Entwicklung von Schnittstellen zur Konfiguration und Verwaltung eines Ambient Intelligence Systems

Diplomarbeit
im
Studiengang Medieninformatik
der
Fachhochschule Stuttgart –
Hochschule der Medien

Markus Heimann

Erstprüfer:	Prof. Johannes Maucher
Zweitprüfer:	Prof. Martin Goik

Bearbeitungszeitraum: 01. Juni 2007 bis 31. Oktober 2007

Herrenberg, Oktober 2007

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Diplomarbeit selbständig angefertigt habe. Es wurden nur die, in der Arbeit ausdrücklich benannten Quellen und Hilfsmittel benutzt. Wörtlich oder sinngemäß übernommenes Gedankengut habe ich als solches kenntlich gemacht.

Ort, Datum

Unterschrift

Inhaltsverzeichnis

Vorwort

Die hier vorgestellte Diplomarbeit wurde im Rahmen des Forschungsprojektes „Ambi-Comp“ an der Hochschule der Medien Stuttgart erstellt.

Herzlich danken möchte ich Herrn Professor Johannes Maucher, für die Sehr Gute Betreuung während der Diplomarbeit, sowie Thomas Suchy und Jürgen Butz für Ihre Hilfe und die vielen produktiven Gespräche.

Ich freue mich, dass ich als Teil des Teams an dem Projekt AmbiComp mitwirken und einen Beitrag dazu leisten konnte.

Überblick

Kapitel 1 bietet einen Einblick in das Gebiet der Ambient Intelligence und das Forschungsprojekt AmbiComp, in dessen Umfeld die Diplomarbeit angesiedelt ist. **Kapitel 2** stellt die Ziele der Diplomarbeit vor. In **Kapitel 3** wird die Systemumgebung beschrieben, in der die Umsetzung der Diplomarbeitsziele erfolgen wird. **Kapitel 4** definiert die Rahmenbedingungen für die Umsetzung und die Anforderungen an die Umsetzung. **Kapitel 5** und **Kapitel 6** dienen der Konzepterstellung für die Schnittstellen und Datenformate. Diese Konzepte werden im Rahmen von **Kapitel 7** in eine reale Applikation eingebunden und damit auf ihre Umsetzbarkeit getestet. In **Kapitel 8** wird auf die Diplomarbeit resümierend zurück geblickt, um zu sehen, ob die gesetzten Ziele erreicht wurden und darüber gesprochen, was für die Zukunft noch zu tun bleibt.

Abstract

Ein Ambient Intelligence System kann Anwendern vielfältige Möglichkeiten zur Steuerung und Konfiguration des Systems geben. Der Zugriff auf diese Funktionalität kann auf verschiedene Arten realisiert werden. Eine ist die Nutzung eines mobilen Gerätes als Zugangsmedium. Damit dies möglich wird, sind Konzepte und Schnittstellen notwendig, die den Informationsaustausch zwischen Anwendern, Gerät und Ambient Intelligence System ermöglichen. Diese Diplomarbeit beschäftigt sich mit der Erarbeitung dieser Grundlagen.

1 Über Ambient Intelligence und AmbiComp

Ambient Intelligence (AmI) ist ein Begriff, der heutzutage immer öfter genannt wird als eine der Entwicklungen der Zukunft. Zahlreiche Firmen und Forschungsinstitute, darunter auch das Forschungsprogramm Information Society Technologies der EU, beschäftigen sich mit diesem Gebiet. Diese Diplomarbeit ist Teil des Forschungsprojektes AmbiComp, welches sich der Schaffung von Grundlagen für AmI Systeme verschrieben hat. In diesem Kapitel werden die grundlegenden Ziele von Ambient Intelligence und in diesem Zusammenhang des Projektes AmbiComp erläutert.

1.1 Ambient Intelligence

1991 verfasste Mark Weiser ein Dokument mit dem Titel: **M.Weiser** (1991) - The Computer for the 21st century. In diesem Dokument erzählte er von einer Vision, die er und seine Kollegen haben. Einer Welt, in der überall Computer sind, ohne dass man sich dessen wirklich bewusst ist oder bewusst sein muss. Mark Weiser prägte damit den Begriff des Ubiquitous Computing (dt. Rechnerallgegenwart). In seiner Vision wird sich die Art, wie Computer verwendet werden, grundlegend ändern. Um Aufgaben zu erledigen, benötigt man dann keine Desktop Computer oder Notebooks als zentrales Element des Zugangs. Eine Vielzahl im Hintergrund agierender Computer wird diese Aufgaben übernehmen. Gleichzeitig wird die Bedienung dieser Computer nicht mit der heutigen Nutzung von komplexen, Aufmerksamkeit fordernden Interfaces vergleichbar sein. Die Bedienung wird rein intuitiv geschehen, so wie Menschen z.B. ihre Umwelt wahrnehmen, ohne dass diese sich aktiv darauf konzentrieren müssen. Mark Weiser erschuf mit seiner Vision ein neues Feld der Forschung, in dem heute Wissenschaftler und Firmen auf der ganzen Welt tätig sind.

Eine Weiterentwicklung des Ubiquitous Computing ist das Gebiet der Ambient Intelligence (AmI) (dt. Umgebungsintelligenz). Es greift die Ansätze von Mark Weiser auf, legt den Fokus aber darauf, Gegenstände der Umwelt nicht nur zu vernetzen, sondern sie mit einer gewissen Intelligenz auszustatten. Fuhrmann (2006, S.1) beschreibt AmI als:

> *„eine Vision, die davon ausgeht, dass eine Vielzahl von eingebetteten Systemen unserer Lebens- und Arbeitsumgebung so mit Rechen- und Kommunikationsleistung ausstattet sind, dass diese eingebetteten Systeme zusammen mit ihren angeschlossenen Sensoren und Akttoren gemeinsam eine bestimmte Funktion erfüllen können.*

Dabei tritt der einzelne Rechner in den Hintergrund, so dass die Umgebung als scheinbar intelligent wahrgenommen wird.“

AmI könnte in einer Vielzahl von Bereichen unseres täglichen Lebens aktiv sein. Doch bevor es soweit ist, müssen erst die Grundlagen dazu geschaffen werden.

1.2 Das AmbiComp Forschungsprojekt

Viele Firmen haben bereits das Potential erkannt, dass in AmI liegt und mit der Forschung und Realisierung erster Projekte begonnen. Doch haben all diese Projekte und Produkte noch ein großes Manko. Sie werden in der Regel von einzelnen großen Firmen entwickelt und als Gesamtsystem konstruiert und vermarktet. Zusätzlich haben die Firmen selbstverständlich ein fundamentales Interesse daran, ihre Entwicklungen zu schützen und es Drittanbietern nicht zu ermöglichen, eigene Produkte in ihre Lösungen zu integrieren. Ein echtes Ambient Intelligence System sollte aber kein konstruiertes System sein, sondern durch das Zusammenwirken zwischen seinen Komponenten entstehen.

Das Forschungsprojekt AmbiComp verfolgt diesen Ansatz und hat sich darüber hinaus noch weitere Ziele gesteckt, die zu einem marktfähigen Ambient Intelligence System führen sollen:

1. Die Entwicklung von Werkzeugen für die Softwareentwicklung.

2. Die Entwicklung der Grundlagen für das Ambient Intelligence System, in dem die entwickelte Software später laufen soll. Dazu gehören z.B. die Ausführungsumgebung oder die Anwendungsschnittstellen.

3. Die Entwicklung der Methodik für die Software Entwicklung.

Die Entwicklungsumgebung und die Methoden sollen vor allem kleinen und mittelständischen Unternehmen, aber auch freien Entwicklern auf Basis einer Open Source Lizenz zur Verfügung gestellt werden. Zusammen mit den entwickelten Kernkomponenten, welche in beliebige Geräte integriert werden können, sind diese somit in der Lage eigene, mit Produkten anderer Hersteller kompatible Ambient Intelligence Geräte und Programme zu entwickeln und zu vermarkten.

2 Zielsetzung der Diplomarbeit

Diese Diplomarbeit ist Teil des Forschungsprojektes AmbiComp und hilft dabei, zu dessen Realisierung beizutragen. Die genauen Ziele und Vorstellungen des gesamten Projektes sind in dem Dokument **Fuhrmann, T.** (2006): „Vorhabensbeschreibung zum Projektantrag AmbiComp“ zusammengefasst.

Das Forschungsprojekt ist so umfangreich, so dass zu dessen Realisierung alle zu bewältigenden Aufgaben in Arbeitspakete aufgeteilt wurden. Diese wurden entsprechend den Kenntnissen und Erfahrungen der Projektteilnehmer verteilt.

Aufgabe dieser Diplomarbeit ist das Arbeitspaket 5.1 PDA-/Handyschnittstellen praktisch zu realisieren. In Rücksprache mit den Projektverantwortlichen wurde, aufbauend auf das Arbeitspaket, folgende Zielsetzung formuliert:

Ziel der Diplomarbeit ist die Entwicklung von Benutzerschnittstellen und Formaten, die es den verschiedenen Benutzergruppen des AmbiComp-Systems ermöglicht, mittels Geräten wie Personal Digital Assistants (PDA) oder Mobiltelefonen[1], mit einem Ambient Intelligence System in Interaktion zu treten und dort Verwaltungsaufgaben zu erledigen. Die Ergebnisse sind im Rahmen einer Beispielanwendung praktisch umzusetzen.

Die Entwicklung von konkreten Verwaltungsapplikationen gehört dabei nicht zu den Aufgaben. Sehr wohl können sich aber Anforderungen an solche Applikationen ergeben.

Die Erstellung der Konzepte sollten die unterschiedlichen Rahmenbedingungen der mobilen Endgeräte, wie Ressourcen (z.B. Rechenleistung, Displaygrößen), vorhandene Ein- / Ausgabemöglichkeiten und Netzwerkverbindungen berücksichtigen.

Für weiterführende Informationen über AmbiComp sei an dieser Stelle auf den offiziellen Webauftritt des Projektes unter www.AmbiComp.de verwiesen.

[1] PDAs und Mobiletelefone werden im folgenden unter „Mobile Geräte“ zusammengefasst

3 Die Systemumgebung

Unter Systemumgebung wird dass AmbiComp-System und seine Anwender verstanden. Zunächst wird erklärt, was ein AmbiComp-System ist und wer mit diesem System arbeitet. Da sich das System zum Zeitpunkt der Erstellung dieser Diplomarbeit noch in der Entwicklung befindet, beruhen die aufgeführten Informationen auf ersten Ergebnissen des Projektes und der Vorhabensbeschreibung für das fertige System.

3.1 Das AmbiComp-System

Gemäß **Wikipedia 1** (12.10.2007) bezeichnet ein System

> *„ein Gebilde, dessen wesentliche Elemente (Teile) so aufeinander bezogen sind und in einer Weise wechselwirken, dass sie (aus einer übergeordneten Sicht heraus) als aufgaben-, sinn- oder zweckgebundene Einheit (d.h. als Ganzes) angesehen werden (können) und sich in dieser Hinsicht gegenüber der sie umgebenden Umwelt auch abgrenzen.“*

Dies trifft auch auf das AmbiComp-System zu. Es besteht in der Regel aus einer Vielzahl einzelner Elemente, die sich zu einem System zusammenschließen. Im AmbiComp Umfeld nennt man diese Elemente AmbiComp-Geräte und AmbiComp-Anwendungen.

3.1.1 AmbiComp Geräte

Folgende Grafik zeigt den Aufbau eines AmbiComp Gerätes:

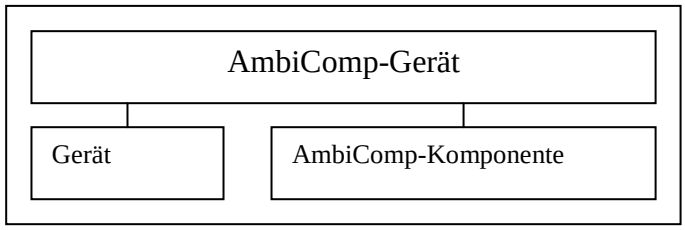

Abbildung 3-1: AmbiComp-Gerät

Ein AmbiComp-Gerät setzt sich aus einem Gerät, z.B. einem Sensor und einer AmbiComp-Komponente zusammen. Das Gerät ist für sich voll funktionstüchtig. Durch die Kombination mit einer AmbiComp-Komponente können die Fähigkeiten des Gerätes jedoch zusätzlich in einem AmbiComp-System genutzt werden.

