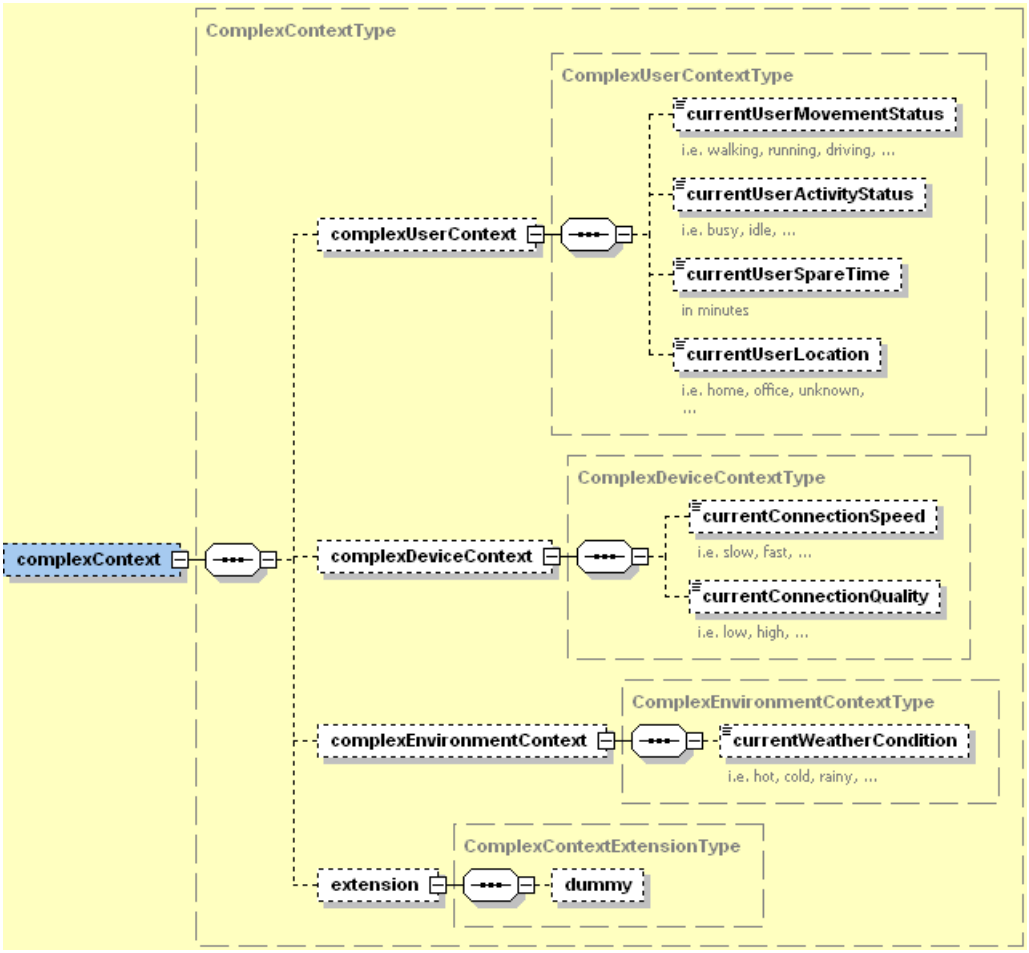

Abbildung 4.16: Complex Context Type

absolute Werte behandelt. Abbildung 4.19 zeigt beispielhaft die Struktur des *DeviceProfileFactType*. Durch die Angabe von *min...* und *max...* Werten, können die Wertebereiche beschrieben werden. Siehe dazu auch das Beispiel in Abbildung 4.18.

Implication

Die Elemente von *implication*, unterscheiden sich syntaktisch kaum von denen des *complexContext* aus Kapitel 4.3.1. Es ist jedoch zu beachten, dass alle XML *sequence* Elemente durch XML *choice* ersetzt wurden. Somit kann sichergestellt werden, dass jeweils nur eine Folgerung pro Regel gezogen wird. Die Struktur von *implication* geht aus Abbildung 4.20 hervor.

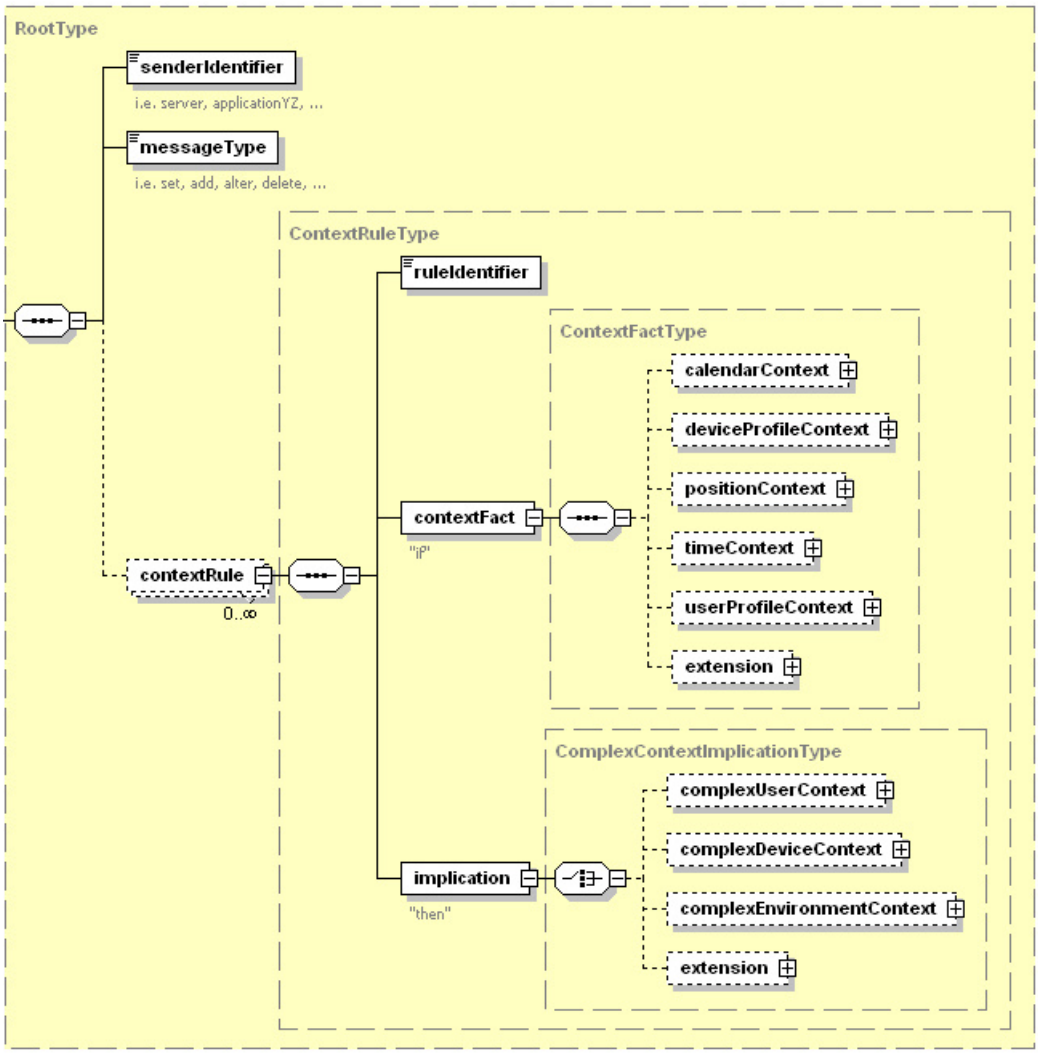

Abbildung 4.17: Rule Schema

Randbedingungen

Zu prüfende Randbedingungen in Bezug auf die Regeln sind:

- Es dürfen nie mehrere Regeln mit dem gleichen *ruleIdentifier* für eine Applikation bestehen.
- Der Versuch, eine nicht existierende Regel zu ändern, ist ein Fehler.
- Der Versuch, eine nicht existierende Regel zu löschen, ist **kein** Fehler und wird ignoriert.

```
<?xml version="1.0" encoding="UTF-8"?>
<root xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="..\RuleSchema.xsd">
   <senderIdentifier>Server</senderIdentifier>
   <messageType>set</messageType>
   <contextRule>
      <ruleIdentifier>movementRule1</ruleIdentifier>
      <contextFact>
         <positionContext>
            <minCurrentSpeed>0</minCurrentSpeed>
            <maxCurrentSpeed>1.9</maxCurrentSpeed>
         </positionContext>
         <userProfileContext>
            <userMobilityPreset>walk</userMobilityPreset>
         </userProfileContext>
      </contextFact>
      <implication>
         <complexUserContext>
            <currentUserMovementStatus>walking</currentUserMovementStatus>
         </complexUserContext>
      </implication>
   </contextRule>

   <contextRule>
      <ruleIdentifier>movementRule2</ruleIdentifier>
      <contextFact>
         <positionContext>
            <minCurrentSpeed>1.9</minCurrentSpeed>
         </positionContext>
         <userProfileContext>
            <userMobilityPreset>walk</userMobilityPreset>
         </userProfileContext>
      </contextFact>
      <implication>
         <complexUserContext>
            <currentUserMovementStatus>running</currentUserMovementStatus>
         </complexUserContext>
      </implication>
   </contextRule>
</root>
```

Abbildung 4.18: Rule XML - Simple Example

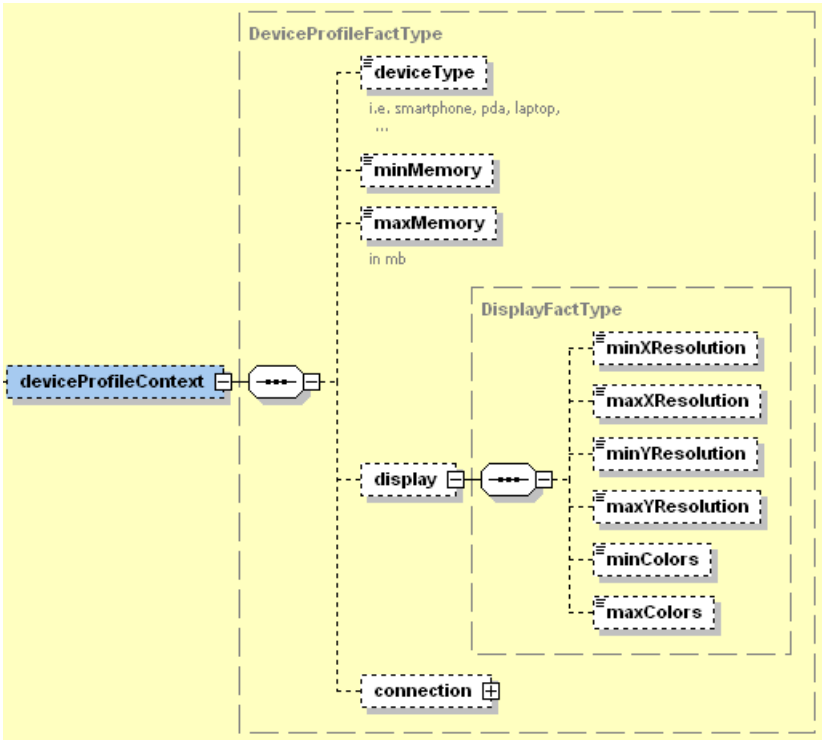

Abbildung 4.19: Rule Fact: Device Profile

- Regeln dürfen nicht im Widerspruch zueinander stehen.
- Die Bedingungen einer Regel (*contextFact*) dürfen nur durch $\wedge$ und nicht durch $\vee$ verkettet sein. Außerdem darf nur eine einzige Folgerung (*implication*) pro Regel gezogen werden (beides wird durch das XML Schema festgelegt). Nach [Beierle et al., 2000] erleichtern diese Einschränkungen eine algorithmische Abarbeitung der Regeln. Dies kann zwar Redundanzen notwendig machen, reduziert aber die Komplexität der Lösung.

