Universität Hamburg
Fakultät für Mathematik,
Informatik und Naturwissenschaften

Verteilte Systeme und Informationssysteme

Diplomarbeit

Integration von Context-Awareness in eine Middleware für mobile Systeme

Hamburg, 29. Juni 2006

Mirwais Turjalei

turjalei@gmx.de
Studiengang Informatik
Matr.-Nr. 5202551
Fachsemester 14

Erstgutachter: Professor Dr. W. Lamersdorf
Zweitgutachter: Professor Dr. H. Züllighoven

Inhaltsverzeichnis

1 Einleitung

Das folgende Kapitel stellt eine Einführung in die Thematik dieser Diplomarbeit dar. Es skizziert zunächst in prägnanter Form die Begründung für die Integration von Context-Awareness in mobilen Systemen und präzisiert die Ziele, die mit Hilfe dieser Arbeit erreicht werden sollen. Der anschließende Abschnitt zur Eingrenzung des Themas legt die genauen Rahmenbedingungen für die Entwicklung eines generischen Kontextkonzepts fest. Der letzte Abschnitt zur geplanten Vorgehensweise schließt dieses Kapitel ab.

1.1 Motivation

Jede Interaktion zwischen Menschen erfolgt in einem bestimmten Kontext. Dieser Kontext ist uns Menschen meist bewusst. Dass wir die Fähigkeiten haben, situationsabhängige Informationen wahrzunehmen, zu interpretieren und in einer zwischenmenschlichen Kommunikation einzusetzen, hängt dabei von verschiedenen Faktoren ab.
Ein entscheidender Vorteil in diesem Zusammenhang ist zunächst das *Reichtum* und *Vielfalt* der menschlichen *Sprache*. Darüber hinaus verfügen wir ein allgemeines Verständnis darüber wie die Dinge in der Welt zueinander in Beziehung stehen. Auch implizites Wissen über alltägliche Situationen und die Fähigkeit zu deren Einschätzung fördert die Anpassungsfähigkeit [Dey01].
Wenn zwei Studenten eine Unterhaltung während einer Vorlesung führen, dann ist Ihnen (meist) bewusst, dass diese Unterhaltung im Kontext „in-der-Vorlesung“ stattfindet. Folglich versuchen sie, wenn sie den Professor und ihre Kommilitonen nicht stören wollen, möglichst unauffällig und leise zu sprechen. Die selbe Unterhaltung hat möglicherweise eine viel höhere Lautstärke, wenn sie den Kontext „bei-einer-Party“ hat. Dies bedeutet implizites Wissen von Menschen über die Umgebung und die Situation, indem eine Aktivität oder eine Kommunikation erfolgt, verschafft den Menschen den Vorteil der aktiven Anpassung an die gegebenen Umstände.
Das Kontextbewusstsein, das in einer natürlichen Form zwischen Menschen genutzt wird, existiert in der Mensch-Maschine- oder gar in der Maschine-Maschine-Interaktion nur rudimentär. Computer besitzen nicht die natürliche Begabung die Umgebung oder Situation, in die eine Aktivität erfolgt, wahrzunehmen, sie zu interpretieren und sich den Umständen entsprechend anzupassen. Im Allgemeinen ist dieses Kontextbewusstsein für stationäre Desktop-Computer nicht in hohem Maß notwendig, da die Umgebung eines stationären Gerätes sich selten ändert und damit eine aktive Anpassung nicht erforderlich ist. Differenzierter ist die Situation für tragbare Computer zu bewerten. Sie befinden sich häufig in wechselnden physikalischen, sozialen und kommunikationstechnischen Umgebungen. Die Wahrnehmung und Interpretation von Umgebungsinformationen, deren prominente Vertreter Zeit und Ort sind, und die Möglichkeit der Adaption des mobilen Gerätes bezüglich ihrer Umwelt birgt ein hohes Potential an neuartigen mobilen Anwendungen und Dienste, die sich deutlich von den bisherigen klassischen Desktop-Anwendungen unterscheiden.
Eine kontextbezogene Anwendung könnte dem Anwender Informationen über die aktuelle Umgebung liefern oder selbst auf eine bestimmte Situation reagieren, d.h. bestimmte Aktionen ausführen, wie beispielsweise die Handylautstärke anpassen oder ein Dokument automatisch auf dem nächsten Drucker drucken. Andere mobile Anwendungen können