Wie dies möglich ist, wird bei Betrachtung der AmbiComp-Komponente deutlich. Die AmbiComp Komponente besteht aus einer Ambient Intelligence Control Unit (kurz: AICU) und bestimmten APIs.

API

Die APIs ermöglichen den Zugriff auf die nativen Funktionen zur Ansteuerung der Komponenten-Hardware. Handelt es sich bei dem Gerät beispielsweise um einen Temperatursensor, so könnte über die APIs die aktuelle vom Sensor gemessene Temperatur abgerufen werden.

AICU

Eine AICU ist laut **Suchy, T.** (2007; S. 6):

> *„ein eingebettetes System mit Mikroprozessor, Speicher und Schnittstellen nach außen. Das Software-Kernsystem beinhaltet eine Java Virtual Machine die mittels einer Implementierung des in Karlsruhe entwickelten SSR-Routingverfahrens mit anderen AICUs kommunizieren kann.“*

Die AICUs sind die entscheidenden Teile des AmbiComp-Systems. Sie bauen das System auf, verbinden alle seine Teile miteinander und ermöglichen den Austausch von Informationen. Außerdem stellen sie den Speicher und die Ausführungsumgebung für APIs und AmbiComp-Anwendungen zur Verfügung.

Dazu bieten sie ein Interface, um auf Nachbar-AICUs zuzugreifen `AICU.getNachbarAICUid()`und um die Produktnummer der AICU zu ermitteln `AICU.getAICUProduktID()`. Damit ist es möglich auf jede AICU im System zuzugreifen und über die Produktnummer herauszufinden, in was für ein AmbiComp Gerät die AICU eingebaut ist.

3.1.2 AmbiComp-Anwendungen

Die AICUs ermöglichen es, AmbiComp-Geräte miteinander zu vernetzen. Es fehlt jedoch noch der Nutzen aus dieser Vernetzung. Diesen Nutzen schaffen die AmbiComp Anwendungen.

3.1.2.1 Was sind AmbiComp-Anwendungen

Eine Anwendung ist laut **Wikipedia 3** (17.09.2007):

> *„ein Computerprogramm, das eine für den Anwender nützliche Funktion ausführt.“*

Nützliche Funktionen gibt es viele. Eine Wichtige ist im AmbiComp-System die Kombination von AICU Fähigkeiten.

Beispiel

Betrachtet wird ein AmbiComp-System bestehend aus zwei AmbiComp-Geräten. Eines stellt einen Temperatursensor dar, das andere eine Sirene. Die Fähigkeit des Temperatursensors besteht darin die aktuelle Temperatur abzufragen. Die Sirene kann auf Kommando einen bestimmten Ton von sich geben. Durch ihre AICUs sind diese Geräte bereits miteinander verbunden, haben sonst bisher aber keine gemeinsame Funktion. Diese Funktion kann eine dafür entwickelte AmbiComp-Anwendung herstellen. Sie könnte im vorliegenden Beispiel die Fähigkeiten der beiden AICUs kombinieren, um ein einfaches Feuermeldesystem zu bilden. Über den Temperatursensor kann ein Anstieg der Temperatur festgestellt und aufgrund dessen über die Sirene einen Warnton ausgegeben werden.

Dazu könnte die Anwendung die Möglichkeit bieten, Einstellungen dafür vorzunehmen, wann und wie Feueralarm ausgelöst wird. Beispielweise könnte die Temperatur, ab der Alarm ausgelöst wird, auf 140 Grad festegelegt werden und die Sirene würde bei Feuer abwechselnd drei kurze und drei lange Töne von sich geben.

Neben diesem offensichtlichen Nutzen von Anwendungen können diese auch für eine Vielzahl weiterer Aufgaben eingesetzt werden. Beispiele wären die Systemüberwachung oder Sicherheitsfunktionen.

3.1.2.2 Wie werden AmbiComp Anwendungen ausgeführt

AmbiComp Anwendungen setzen auf den AICUs des AmbiComp-Systems auf.

Eine AICU stellt eine Schnittstelle bereit, die den Zugriff auf die im Speicher vorhandenen Anwendungen ermöglicht. Über den Aufruf `getAICUAnwendungen()` erhält man eine Liste aller auf der AICU befindlichen Anwendungen, deren Aufrufadressen und deren Status (Installiert/Nicht Installiert, Ausgeführt/Nicht Ausgeführt).

Über diese Aufrufadressen kann direkt mit den Anwendungen gearbeitet werden. Dies ermöglicht das Starten und Beenden und die Installation bzw. Deinstallation. Außerdem kann über `Aufrufadresse.getGUIBeschreibung()`eine Beschreibung der graphischen Bedienoberfläche einer Anwendung abgerufen werden, mittels derer diese bedient werden kann.

3.2 Anwender und Rollen im AmbiComp Umfeld

Unter Anwendern versteht man im Allgemeinen reelle Personen, Firmen oder Organisationen. Mehrere Anwender allgemeingültig zu beschreiben, ist in der Regel sehr schwierig, da sie eine Vielzahl von Eigenschaften und Fähigkeiten haben, die sie voneinander unterscheiden. Daher wird im Rahmen der Diplomarbeit das Konzept der Rolle benutzt. Eine Rolle ist für alle, die sie einnehmen, gleich. Dies ermöglicht es, unterschiedliche Anwender gleich zu behandeln.

Wann immer Anwender mit einem AmbiComp-System arbeiten wollen, werden sie dies in Ausübung einer bestimmten Rolle machen. Die Handlungsmöglichkeiten der Anwender werden durch die möglichen Rollen vorgegeben.

Die Rollen sind:

Der Hardware-Entwickler

- entwickelt und testet die Hardware und Gerätefunktionen für Ambient Intelligence Control Units.

Der Software-Entwickler

- entwickelt und testet Software für AmbiComp-Systeme.

Der Systemintegrator

- plant, installiert, konfiguriert und wartet AmbiComp-Systeme.

Der AmbiComp-Endanwender

- beauftragt Systemintegrator, um z.B. ein AmbiComp-System zu installieren.
- ist Nutzer eines fertig eingerichteten AmbiComp-Systems.

Alle Rollen treten aus unterschiedlichen Gründen und aus unterschiedlichen Perspektiven mit einem AmbiComp-System in Kontakt, um dort Aufgaben zu erledigen. Sie lassen sich aufteilen in solche, die die AmbiComp Geräte und Applikationen entwickeln und umsetzen, und jene, die sie einsetzen.

Deutlicher wird diese Aufteilung durch die Ergebnisse einer Anwendungsfallanalyse, die für alle Rollen durchgeführt wurde.

Die Ergebnisse zeigen, dass die Rollen Hardwareentwickler und Softwareentwickler vor allem mit einzelnen AICUs und deren Grundfunktionen arbeiten. Der Hardwareentwickler deswegen, weil er diese AICUs baut und die zugehörigen Grundfunktionen implementiert und der Softwareentwickler, weil seine AmbiComp-Anwendungen auf diese Grundfunktionen aufsetzen.

Der Systemintegrator nutzt die Ergebnisse der Entwicklerrollen, um daraus auf Endanwender zugeschnittene Systeme zu erstellen. Um diese Aufgabe zu erfüllen, benötigt er keine Kenntnisse über den Aufbau und die Grundfunktionalität einzelner AICUs. Er orientiert sich an den Anforderungen der AmbiComp-Anwendungen, um zu erkennen, welche AICUs, auf welche Weise, in das System integriert werden müssen. Dazu nutzt er die Möglichkeiten der vorhandenen AmbiComp-Anwendungen, um das System zu konfigurieren. Mit diesem Wissen plant und installiert der Systemintegrator AmbiComp-Systeme für Endanwender.

Der Endanwender schließlich ist der Nutzer eines fertigen AmbiComp-System. Ihm ist bekannt, was das AmbiComp-System für ihn leisten kann und er nimmt diese Leistungen nach Bedarf in Anspruch. Er benötigt dafür kein Wissen über die interne Funktionsweise des Systems.

Fazit

Das Thema der Diplomarbeit ist die „*Entwicklung von Schnittstellen zur Konfiguration und Verwaltung eines Ambient Intelligence Systems*“. Durch die Ergebnisse der Anforderungsanalyse ist bekannt geworden, dass nur die Rolle Systemintegrator solche Aufgaben in einem AmbiComp-System wahrnimmt. Die anderen Rollen schaffen entweder die notwendigen Grundlagen (Geräte, Programme) oder sind Nutzer des fertigen Systems. Somit ist es Ziel der Diplomarbeit, die Grundlagen zu schaffen, damit ein Systemintegrator seine Aufgaben erfüllen kann.

4 Analyse der Aufgabenstellung

Durch die Vorstellung der Ziele der Diplomarbeit wurde festgelegt, welches Ergebnis erwartet wird und durch das Kapitel „Systemumgebung“, in welchem Umfeld die Umsetzung stattfinden wird. Nun müssen die Rahmenbedingungen für die Umsetzung, sowie die zu erfüllenden Anforderungen, festgelegt werden.

Begonnen wird mit der Festlegung der Rahmenbedingungen. Diese definieren Grenzen für die Anforderungen, aber auch für spätere Schritte, wie Konzeption, Design und Ausführung fest.

Im Anschluss daran wird ein Anforderungskatalog erstellt. Dieser Katalog wird am Ende der Diplomarbeit als Kriterium herangezogen werden, um zu sehen, ob die Ziele erreicht wurden.

Als Basis für die Rahmenbedingungen und die Anforderungserstellung dienen hierbei die Zieldefinition der Diplomarbeit und die damit zusammenhängenden Dokumente.

4.1 Rahmenbedingungen

Es ist notwendig, einen festen Rahmen zu definieren, innerhalb dessen die Diplomarbeit umgesetzt wird, um Unklarheiten während der Bearbeitung zu vermeiden.

Die Beschreibung jeder Rahmenbedingung erfolgt nach einem festgelegten Schema:

1. Dem Titel der Rahmenbedingung
2. Dem Ursprung der Rahmenbedingung

   Die Grundlage aufgrund derer die Rahmenbedingung festgelegt wurde.
3. Einer genauen Beschreibung der Rahmenbedingung

   Die Beschreibung legt eindeutig fest, um was es sich bei der Rahmenbedingung genau handelt.

Die Rahmenbedingungen sind fest und werden während der Umsetzung der Diplomarbeit nicht mehr verändert.

Entwicklungssprache

Ursprung:

„Vorhabensbeschreibung zum Projektantrag AmbiComp“ (**Fuhrmann, T.** (2006; S.2)):

> *„Insbesondere sollen die Arbeiten auf die Programmiersprache Java ausgerichtet[…]werden“*

Beschreibung:

Eines der Ziele von AmbiComp ist es, eine möglichst große Akzeptanz zu erreichen. Die Verwendung der Programmiersprache Java im gesamten Projekt ist eine wichtige Grundlage dafür. Die Sprache ist allgemein bekannt und universell einsetzbar. Dadurch senken sich die Hürden für Einsteiger.

Zielplattform(en)

Ursprung:

„Vorhabensbeschreibung zum Projektantrag AmbiComp“

(**Fuhrmann, T.** (2006; S.27-28)) :

> *„PDA/Handy Schnittstellen“*

Beschreibung:

Unter Plattform versteht man nach der Definition der **Wikipedia** (27.09.2007):

> *„[…]im Zusammenhang mit Computern ein System, auf dem ein Computerprogramm (Software) ausgeführt wird. Meist ist damit eine Kombination von Betriebssystem und der Hardware gemeint“*.

Im Rahmen dieser Diplomarbeit sind die Zielplattformen alle mobilen Geräte. Durch die Festsetzung von Java als Programmiersprache werden die möglichen Zielplattformen aber dennoch eingeschränkt. Obwohl Java als plattformunabhängig und universell einsetzbar gilt, ergibt sich in der Praxis ein anderes Bild. Nicht alle Geräte der Zielgruppe unterstützen Java. Und selbst wenn es eine Java Unterstützung gibt, so unterscheidet sich diese von Hersteller zu Hersteller und von Gerät zu Gerät. Es existiert natürlich die Möglichkeit, die Java-Fähigkeit nachträglich zu installieren, doch funktioniert dies nicht immer problemlos.

Zusammengefasst bedeutet dies, dass die Zielplattformen für diese Diplomarbeit nicht die Gruppe aller mobilen Geräte, sondern die Gruppe aller mobilen Geräte mit Java-Fähigkeit sind. Dazu muss die Java Installation bestimmten Standards genügen. So muss das Gerät das mobile Information Device Profile (MIDP) 2.0 der Java 2 mobile Edition (J2ME) vollständig unterstützten.