Die ersten drei Randbedingungen lassen sich mit wenig Implementierungsaufwand in der *Rule Engine* realisieren. Wie beschrieben, wird die letzte Randbedingung bereits durch das XML Schema erfüllt. Aus den Randbedingungen folgt außerdem, dass ein Feedback (mindestens) im Fehlerfall an den Sender der Regeln übertragen werden muss. Etwas mehr Aufwand erfordert die Überprüfung von Regeln auf Widerspruchsfreiheit. Zu diesem Zweck muss die *Rule Engine* die folgenden konkreten Prüfungen implementieren, die beim Einlesen einer Regel entscheiden, ob die neue Regel zu den Regeln einer konkreten Applikation oder

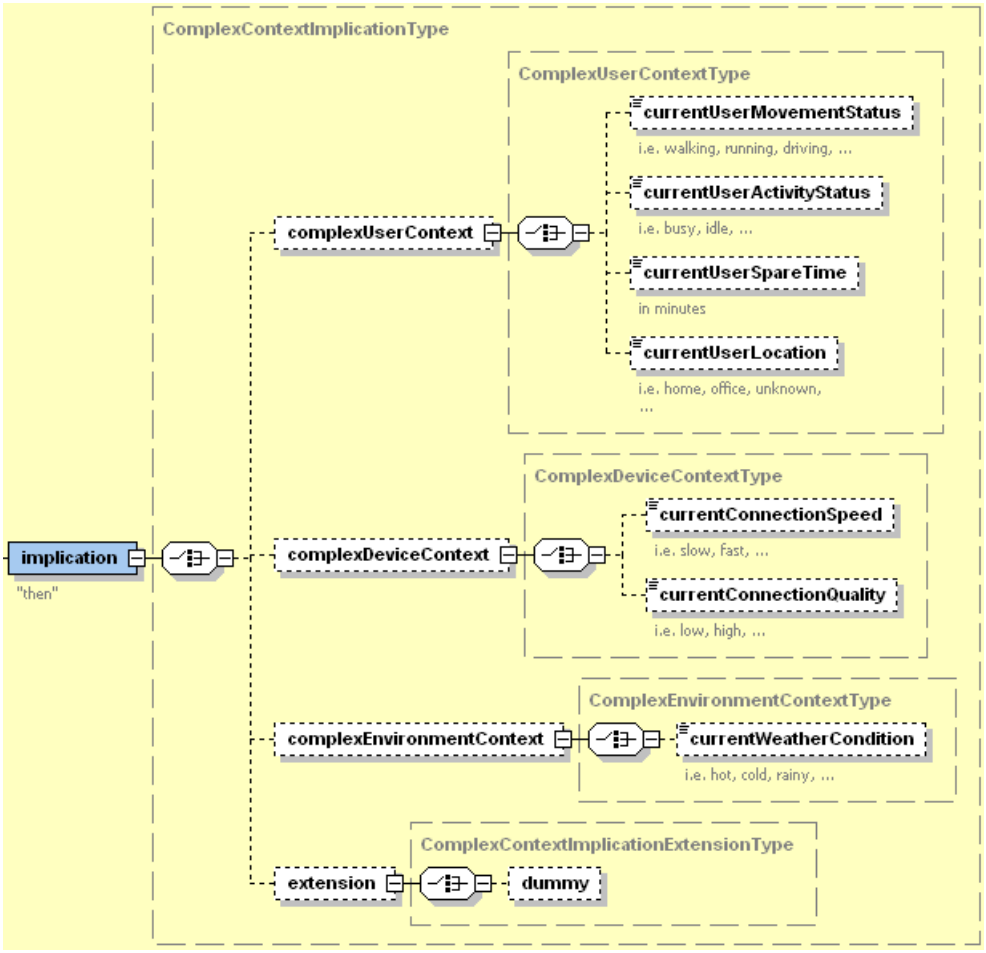

Abbildung 4.20: Rule Implication

des *Context Servers* hinzugefügt werden darf[1]:

- Wenn die Bedingung der Regel die gleichen Attribute wie eine bestehende Regel hat
  - und deren Werte übereinstimmen, darf
    - eine Folgerung mit gleichem Attribut und Wert gezogen werden. (Duplikat)
    - keine Folgerung mit gleichem Attribut und unterschiedlichem Wert gezogen werden. (Widerspruch)
    - eine Folgerung mit unterschiedlichem Attribut gezogen werden. (neues Wissen)

[1]Regeln müssen nur innerhalb der Regeln einer Applikation oder des *Context Servers* widerspruchsfrei sein.

  - und deren Werte nicht übereinstimmen, darf
    - keine Folgerung mit gleichem Attribut und Wert gezogen werden. (Widerspruch)
    - eine Folgerung mit gleichem Attribut und unterschiedlichem Wert gezogen werden. (neues Wissen)
    - eine Folgerung mit unterschiedlichem Attribut gezogen werden. (neues Wissen)

- Wenn die Bedingung der Regel andere Attribute als die bestehenden Regeln hat, darf sie in jedem Fall hinzugefügt werden.
- Wenn die Bedingung der Regel teilweise die gleichen Attribute wie eine bestehende Regel hat (Überschneidung)
  - und deren Werte übereinstimmen, darf
    - eine Folgerung mit gleichem Attribut und Wert gezogen werden.
    - evtl. eine Folgerung mit gleichem Attribut und unterschiedlichem Wert gezogen werden. Dies ist beim Einlesen nicht entscheidbar und muss zur Laufzeit bewertet werden. (potentiell Widerspruch oder neues Wissen)
    - eine Folgerung mit unterschiedlichem Attribut gezogen werden. (neues Wissen)
  - und deren Werte nicht übereinstimmen, darf
    - evtl. eine Folgerung mit gleichem Attribut und Wert gezogen werden. Dies ist beim Einlesen nicht entscheidbar und muss zur Laufzeit bewertet werden. (potentiell Widerspruch oder neues Wissen)
    - eine Folgerung mit gleichem Attribut und unterschiedlichem Wert gezogen werden. (neues Wissen)
    - eine Folgerung mit unterschiedlichem Attribut gezogen werden. (neues Wissen)

Bis auf zwei Sonderfälle können also alle Prüfungen bereits beim Einlesen der Regeln erfolgen. Für diese Sonderfälle muss eine Regelung gefunden werden, die festlegt, welche Regel im Zweifelsfall den Vorrang erhält. Folgende Regelung bietet sich an:

- Für den Fall, dass die Attribute einer Bedingung eine Untermenge der Attribute einer anderen Bedingung sind, erhält die Regel, welche mehr Attribute hat, den Vorrang.

  Beispiel: Angenommen, eine Regel hat 2 Attribute in ihrer Bedingung und eine andere Regel hat 3 (davon 2 gemeinsam mit der ersten Regel). Wenn lediglich die beiden gemeinsamen Attribute erfüllt sind, tritt die erste Regel in Kraft (da die zweite Regel nicht erfüllt ist). Sind allerdings alle 3 Attribute der zweiten Regel erfüllt, erhält sie Vorrang vor der ersten Regel.

- Für den Fall, dass die Attribute einer Bedingung eine Schnittmenge mit den Attributen einer anderen Bedingung bilden, ist nicht entscheidbar, welche Regel den Vorrang erhalten soll. Zum Problem wird dies allerdings nur, wenn alle Attribute der verschiedenen Bedingungen (bzw. Regeln) gleichzeitig erfüllt sind. Da in jedem Fall eine Entscheidung getroffen werden muss, bietet sich hier an, nach dem *ruleIdentifier* (alphabetisch) zu entscheiden. Dies bietet den Vorteil, dass Entwickler beeinflussen können, welche Regel im Zweifelsfall anzuwenden ist, indem sie die *ruleIdentifier* nach Priorität wählen.

Die Tabelle in Abbildung 4.21 stellt die verschiedenen vorgestellten Fälle als Übersicht dar.

	B. u. W.	**B. u. ¬W.**	**¬B.**	**Ü.: B. u. W.**	**Ü.: B. u. ¬W.**
F. u. W.	ja	nein	ja	ja	?
F. u. ¬W.	nein	ja	ja	?	ja
¬F.	ja	ja	ja	ja	ja

B. = Bedingung-Attribut, W. = Wert, F. = Folgerung-Attribut,
¬ = verschieden, Ü = Überschneidung, ? = nur zur Laufzeit entscheidbar

Abbildung 4.21: Randbedingungen: Regeln

4.3.3 Context Server - Application Stub

In den vorangehenden Kapiteln wurden bereits die XML Schemata für den Austausch von Context Informationen und Regeln vorgestellt. Im Folgenden wird gezeigt, wie mit Hilfe von auf den Schemata basierenden Nachrichten eine Interaktion zwischen *Context Server* und *Application Stub* erfolgen kann.

Es wird vorausgesetzt, dass der *Context Server* in Betrieb und seine IP Adresse bekannt ist. Beim Start einer Applikation, die auf dem Framework basiert (also von einem *Application*

Stub erbt), muss eine Verbindung zum *Context Server* aufgenommen werden. In diesem Schritt können auch applikationsspezifische Regeln an den *Context Server* übertragen werden. Zu diesem Zweck, sendet der *Application Stub* eine ”set” Nachricht (siehe Kapitel 4.3.2) an den *Context Server*. Falls keine applikationsspezifischen Regeln übertragen werden sollen, bleibt der *contextRule* Teil der ”set” Nachricht leer. Somit ist die Applikation beim *Context Server* registriert.

Ein weiterer Informationsfluss findet aufgrund der Registrierung noch nicht statt. Um Context Informationen vom *Context Server* zu erhalten, müssen weitere Nachrichten vom *Application Stub* gesendet werden, die spezifische Context Informationen abonnieren und/oder explizit anfragen. Abonnierte Context Informationen sendet der *Context Server* dann, wenn sich diese Context Informationen ändern. Explizit angefragte Informationen hingegen nur einmalig, als Antwort auf die Anfrage. Die dafür benötigten Nachrichtenformate wurden in Kapitel 4.3.1 vorgestellt.

Die Sequenzdiagramme in Abbildung 4.22 und Abbildung 4.23 zeigen zwei Beispiele für die Interaktion, bzw. den Informationsfluss zwischen *Context Server* und *Application Stub*.