4.2 Anforderungsdefinition

Aufgabe der Anforderungsdefinition ist die genaue Beschreibung jeder Anforderung, die zu erfüllen ist, damit das Gesamtziel der Diplomarbeit als erreicht gilt. Dabei wird zwischen funktionalen und nichtfunktionalen Anforderungen unterschieden. Funktionale Anforderungen sind dabei alle jene, die beschreiben, was das Produkt später erfüllen muss. Nichtfunktionale Anforderungen beschreiben Eigenschaften des Produktes.

Beispiel

Angenommen es geht um ein Produkt, das Videos abspielen soll. Dann wäre eine funktionale Anforderung, dass die Applikation .mpg und .avi Videos abspielen kann. Eine nichtfunktionale Funktion wäre es, das die Anwendung nicht 3 Stunden benötigt, um ein Video zu laden, sondern dies, je nach Videolänge, in angemessener Zeit schafft.

Eine Anforderungsdefinition setzt sich, genau wie bei den Rahmenbedingungen, aus den folgenden Teilen zusammen:

1. Dem Titel der Anforderung
2. Dem Ursprung der Anforderung (entfällt bei nichtfunktionalen Anforderungen)
3. Einer genauen Beschreibung der Anforderung

4.2.1 Funktionale Anforderungen

Benutzerschnittstelle für die Erledigung von Verwaltungsaufgaben

Ursprung:

„Vorhabensbeschreibung zum Projektantrag AmbiComp“

(**Fuhrmann, T.** (2006; S.27-28)) :

„PDA/Handy Schnittstellen“

Beschreibung:

Ein AmbiComp-System besteht aus Ambient Intelligence Control Units (AICU). Die AICUs können theoretisch alles Mögliche darstellen, einfache Sensoren oder Aktoren, Küchengeräte, Computer, etc.

Alle AICUs können innerhalb des Systems miteinander kommunizieren. Es ist gewünscht, dass man mit mobilen Geräten Zugriff auf die Verwaltungsfunktionen des Systems erhält. Das mobile Gerät selbst ist aber nicht Teil des AmbiComp-Systems. Es muss also ein Weg gefunden werden, mittels dem externe Geräte auf die interne Funktionalität des Systems zugreifen können. Dieser Weg muss Anwendern des AmbiComp-Systems zugänglich sein.

Entwicklung eines Prototyps

Ursprung:

Diplomarbeit: Entwicklung von Schnittstellen zur Konfiguration und Steuerung eines Ambient Intelligence Systems, Kapitel 1 - Ziele

Beschreibung:

Es sollen nicht nur theoretische Konzepte entwickelt werden, wie Benutzer auf die Verwaltungsfunktionen eines AmbiComp-Systems zugreifen können, sondern diese Konzepte sollen im Rahmen einer Beispielanwendung praktisch umgesetzt werden.

4.2.2 Nichtfunktionale Anforderungen

Dokumentation

Beschreibung:

Es ist wichtig, dass alle, ihm Rahmen der Diplomarbeit erstellten Schnittstellen, der Quellcode und die Datenformate dokumentiert werden. Die Dokumentation soll Dritten helfen, die Arbeit besser zu verstehen und nachvollziehen zu können.

Reaktionszeiten

Beschreibung:

Es ist zu erwarten, dass bestimmte Anfragen an das AmbiComp-System mehr Zeit zur Ausführung benötigen als andere. Die Anwendung sollte dem Anwender daher unvermeidbare Wartezeiten, z.B. bei der Netzwerkkommunikation, anzeigen.

Speicherbedarf

Beschreibung:

Aufgrund des eingeschränkten Speichers eines mobilen Gerätes sollten nicht benötigte Ressourcen so bald wie möglich freigegeben werden.

5 Schnittstellenkonzepte

In diesem Kapitel werden die Berührungspunkte zwischen den beteiligten Systemen, die Schnittstellen, betrachtet und Konzepte, die zu deren Umsetzung notwendig sind, erarbeitet.

Wichtig ist laut **Fuhrmann, T** (2006: S.27-28) die:

„… Entwicklung von Benutzerschnittstellen und Formaten, die es den verschiedenen Benutzergruppen des AmbiComp-Systems, ermöglicht, mittels Geräten wie Personal Digital Assistants (PDA) oder Mobiltelefonen, mit einem Ambient Intelligence System in Interaktion zu treten und dort Verwaltungsaufgaben zu erledigen. Die Ergebnisse sind im Rahmen einer Beispielanwendung praktisch umzusetzen.“

Die Aufgabenstellung und in diesem Zusammenhang auch die Systemumgebung lässt bereits die Punkte erkennen, an denen Schnittstellen notwendig werden. Genauer gesagt geht es darum, **Nutzern** mittels **mobilen Geräten** Zugang zu den Verwaltungsfunktionen eines **AmbiComp-System**s zu ermöglichen.

Daraus ergeben sich zwei Schnittstellen. Eine zwischen einem Anwender und einem mobilen Gerät und eine zwischen einem mobilen Gerät und einem AmbiComp-System. Durch diese Schnittstellen wird der Informationsaustausch zwischen den Systemen erst ermöglicht. Im folgenden Abschnitt werden diese Schnittstellen beschrieben und ausgearbeitet.

5.1 Die Mensch-Gerät-Schnittstelle (MGS)

Die MGS liegt zwischen Anwender und mobilem Gerät. Sie ermöglicht einem Anwender den Zugang zu den Methoden des Gerätes. Die MGS versteckt die darunter liegenden Schichten vor dem Anwender, so gut dies möglich ist. Ein Benutzer kommuniziert somit ausschließlich über diese Schnittstelle und das Gerät muss sich darum kümmern, dass die Eingaben des Benutzers an das AmbiComp-System weitergeleitet werden. Umgekehrt zeigt die Schnittstelle auch jene Informationen an, die vom AmbiComp-System kommen und für den Anwender bestimmt sind.

Das Mittel, das dem Anwender den Zugang zu dieser Schnittstelle ermöglicht, ist das mobile Gerät. Die erste Frage, die jetzt geklärt wird ist, von welcher Art die Benutzerschnittstelle ist, die das mobile Gerät dem Benutzer anbietet. Es gibt eine Vielzahl von Möglichkeiten. Die Bekanntesten sind Kommandozeilenbasierte (z.B. Unix Shell) und graphische Benutzeroberflächen (engl. Graphical User Interface (GUI)). Da auf einem mobilen Gerät der Einsatz von graphischen Benutzeroberflächen der Normalfall ist, wird dieser Ansatz gewählt.

5.1.1 Eigenschaften der graphischen Benutzerschnittstelle

Ein Anwender hat an die Benutzung einer graphischen Benutzeroberfläche bestimmte Anforderungen. Er erwartet, dass er diese entsprechend den Möglichkeiten des mobilen Gerätes bedienen kann. Hat das mobile Gerät eine Tastatur als Steuerungs- und Eingabemedium, so sollten darüber alle Teile der GUI steuerbar sein. Ist ein berührungssensitives Display vorhanden, gilt das Gleiche.

Ein anderer Aspekt neben der Bedienbarkeit ist das Aussehen der GUI. In der Regel orientiert sich dieses am generellen „Lock and Feel“ des mobilen Gerätes. Dies bringt den Vorteil der einfacheren Wiedererkennbarkeit von Elementen und deren Anordnung. Abweichungen sollte es nur dort geben, wo es absolut notwendig ist oder wo es bewusst gewollt ist. Ein Problem ist, dass das Aussehen der GUI erst zur Laufzeit feststeht. Je nach dargestellter Anwendung kann sich diese ändern. Die GUI muss zur Laufzeit erzeugbar, darstellbar und veränderbar sein. Dazu ist die Ausführung von Aktionen ein Thema. Wenn eine GUI Benutzereingaben entgegennimmt, müssen diese auch verarbeitet werden, so wie der Anwender dies erwartet. Auch hier besteht das Problem, dass die Art der Aktionen erst zur Laufzeit feststehen. Es muss somit eine Möglichkeit gefunden werden, wie zur Laufzeit sowohl die Erzeugung der GUI als auch die Handhabung von Aktionen möglich ist.

5.1.1.1 Darstellung von graphischen Elementen zur Laufzeit

Zwei Ansätze sind an dieser Stelle interessant.

Der erste Ansatz basiert darauf, die graphische Oberfläche in einem bestimmten Format zu beschreiben. Dieses Format wird zur Laufzeit interpretiert und aus der entnommenen Beschreibung wird durch ein Programm eine graphische Oberfläche generiert und angezeigt. HTML (Format) und Webbrowser (Programm) sind ein Beispiel für diese Art der Oberflächenerzeugung.

Der zweite Ansatz beruht darauf, die Beschreibung der graphischen Oberfläche in einer Programmiersprache vorzunehmen. Dann müssen die entsprechenden Quelldateien zur Laufzeit geladen und ausgeführt werden.

Beide Darstellungsansätze haben, die in der folgenden Tabelle aufgeführten, Vor- und Nachteile.

Tabelle 1: Vorteile und Nachteile der Darstellungsansätze

Vorteile des ersten Ansatzes	Vorteile des zweiten Ansatzes
- Unabhängig von bestimmter Programmiersprache oder Zielplattform.	- In Quellcode geschriebene Anweisungen können direkt ausgeführt werden und sind somit schneller.
- Nicht-Programmierer können mit diesem Ansatz recht einfach graphische Oberflächen erstellen.	- Verwendung aller bekannten graphischen Elemente der Programmiersprache möglich.
Nachteile des ersten Ansatzes	Nachteile des zweiten Ansatzes
- Die Beschreibung ist in der Regel ausführlicher als wenn sie direkt im Quellcode einer Programmiersprache geschrieben wir.	- Abhängig von bestimmter Programmiersprache und damit nicht überall ausführbar.
- Es ist ein Parser notwendig, der die Beschreibung in konkreten Code umsetzt.	- Eventuell abhängig vom Vorhandensein bestimmter Bibliotheken.
- Der Umfang der darstellbaren graphischen Elemente ist direkt abhängig vom Umfang der Beschreibungssprache.	

Es wurde entschieden, dem ersten Ansatz zu folgen. Das wichtigste Argument dafür ist die damit erzielbare Plattformunabhängigkeit. Um das Format graphisch anzuzeigen, ist lediglich ein Interpreter notwendig, der in einer beliebigen Programmiersprache verfasst sein kann.

Auswahl des Beschreibungsformats

Das Beschreibungsformat muss mindestens folgenden Anforderungen genügen:

- Beschreibung von graphischen Elementen wie Buttons, Textfeldern oder Listen
- Beschreibung von Aktionen, die zu graphischen Elementen gehören
- Es muss lizenzfrei nutzbar sein.

Eine Recherche im Bereich der Beschreibungssprachen für graphische Oberflächen zeigt, dass es bereits eine Vielzahl von Entwicklungen gibt. Die Bekanntesten setzen dabei auf das Grundgerüst der Extensible Markup Language (XML) auf. Folgend wird nach einer, für AmbiComp geeignete, Beschreibungssprache gesucht.

Für die Umsetzung interessant sind jene Beschreibungssprachen, die auch für mobile Endgeräte geeignet sind. Formate wie XUL der Mozilla Foundation oder Qt von Trolltech sind sehr mächtige Sprachen. Der Webbrowser Firefox basiert beispielsweise auf XUL. Leider hat diese Mächtigkeit hohe Anforderung an Speicher und Ressourcenleistung. Somit sind diese Formate nicht für mobile Geräte geeignet.

Als möglicher Kandidat für mobile Geräte wurde die WML (wireless markup language) identifiziert. WML implementiert ein Teil des XHTML Standards und ist Teil des Wireless Application Protocoll (WAP), das speziell für Darstellung flexibler Inhalte auf mobile Geräte entwickelt worden ist. Bei genauerer Betrachtung fällt aber auf, dass der Einsatz von WAP im Rahmen des Projektes mit sehr viel Aufwand verbunden ist. Um mit WML und WAP zu arbeiten, muss die entsprechende Infrastruktur aufgebaut werden (WAP-Gateway, etc.). Dazu ist WAP für die Kommunikation via GPRS oder ähnlichen Technologien ausgelegt, nicht für den Austausch von Informationen über beispielsweise Bluetooth. Um dies zu umgehen, wäre es möglich, nur den WML-Standard zu verwenden und für diesen einen eigenen Browser, der sich für die Kommunikation mit einem AmbiComp-System eignet, zu entwickeln. Somit würden alle sonstigen Elemente von WAP wegfallen. Hierbei müssten aber auch Elemente des Standards implementiert werden, die für die Anwendung im AmbiComp Umfeld unnötig sind. Außerdem könnten keine eigenen Erweiterungen, die für AmbiComp eventuell notwendig sind, mit einbezogen werden. WML ist somit kein für die Umsetzung geeigneter XML-Dialekt.