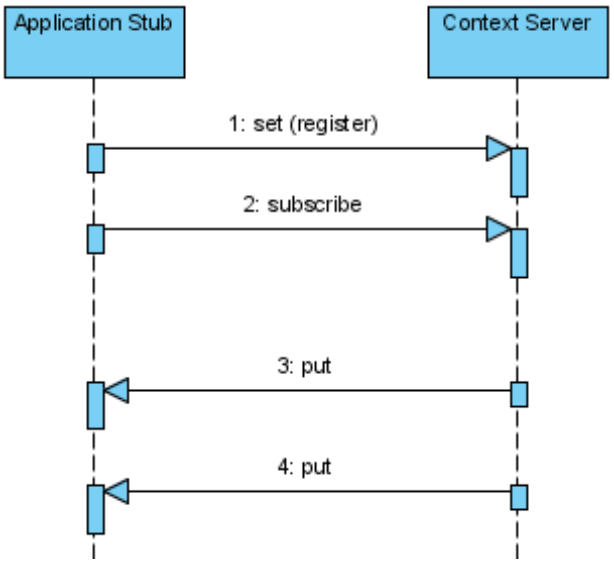

Abbildung 4.22: Context Server - Application Stub: Interaktion 1

4.3.4 Context Server - Context Adapter

Wie bereits erwähnt, wird vorausgesetzt, dass dem *Context Server* beim Start alle potentiell verfügbaren *Context Adapter* bekannt sind. Das bedeutet jedoch nicht, dass all diese

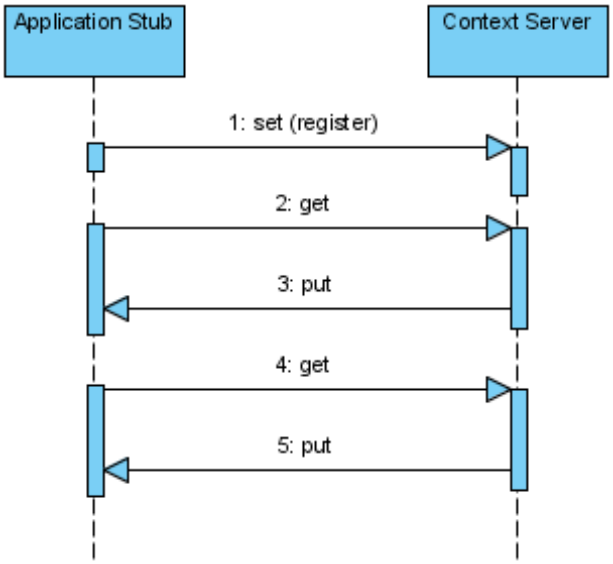

Abbildung 4.23: Context Server - Application Stub: Interaktion 2

Adapter beim Start vorhanden/verfügbar sein müssen. Im Folgenden wird jedoch davon ausgegangen, dass die *Context Adapter* mit dem *Context Server* verbunden sind.

Zwischen *Context Server* und *Context Adapter* spielen die Regeln (Kapitel 4.3.2) keine Rolle. Da alle *Context Adapter* zu einer spezifischen Domain gehören und ausschließlich mit dem *simpleContext* (Kapitel 4.3.1) ihrer Domain arbeiten.

Abhängig davon, ob es sich um aktive oder passive *Context Adapter* handelt, senden diese entweder ihre Context Informationen bei Änderungen oder auf Anfrage des *Context Server*.

Die Sequenzdiagramme in Abbildung 4.24 und Abbildung 4.25 zeigen zwei Beispiele für die Interaktion, bzw. den Informationsfluss zwischen *Context Server* und *Context Adapter*.

4.4 Szenarien: Framework Erweiterung

Eine der wichtigsten Anforderungen an das Framework ist die Erweiterbarkeit. Im Folgenden werden einige Szenarien für die Verwendung des Frameworks und die damit verbundenen notwendigen Erweiterungen aufgezeigt.

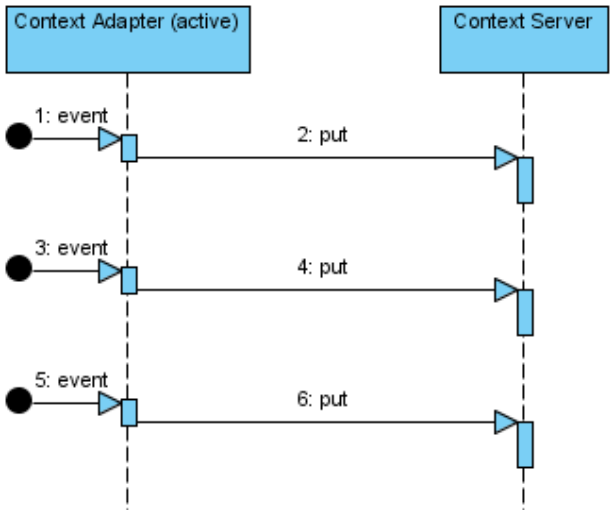

Abbildung 4.24: Context Quelle - Context Server: Interaktion 1

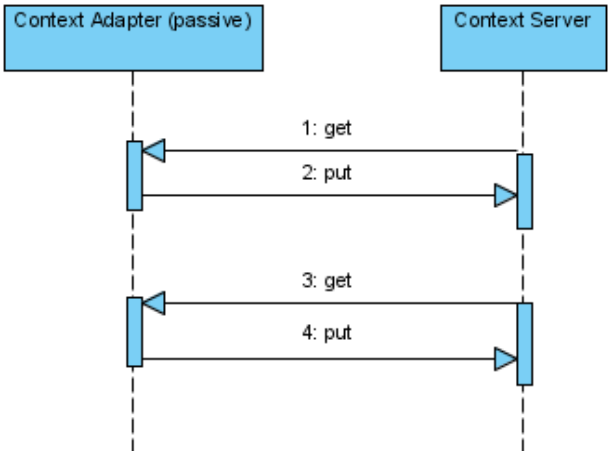

Abbildung 4.25: Context Quelle - Context Server: Interaktion 2

4.4.1 Neue Applikation

Szenario 1: Applikation ohne eigene Regeln, die nur Context von dem Framework bekannten Context Quellen benötigt.

In diesem Fall muss das Framework nicht erweitert werden. Die Applikation kann von einem *Application Stub* des Frameworks erben und die benötigten Context Informationen beim *Context Server* erfragen oder abonnieren.

Szenario 2: Applikation mit eigenen Regeln, die nur Context von dem Framework bekannten Context Quellen benötigt.

In diesem Fall muss das Framework ebenfalls nicht erweitert werden. Die Applikation kann von einem *Application Stub* des Frameworks erben und die benötigten Context Informationen beim *Context Server* erfragen oder abonnieren. Zusätzlich müssen die applikationsspezifischen Regeln definiert werden, welche dann beim Anmelden der Applikation an den *Context Server* übertragen werden.

Szenario 3: Applikation ohne eigene Regeln, die Context Informationen von Context Quellen benötigt, die dem Framework nicht bekannt sind. Die Context Informationen gehören jedoch zu einer bekannten Context Domain.

Das Framework muss durch Anbindung der benötigten Context Quelle(n) erweitert werden. Details dazu werden in Kapitel 4.4.2 beschrieben. Die XML Schemata und die *Rule Engine* des *Context Server* müssen **nicht** erweitert werden.

Szenario 4: Applikation mit eigenen Regeln, die Context Informationen von Context Quellen benötigt, die dem Framework nicht bekannt sind. Die Context Informationen gehören jedoch zu einer bekannten Context Domain.

Wie Szenario 3. Zusätzlich müssen die applikationsspezifischen Regeln definiert werden, welche dann beim Anmelden der Applikation an den *Context Server* übertragen werden.

Szenario 5: Applikation ohne eigene Regeln, die Context Informationen von Context Quellen benötigt, die dem Framework nicht bekannt sind. Die Context Informationen gehören **nicht** zu einer bekannten Context Domain.

Das Framework muss um die neue(n) Context Domain(s) erweitert werden. Dies wird in Kapitel 4.4.3 näher beschrieben. Das Framework muss dann durch Anbindung der benötigten Context Quelle(n) erweitert werden. Details dazu werden in Kapitel 4.4.2 beschrieben. Außerdem müssen die XML Schemata und die *Rule Engine* des *Context Server* erweitert werden.

Szenario 6: Applikation mit eigenen Regeln, die Context Informationen von Context Quellen benötigt, die dem Framework nicht bekannt sind. Die Context Informationen gehören **nicht** zu einer bekannten Context Domain.

Wie Szenario 5. Zusätzlich müssen die applikationsspezifischen Regeln definiert werden, welche dann beim Anmelden der Applikation an den *Context Server* übertragen werden.

4.4.2 Neue Context Quelle

Im Folgenden wird davon ausgegangen, dass die neue Context Quelle zu einer bekannten Context Domain gehört. Die Behandlung von unbekannten Context Domains wird in Kapitel 4.4.3 beschrieben.

Für die neue Context Quelle muss ein *Context Adapter* erstellt werden. Dieser muss von einer der abstrakten *Context Adapter* Klassen des Frameworks erben (siehe Kapitel 4.2.2). Durch den *Context Adapter* muss eine Überführung der individuellen Context Informationen der Context Quelle in das vorgegebene Context Format der zugehörigen Context Domain erfolgen. Da die XML Schemata und die *Rule Engine* nur mit Domain Context arbeiten, müssen sie **nicht** angepasst werden.

4.4.3 Neue Context Domain

Die Anbindung von Context Quellen, die zu keinem bekannten Domain Context gehören, ist mit Abstand der aufwändigste Fall in Bezug auf die Framework Erweiterung.

Die XML Schemata für die Context Informationen und für die Regeln müssen erweitert werden. Dies kann entweder durch die vorgesehenen Extensions erfolgen, oder durch permanente Erweiterung der Schemata.

Entsprechend der XML Schemata muss zusätzlich eine Erweiterung der Context Objekte erfolgen (siehe Kapitel 4.2.4).

Zusätzlich sollte ein neuer abstrakter *Domain Context Adapter* (siehe Kapitel 4.2.2) erstellt werden, um die Implementierung der zugehörigen konkreten Context Adapter zu vereinfachen. Soll das Framework nicht permanent erweitert werden, kann jedoch evtl. auf die Implementierung eines abstrakten *Domain Context Adapter* verzichtet werden.

Eine Erweiterung der *Rule Engine* (siehe Kapitel 4.2.1) ist aufgrund ihrer generischen Struktur nicht notwendig. Allerdings können bei Bedarf Server eigene Regeln für den neuen Domain Context erstellt werden.

4.5 Abgleich mit Anforderungen

Der in diesem Kapitel vorgestellte Entwurf eines Frameworks für Context-Awareness auf mobilen Geräten, soll die in Kapitel 3 aufgestellten Anforderungen erfüllen. Um sicher zu stellen, dass alle Anforderungen berücksichtigt wurden, werden sie im Folgenden mit den zugehörigen Aspekten des vorgestellten Entwurfs abgeglichen.

Die Tabelle in Abbildung 4.26 fasst die Ergebnisse des Abgleichs zusammen.

Auch wenn der vorgestellte Entwurf die funktionalen Anforderungen erfüllt, muss letzten Endes die Implementierung des Entwurfs gegen die Anforderungen getestet werden. Im nachfolgenden Kapitel 4.6 wird ein Testkonzept vorgestellt, welches beschreibt, wie eine Implementierung des Entwurfs in der Praxis getestet werden könnte.

A.1 Anbindung von Context Quellen Die Anforderungen aus Kapitel 3.2.1/A.1.1 - 5, sowie 3.2.1/A.1.6 - 9 werden durch die Komponenten aus Kapitel 4.2 (speziell durch die *Context Adapter*, Kapitel 4.2.2) und das vorgestellte Konzept zur Interaktion (Kapitel 4.3) erfüllt. Dies geht auch aus den Szenarien zur Erweiterbarkeit hervor (Kapitel 4.4).

A.2 Anbindung von Applikationen Die Anforderungen aus Kapitel 3.2.1/A.2.1 - 6 werden durch die Konzepte aus Kapitel 4.2.3 und 4.3 erfüllt (*Application Stub* und Interaktion).

A.3 Verarbeitung von applikationsspezifischer Logik Die Anforderungen 3.2.1/A.3.1 und 3.2.1/A.3.2 - 4 müssen getrennt voneinander betrachtet werden. Das Konzept des *Context Servers* (speziell *Rule Engine* und *Context Store*, Kapitel 4.2.1 und 4.2.1) in Verbin-

dung mit dem Rule Format (Kapitel 4.3.2) erfüllen Anforderung 3.2.1/A.3.1. Die Anforderungen aus Kapitel 3.2.1/A.3.2 - 4 werden durch die *Application Stubs* (Kapitel 4.2.3) erfüllt. Außerdem ist Kapitel 4.3 (Interaktion) für alle Anforderungen unter 3.2.1/A.3 relevant.

B.1 Registrierung von Context Quellen Die Anforderung 3.2.2/B.1 wird durch die Konzepte aus Kapitel 4.3.1 (Nachrichtenformat) und 4.2.2 (*Context Adapter*) erfüllt.

B.2 Registrierung von Applikationen Kapitel 4.3.1 (Nachrichtenformat) und 4.2.3 (*Application Stub*) erfüllen die Anforderung 3.2.2/B.2.

B.3 Applikationsspezifische Regeln Die *Rule Engine* und das Rule Format (Kapitel 4.2.1 u. 4.3.2) erfüllen die Anforderung 3.2.2/B.3.

C.1 - 3 Context Akkumulation, Context Abstraktion, Context Distribution Diese Anforderungen unter Kapitel 3.2.3 hängen sehr eng zusammen und werden daher gemeinsam betrachtet. Zu Ihrer Erfüllung dienen vor allem die *Rule Engine* (Kapitel 4.2.1), der *Context Store* (Kapitel 4.2.1), sowie die *Context Adapter* (Kapitel 4.2.2). Weiterhin spiele die vorgestellten Protokolle zur Interaktion (Kapitel 4.3) eine wichtige Rolle. Ebenfalls beteiligt sind der *Context Server* (Kapitel 4.2.1) und die Context Objekte (Kapitel 4.2.4).

Persistenz von Context Informationen Die Anforderung aus Kapitel 3.2.4 ist im Gegensatz zu den anderen Anforderungen eine bewusste Einschränkung (keine Persistenz). Da im Entwurf keine Komponente für die dauerhafte Persistenz von Context Informationen vorhanden ist, wird auch diese Anforderung erfüllt.

Nichtfunktionale Anforderungen Im Gegensatz zu den funktionalen Anforderungen, können die nichtfunktionalen Anforderungen nur teilweise durch den Entwurf abgedeckt werden. Daher ist auch hier ein Test gegen die Implementierung des Entwurfs von entscheidender Bedeutung. Lediglich die Anforderungen aus Kapitel 3.3.4 (Änderbarkeit) und 3.3.5 (Übertragbarkeit) werden bereits durch den vorgestellten Entwurf erfüllt (Kapitel 4.1.1 und 4.4). Natürlich müssen auch sie gegen die Implementierung getestet werden.

	A.1[1]	A.2[2]	A.3.1[3]	A.3.2 - 4[4]	B.1[5]	B.2[6]	B.3[7]	C[8]
Context Server	(✓)	(✓)	(✓)		(✓)	(✓)		(✓)
Rule Engine			✓				✓	✓
Context Store			✓					✓
Context Adapter	✓				✓			✓
Application Stub		✓		✓		✓		
Context Objekte	✓							(✓)
Nachrichtenformat	✓	✓	(✓)	✓	✓	✓		✓
Rule Format			✓				✓	✓
Erweiterung	✓							

1. Anforderung A.1.1 - 9: Anbindung von Context Quellen
2. Anforderung A.2.1 - 6: Anbindung von Applikationen
3. Anforderung A.3.1: Verarbeitung von applikationsspezifischer Logik (Framework)
4. Anforderung A.3.2 - 4: Verarbeitung von applikationsspezifischer Logik (Applikation)
5. Anforderung B.1: Registrierung von Context Quellen
6. Anforderung B.2: Registrierung von Applikationen
7. Anforderung B.3: Applikationsspezifische Regeln
8. Anforderung C.1 - 3: Verarbeitung von Context Informationen

Abbildung 4.26: Abgleich: funktionale Anforderungen

4.6 Testkonzept

Die Durchführung von Tests an Context-Sensitiven Systemen ist generell schwierig, da sich viele reale Context Informationen (Testdaten) dynamisch ändern. Testfälle können somit nicht unter den gleichen Testbedingungen wiederholt werden. Eine systematische Testauswertung wird dadurch verhindert.

Daher ist es notwendig, Wege zu finden, wie trotzdem aussagekräftige Tests durchgeführt werden können. Eine gute Möglichkeit, die grundlegende Funktionalität eines Context-Sensitiven Systems zu testen, besteht in der Simulation von Context Quellen. Somit lassen sich konstante Testfälle und Testdaten garantieren. Die simulierten Context Quellen müssen dazu vorher festgelegte Context Informationen als Testdaten ausgeben. Abschließend sollte jedoch auch die Interaktion mit realen Context Quellen (reale Testdaten) getestet werden, um zu zeigen, dass das System nicht ausschließlich für vereinfachte Testfälle funktioniert.

Bezogen auf den vorgestellten Entwurf bedeutet das:

- Es müssen diverse ”dummy” Context Quellen erstellt und mit *Context Adaptern* verbunden werden. Der einfachste Weg dafür ist, eine Klasse zu erstellen, die von einem *Context Adapter* erbt und in dieser Klasse die gewünschte Funktionalität der ”dummy” Context Quelle zu implementieren.
- Der Informationsfluss von den ”dummy” Context Quellen zum *Context Server*, die Verarbeitung der Context Informationen, sowie der Informationsfluss vom *Context Server* zu einer Beispiel Applikation sollten mit Hilfe von White-Box Tests getestet werden. Die Testumgebung für die White-Box Tests muss dazu einen Einblick in die internen Abläufe der Framework Komponenten ermöglichen.
- Für den abschließenden Test mit realen Context Quellen bietet sich ein Black-Box Test an.

In der *Rule Engine*, bzw. der Verarbeitung der XML Regeln, steckt die größte Komplexität und somit auch das größte Potential für Fehler innerhalb des vorgestellten Entwurfs. Bei der Erstellung von Testfällen, der anschließenden Testdurchführung und Testauswertung sollte dieser Komponente daher besondere Aufmerksamkeit gewidmet werden.

Kapitel 5

Realisierung

Dieses Kapitel beschäftigt sich mit der Realisierung des in Kapitel 4 vorgestellten Entwurfs. Es war und ist nicht Ziel dieser Arbeit, ein marktreifes Produkt zu implementieren. Der Fokus liegt eindeutig auf dem konzeptionellen Teil. Um die generelle Realisierbarkeit des vorgestellten Entwurfs zu zeigen, erfolgt die Implementierung daher lediglich in Form eines Prototyps. Weiterhin soll durch die Implementierung nicht gezeigt werden, wie sich möglichst viele reale Context Quellen an das Framework anbinden lassen, sondern wie die Komponenten des Framework prinzipiell funktionieren und interagieren können. Daher werden ausschließlich vereinfachte, simulierte Context Quellen verwendet.

5.1 Entwicklungsumgebung

Als Entwicklungsumgebung für die Realisierung wurde Microsoft Visual Studio eingesetzt (siehe Abbildung 5.1). Zusätzlich kam das .NET Compact Framework [Microsoft, 2004] [Dröge et al., 2006] und die Programmiersprache C# zu Einsatz. Als Zielplattform wurde ein HP iPAQ Smartphone [HP] verwendet (siehe Abbildung 5.2), welcher per USB mit der Entwicklungsplattform verbunden wurde. Als Entwicklungsplattform diente ein Standard Windows PC.

Microsoft Visual Studio ermöglicht eine sehr komfortable Entwicklung von mobilen Applikationen. Die Applikationen können theoretisch vollständig auf der Entwicklungsplattform entwickelt und getestet werden, da die Entwicklungsumgebung einen Emulator für mobile Geräte anbietet. Zusätzlich kann direkt auf dem mobilen Gerät getestet werden, wobei das