Der XML-Dialekt der Firma **Netads** (Juni 2005), der im Zusammenhang mit dem BTKit veröffentlich wurde, ist eine weitere Möglichkeit für die Umsetzung. Die Verwendung dieses Standards wird im Rahmen des Arbeitspaketes 5.1 des AmbiComp Projektes (Fuhrmann, T. (2006; S.27)) als Grundlage empfohlen. Sein Umfang beschränkt sich auf wenige Elemente und ließe sich somit leicht implementieren, ohne auf den Interpreter der Firma Netads angewiesen zu sein.

Das Problem, auch hier, ist, dass dieser speziell für die Arbeit mit einem BTKit ausgelegt wurde. Dementsprechend finden sich auch hier Elemente, die im AmbiComp Umfeld überflüssig sind, während andere fehlen. Eigene Erweiterungen sind nicht möglich, will man sich an dem Standard orientieren. Somit ist dieses Format, wie WML auch, nicht geeignet.

Weitere in Frage kommende Standards konnten im Rahmen der Recherche nicht ausgemacht werden.

Da kein Standard gefunden werden konnte, der für die vorgesehene Aufgabe geeignet ist, wurden Überlegungen begonnen, eine eigene Beschreibungssprache auf Basis von XML zu entwickeln. Aufgrund der Recherche konnten Ideen über den Aufbau von Beschreibungssprachen gesammelt werden. Es zeigt sich, dass mit einigen Änderungen ein einfacher Standard wie der des BTKit ausreichen würde, um die aktuellen Anforderungen zu erfüllen. Des Weiteren kann eine eigene Beschreibungssprache mit fortschreitender Entwicklung des Systems an möglicherweise gewachsene Bedürfnisse angepasst werden.

Entwicklung eines Beschreibungsformates für graphische Oberflächen

Das Beschreibungsformat wird aus zwei Teilen bestehen. Einer XML Datei, die den Aufbau der graphischen Oberfläche beschreibt, und einer Document Type Definition (DTD). Die DTD beschreibt, wie die XML Datei selbst aufgebaut ist, d.h. die Reihenfolge der Elemente, ihren Inhalt und die erlaubten Attribute. Für Informationen über XML selbst siehe **World Wide Web Consortium** (2007).

Das Format muss folgende Anforderungen umsetzen:

- Beschreibung von graphischen Elementen wie Buttons, Textfeldern oder Listen
- Beschreibung von Aktionen, die zu graphischen Elementen gehören
- Es muss lizenzfrei nutzbar sein.

und die Möglichkeit bieten, den Standard in Zukunft zu erweitern.

Die Elemente und Attribute des Beschreibungsformates

Das XML Format setzt sich aus Strukturierungselementen und GUI Elementen zusammen. Folgend werden die einzelnen Teile definiert und erläutert. Es ist die Aufgabe der Benutzerschnittstelle dieses Format auf Basis der Element- und Attributbeschreibung richtig zu interpretieren und darzustellen.

Strukturierungselemente

Strukturierungselemente dienen der besseren Gliederung der graphischen Elemente. Sie werden selbst nicht angezeigt.

`<ambiCompGui>`

Repräsentiert das Wurzelelement der XML Struktur.

Attribute: id – Einmalige ID, die das Element eindeutig identifiziert. REQUIRED.

label – Name der Applikation

version – Gibt die verwendete Version an, die zur Erstellung der GUI verwendet wurde. REQUIRED

applicationAddress – Einmalige ID, welche die Applikation identifiziert, zu welcher die GUI gehört. REQUIRED

Erlaubte Unterelemente:

page

Graphische Elemente

Unter graphischen Elementen werden all jene verstanden, die sichtbar auf einem Display dargestellt werden können. Dabei treten manche Elemente mehr in den Vorgrund als andere.

`<page>`

Repräsentiert eine Seite (Screen) auf dem Zielgerät. Die Seite kann größer sein als die Displaygröße eines Gerätes. Die Seite hat eine Größe, die durch die Angaben von „width“ und „height“ vorgegeben wird. Falls sich GUI Elemente außerhalb dieses Rahmens befinden, sollen sie nicht angezeigt werden.

Attribute: id – Einmalige ID, die das Element eindeutig identifiziert. REQUIRED.

title – Titel der Seite.

width – Breite der Seite in Pixel.

height – Höhe der Seite in Pixel.

Erlaubte Unterelemente:

button, label, inputField, inputBox, radioBox, text, choiceGroup, group, link, menu

`<button>`

Repräsentiert ein Button-Element. Bei der Aktivierung eines Button Elements sollen die ihm untergeordneten Aktionen ausgeführt werden. „width“ und „height“ geben die Größe des Elements wieder, „xPos“ und „yPos“ seine Position innerhalb des übergeordneten Elements.

Attribute: id – Einmalige ID, die das Element eindeutig identifiziert. REQUIRED.

label – Bezeichnung des Buttons.

width – Breite des Buttons in Pixel.

height – Höhe des Buttons in Pixel.

xPos - x-Koordinate der oberen linken Ecke im übergeordneten Element.

yPos - y-Koordinate der oberen linken Ecke im übergeordneten Element.

Erlaubte Unterelemente:

action

`<label>`

Repräsentiert ein Label. Ein Label ist ein kurzer Schriftzug, der auf dem Bildschirm positioniert werden kann. „width“ und „height“ geben die Größe des Elements wieder, „xPos“ und „yPos“ seine Position innerhalb des übergeordneten Elements.

Attribute: id – Einmalige ID, die das Element eindeutig identifiziert. REQUIRED.

label – Gibt den Text des Labels an.

width – Breite des Labels in Pixel.

height – Höhe des Labels in Pixel.

xPos - x-Koordinate der oberen linken Ecke im übergeordneten Element.

yPos - y-Koordinate der oberen linken Ecke im übergeordneten Element.

Erlaubte Unterelemente:

Keine

`<inputField>`

Repräsentiert ein einzeiliges Eingabefeld. In ein Eingabefeld können beliebige Eingaben gemacht werden. width“ und „height“ geben die Größe des Elements wieder, „xPos“ und „yPos“ seine Position innerhalb des übergeordneten Elements. Das Unterelement „text“ bestimmt den aktuellen Inhalt des Eingabefeldes.

Attribute: id – Einmalige ID, die das Element eindeutig identifiziert. REQUIRED.

value – Aktueller Inhalt der input box

width – Breite des Eingabefelds in Pixel.

height – Höhe des Eingabefelds in Pixel.

xPos - x-Koordinate der oberen linken Ecke im übergeordneten Element.

yPos - y-Koordinate der oberen linken Ecke im übergeordneten Element.

Erlaubte Unterelemente:

Keine

`<inputBox>`

Repräsentiert eine Eingabebox. Eine Eingabebox hat im Vergleich zum Eingabefeld einen Scrollbalken. In eine Eingabebox können beliebige Eingaben gemacht werden. width“ und „height“ geben die Größe des Elements wieder, „xPos“ und „yPos“ seine Position innerhalb des übergeordneten Elements. Das Unterelement „text“ bestimmt den aktuellen Inhalt der Eingabebox.

Attribute: id – Einmalige ID, die das Element eindeutig identifiziert. REQUIRED.

width – Breite der Eingabebox in Pixel.

height – Höhe der Eingabebox in Pixel.

xPos - x-Koordinate der oberen linken Ecke im übergeordneten Element.

yPos - y-Koordinate der oberen linken Ecke im übergeordneten Element.

Erlaubte Unterelemente:

Keine

`<radioBox>`

Repräsentiert eine Radiobox. Eine Radiobox beinhaltet eine Liste von Radio Box Einträgen, von denen genau einer ausgewählt werden kann. Wird ein Eintrag selektiert, wird ein anderer, der bereits ausgewählt wurde, deselektiert.

Attribute: id – Einmalige ID. die das Element eindeutig identifiziert. REQUIRED.

width – Breite der Radiobox in Pixel.

height – Höhe der Radiobox in Pixel.

xPos - x-Koordinate der oberen linken Ecke im übergeordneten Element.

yPos - y-Koordinate der oberen linken Ecke im übergeordneten Element.

Erlaubte Unterelemente:

radioBoxEntry

`<radioBoxEntry>`

Repräsentiert einen Eintrag einer RadioBox. „label“ bezeichnet den angezeigten Text der zu dem Radio Box Eintrag gehört, „selected“ gibt an, ob der Eintrag aktuell ausgewählt ist.

Attribute: id – Einmalige ID die das Element eindeutig identifiziert. REQUIRED.

label – Zum Eintrag gehörendes Label.

selected – Erlaubte Werte: "true", "false", Default ist “false“.

Erlaubte Unterelemente:

Keine

`<choiceGroup>`

Repräsentiert eine Auswahl von Elementen. Choice Groups lassen die gleichzeitige Auswahl mehrerer choiceGroupEntrys zu.

Attribute: id – Einmalige ID, die das Element eindeutig identifiziert. REQUIRED.

width – Breite der choiceGroup in Pixel.

height – Höhe der choiceGroup in Pixel.

xPos - x-Koordinate der oberen linken Ecke im übergeordneten Element.

yPos - y-Koordinate der oberen linken Ecke im übergeordneten Element.

Erlaubte Unterelemente:

chocieGroupEntry

`<choiceGroupEntry>`

Repräsentiert einen Choice Group Eintrag. Ein Choice Group Eintrag kann ausgewählt oder nicht ausgewählt sein. „label“ bezeichnet den angezeigten Text, der zu dem Choice Group Eintrag gehört. „selected“ gibt an, ob der Eintrag aktuell ausgewählt ist.

Attribute: id – Einmalige ID, die das Element eindeutig identifiziert. REQUIRED.

label – Zum Eintrag gehörendes Label.

selected - Erlaubte Werte: "true", "false", Default ist “false“.

Erlaubte Unterelemente:

Keine

`<list>`

Repräsentiert eine Liste. Eine Liste kann beliebig viele listEntrys enthalten.

Attribute: id – Einmalige ID, die das Element eindeutig identifiziert. REQUIRED.

width – Breite der Liste in Pixel.

height – Höhe der Liste in Pixel.

xPos - x-Koordinate der oberen linken Ecke im übergeordneten Element.

yPos - y-Koordinate der oberen linken Ecke im übergeordneten Element.

Erlaubte Unterelemente:

listEntry

`<listEntry>`

Repräsentiert einen Listeneintrag. „label“ bezeichnet den angezeigten Text, der zu dem Listeneintrag gehört. „selected“ gibt an, ob der Eintrag aktuell ausgewählt ist.

Attribute: id – Einmalige ID, die das Element eindeutig identifiziert. REQUIRED.

label – Zum Eintrag gehörendes Label.

Erlaubte Unterelemente:

Keine

`<text>`

Steht für reinen Text ohne weitere Funktion.

Attribute: id – Einmalige ID, die das Element eindeutig identifiziert. REQUIRED.

Erlaubte Unterelemente:

#CDATA

`<link>`

Verweist auf ein anderes Element. Bei Auswahl wird zu dem Element gesprungen, dessen ID mittels „idTarget“ angegeben wurde. Aktuell sind nur <page>-Elemente als Zielelemente erlaubt.

Attribute: id – Einmalige ID, die das Element eindeutig identifiziert. REQUIRED.

label – Angezeigter Text der Verlinkung

idTarget - ID des Elements auf das verlinkt wird.

Erlaubte Unterelemente:

Keine

`<action>`

Repräsentiert eine Aktion. Bei Auslösung der Aktion wird der CDATA Bereich der Aktion an die Applikation zu der die GUI gehört übertragen. Die Adresse der Applikation wird durch das Attribut „applicationKey“ des Elements <ambiCompGui> angegeben. Der Inhalt eines `<action>`-Elements kann beliebig sein. Da es aber in ein XML-Format eingebettet ist, muss der Inhalt mit `<![CDATA[ … ]]>` umschlossen werden. Andernfalls kann ein XML Parser das Dokument eventuell nicht fehlerfrei parsen. Auslöser für eine Aktion kann beispielsweise das Drücken eines Buttons sein.

Attribute: id – Einmalige ID, die das Element eindeutig identifiziert. REQUIRED.