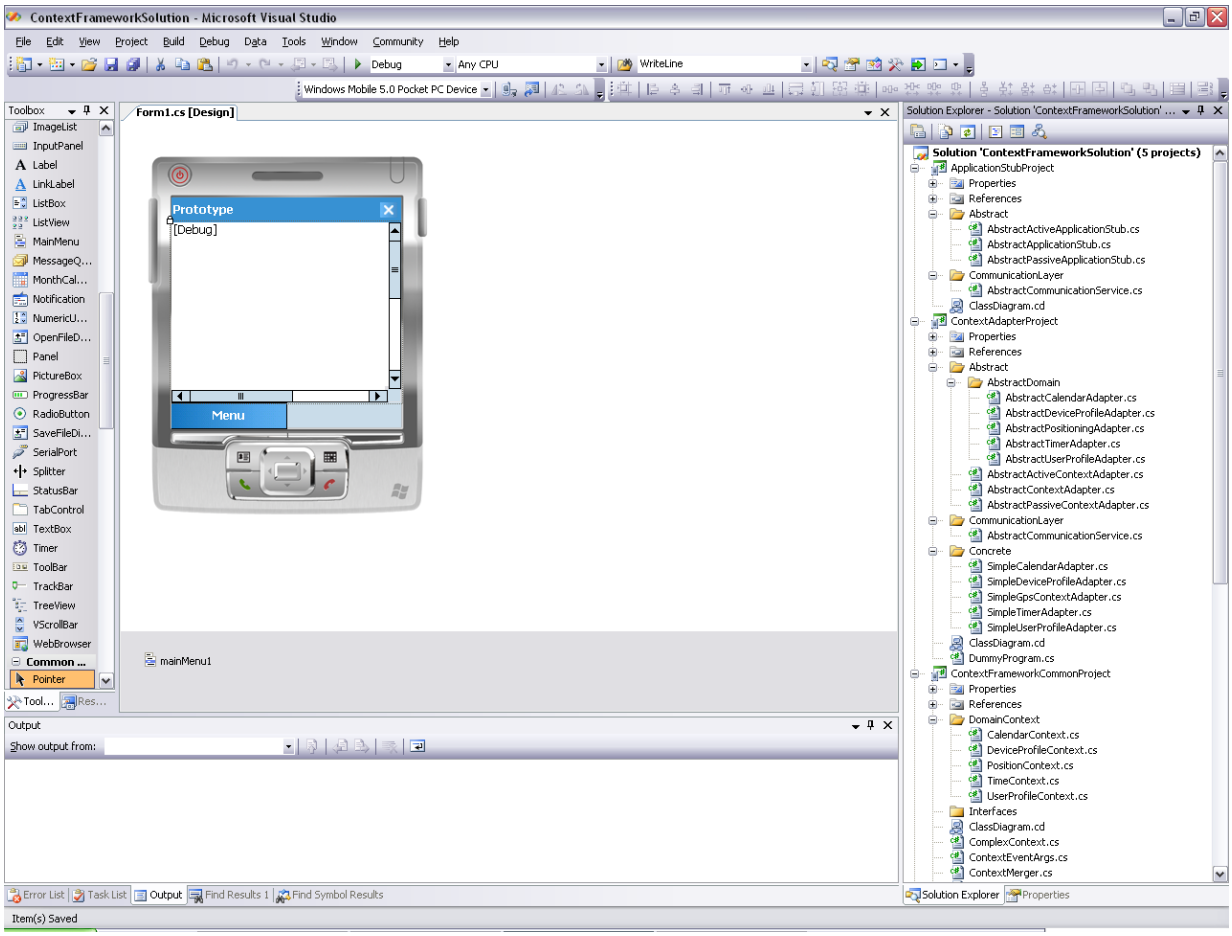

Abbildung 5.1: Microsoft Visual Studio

Debugging aus der Entwicklungsumgebung heraus erfolgt. Der zweite Ansatz ist performanter und erfordert keinen nennenswerten technischen Aufwand.

Für die Entwicklung der in Kapitel 4.3 vorgestellten XML Schemata und XML Beispiele, wurde ein XML Editor eingesetzt.

5.2 Funktionsumfang

Im Folgenden wird beschrieben, welche Aspekte des Entwurfs im Prototyp realisiert wurden und worauf bewusst verzichtet wurde.

Abbildung 5.2: HP iPAQ [HP]

5.2.1 Komponenten

Context Server

Eine der wichtigsten Eigenschaften des *Context Servers* ist, mit mehreren Applikationen und Context Quellen gleichzeitig verbunden sein zu können. Im Protoyp wurden daher zwei Multithreading fähige Kommunikations-Services implementiert. Der *AbstractContextCommunicationService* dient dabei der Kommunikation mit den Context Quellen und der *AbstractApplicationCommunicationService* der Kommunikation mit den Applikationen. Die Kommunikation findet über TCP/IP Verbindungen statt. *AbstractContextCommunicationService* wartet auf eingehende Verbindungen auf Port 4712 und *AbstractApplicationCommunicationService* auf Port 4711. Zur Datenübertragung werden aus technischen Gründen einfache Byte Streams verwendet. Da die Kommunikation auf konzeptioneller Ebene ausschließlich über XML Nachrichten stattfindet (siehe Kapitel 4.3), ist eine Übertragung per Byte Stream unproblematisch. Innerhalb Framework Komponenten wird allerdings ausschließlich mit den in Kapitel 4.2.4 beschriebenen Context Objekten gearbeitet. Daher ist eine Überführung der XML Nachrichten in Context Objekte, sowie eine Rücküberführung

notwendig. Diese Aufgabe wird von speziellen Hilfsklassen übernommen (*ContextToXmlMapper* und *XmlToContextMapper*). Der Inhalt der versendeten Nachrichten ist für die beiden Kommunikations-Services irrelevant. Dieser wird erst auf einer höheren Ebene ausgewertet. Somit können die Kommunikations-Services auf einen überschaubaren Funktionsumfang beschränkt werden.

Wie bereits im Entwurf dargelegt wurde, ist die *Rule Engine* die komplexeste Komponenten des Frameworks. Um den Implementierungsaufwand für den Prototyp in einem angemessenen Rahmen zu halten, ist es daher notwendig, die *Rule Engine* auf eine rudimentäre Basisfunktionalität zu beschränken. Besonders die generischen und dynamischen Aspekte der *Rule Engine* bedeuten einen zu hohen Aufwand für die Implementierung eines Prototyps. Daher werden für die *Rule Engine* einige Regeln in hart codierter Form implementiert, die für die Realisierung eines Beispiel Szenarios benötigt werden (siehe Kapitel 5.2.2).

Der *Context Store* wird ebenfalls auf seine Basisfunktionalität beschränkt und besteht im Wesentlichen aus einer Klasse, die die beiden Context Objekte, welche in Kapitel 4.2.4 beschrieben wurden, verwaltet.

Context Adapter

Wie im Entwurf beschrieben wurde, sind die *Context Adapter* für die Kommunikation mit dem *Context Server* zuständig. Zu diesem Zweck stellen die *Context Adapter* bei ihrem Start eine TCP/IP Verbindung (Port 4712) mit dem *Context Server* her. Für den Prototyp wird darauf verzichtet, die Funktionalität für explizite Anfragen des *Context Server* an die *Context Adapter* zu realisieren. Context Informationen werden ausschließlich aktiv an den *Context Server* gesendet. Entsprechend dem Entwurf, sollen die *Context Adapter* intern mit Simple Context Objekten (Kapitel 4.2.4) arbeiten und diese nur für die Kommunikation mit dem *Context Server* in XML umwandeln. Zur Vereinfachung werden im Prototyp jedoch vorgefertigte XML Nachrichten periodisch an den *Context Server* gesendet.

Application Stub

Die *Application Stubs* sind, wie auch die *Context Adapter* hauptsächlich für die Kommunikation mit dem *Context Server* zuständig. Beim Start stellen auch sie eine TCP/IP Verbindung (Port 4711) mit dem *Context Server* her. Im Prototyp sind keine aktiven Anfragen der *Application Stubs* an den *Context Server* vorgesehen. Es werden ausschließlich

Nachrichten vom *Context Server* an die *Application Stubs* gesendet (basierend auf hart codierten Regeln).

5.2.2 Prototyp: Beispiel Anwendung

Um die Funktionalität des Prototyps zu zeigen, wird eine einfach Anwendung erstellt. Die Vorlage bietet das Beispiel aus Kapitel 4.3.2, bzw. Abbildung 4.18, in dem auf den aktuellen *currentUserMovementStatus* geschlossen wird. Zu diesem Zweck werden zwei simulierte Context Quellen erstellt, die über *Context Adapter* an den *Context Server* angebunden werden. Die Context Quellen liefern periodisch Context Informationen zur aktuellen Geschwindigkeit und zur aktuellen Mobilität des Benutzers. Im *Context Server*, bzw. der *Rule Engine* werden hart codierte Regeln erstellt, die diese *simpleContext* Informationen, entsprechend dem genannten Beispiel in *complexContext* Informationen überführen (*currentUserMovementStatus*). Die Information zum *currentUserMovementStatus* wird dann an den *Application Stub* einer Anwendung geschickt, die die erhaltene Information auf dem Display des mobilen Geräts ausgibt. Eine weitere Verarbeitung findet nicht statt. Alle beschriebenen Komponenten werden gemeinsam auf einem mobilen Gerät betrieben.

Testergebnisse

In Kapitel 4.6 wurde ein Testkonzept für das Framework vorgestellt. Entsprechend des Testkonzepts wurden für den Prototyp ausschließlich simulierte Context Quellen verwendet. In Verbindung mit der eingesetzten Entwicklungsumgebung (Kapitel 5.1), konnten die geforderten White-Box Tests erfolgreich durchgeführt werden. In den Tests hat sich gezeigt, dass bereits die einfache Prototyp Implementierung des Frameworks in der Lage ist, Context Informationen von verschiedenen Context Quellen (parallel) zu verarbeiten. Eine Applikation konnte mit minimalem Aufwand an das Framework angebunden werden.

Kapitel 6

Fazit

Die Entwicklung von Context-Sensitiven Applikationen ist nicht trivial. Im Vergleich zu herkömmlichen Applikationen erfordern sie die Integration von Context Informationen, welche unter Umständen von einer Vielzahl von Context Quellen bezogen werden müssen. Das Anbinden der verschiedenen Context Quellen kann dabei einen erheblichen Implementierungsaufwand bedeuten. Dieser Implementierungsaufwand stellt einen Overhead dar, der nicht der Realisierung der eigentlichen Applikationslogik dient.

In dieser Arbeit wurde daher ein Framework entworfen, dessen Aufgabe es ist, die Entwicklung von Context-Sensitiven Applikationen zu erleichtern (Kapitel 4). Der Fokus liegt dabei speziell auf mobilen Geräten. Es wurde gezeigt, dass der vorgestellte Entwurf die Anforderungen erfüllt, die im Rahmen dieser Arbeit ermittelt wurden (Kapitel 4 und 4.5). Weiterhin wurde seine Realisierbarkeit erfolgreich überprüft (Kapitel 5).

Der Entwurf zeichnet sich vor allem durch seine hohe Flexibilität aus. Da nicht zwischen lokalen und verteilten Applikationen unterschieden wird, lassen sich eine Vielzahl von Szenarien realisieren (Kapitel 4.1.1). Die hohe Flexibilität wird außerdem durch die verschiedenen Möglichkeiten der Erweiterbarkeit verdeutlicht (Kapitel 4.4).

Ein weiterer wichtiger Aspekt des Entwurfs ist das Protokoll für die Interaktion zwischen den einzelnen Framework Komponenten (Kapitel 4.3). Dieses ist einerseits sehr einfach, andererseits jedoch auch sehr mächtig.

Im Gegensatz zu den meisten anderen Arbeiten (Kapitel 3.4), deckt die vorgestellte Lösung alle wichtigen Aspekte von Context-Sensitiven Applikationen, bzw. Systemen ab (Context Akkumulation, Context Abstraktion und Context Distribution) und ist nicht auf spezielle

Context Informationen beschränkt.

6.1 Ausblick

Der Schwerpunkt dieser Arbeit liegt auf dem Entwurf und nicht auf der Implementierung eines Frameworks. Obwohl die generelle Realisierbarkeit mit Hilfe eines Prototyps gezeigt wurde, bedarf es noch einiger Arbeit, alle Aspekte des Entwurfs vollständig umzusetzen. Besonders die Implementierung der *Rule Engine* stellt dabei voraussichtlich den größten Umfang dar.

Im Rahmen dieser Arbeit wurde bewusst auf den Einsatz von Ontologien und/oder RDF + RDFS verzichtet (Kapitel 2.4 und 4.3), um die Komplexität der Lösung nicht unnötig zu erhöhen. In Arbeiten, die auf die Ergebnisse dieser Arbeit aufbauen, wäre es jedoch trotzdem interessant zu untersuchen, wie der Einsatz von Ontologien, bzw. RDF + RDFS die Mächtigkeit der Lösung verstärken könnte.

Literaturverzeichnis

[Weiser, 1991] Mark Weiser: **The Computer for the 21st Century**, Scientific American Ubicomp Paper (1991).

[Schmidt et al., 1998] A. Schmidt, M. Beigl, H.-W. Gellersen: **There is more to Context than Location**, IEEE (1998).