Erlaubte Unterelemente:

#CDATA

Die Document Type Definition (DTD)

Die DTD beschreibt den genauen Aufbau des XML-Formates, z.B. welches Element an welcher Stelle stehen darf oder muss oder welche Attribute es haben darf. Durch den vorgegebenen Aufbau wird sichergestellt, dass das XML-Format richtig interpretiert werden kann. Die DTD für das Beschreibungsformat wird im **Anhang A** näher erklärt.

Beispiel für eine GUI Beschreibung

```
<ambiCompGui id="gui1" applicationKey="R21323GHT">
   <page id="page1">
      <button id="button1">
         <action>
            <![CDATA[
               <methodCall>
                  <methodName>
                     ButtonPressed
                  </methodName>
                  <methodParameter>
                     20
                  </methodParameter>
               </methodCall>
            ]]>
         </action>
      </button>
   </page>
</ambiCompGui>
```

5.1.1.2 Update der graphischen Oberfläche zur Laufzeit

Der Inhalt einer graphischen Oberfläche kann sich zur Laufzeit der Applikation jederzeit ändern. Ursachen für solche Veränderungen können vielfältig sein, z.B. Eingaben durch den Benutzer, das Öffnen oder Schließen weiterer Anwendungen, die Anzeige von Warnungen, das Abrufen neuer Informationen, usw. Daher muss ein Mechanismus installiert sein, der die Benutzerschnittstelle über diese Änderungen informiert und ihr die Möglichkeit gibt, ihre Anzeige entsprechend zu aktualisieren. Für diese Aufgabe wird ein eigenes Format entwickelt.

Das Update-Format

In Kapitel 5.1.1.1 wurde festegelegt, dass die graphischen Elemente für die Benutzerschnittstelle durch das erarbeitete XML-Format beschrieben werden. Die Aufgabe der Benutzerschnittstelle besteht darin, dieses XML-Format zu interpretieren und graphisch darzustellen.

Sollte sich nun eines oder mehrere Elemente der Benutzerschnittstelle ändern, z.B. der Inhalt eine Textfeldes, so ist dies für die Benutzerschnittstelle in der aktuellen Ausarbeitung ohne Bedeutung. Sie hat bisher nur die Aufgabe eine graphische Benutzeroberfläche auf Basis ihrer Beschreibung darzustellen. Die einzige Möglichkeit, Änderungen anzuzeigen, besteht somit darin, nochmals eine komplette Beschreibung der Oberfläche, die die Änderungen enthält, an die Schnittstelle zu senden, damit diese die alte Darstellung verwirft und die Neue anzeigt. Ein Verfahren, dass mit sehr viel Overhead verbunden ist.

Um dies zu vereinfachen, wird ein Update Format entwickelt. Ein solches Update kann sich auf beliebige Elemente der graphischen Oberfläche beziehen und ermöglicht es, nur diese Elemente zu aktualisieren.

Damit dies funktioniert, muss die Benutzerschnittstelle in der Lage sein, dieses Update Format zu verstehen und umzusetzen.

Zunächst wird dieses Format erarbeitet. Im Anschluss daran erfolgt eine Erklärung wie ein Update ablaufen wird.

XML-Syntax

`<ambiCompGuiUpdate>`

Wenn eine Benutzerschnittstelle eine Updateanweisung bekommt, so erkennt sie anhand des Attributes „applicationAddress“, um welche Anwendung es sich handelt und anhand der „targetID“, welches Element aktualisiert werden soll.

Attribute: applicationAddress – Gibt die Anwendung an, von der das Update gesendet wurde.

targetID – Gibt die ID des Elements an, dass geändert werden soll. Eine Änderung beeinflusst alle Unterelemente des Zielelements mit.

Erlaubte Unterelemente:

Alle Elemente, die zur Beschreibung einer graphischen Oberfläche erlaubt sind: page, button, label, inputField, radioBox, text, textBox, choiceGroup, group, link, menu

Die Document Type Definition

Die DTD für das Update-Format befindet sich im **Anhang A**.

Funktionsweise des Updates

Erhält eine Benutzerschnittstelle eine Updateanweisung, kann sie anhand des Attributes targetID des Elements `<ambiCompGuiUpdate>` erkennen, welcher Teil der graphischen Oberfläche ein Update erhalten soll. Das Attribut targetID bezieht sich dabei auf die ID des Zielelementes. Alle Unterelemente und Inhalte des Zielelementes werden daraufhin durch den in der Updateanweisung enthaltenen Code ersetzt. Folgendes Beispiel zeigt die Funktionsweise:

Beispiel

XML Struktur vor Update

```
<ambiCompGui id="gui1" applicationAddress="R21323GHT">
   <page id="page1">
      <button id="button1">
         <action>
            Irgendeine Aktion
         </action>
      </button>
   </page>
</ambiCompGui>
```

Update Anweisung

```
<ambiCompGuiUpdate  applicationAddress ="R21323GHT"
                    targetID=”page1”>
   <button id=”button3”>
      <action>
         <![CDATA[
            Irgendeine Aktion
         ]]>
      </action>
   </button>
   <button id=”button4”>
      <action>
         <![CDATA[
            Irgendeine Aktion
        ]]>
      </action>
   </button>
</ ambiCompGuiUpdate >
```

XML Struktur nach Update

```
<ambiCompGui id="gui1" applicationAddress="R21323GHT">
   <page id="page1">
      <button id=”button3”>
         <action>
            <![CDATA[
               Irgendeine Aktion
            ]]>
         </action>
      </button>
      <button id=”button4”>
         <action>
            <![CDATA[
               Irgendeine Aktion
            ]]>
         </action>
      </button>
   </page>
</ambiCompGui>
```

5.2 Gerät-AmbiComp-Schnittstelle (GAS)

Über die GAS verbindet sich das mobile Gerät mit dem AmbiComp-System. Über diese Schnittstelle können Informationen ausgetauscht werden, die für das AmbiComp-System bestimmt sind und umgekehrt.

5.2.1 Eigenschaften der Gerät-AmbiComp-Schnittstelle

Über diese Schnittstelle können Informationen zwischen einem AmbiComp-System und einem Gerät ausgetauscht werden. Eine wichtige Eigenschaft ist die Flexibilität der Schnittstelle. Noch ist nicht bekannt, über welche Protokolle Informationen ausgetauscht werden und wie diese Informationen aufgebaut sind. Auch die Verbindungsmöglichkeiten (Bluetooth, WLAN, etc.), die einem mobilen Gerät oder auf Seiten des AmbiComp-Systems zur Verfügung stehen, sind noch offen. Es sollte möglich sein, beliebige Daten über eine beliebige Verbindung zu schicken und damit ein identisches Ergebnis zu erreichen. Es sollte aber auch berücksichtigt werden, dass nicht jedes Gerät oder jedes AmbiComp-System über jede Verbindungsmöglichkeiten verfügt.

Die Schnittstelle selbst hat nicht die Aufgabe ankommende Daten zu interpretieren. Sie muss nur das Senden und Empfangen dieser über entsprechende Wege ermöglichen. Kommen Daten an, werde diese an jene Stellen geschickt, die auf den Empfang von Daten warten. Wenn Daten gesendet werden sollen, so gibt derjenige, der die Sendeoption aufruft, den Empfänger vor.

5.2.2 Konzept für die Umsetzung der GAS

Je nach der Ausstattung mit Netzwerkschnittstellen können die Daten über unterschiedliche Kanäle gesendet und empfangen werden.

Die Aufgabe der GAS ist es, auf ankommende Daten zu warten und diese an das vorgegebene Ziel weiterzuleiten. Sie stellt damit einen Service zur Verfügung, der genutzt werden kann, um Daten zu übermitteln.

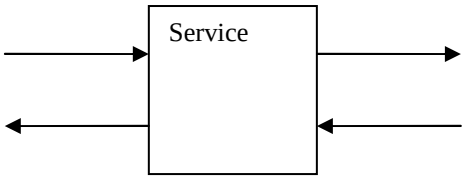

Abbildung 5-1: GAS-Service

Für mobile Geräte gibt es eine Vielfalt an möglichen Netzwerkschnittstellen. Die aktuell bekannteste ist Bluetooth. Weitere Möglichkeiten sind Infrarot und vor allem bei Smartphones und PDAs WLAN. Es gibt aber neben diesen Schnittstellen auch noch weitere weniger weit verbreitete oder weniger bekannte Netzwerkschnittstellen. Diese könnten ebenso zur Kommunikation genutzt werden, wenn sie auch auf Seiten des AmbiComp-System zur Verfügung stehen.

5.3 Zusammenfassung

Die folgende Grafik veranschaulicht noch einmal den Zusammenhang zwischen den beteiligten Anwendern, mobilem Gerät und AmbiComp-System.

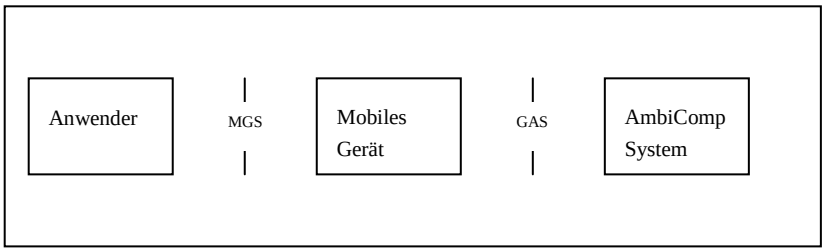

Abbildung 5-2: Schnittstellenübersicht

Man kann erkennen, dass die Schnittstellen die Verbindungstücke zwischen den Systemen darstellen. Sie ermöglichen es den Systemen, miteinander zu kommunizieren. Was sie aber nicht regeln, ist der Ablauf des Datenaustausches und die Art der Daten, die zwischen den Beteiligten ausgetauscht werden. Auch geben sie keine Auskunft darüber, wie Datenpakete gehandhabt werden und was im mobilen Gerät oder im AmbiComp-System mit den empfangenen Daten geschieht. Diese Gesichtspunkte werden in nachfolgenden Kapiteln behandelt.

6 Konzepte für Datenformate

Im Rahmen des Kapitels Schnittstellenkonzepte wurden die Schnittstellen festgelegt, über die Daten ausgetauscht werden können. Über den Aufbau der ausgetauschten Daten und wie sie zu interpretieren sind, werden dort keine Angaben gemacht. Dies hat den Vorteil, dass die Schnittstellen mit beliebigen Daten arbeiten können. Die Daten verarbeitenden Teile der Systeme müssen die übertragenen Daten aber interpretieren und verarbeiten können. Daher muss ihnen das Format der übertragenen Daten bekannt sein.

> **Wikipedia 2** (01.10.2007):
>
> *„Ein Datenformat ist eine Spezifikation der Datenverarbeitung, die festlegt, wie Daten beim Laden, Speichern oder Verarbeiten programmtechnisch zu interpretieren sind.“*

Als Grundlage für die Beschreibung der Datenformate im Rahmen dieser Arbeit wurde XML gewählt. XML bietet den Vorteil, dass es sich für die Beschreibung beliebiger Daten nutzen lässt und dabei einfach erweiterbar ist, wenn sich neue Anforderungen ergeben.

6.1 Rahmenformat für die Übertragung von Daten

Zwischen einem AmbiComp-System und einem mobilen Gerät können theoretisch beliebige Daten ausgetauscht werden. Damit die annehmende Stelle erkennen kann, in welchem Format die ankommenden Daten verpackt sind, werden diese Daten in ein Rahmenformat verpackt. Der Einsatz dieses Rahmenformats schafft eine große Flexibilität, da es die Übertragung beliebiger Daten ermöglicht und der empfangenden Seite gleichzeitig anzeigt, welches Datenformat die enthaltenen Daten haben.

Die Elemente des Rahmenformats

Der Aufbau des Rahmenformats ist bewusst einfach gehalten, um einen unnötigen Overhead zu vermeiden.

Das Wurzelelement

„ambiComp“ ist das einzige Element. Es zeigt an, dass es sich um eine Nachricht aus dem AmbiComp Umfeld handelt.

Attribute: - „version“ gibt die Versionsnummer des verwendeten Rahmenformates an. Aktuell ist es die Versionsnummer 1.0. Aufgrund von zusätzlichen Anforderungen kann es jedoch notwendig werden, das Rahmenformat anzupassen. Eine Applikation könnte z.B. die Annahme eines Datenpaketes ablehnen, wenn die Versionsnummer nicht passt.

- „type“ gibt an, wie der Inhalt zwischen dem Start- und Endelement zu interpretieren ist. type=“xml“ würde beispielsweise anzeigen, dass es sich um ein XML-Format handelt. Der Typ ist dazu gedacht, die Applikation, abhängig vom Typ entsprechender Interpreter, benutzen zu können, um die Daten richtig zu verstehen.

Zwischen dem öffnenden und schließenden AmbiComp-Element können beliebige Daten stehen. Applikationen erkennen anhand des „type“-Attributs wie diese Informationen zu interpretieren sind. Damit bei der Interpretation des Inhaltes, z.B. bei XML-Daten, keine Probleme auftreten, sind die Daten in eine CDATA-Section einzuschließen. Die CDATA Section verhindert, dass die Daten von einem XML-Parser interpretiert werden. Wird dies nicht getan, können aufgrund von Sonderzeichen innerhalb der Daten Fehlinterpretationenentstehen.

Die DTD des Rahmenformates

```
<!ELEMENT ambiComp #CDATA>
<!ATTLIST ambiComp
   version CDATA #REQUIRED
   type #CDATA #REQUIRED
>
```

Beispiel

```
<ambiComp version=“1.0“ type=“xml“>
   <![CDATA[
      <testTag>
         Hallo Welt!
      </testTag>
   ]]>
</ambiComp>
```

6.2 Datenformat für die Übertragung von Methodenaufrufen

Im Kapitel: Schnittstellenkonzepte - Darstellung von graphischen Elementen zur Laufzeit, wird das „action“-Element aufgeführt. Beim Auslösen einer Aktion, z.B. dem drücken eines Knopfes, soll der Inhalt des zugehörigen „action“-Elements an die entsprechende Applikation übergeben werden. Gemäß der Definition kann das Element einen beliebigen Text enthalten. Damit eine Applikation aber versteht, was zu tun ist, müssen die Daten innerhalb eines „action“-Elements in einer besonderen Art formatiert sein, die von der Applikation richtig interpretiert werden kann.

Die Elemente und Attribute des Formates

`<methodCall>`

Wurzelelement.

Attribute: id – Identifikationsnummer für diesen Methoden-Aufruf. Kann für Referenzzwecke genutzt werden.

applicationAddress – Adresse der Ziel-Applikation für diesen Aufruf. Entspricht dem gleichen Eintrag wie dem Attribut „applicationAddress“ des Elements <ambiComp>.

Erlaubte Unterelemente:

methodName, methodParameter

`<methodName>`

Name der Methode, die aufgerufen werden soll.

Attribute: Keine.

Erlaubte Unterelemente:

CDATA (Surrounded by a CDATA-Section `<![CDATA[ … ]]>`)

`<methodParameter>`

Attribute: parameterType – Art des Parameters, e.g. int, string, Boolean (true/false), etc.

Erlaubte Unterelemente:

CDATA (Surrounded by a CDATA-Section `<![CDATA[ … ]]>`)

Die DTD

```
<!ELEMENT methodCall (methodName, methodParameter*)>
<!ATTLIST methodCall
   id CDATA #REQUIRED
   applicationAddress CDATA #REQUIRED
>
<!ELEMENT methodName (#CDATA)>
<!ELEMENT methodParameter> (#CDATA)>
<!ATTLIST methodParameter
   parameterType CDATA #REQUIRED
>
```

Beispiel

```
<methodCall id=”No1” applicationAddress =“r8g88f.App“>
   <methodName>
      <![CDATA[ ButtonPressed ]]>
   </methodName>
   <methodParameter parameterType=”int”>
      <![CDATA [ 20 ]]>
   </methodParameter>
   <methodParameter parameterType=”boolean”>
      <![CDATA[ True ]]>
   </methodParameter>
</methodCall>
```

7 Prototypenentwicklung

In den vorhergehenden Kapiteln wurden Konzepte erarbeitet, die es ermöglich, die Zielsetzung der Diplomarbeit zu erfüllen. Aufgabe dieses Kapitels ist es nun, die Konzepte in konkrete Anwendungen umzusetzen und deren Praxistauglichkeit zu erproben.

7.1 Überblick

Im Kapitel „Schnittstellenkonzepte“ wurden drei konkrete Systeme eruiert, die miteinander kommunizieren können, der Anwender, das mobile Geräte und das AmbiComp-System. Die Beziehung zwischen den drei Systemen wird durch folgende Grafik verdeutlicht:

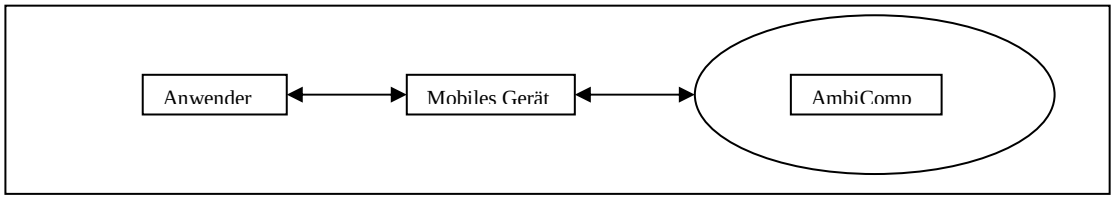

Abbildung 7-1: Systemübersicht

Der Anwender kommuniziert ausschließlich mit dem mobilen Gerät, welches wiederum mit dem AmbiComp-System kommuniziert.

Ausgehend von dieser Übersicht scheint alles soweit zu funktionieren, wie gedacht. Doch genauer betrachtet, ergibt sich bei der Kommunikation mit dem AmbiComp-System ein Problem. Bisher ist ein AmbiComp-System ein geschlossenes System. Dies bedeutet, dass nur im System integrierte AICUs (Siehe Kapitel 3.1.1) miteinander kommunizieren können. Externe Geräte sind davon ausgenommen. Es muss deshalb auf Seiten des AmbiComp-Systems ein Zugang, im folgenden Gateway genannt, geschaffen werden, der die Kommunikation zwischen internen und externen Geräten ermöglicht. Dies führt zu folgender Systemsicht:

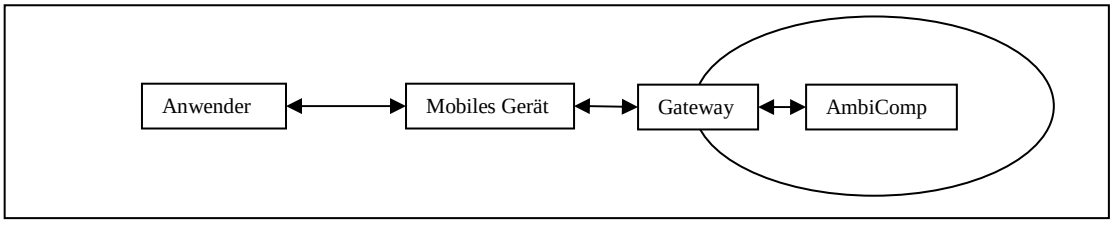

Abbildung 7-2: Erweiterte Systemübersicht

Der Gateway ist eine AICU auf der eine spezielle Applikation läuft, die Gateway-Applikation.

Wie die Gateway-Applikation funktioniert und aufgebaut ist, wird im Kapitel 7.2 erläutert. Kapitel 7.3 erklärt die Arbeitsweise der Client Applikation und Kapitel 7.4 beschreibt ein Testszenario, in dem die Anwendungen und damit die dort umgesetzten Konzepte auf ihre Praxistauglichkeit überprüft werden.

7.2 Entwicklung der AmbiComp Gateway Applikation

Die Gateway Applikation ist die Verbindung zwischen einem AmbiComp-System und der Außenwelt. Ihre Aufgabe ist es, Anfragen von außerhalb anzunehmen, aufzubereiten und an das AmbiComp-System weiterzuleiten. Wird eine Rückantwort erwatet, kann diese aus dem AmbiComp-System über den Gateway an ein externes Gerät geschickt werden.

Die folgende Graphik veranschaulicht den groben Aufbau dieser Applikation:

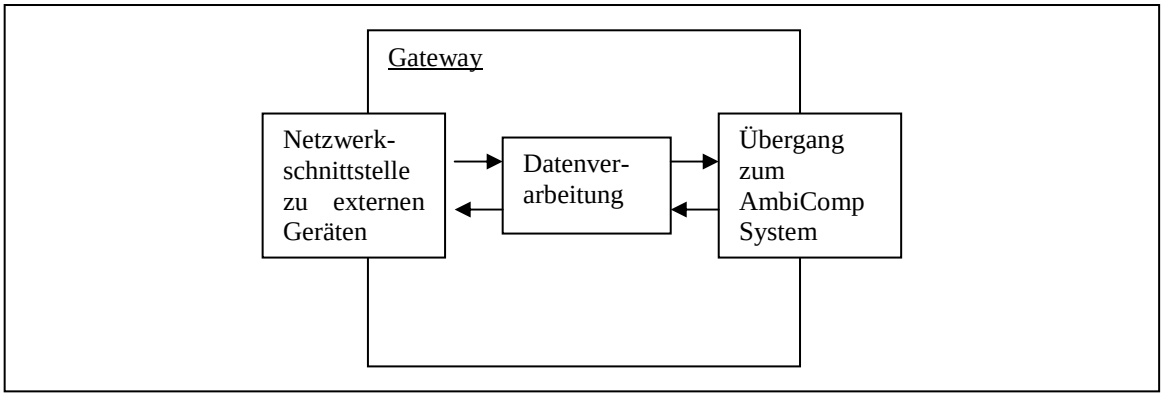

Abbildung 7-3: Aufbau der Gateway Applikation

7.2.1 Netzwerkschnittstelle zu externen Geräten

Über diese Schnittstelle können Daten gesandt und empfangen werden. Als Grundlage dient das Konzept der Gerät-AmbiComp-Schnittstelle (GAS). Die GAS stellt ihre Dienste als Service zur Verfügung, der genutzt werden kann.

Implementiert wird die Schnittstelle unter dem Titel **Gateway Services**. Innerhalb dieser Schnittstelle wird jede Netzwerkverbindung als Service dargestellt und angeboten.

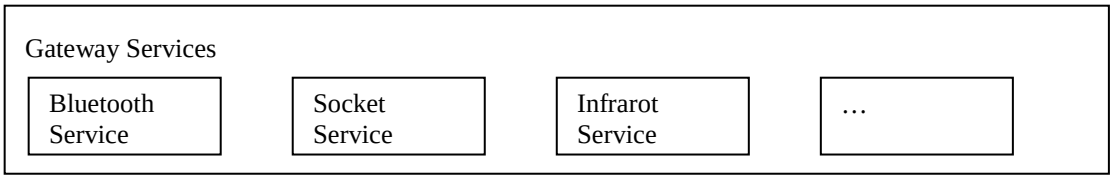

Abbildung 7-4: Mögliche Gateway Services

7.2.1.1 Funktionsweise eines Services

Unabhängig von der Art der Netzwerkverbindung bieten alle Services eine einheitliche Schnittstelle nach außen. Die Komplexität der jeweiligen Netzwerkverbindung kann so vor dem Rest des Systems verborgen werden. Nachfolgend werden die Elemente der Schnittstelle erläutert.

Service starten und stoppen

Bevor ein Service genutzt werden kann, muss dieser gestartet werden: `startGatewayService()`

Beim Start baut der Service eine Verbindung zu einer bestimmten Netzwerkschnittstelle auf und lauscht von nun an auf ankommenden Daten.

Um den Service zu beenden wird `stopGatewayService()` aufgerufen. Der Service gibt dabei belegte Ressourcen wieder frei.

Registrierung am Service

Ein Service ist sich seiner Umgebung nicht bewusst. Er kennt nur seine Aufgabe, Daten zu empfangen, weiterzuleiten und Daten zu senden.

Damit ein Service ankommende Daten richtig weiterleitet, müssen sich die an diesen Daten interessierten Stellen bei ihm registrieren. Als Vorbild dient hierfür das Observer Pattern von **Gamma** (1995; S.293ff.).