[Hofer et al., 2003] T. Hofer et al.: **Context-Awareness on Mobile Devices the Hydrogen Approach**, IEEE (2003).

[Yan und Sere, 2004] L. Yan, K. Sere: **A Formalism for Context-Aware Mobile Computing**, IEEE (2004).

[Eckstein, 2004] R. Ecksten, S. Eckstein: **XML und Datenmodellierung**, dpunkt.verlag (2004).

[BSI, 2006] Bundesamt für Sicherheit in der Informationstechnik: **Pervasive Computing: Entwicklungen und Auswirkungen**, BSI (2006).

[Ganek und Corbi, 2003] A.G. Ganek, T.A. Corbi: **The dawning of the autonomic computing era**, IBM Systems Journal 42(1) (2003).

[Barkhuus und Dey, 2003] L. Barkhuus, A. Dey: **Is Context-Aware Computing Taking Control Away from the User?**, Proceedings of UbiComp (2003).

[Powers, 2003] Shelley Powers: **Practical RDF**, O'Reilly (2003).

[Gamma et al., 1995] E. Gamma et al.: **Entwurfsmuster**, Addison-Wesley (2004).

[Chen et al., 2004] G. Chen et al.: **Design and Implementation of a Large-Scale Context Fusion Network**, IEEE (2004).

[Chen und Kotz, 2000] G. Chen, D. Kotz: **A Survey of Context-Aware Mobile Computing Research**, IEEE (2000).

[Tan et al., 2003] C.-P. Tan et al.: **Context-aware Service Protocol**, IEEE (2003).

[Strimpakou et al., 2006] M. A. Strimpakou et al.: **A Context Ontology for Pervasive Service Provision**, IEEE (2006).

[Li et al., 2006] J. Li et al.: **FollowMe: On Research of Pluggable Infrastructure for Context-Awareness**, IEEE (2006).

[Kouadri et al., 2004] G. Kouadri et al.: **Context-Aware Computing: A Guide for the Pervasive Computing Community**, IEEE (2004).

[Moran und Dourish, 2001] T. P. Moran, P. Dourish: **Introduction to This Special Issue on Context-Aware Computing**, Special Issue of Human-Computer Interaction, Volume 16 (2001).

[Parsons et al., 1999] D. Parsons et al.: **A "Framework" for Object Oriented Frameworks Design**, IEEE (1999).

[Pashtan, 2005] Ariel Pashtan: **Mobile Web Services**, Cambridge (2005).

[Beierle et al., 2000] C. Beierle, G. Kern-Isberner: **Methoden wissensbasierter Systeme**, Verlag Vieweg (2000).

[Reussner et al., 2006] R. Reussner, W. Hasselbring: **Handbuch der Software-Architektur**, dpunkt.verlag (2006).

[Ebert, 2005] C. Ebert: **Systematisches Requirement Management**, dpunkt.verlag (2005).

[Rupp et al., 2007] C. Rupp et al.: **Requirements-Engineering und Management**, Hanser (2007).

[Hohpe et al., 2004] G. Hohpe, B. Woolf: **Enterprise Integration Patterns**, Addison-Wesley (2004).

[Szyperski, 2002] C. Szyperski et al.: **Component Software**, Addison-Wesley (2002).

[Salvatori, 2006] P. Salvatori: **Anwendungsentwicklung für den Pocket-PC**, Data Becker (2006).

[Dröge et al., 2006] R. Dröge, P. Nowak, T. Weber: **Programmieren mit dem .NET Compact Framework**, Microsoft Press (2006).

[Horridge et al., 2007] M. Horridge et al.: **A Practical Guide To Building OWL Ontologies Using Protégé 4**, The University Of Manchester (2007).

[IBM, 2001] ibm.com: **Autonomic Computing**, (2001). Im Internet zu finden unter <http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf> (März 2008)

[Winograd, 2001] T. Winograd: **Architectures for context.**, (2001). Im Internet zu finden unter <http://hci.stanford.edu/w̃inograd/papers/context/context.pdf> (März 2008)

[Gruber, 1992] T. Gruber: **What is an Ontology?**, (1992). Im Internet zu finden unter <http://www-ksl.stanford.edu/kst/what-is-an-ontology.html> (März 2008)

[W3C 1, 2004] w3.org: **Resource Description Framework**, (2004). Im Internet zu finden unter <http://www.w3.org/RDF/> (März 2008)

[W3C 2, 2004] w3.org: **RDF/XML Syntax Specification**, (2004). Im Internet zu finden unter <http://www.w3.org/TR/rdf-syntax-grammar/> (März 2008)

[W3C 3, 2001] w3.org: **W3C Semantic Web Activity**, (2001). Im Internet zu finden unter <http://www.w3.org/2001/sw/> (März 2008)

[W3C 4, 2004] w3.org: **RDF Vocabulary Description Language 1.0: RDF Schema**, (2004). Im Internet zu finden unter <http://www.w3.org/TR/rdf-schema/> (März 2008)

[W3C 5, 2007] w3.org: **Composite Capabilities/Preference Profiles: Structure and Vocabularies 2.0**, (2007). Im Internet zu finden unter <http://www.w3.org/Mobile/CCPP/> (März 2008)

[W3C 6, 2004] w3.org: **Web Ontology Language (OWL)**, (2004). Im Internet zu finden unter <http://www.w3.org/2004/OWL/> (März 2008)

[DAML, 2000] daml.org: **The DARPA Agent Markup Language Homepage**, (2000). Im Internet zu finden unter <http://www.daml.org/> (März 2008)

[TALIS, 2005] talis.com: **An Introduction to RDF**, (2005). Im Internet zu finden unter <http://research.talis.com> (März 2008)

[Microsoft, 2004] microsoft.com: **Das .NET Compact Framework**, (2004). Im Internet zu finden unter <http://msdn2.microsoft.com/de-de/library/bb979027.aspx> (März 2008)

[Sun, 2008] sun.com: **Java ME**, (2008). Im Internet zu finden unter <http://java.sun.com/javame> (März 2008)

[UbiComp] UbiComp @ Informatik.HAW Hamburg. Im Internet zu finden unter <http://users.informatik.haw-hamburg.de/ ubicomp/projects.html> (März 2008)

[DIN 66272] de.wikipedia.org: **DIN 66272**, (1994). Im Internet zu finden unter <http://de.wikipedia.org/wiki/DIN_66272> (März 2008)

[ISO/IEC 9126] de.wikipedia.org: **ISO/IEC 9126**. Im Internet zu finden unter <http://de.wikipedia.org/wiki/ISO/IEC_9126> (März 2008)

[HP] hp.com Im Internet zu finden unter <http://www.hp.com/> (März 2008)

[Weindorf, 2007] M. Weindorf: **AW2 Projektbericht, Pervasive Gaming Framework**, (2007). Im Internet zu finden unter <http://users.informatik.haw-hamburg.de/ ubicomp/projects.html> (März 2008)

Anhang A

Inhalt der CD

Die beiliegende CD enthält folgendes:

- Masterarbeit im PDF Format
- Sourcecode des Framework-Prototyp
- XML Schemata und XML Beispiele

Anhang B

XML Schema: Context

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema elementFormDefault="qualified" attributeFormDefault="unqualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:include schemaLocation="extensions/simpleContextExtension.xsd"/>
   <xs:include schemaLocation="extensions/complexContextExtension.xsd"/>
   <xs:element name="contextRoot" type="ContextType">
      <xs:annotation>
         <xs:documentation>root element</xs:documentation>
      </xs:annotation>
   </xs:element>
   <xs:complexType name="ContextType">
      <xs:annotation>
         <xs:documentation>root context type</xs:documentation>
      </xs:annotation>
      <xs:sequence>
         <xs:element name="senderIdentifier" type="xs:string">
            <xs:annotation>
               <xs:documentation>i.e. contextServer, contextAdapterXY, applicationYZ, ...</xs:documentation>
            </xs:annotation>
         </xs:element>
         <xs:element name="messageType" type="xs:string">
            <xs:annotation>
               <xs:documentation>i.e. subsribe, unsubscribe, get, put, ...</xs:documentation>
            </xs:annotation>
         </xs:element>
         <xs:element name="simpleContext" type="SimpleContextType" minOccurs="0"/>
         <xs:element name="complexContext" type="ComplexContextType" minOccurs="0"/>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="SimpleContextType">
      <xs:annotation>
         <xs:documentation>root type for simple context</xs:documentation>
      </xs:annotation>
      <xs:sequence>
         <xs:element name="calendarContext" type="CalendarContextType" minOccurs="0" maxOccurs="unbounded"/>
         <xs:element name="deviceProfileContext" type="DeviceProfileContextType" minOccurs="0"/>
         <xs:element name="positionContext" type="PositionContextType" minOccurs="0"/>
         <xs:element name="timeContext" type="TimeContextType" minOccurs="0"/>
         <xs:element name="userProfileContext" type="UserProfileContextType" minOccurs="0"/>
         <xs:element name="extension" type="SimpleContextExtensionType" minOccurs="0"/>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="ComplexContextType">
      <xs:annotation>
         <xs:documentation>root type for complex context</xs:documentation>
      </xs:annotation>
      <xs:sequence>
         <xs:element name="complexUserContext" type="ComplexUserContextType" minOccurs="0"/>
         <xs:element name="complexDeviceContext" type="ComplexDeviceContextType" minOccurs="0"/>
         <xs:element name="complexEnvironmentContext" type="ComplexEnvironmentContextType" minOccurs="0"/>
         <xs:element name="extension" type="ComplexContextExtensionType" minOccurs="0"/>
      </xs:sequence>
   </xs:complexType>
```

```
<xs:complexType name="CalendarContextType">
   <xs:annotation>
      <xs:documentation>simple</xs:documentation>
   </xs:annotation>
   <xs:sequence>
      <xs:element name="calendarEntryIdentifier" type="xs:string" minOccurs="0"/>
      <xs:element name="description" type="xs:string" minOccurs="0"/>
      <xs:element name="type" type="xs:string" minOccurs="0">
         <xs:annotation>
            <xs:documentation>i.e. birthday, meeting, reminder, ...</xs:documentation>
         </xs:annotation>
      </xs:element>
      <xs:element name="startTime" type="xs:string" minOccurs="0">
         <xs:annotation>
            <xs:documentation>date + time</xs:documentation>
         </xs:annotation>
      </xs:element>
      <xs:element name="endTime" type="xs:string" minOccurs="0">
         <xs:annotation>
            <xs:documentation>date + time</xs:documentation>
         </xs:annotation>
      </xs:element>
      <xs:element name="location" type="PositionType" minOccurs="0"/>
   </xs:sequence>
</xs:complexType>
<xs:complexType name="DeviceProfileContextType">
   <xs:annotation>
      <xs:documentation>simple</xs:documentation>
   </xs:annotation>
   <xs:sequence>
      <xs:element name="deviceType" type="xs:string" minOccurs="0">
         <xs:annotation>
            <xs:documentation>i.e. smartphone, pda, laptop,  ...</xs:documentation>
         </xs:annotation>
      </xs:element>
      <xs:element name="memory" type="xs:int" minOccurs="0">
         <xs:annotation>
            <xs:documentation>in mb</xs:documentation>
         </xs:annotation>
      </xs:element>
      <xs:element name="display" type="DisplayType" minOccurs="0"/>
      <xs:element name="connection" type="ConnectionType" minOccurs="0"/>
   </xs:sequence>
</xs:complexType>
<xs:complexType name="PositionContextType">
   <xs:annotation>
      <xs:documentation>simple</xs:documentation>
   </xs:annotation>
   <xs:sequence>
      <xs:element name="positionDeviceType" type="xs:string" minOccurs="0">
         <xs:annotation>
            <xs:documentation>i.e. gps, cricket, ...</xs:documentation>
         </xs:annotation>
      </xs:element>
      <xs:element name="precision" type="xs:double" minOccurs="0">
         <xs:annotation>
            <xs:documentation>precision in meters</xs:documentation>
         </xs:annotation>
      </xs:element>
      <xs:element name="currentPosition" type="PositionType" minOccurs="0"/>
      <xs:element name="currentSpeed" type="xs:double" minOccurs="0">
         <xs:annotation>
            <xs:documentation>in meter/sec</xs:documentation>
         </xs:annotation>
      </xs:element>
      <xs:element name="currentDirection" type="xs:string" minOccurs="0">
         <xs:annotation>
            <xs:documentation>i.e. N, E, S, W, NW, ...</xs:documentation>
         </xs:annotation>
      </xs:element>
   </xs:sequence>
</xs:complexType>
<xs:complexType name="TimeContextType">
   <xs:annotation>
      <xs:documentation>simple</xs:documentation>
   </xs:annotation>
   <xs:sequence>
      <xs:element name="currentTime" type="xs:string" minOccurs="0"/>
   </xs:sequence>
</xs:complexType>
<xs:complexType name="UserProfileContextType">
```

```
      <xs:annotation>
         <xs:documentation>simple</xs:documentation>
      </xs:annotation>
      <xs:sequence>
         <xs:element name="userName" type="xs:string" minOccurs="0"/>
         <xs:element name="userMobilityPreset" minOccurs="0">
            <xs:annotation>
               <xs:documentation>i.e. walk, bicycle, car</xs:documentation>
            </xs:annotation>
         </xs:element>
         <xs:element name="userLocation" type="UserLocationType" minOccurs="0" maxOccurs="unbounded"/>
         <xs:element name="userInterest" type="UserInterestType" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="PositionType">
      <xs:annotation>
         <xs:documentation>position</xs:documentation>
      </xs:annotation>
      <xs:sequence>
         <xs:element name="radius" type="xs:double" minOccurs="0"/>
         <xs:element name="xPosition" type="xs:double" minOccurs="0"/>
         <xs:element name="yPosition" type="xs:double" minOccurs="0"/>
         <xs:element name="zPosition" type="xs:double" minOccurs="0"/>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="DisplayType">
      <xs:annotation>
         <xs:documentation>display</xs:documentation>
      </xs:annotation>
      <xs:sequence>
         <xs:element name="xResolution" type="xs:int" minOccurs="0"/>
         <xs:element name="yResolution" type="xs:int" minOccurs="0"/>
         <xs:element name="colors" type="xs:int" minOccurs="0"/>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="ConnectionType">
      <xs:annotation>
         <xs:documentation>connection</xs:documentation>
      </xs:annotation>
      <xs:sequence>
         <xs:element name="connectionType" type="xs:string" minOccurs="0">
            <xs:annotation>
               <xs:documentation>i.e. wlan, umts, bluetooth, gprs, infrared, ...</xs:documentation>
            </xs:annotation>
         </xs:element>
         <xs:element name="maxSpeed" type="xs:double" minOccurs="0">
            <xs:annotation>
               <xs:documentation>in kbit/sec</xs:documentation>
            </xs:annotation>
         </xs:element>
         <xs:element name="currentSpeed" type="xs:double" minOccurs="0">
            <xs:annotation>
               <xs:documentation>in kbit/sec</xs:documentation>
            </xs:annotation>
         </xs:element>
         <xs:element name="currentQuality" type="xs:int" minOccurs="0">
            <xs:annotation>
               <xs:documentation>from 1=low to 10=high</xs:documentation>
            </xs:annotation>
         </xs:element>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="UserLocationType">
      <xs:annotation>
         <xs:documentation>user location</xs:documentation>
      </xs:annotation>
      <xs:sequence>
         <xs:element name="locationIdentifier" type="xs:string" minOccurs="0">
            <xs:annotation>
               <xs:documentation>i.e. home, office, ...</xs:documentation>
            </xs:annotation>
         </xs:element>
         <xs:element name="location" type="PositionType" minOccurs="0"/>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="UserInterestType">
      <xs:annotation>
         <xs:documentation>user interest</xs:documentation>
      </xs:annotation>
      <xs:sequence>
         <xs:element name="type" type="xs:string" minOccurs="0">
```