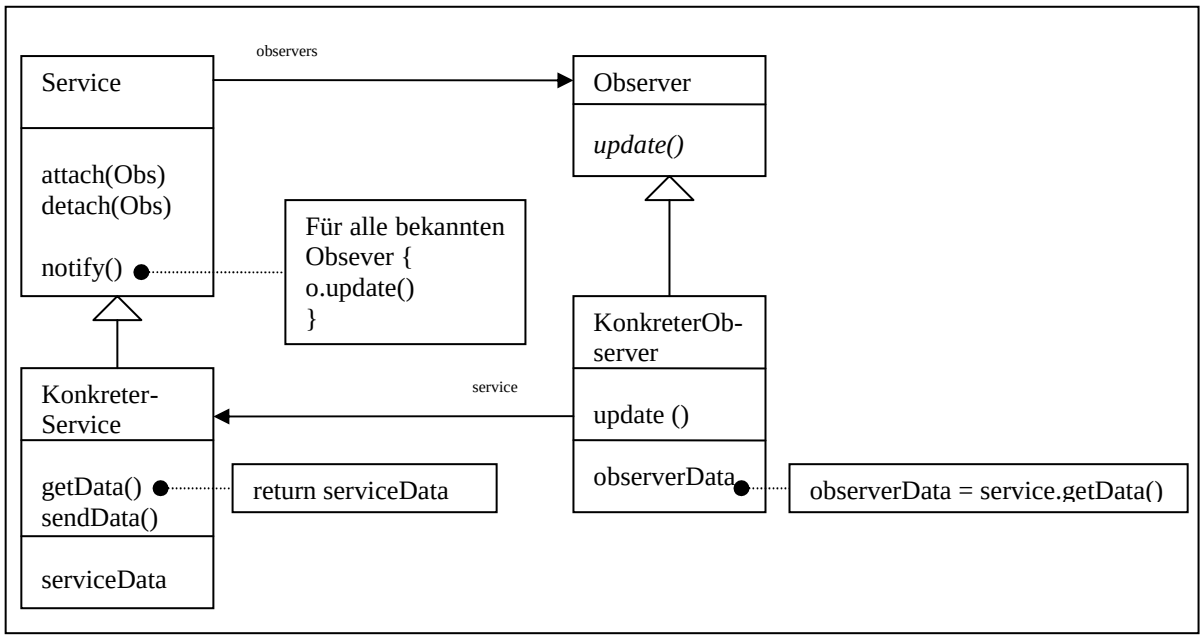

Abbildung 7-5: Gateway Service - Observer Pattern

Mittels `attachObserver(Observer obs)` registriert sich ein Observer (dt. Beobachter) beim Service. Durch den Aufruf von `detachObserver(Observer obs)` wird diese Registrierung wieder aufgehoben.

Behandlung ankommender Daten

Empfängt der Service Daten, werden die Beobachter des Service umgehend darüber informiert `notifyObserver()`. Die informierten Observer können dann mittels `getData()` die Daten abrufen.

Senden von Daten

Das Senden von Daten ist nur unmittelbar nach dem Empfang möglich. Danach wird die Verbindung zum Sender wieder geschlossen. Nachdem die Observer mittels `notifyObserver()` über die Ankunft von Daten informiert wurden, wartet der Service bis mittels `sendData(byte[] message)` eine Rückantwort an den Sender der Daten übermittelt wurde. Danach nimmt er seinen Dienst wieder auf und wartet auf die Ankunft neuer Daten an der Netzwerkschnittstelle.

7.2.1.2 Fazit

Dadurch, dass alle Services die gleichen Funktionen implementieren und diese die gleichen Rückgabewerte haben, ist es für einen Observer irrelevant, über welche Netzwerkschnittstelle die Daten empfangen wurden. Er wird sie in einem einheitlichen Format überreicht bekommen. Das Gleiche gilt, wenn der Observer eine Rückantwort schicken will. Er ruft dazu bei allen Services die gleiche Funktion auf und kann das gleiche Ergebnis erwarten.

7.2.2 Datenverarbeitung

Die Aufgabe der Datenverarbeitung ist es, ankommende Daten zu prüfen, aufzubereiten und weiterzuleiten.

Kommen Daten bei den Gateway Services an, werden diese an die Datenverarbeitung weitergeleitet. Dort werden sie überprüft und aufbereitet bevor sie an die Zieladresse innerhalb des AmbiComp-Systems gesendet werden. Eine Antwort wird auf dem umgekehrten Weg an den Sender zurückgesandt.

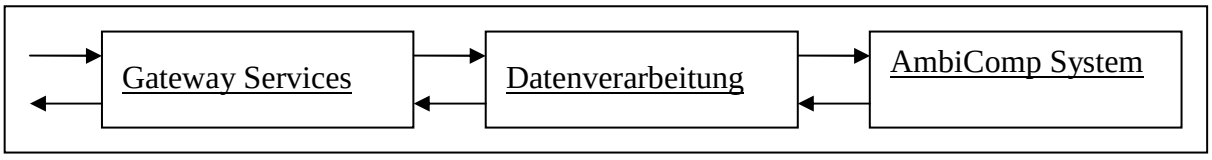

Abbildung 7-6: Ablauf beim Empfang von Daten

7.2.2.1 Die Funktionsweise im Detail

Kern der Datenverarbeitung ist die Klasse `DataHandlerMain`.

Diese registriert sich bei den gewünschten Gateway Services (Siehe: Abbildung 7-5: Gateway Service - Observer Pattern) und wartet darauf über neu angekommene Daten informiert zu werden.

Sobald Daten ankommen, wird `DataHandlerMain` diese über `gwService.getData()` abrufen und mit der Verarbeitung beginnen.

Erster Schritt der Datenverarbeitung ist das Parsen der empfangenen XML Struktur (Siehe: Kapitel 6.1 - Rahmenformat für die Übertragung von Daten):

```
dataHandlerMain.parse(xmlData)
```

Ist der Parsvorgang erfolgreich, wird versucht, den Dateninhalt zu ermitteln:

```
dataHandlerMain.getDataType(xmlDataTree)
```

Basierend auf dem Dateninhalt wird ein entsprechender Handler initialisiert, der die weitere Verarbeitung und Aufbereitung der Daten übernimmt:

```
DataHandler dataHandler = new dataHandler(xmlDataTree);
dataHandler.start();
```

Der Handler wird überwacht, um mitzubekommen wann er seine Aufgabe abgeschlossen hat:

```
dataHandler.attachObserver(this);
```

Ist die Aufbereitung der Daten abgeschlossen, wird der Observer darüber informiert. Damit er die Daten abrufen kann:

```
dataHandler.notify()
dataHandler.getHandledData()
```

Die Daten können in dieser Form an die Zielapplikation übermittelt werden.

```
dataHandlerMain.exectuteData(handledData)
```

Nachdem die Zielapplikation die Daten verarbeitet hat, sendet sie eine Rückantwort, welche über den Gateway an den Sender der Daten zurückgesandt wird.

7.2.3 Übergang zum AmbiComp-System

An dieser Stelle werden Daten in das AmbiComp-System eingespeist. SSR, das Routingprotokoll von AmbiComp wird Methodenaufrufe automatisch an die richtige Zieladresse, in diesem Fall einer Appliaktion, weiterleiten. Eine Antwort der Applikation wird über den Gateway, zurückgesandt

Dieser Teil wird nicht im Rahmen der Diplomarbeit entwickelt, sondern als vorhanden angenommen.

7.3 Entwicklung der AmbiComp Client Applikation

Die Client Applikation verbindet einen Anwender mit einem AmbiComp-System, genauer, mit einem Gateway, der den Zugang zu dem System ermöglicht.

Die Client Applikation besteht aus drei zentralen Teilen, welche in der folgenden Grafik dargestellt sind:

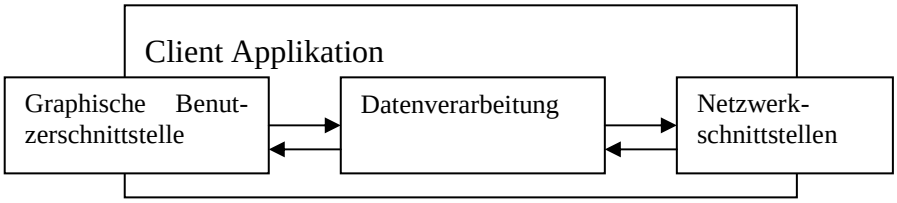

Abbildung 7-7: Aufbau der Client Applikation

Die graphische Benutzerschnittstelle ist für die Darstellung von graphischen Elementen zuständig. Die Datenverarbeitung übernimmt, wie bei der Gateway-Applikation, die Aufbereitung der Daten für die jeweiligen Schnittstellen. Die Netzwerkschnittstelle ist für das Senden und Empfangen von Daten zuständig.

7.3.1 Die graphische Benutzerschnittstelle

Die graphische Benutzerschnittstelle ist die Verbindung zwischen Anwender und mobilem Gerät. Sie gliedert sich in zwei Teile.

Der erste Teil besteht aus einer Reihe bereits implementierter graphischer Elemente, die dem Anwender den Zugang zu den Grundfunktionalitäten ermöglichen. Eine Grundfunktionalität ist beispielsweise die Suche nach AmbiComp Gateways in der Umgebung.

Der zweite Teil besteht aus einem Interpreter, der zur Laufzeit graphische Oberflächen anhand ihrer Beschreibung erzeugt und darstellt.

7.3.2 Die Datenverarbeitung

Aufgabe der Datenverarbeitung ist die Aufbereitung von Daten für die jeweiligen Schnittstellen. Die graphische Benutzerschnittstelle leitet Kommandos (Aktionen), die dort ausgewählt wurden (z.B. durch das Drücken eines Knopfes), an die Datenverarbeitung weiter. Diese verpackt die Daten in ein entsprechendes Rahmenformat und sendet diese über die Netzwerkschnittstellen an das AmbiComp System. Kommt vom System eine Rückantwort, wird diese für die Benutzerschnittstelle aufbereitet und dieser übergeben.

7.3.3 Die Netzwerkschnittstellen

Über die Netzwerkschnittstellen kann ein mobiles Gerät mit einem AmbiComp System in Verbindung treten und Daten austauschen. Alle Schnittstellen stellen dafür ein einheitliches Interface zur Verfügung:

Mittels `searchForDevices()` werden AmbiComp Gateways gesucht, die aktuell über diese Schnittstelle erreicht werden können.

Mittels `connectToDevice(selectedDevice)` wird eine Verbindung zum gewählten Gerät aufgebaut. Mittels `disconnectFromDevice()` wird diese Verbindung wieder beendet.

Die Funktion `sendAndRecieveData()` ermöglicht schließlich das Senden von Daten und das Empfangen von Antworten. Es ist zu beachten, dass nur empfangen werden kann, wenn davor etwas gesendet wurde.

7.3.4 Funktionsweise der Client Applikation

Nach dem Start der Applikation muss der Anwender sich zuerst mit einem AmbiComp Gateway verbinden. Dazu werden nacheinander drei Schritte ausgeführt:

1. Auswahl der Verbindungsart (z.B. Bluetooth)
2. Auswahl des gewünschten Gateways, wenn einer gefunden wurde
3. Verbindungsaufbau zum Gateway

Der Gateway wird daraufhin eine XML-Beschreibung seiner graphischen Oberfläche übermitteln, welche von dem Gerät interpretiert und angezeigt wird. Über die graphische Oberfläche bietet der Gateway seine Dienste an, beispielsweise den Zugriff auf eine bestimmte Applikation im AmbiComp System. Wählt der Benutzer dies aus, so wird ihm die GUI der entsprechenden Applikation übermittelt und er kann diese bedienen.

7.4 Testen der Prototypen

In diesem Kapitel wird überprüft, ob die entwickelten Anwendungen gemäß den Anforderungen funktionieren. Dadurch wird der Beweis erbracht, dass die in Kapitel 5 entwickelten Schnittstellen und in Kapitel 6 erarbeiteten Datenformate die Erwartungen erfüllen und in die Praxis umgesetzt werden können.

7.4.1 Das Testszenario

Das Testszenario besteht aus der Simulation von zwei AmbiComp Geräten und einem Mobiltelefon als Client Gerät.

7.4.2 Testumgebung

Es gibt noch keine Hardware, mit der das Testszenario umgesetzt werden könnte. Daher wird die Systemumgebung auf einem normalen Computer simuliert.

Gateway und Temperatursensor werden durch zwei Java-Anwendungen repräsentiert. Die Gateway-Applikation kann die Bluetooth Schnittstelle des Testrechners zur Kommunikation nach außen nutzen.

Die Client-Applikation wird auf einem Mobiltelefon ausgeführt und kann mittels Bluetooth eine Verbindung zum AmbiComp Gateway aufbauen.