```
            <xs:annotation>
               <xs:documentation>i.e. restaurant, entertainment, ...</xs:documentation>
            </xs:annotation>
         </xs:element>
         <xs:element name="description" type="xs:string" minOccurs="0">
            <xs:annotation>
               <xs:documentation>i.e. modern art, cars, ...</xs:documentation>
            </xs:annotation>
         </xs:element>
         <xs:element name="intensity" type="xs:int" minOccurs="0">
            <xs:annotation>
               <xs:documentation>from 1=low interest to 10=high interest</xs:documentation>
            </xs:annotation>
         </xs:element>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="ComplexUserContextType">
      <xs:annotation>
         <xs:documentation>complex</xs:documentation>
      </xs:annotation>
      <xs:sequence>
         <xs:element name="currentUserMovementStatus" type="xs:string" minOccurs="0">
            <xs:annotation>
               <xs:documentation>i.e. walking, running, driving, ...</xs:documentation>
            </xs:annotation>
         </xs:element>
         <xs:element name="currentUserActivityStatus" type="xs:string" minOccurs="0">
            <xs:annotation>
               <xs:documentation>i.e. busy, idle, ... </xs:documentation>
            </xs:annotation>
         </xs:element>
         <xs:element name="currentUserSpareTime" type="xs:int" minOccurs="0">
            <xs:annotation>
               <xs:documentation>in minutes</xs:documentation>
            </xs:annotation>
         </xs:element>
         <xs:element name="currentUserLocation" type="xs:string" minOccurs="0">
            <xs:annotation>
               <xs:documentation>i.e. home, office, unknown, ...</xs:documentation>
            </xs:annotation>
         </xs:element>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="ComplexEnvironmentContextType">
      <xs:annotation>
         <xs:documentation>complex</xs:documentation>
      </xs:annotation>
      <xs:sequence>
         <xs:element name="currentWeatherCondition" type="xs:string" minOccurs="0">
            <xs:annotation>
               <xs:documentation>i.e. hot, cold, rainy, ...</xs:documentation>
            </xs:annotation>
         </xs:element>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="ComplexDeviceContextType">
      <xs:annotation>
         <xs:documentation>complex</xs:documentation>
      </xs:annotation>
      <xs:sequence>
         <xs:element name="currentConnectionSpeed" type="xs:string" minOccurs="0">
            <xs:annotation>
               <xs:documentation>i.e. slow, fast, ...</xs:documentation>
            </xs:annotation>
         </xs:element>
         <xs:element name="currentConnectionQuality" type="xs:string" minOccurs="0">
            <xs:annotation>
               <xs:documentation>i.e. low, high, ...</xs:documentation>
            </xs:annotation>
         </xs:element>
      </xs:sequence>
   </xs:complexType>
</xs:schema>
```

Anhang C

XML Schema: Rules

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema elementFormDefault="qualified" attributeFormDefault="unqualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:include schemaLocation="extensions/contextFactExtension.xsd"/>
   <xs:include schemaLocation="extensions/complexContextImplicationExtension.xsd"/>
   <xs:element name="root" type="RootType">
      <xs:annotation>
         <xs:documentation>root element</xs:documentation>
      </xs:annotation>
   </xs:element>
   <xs:complexType name="RootType">
      <xs:annotation>
         <xs:documentation>root type</xs:documentation>
      </xs:annotation>
      <xs:sequence>
         <xs:element name="senderIdentifier" type="xs:string">
            <xs:annotation>
               <xs:documentation>i.e. server, applicationYZ, ...</xs:documentation>
            </xs:annotation>
         </xs:element>
         <xs:element name="messageType" type="xs:string">
            <xs:annotation>
               <xs:documentation>i.e. set, add, alter, delete, ...</xs:documentation>
            </xs:annotation>
         </xs:element>
         <xs:element name="contextRule" type="ContextRuleType" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="ContextRuleType">
      <xs:annotation>
         <xs:documentation>root rule type</xs:documentation>
      </xs:annotation>
      <xs:sequence>
         <xs:element name="ruleIdentifier" type="xs:string"/>
         <xs:element name="contextFact" type="ContextFactType">
            <xs:annotation>
               <xs:documentation>"if"</xs:documentation>
            </xs:annotation>
         </xs:element>
         <xs:element name="implication" type="ComplexContextImplicationType">
            <xs:annotation>
               <xs:documentation>"then"</xs:documentation>
            </xs:annotation>
         </xs:element>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="ContextFactType">
      <xs:annotation>
         <xs:documentation>root type or context facts</xs:documentation>
      </xs:annotation>
      <xs:sequence>
         <xs:element name="calendarContext" type="CalendarFactType" minOccurs="0"/>
         <xs:element name="deviceProfileContext" type="DeviceProfileFactType" minOccurs="0"/>
         <xs:element name="positionContext" type="PositionFactType" minOccurs="0"/>
         <xs:element name="timeContext" type="TimeFactType" minOccurs="0"/>
         <xs:element name="userProfileContext" type="UserProfileFactType" minOccurs="0"/>
```

```
      <xs:element name="extension" type="ContextFactExtensionType" minOccurs="0"/>
   </xs:sequence>
</xs:complexType>
<xs:complexType name="ComplexContextImplicationType">
   <xs:annotation>
      <xs:documentation>root type for complex context implications</xs:documentation>
   </xs:annotation>
   <xs:choice>
      <xs:element name="complexUserContext" type="ComplexUserContextType" minOccurs="0"/>
      <xs:element name="complexDeviceContext" type="ComplexDeviceContextType" minOccurs="0"/>
      <xs:element name="complexEnvironmentContext" type="ComplexEnvironmentContextType" minOccurs="0"/>
      <xs:element name="extension" type="ComplexContextImplicationExtensionType" minOccurs="0"/>
   </xs:choice>
</xs:complexType>
<xs:complexType name="CalendarFactType">
   <xs:annotation>
      <xs:documentation>simple fact</xs:documentation>
   </xs:annotation>
   <xs:sequence>
      <xs:element name="type" type="xs:string" minOccurs="0">
         <xs:annotation>
            <xs:documentation>i.e. birthday, meeting, reminder, ...</xs:documentation>
         </xs:annotation>
      </xs:element>
      <xs:element name="startFromCurrentTime" type="xs:int" minOccurs="0">
         <xs:annotation>
            <xs:documentation>current time + minutes to start</xs:documentation>
         </xs:annotation>
      </xs:element>
      <xs:element name="startFromCurrentTime" type="xs:int" minOccurs="0">
         <xs:annotation>
            <xs:documentation>current time + minutes to end</xs:documentation>
         </xs:annotation>
      </xs:element>
      <xs:element name="location" type="PositionType" minOccurs="0"/>
   </xs:sequence>
</xs:complexType>
<xs:complexType name="DeviceProfileFactType">
   <xs:annotation>
      <xs:documentation>simple fact</xs:documentation>
   </xs:annotation>
   <xs:sequence>
      <xs:element name="deviceType" type="xs:string" minOccurs="0">
         <xs:annotation>
            <xs:documentation>i.e. smartphone, pda, laptop,  ...</xs:documentation>
         </xs:annotation>
      </xs:element>
      <xs:element name="minMemory" type="xs:int" minOccurs="0"/>
      <xs:element name="maxMemory" type="xs:int" minOccurs="0">
         <xs:annotation>
            <xs:documentation>in mb</xs:documentation>
         </xs:annotation>
      </xs:element>
      <xs:element name="display" type="DisplayFactType" minOccurs="0"/>
      <xs:element name="connection" type="ConnectionType" minOccurs="0"/>
   </xs:sequence>
</xs:complexType>
<xs:complexType name="PositionFactType">
   <xs:annotation>
      <xs:documentation>simple fact</xs:documentation>
   </xs:annotation>
   <xs:sequence>
      <xs:element name="positionDeviceType" type="xs:string" minOccurs="0">
         <xs:annotation>
            <xs:documentation>i.e. gps, cricket, ...</xs:documentation>
         </xs:annotation>
      </xs:element>
      <xs:element name="minPrecision" type="xs:double" minOccurs="0">
         <xs:annotation>
            <xs:documentation>precision in meters</xs:documentation>
         </xs:annotation>
      </xs:element>
      <xs:element name="maxPrecision" type="xs:double" minOccurs="0">
         <xs:annotation>
            <xs:documentation>precision in meters</xs:documentation>
         </xs:annotation>
      </xs:element>
      <xs:element name="currentPosition" type="PositionType" minOccurs="0"/>
      <xs:element name="minCurrentSpeed" type="xs:double" minOccurs="0">
         <xs:annotation>
            <xs:documentation>in meter/sec</xs:documentation>
```

```
         </xs:annotation>
      </xs:element>
      <xs:element name="maxCurrentSpeed" type="xs:double" minOccurs="0">
         <xs:annotation>
            <xs:documentation>in meter/sec</xs:documentation>
         </xs:annotation>
      </xs:element>
      <xs:element name="minCurrentDirection" type="xs:string" minOccurs="0">
         <xs:annotation>
            <xs:documentation>i.e. N, E, S, W, NW, ...</xs:documentation>
         </xs:annotation>
      </xs:element>
      <xs:element name="maxCurrentDirection" type="xs:string" minOccurs="0">
         <xs:annotation>
            <xs:documentation>i.e. N, E, S, W, NW, ...</xs:documentation>
         </xs:annotation>
      </xs:element>
   </xs:sequence>
</xs:complexType>
<xs:complexType name="TimeFactType">
   <xs:annotation>
      <xs:documentation>simple fact</xs:documentation>
   </xs:annotation>
   <xs:sequence>
      <xs:element name="startRange" type="xs:string" minOccurs="0">
         <xs:annotation>
            <xs:documentation>date + time</xs:documentation>
         </xs:annotation>
      </xs:element>
      <xs:element name="endRange" type="xs:string" minOccurs="0">
         <xs:annotation>
            <xs:documentation>date + time</xs:documentation>
         </xs:annotation>
      </xs:element>
   </xs:sequence>
</xs:complexType>
<xs:complexType name="UserProfileFactType">
   <xs:annotation>
      <xs:documentation>simple fact</xs:documentation>
   </xs:annotation>
   <xs:sequence>
      <xs:element name="userName" type="xs:string" minOccurs="0"/>
      <xs:element name="userMobilityPreset" minOccurs="0">
         <xs:annotation>
            <xs:documentation>i.e. walk, bicycle, car</xs:documentation>
         </xs:annotation>
      </xs:element>
      <xs:element name="userLocation" type="UserLocationType" minOccurs="0"/>
      <xs:element name="userInterest" type="UserInterestType" minOccurs="0"/>
   </xs:sequence>
</xs:complexType>
<xs:complexType name="PositionType">
   <xs:annotation>
      <xs:documentation>position</xs:documentation>
   </xs:annotation>
   <xs:sequence>
      <xs:element name="radius" type="xs:double" minOccurs="0"/>
      <xs:element name="xPosition" type="xs:double" minOccurs="0"/>
      <xs:element name="yPosition" type="xs:double" minOccurs="0"/>
      <xs:element name="zPosition" type="xs:double" minOccurs="0"/>
   </xs:sequence>
</xs:complexType>
<xs:complexType name="DisplayFactType">
   <xs:annotation>
      <xs:documentation>display</xs:documentation>
   </xs:annotation>
   <xs:sequence>
      <xs:element name="minXResolution" type="xs:int" minOccurs="0"/>
      <xs:element name="maxXResolution" type="xs:int" minOccurs="0"/>
      <xs:element name="minYResolution" type="xs:int" minOccurs="0"/>
      <xs:element name="maxYResolution" type="xs:int" minOccurs="0"/>
      <xs:element name="minColors" type="xs:int" minOccurs="0"/>
      <xs:element name="maxColors" type="xs:int" minOccurs="0"/>
   </xs:sequence>
</xs:complexType>
<xs:complexType name="ConnectionType">
   <xs:annotation>
      <xs:documentation>connection</xs:documentation>
   </xs:annotation>
   <xs:sequence>
      <xs:element name="connectionType" type="xs:string" minOccurs="0">
```

```
            <xs:annotation>
               <xs:documentation>i.e. wlan, umts, bluetooth, gprs, infrared, ...</xs:documentation>
            </xs:annotation>
         </xs:element>
         <xs:element name="minMaxSpeed" type="xs:double" minOccurs="0">
            <xs:annotation>
               <xs:documentation>in kbit/sec</xs:documentation>
            </xs:annotation>
         </xs:element>
         <xs:element name="maxMaxSpeed" type="xs:double" minOccurs="0">
            <xs:annotation>
               <xs:documentation>in kbit/sec</xs:documentation>
            </xs:annotation>
         </xs:element>
         <xs:element name="minCurrentSpeed" type="xs:double" minOccurs="0">
            <xs:annotation>
               <xs:documentation>in kbit/sec</xs:documentation>
            </xs:annotation>
         </xs:element>
         <xs:element name="maxCurrentSpeed" type="xs:double" minOccurs="0">
            <xs:annotation>
               <xs:documentation>in kbit/sec</xs:documentation>
            </xs:annotation>
         </xs:element>
         <xs:element name="minCurrentQuality" type="xs:int" minOccurs="0">
            <xs:annotation>
               <xs:documentation>from 1=low to 10=high</xs:documentation>
            </xs:annotation>
         </xs:element>
         <xs:element name="maxCurrentQuality" type="xs:int" minOccurs="0">
            <xs:annotation>
               <xs:documentation>from 1=low to 10=high</xs:documentation>
            </xs:annotation>
         </xs:element>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="UserLocationType">
      <xs:annotation>
         <xs:documentation>user location</xs:documentation>
      </xs:annotation>
      <xs:sequence>
         <xs:element name="description" type="xs:string" minOccurs="0">
            <xs:annotation>
               <xs:documentation>i.e. home, office, ...</xs:documentation>
            </xs:annotation>
         </xs:element>
         <xs:element name="location" type="PositionType" minOccurs="0"/>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="UserInterestType">
      <xs:annotation>
         <xs:documentation>user interest</xs:documentation>
      </xs:annotation>
      <xs:sequence>
         <xs:element name="type" type="xs:string" minOccurs="0">
            <xs:annotation>
               <xs:documentation>i.e. restaurant, entertainment, ...</xs:documentation>
            </xs:annotation>
         </xs:element>
         <xs:element name="description" type="xs:string" minOccurs="0">
            <xs:annotation>
               <xs:documentation>i.e. modern art, cars, ...</xs:documentation>
            </xs:annotation>
         </xs:element>
         <xs:element name="minIntensity" type="xs:int" minOccurs="0">
            <xs:annotation>
               <xs:documentation>from 1=low interest to 10=high interest</xs:documentation>
            </xs:annotation>
         </xs:element>
         <xs:element name="maxIntensity" type="xs:int" minOccurs="0">
            <xs:annotation>
               <xs:documentation>from 1=low interest to 10=high interest</xs:documentation>
            </xs:annotation>
         </xs:element>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="ComplexUserContextType">
      <xs:annotation>
         <xs:documentation>complex</xs:documentation>
      </xs:annotation>
      <xs:choice>
```

```
         <xs:element name="currentUserMovementStatus" type="xs:string" minOccurs="0">
            <xs:annotation>
               <xs:documentation>i.e. walking, running, driving, ...</xs:documentation>
            </xs:annotation>
         </xs:element>
         <xs:element name="currentUserActivityStatus" type="xs:string" minOccurs="0">
            <xs:annotation>
               <xs:documentation>i.e. busy, idle, ... </xs:documentation>
            </xs:annotation>
         </xs:element>
         <xs:element name="currentUserSpareTime" type="xs:int" minOccurs="0">
            <xs:annotation>
               <xs:documentation>in minutes</xs:documentation>
            </xs:annotation>
         </xs:element>
         <xs:element name="currentUserLocation" type="xs:string" minOccurs="0">
            <xs:annotation>
               <xs:documentation>i.e. home, office, unknown, ...</xs:documentation>
            </xs:annotation>
         </xs:element>
      </xs:choice>
   </xs:complexType>
   <xs:complexType name="ComplexEnvironmentContextType">
      <xs:annotation>
         <xs:documentation>complex</xs:documentation>
      </xs:annotation>
      <xs:choice>
         <xs:element name="currentWeatherCondition" type="xs:string" minOccurs="0">
            <xs:annotation>
               <xs:documentation>i.e. hot, cold, rainy, ...</xs:documentation>
            </xs:annotation>
         </xs:element>
      </xs:choice>
   </xs:complexType>
   <xs:complexType name="ComplexDeviceContextType">
      <xs:annotation>
         <xs:documentation>complex</xs:documentation>
      </xs:annotation>
      <xs:choice>
         <xs:element name="currentConnectionSpeed" type="xs:string" minOccurs="0">
            <xs:annotation>
               <xs:documentation>i.e. slow, fast, ...</xs:documentation>
            </xs:annotation>
         </xs:element>
         <xs:element name="currentConnectionQuality" type="xs:string" minOccurs="0">
            <xs:annotation>
               <xs:documentation>i.e. low, high, ...</xs:documentation>
            </xs:annotation>
         </xs:element>
      </xs:choice>
   </xs:complexType>
</xs:schema>
```

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung des Masterstudiengangs Informatik an der Hochschule für Angewandte Wissenschaften Hamburg vom 22. November 2001, geändert am 7. Dezember 2004, nach §22(4) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, den 09.04.2008

Maik Weindorf