7.4.3 Ablauf des Tests

- Starten der Gateway- und Temperatursensorsimulation
- Client verbindet sich über Bluetooth mit dem Gateway.
- Gateway GUI wird übertragen und dargestellt (Liste der verfügbaren Applikationen)
- Anwender wählt Temperatursensor-Applikation aus.
- Gateway überträgt die GUI-Beschreibung der Applikation.
- Client zeigt eine auf der Beschreibung basierende graphische Oberfläche an.
- Client verändert über die graphische Oberfläche einen Parameter und sendet die Änderung an die Temperatursensor-Applikation.
- Die Appliaktion übernimmt die Änderung und sendet dem Client ein GUI-Update mit den geänderten Werten.
- Der Client übernimmt das Update und aktualisiert die graphische Oberfläche entsprechend.

Der Test ist damit erfolgreich abgeschlossen.

7.4.4 Ergebnis des Tests

Der Test hat gezeigt, dass ein Client unterschiedliche GUI-Beschreibungen (eine des Gateways und eine des Temperatursensors) in graphische Oberflächen umsetzen kann. Über diese Oberflächen können Aktionen ausgeführt werden, die entsprechende Reaktionen des AmbiComp Systems (z.B. Übertragung von GUI-Updates) auslösen. Diese Reaktionen (z.B. Updates der graphischen Oberfläche) wiederum können vom Client verarbeitet und dargestellt werden.

Damit wurde bewiesen, dass,

- die Beschreibung der graphischen Oberfläche in reale graphische Oberflächen umgesetzt werden kann.d
- der Mechanismus zur Übertragung von GUI-Updates funktioniert und entsprechend umgesetzt werden kann und
- ein AmbiComp Gateway in der Lage ist, Daten von Clients anzunehmen, zu verarbeiten und weiterzuleiten, sowie Antworten an den Client zurück zu senden.

Damit sind die Grundvoraussetzungen geschaffen, die nach Anforderungsbeschreibung gefordert wurden. Es ist möglich, mittels eines Client mit einem AmbiComp System in Kontakt zu treten und von dort Informationen abzurufen. Diese können wiederum einem Anwender graphisch zur Verfügung gestellt werden. Ein Anwender kann dann durch vorgegebene Möglichkeiten Anfragen an das AmbiComp System senden, die dort aufgenommen und weitergeleitet werden können.

8 Resümee

War die Diplomarbeit ein Erfolg? Dies kann mit einem klaren „Ja“ beantwortet werden. Die Implementierung im Rahmen einer Beispielanwendung hat gezeigt, dass die erarbeiteten Konzepte in der Praxis umsetzbar sind. Damit ist die Grundlage für eine Anwendung geschaffen, die es Anwendern ermöglicht, ein AmbiComp-System zu verwalten. Das Hauptziel der Diplomarbeit ist somit erreicht. Das bestätigt auch ein Vergleich der Anforderungen (siehe Kapitel 4) mit dem Ist-Zustand. Darüber hinaus sind die entwickelten Konzepte derart gestaltet, dass sie ohne weiteres an zukünftige Entwicklungen angepasst werden können. Dies ist im Hinblick auf den aktuellen Status des AmbiComp Projektes von besonderer Bedeutung. Da ein fertiges System noch fehlt, mussten bestimmte Annahmen über die Funktionsweise des Systems getroffen werden. Ob diese Annahmen zutreffen, kann jetzt noch nicht sicher gesagt werden. Sollten sich einige nicht bestätigen, so sind die Konzepte zu überprüfen und gegebenenfalls an die neue Situation anzupassen.

Auch an der Applikation muss noch gearbeitet werden. Die erste Implementierung hat nur bewiesen, dass die Konzepte funktionieren. Sie kann aber keineswegs als fertiges Produkt für Anwender angesehen werden. Die Erweiterungsmöglichkeiten, vor allem im Hinblick auf den Bedienkomfort sind groß. Der Autor ist aber der Ansicht, dass die Beispielimplementierung sehr gut als Grundlage für zukünftige Entwicklungen dienen kann.

Anhang A – Document Type Definitions (DTD)

DTD für das Beschreibungsformat zur Darstellung von graphischen Elementen zur Laufzeit (Kapitel 5.1.1.2):

```
<!ELEMENT ambiCompGui (page*)>
<!ATTLIST ambiCompGui
   id ID #REQUIRED
   label CDATA #REQUIRED
   version CDATA #REQUIRED
   applicationAddress CDATA #REQUIRED
>
<!ELEMENT page ( button*, label*, inputField*, inputBox*,
radioBox*, text*, choiceGroup*, group*, link*, menu*)>
<!ATTLIST page
   id ID #REQUIRED
   title CDATA #REQUIRED
   width CDATA #IMPLIED
   height CDATA #IMPLIED
>
<!ELEMENT button (action)>
<!ATTLIST button
   id ID #REQUIRED
   label CDATA #REQUIRED
   width CDATA #IMPLIED
   height CDATA #IMPLIED
   xPos CDATA #IMPLIED
   yPos CDATA #IMPLIED
>
<!ELEMENT label (EMPTY)>
<!ATTLIST label
   id ID #REQUIRED
   label CDATA #REQUIRED
   width CDATA #IMPLIED
   height CDATA #IMPLIED
   xPos CDATA #IMPLIED
   yPos CDATA #IMPLIED
>
<!ELEMENT inputField (EMPTY)>
<!ATTLIST inputField
   id ID #REQUIRED
   value CDATA #IMPLIED
   width CDATA #IMPLIED
   height CDATA #IMPLIED
   xPos CDATA #IMPLIED
   yPos CDATA #IMPLIED
>
<!ELEMENT inputBox (EMPTY)>
<!ATTLIST inputBox
   id ID #REQUIRED
   width CDATA #IMPLIED
   height CDATA #IMPLIED
```

```
   xPos CDATA #IMPLIED
   yPos CDATA #IMPLIED
>
<!ELEMENT radioBox (radioBoxEntry+)>
<!ATTLIST radioBox
   id ID #REQUIRED
   width CDATA #IMPLIED
   height CDATA #IMPLIED
   xPos CDATA #IMPLIED
   yPos CDATA #IMPLIED
>
<!ELEMENT radioBoxEntry (EMPTY)>
<!ATTLIST radioBoxEntry
   id ID #REQUIRED
   label CDATA #REQUIRED
   selected (true | false) "false" #REQUIRED
>
<!ELEMENT choiceGroup (choiceGroupEntry)>
<!ATTLIST choiceGroup
   id ID #REQUIRED
   width CDATA #IMPLIED
   height CDATA #IMPLIED
   xPos CDATA #IMPLIED
   yPos CDATA #IMPLIED
>
<!ELEMENT choiceGroupEntry (EMPTY)>
<!ATTLIST choiceGroupEntry
   id ID #REQUIRED
   label CDATA #REQUIRED
   selected (true | false) "false" #REQUIRED
>
<!ELEMENT list (listEntry+)
<!ATTLIST list
   id ID #REQUIRED
   width CDATA #IMPLIED
   height CDATA #IMPLIED
   xPos CDATA #IMPLIED
   yPos CDATA #IMPLIED
>
<!ELEMENT listEntry (EMPTY)
<!ATTLIST listEntry
   id ID #REQUIRED
   label CDATA #REQUIRED
>
<!ELEMENT link (EMPTY)>
<!ATTLIST link
   id ID #REQUIRED
   label CDATA #REQUIRED
   idTarget CDATA #REQUIRED
>
<!ELEMENT action (#CDATA)>
<!ATTLIST action
   id ID #REQUIRED
>
<!ELEMENT text (#CDATA)>
<!ATTLIST text
   id ID #REQUIRED
>
```

DTD für das Update-Format zur Darstellung von graphischen Elementen zur Laufzeit (Kapitel 5.1.1.3):

```
<!ELEMENT ambiCompGuiUpdate ( page*, button*, label*,
                              inputField*,       radioBox*,
                              text*,    textbox*,    choice-
                              Group*, group*, link*, menu*)
>
<!ATTLIST ambiCompGuiUpdate
   applicationAddress ID #REQUIRED
   targetID  CDATA #REQUIRED
>
<!ELEMENT page ( button*,  label*,  inputField*,  inputBox*,
radioBox*, text*, choiceGroup*, group*, link*, menu*)>
<!ATTLIST page
   id ID #REQUIRED
   title CDATA #REQUIRED
   width CDATA #IMPLIED
   height CDATA #IMPLIED
>
<!ELEMENT button (action)>
<!ATTLIST button
   id ID #REQUIRED
   label CDATA #REQUIRED
   width CDATA #IMPLIED
   height CDATA #IMPLIED
   xPos CDATA #IMPLIED
   yPos CDATA #IMPLIED
>
<!ELEMENT label (EMPTY)>
<!ATTLIST label
   id ID #REQUIRED
   label CDATA #REQUIRED
   width CDATA #IMPLIED
   height CDATA #IMPLIED
   xPos CDATA #IMPLIED
   yPos CDATA #IMPLIED
>
<!ELEMENT inputField (EMPTY)>
<!ATTLIST inputField
   id ID #REQUIRED
   value CDATA #IMPLIED
   width CDATA #IMPLIED
   height CDATA #IMPLIED
   xPos CDATA #IMPLIED
   yPos CDATA #IMPLIED
>
<!ELEMENT inputBox (EMPTY)>
<!ATTLIST inputBox
   id ID #REQUIRED
   width CDATA #IMPLIED
   height CDATA #IMPLIED
   xPos CDATA #IMPLIED
   yPos CDATA #IMPLIED
>
```

```
<!ELEMENT radioBox (radioBoxEntry+)>
<!ATTLIST radioBox
   id ID #REQUIRED
   width CDATA #IMPLIED
   height CDATA #IMPLIED
   xPos CDATA #IMPLIED
   yPos CDATA #IMPLIED
>
<!ELEMENT radioBoxEntry (EMPTY)>
<!ATTLIST radioBoxEntry
   id ID #REQUIRED
   label CDATA #REQUIRED
   selected (true | false) "false" #REQUIRED
>
<!ELEMENT choiceGroup (choiceGroupEntry)>
<!ATTLIST choiceGroup
   id ID #REQUIRED
   width CDATA #IMPLIED
   height CDATA #IMPLIED
   xPos CDATA #IMPLIED
   yPos CDATA #IMPLIED
>
<!ELEMENT choiceGroupEntry (EMPTY)>
<!ATTLIST choiceGroupEntry
   id ID #REQUIRED
   label CDATA #REQUIRED
   selected (true | false) "false" #REQUIRED
>
<!ELEMENT list (listEntry+)
<!ATTLIST list
   id ID #REQUIRED
   width CDATA #IMPLIED
   height CDATA #IMPLIED
   xPos CDATA #IMPLIED
   yPos CDATA #IMPLIED
>
<!ELEMENT listEntry (EMPTY)
<!ATTLIST listEntry
   id ID #REQUIRED
   label CDATA #REQUIRED
>
<!ELEMENT link (EMPTY)>
<!ATTLIST link
   id ID #REQUIRED
   label CDATA #REQUIRED
   idTarget CDATA #REQUIRED
>
<!ELEMENT action (#CDATA)>
<!ATTLIST action
   id ID #REQUIRED
>
<!ELEMENT text (#CDATA)>
<!ATTLIST text
   id ID #REQUIRED
>
```

Literaturverzeichnis

Fuhrmann, T. (2006): Vorhabensbeschreibung zum Projektantrag AmbiComp

Gamma (1995; S.293ff.): Design Patterns – Addison Wesley Professional Computing Series

Netads (Juni 2005): An XML based embedded device - XML Configuration

Suchy, T. (2007; S. 6): Anforderungsanalyse v0.8.3

Weiser, M. (1991): The Computer for the 21st century
Quelle: http://www.ubiq.com/hypertext/weiser/SciAmDraft3.html

Wikipedia 1 (12.10.2007): System

Quelle: http://de.wikipedia.org/wiki/System

Wikipedia 2 (2007): Datenformat

Quelle: http://de.wikipedia.org/wiki/Datenformat

Wikipedia 3 (17.09.2007): Anwendungsprogramm

Quelle: http://de.wikipedia.org/wiki/Anwendungsprogramm

World Wide Web Consortium (2007): Extensible Markup Language (XML)

Quelle: http://www.w3.org/XML/

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

AICU Ambient Intelligence Control Unit

Ami Ambient Intelligence

DTD Document Type Definition

GAS Gerät-AmbiComp-Schnittstelle

GUI Graphical User Interface (graphische Benutzeroberfläche)

HdM Hochschule der Medien

J2ME Java 2 mobile Edition

MGS Mensch-Gerät-Schnittstelle

MIDP mobile Information Device Profile

WAP Wireless Application Protokoll

WML Wireless Markup Language

Stichwortverzeichnis