beispielsweise eher an der Identität, dem Ort, der Situation eines Benutzers oder andere Personen und Objekte in seiner Nähe interessiert sein. Kontextinformationen können also von unterschiedlichsten Anwendungen genutzt werden und sind daher anwendungsspezifisch. Die Fähigkeit eines (mobilen) Systems sich auf veränderte Bedingungen einzustellen, ist mit einer Vielzahl von Vorteilen verbunden: Angebotene Informationen und Dienste können kontextbezogen selektiert werden. Das Angebot kann davon abhängig sein, wo, zu welcher Zeit und in welcher Situation sich ein Benutzer befindet. Ob er als Tourist oder Geschäftsreisender unterwegs ist, wie schnell und mit welchen Verkehrsmittel er sich bewegt. Die Präsentation der Informationen kann ebenfalls kontextabhängig sein. Beispielsweise sind Informationen für einen Fußgänger in einer anderen Form darzustellen, als die für einen Autofahrer, der mit hoher Geschwindigkeit unterwegs ist.
Ohne Informationen zum aktuellen Kontext müsste der mobile Anwender selbst die erforderlichen Aktionen selektieren und deren Ausführung einleiten. Im letzteren Fall zeigt sich, dass ein Defizit an Kontextbewusstsein nicht nur ein relativ starres und nicht adaptives Verhalten von mobilen Geräten zur Folge hat, sondern auch, dass die Benutzer mit möglichen wiederholenden Routineaufgaben beschäftigt werden. Die Möglichkeit zur automatischen Ausführung von kontextbezogenen Aktionen könnte den Wunsch des Benutzers nach mehr *Benutzerfreundlichkeit* erfüllen. Zusätzlich trägt kontextabhängige Informationspräsentation zu einem *sicheren Umgang* mit mobilen Geräten bei: Ein Autofahrer lässt sich z.B. bei hoher Geschwindigkeit weniger durch die akustischen Hinweise eines Navigationssystems ablenken als durch die detaillierte Landkarte auf dem Borddisplay.
Kontextbewusstes Verhalten verleiht also mobilen Anwendungen nicht nur Anpassungsfähigkeit, sondern erweitert auch die Bandbreite der Mensch-Maschine-Interaktion in einer natürlichen Form, so wie sie bei Menschen von Natur aus gegeben ist [Dey01].
Adaption bedeutet in diesem Zusammenhang allerdings auch, dass einer mobilen Anwendung die Möglichkeit gegeben wird, während der Ausführung (durch Zugriff auf aktuellen Kontextinformationen) zwischen *optionalen Strategien* zu wählen. Beispielsweise kann eine Anwendung mit Hilfe von (Kontext-) Information zur verfügbaren Kommunikationsinfrastruktur der Umgebung autonom entscheiden, ob ein Dokument über das kostenpflichtige GSM-Netz oder über die gerade verfügbare kostenlose WLAN-Verbindung eines Cafés gesendet werden soll. Die Möglichkeit zur Wahrnehmung von optionalen Strategien zur Ausführung einer Aktion erhöht die Chance den eigenen Dienst zu erbringen oder den Dienst eines entfernten Gerätes in Anspruch zu nehmen.
Die Integration von Context-Awareness in mobile Systeme ist ein wichtiger Aspekt des *Ubiquitous* und *Pervasive Computing*. Das Registrieren und Kommunizieren von Informationen über die Umgebung wird begünstigt durch die technischen Entwicklungen der letzten Jahre [RBB03]:
Die rasante Entwicklung der *drahtlosen Kommunikation* für mobile Systeme wie beispielsweise „Wireless cellular networks“ (z.B. GSM-Netz), „Wireless LAN networks“, „Wireless PAN (Personal Area Networks)“, „BAN (Body Area Networks)“ und „Ad-hoc networks“ ermöglicht die Kommunikation von mobilen Endgeräten in verschiedenen Netztopologien.
Vor dem Hintergrund der fortschreitenden Verbreitung der *Sensortechnik* ist es absehbar, dass zukünftig viele Milliarden von Sensoren existieren werden, die kontinuierlich den Zustand der realen Welt erfassen und kommunizieren.
Bei der Forschung auf dem Gebiet der *eingebetteten Systeme* entstehen „smarte“ Alltagsgegenstände, die häufig über sensorische Fähigkeiten verfügen und zudem untereinander vernetzt sind.

Die Entwicklung und Verbreitung von immer leistungsfähigeren *tragbaren Endgeräten* wie Handys, Smartphones, „Personal Digital Assistants“ (PDAs), Organizer, Notebooks und „Wearable Computers“ spielen im Zuge der Miniaturisierung von Computern eine entscheidende Rolle.
Dem heute flächendeckend verfügbare Mobilfunknetz der zweiten Generation ist bereits das leistungsfähigere UMTS-Netz gefolgt. Für die oben genannten „Personal Area Networks“ steht mit der Bluetooth-Technologie die Vernetzung von „intelligenten“ Alltagsdingen zu sehr geringen Kosten zur Verfügung. Einige Multifunktionsgeräte, die Kommunikation-, Rechner- und Sensorfunktionen integrieren, existieren bereits heute.

1.2 Einführendes Beispiel

Das folgende Beispiel dient als Einführung in die Thematik Context-Awareness in mobilen Systemen. Es soll einen ersten Eindruck darüber vermitteln wie die Lokation des Benutzers technisch wahrgenommen werden kann und vor allem *wie* eine kontextbezogene Anwendung diese Informationen nutzt.
Im Rahmen des CoolTown-Projekts [KiBa00, Rot02] realisierte die Firma Hewlett Packard beispielhaft einen elektronischen Museumsführer. Dabei wurde jeder Museumsbesucher beim Eintreten des Museums mit einem mobilen Gerät, z.B. ein PDA, ausgestattet. Unmittelbar in der Nähe jedes Exponats des CoolTown-Museums wurde ein Infrarot-Beacon installiert, die periodisch eine gespeicherte WWW-URL sendet. Die Internetseite zu der gesendeten URL beinhaltet weiterführende Informationen zu dem betreffenden Exponat. Das mobile Gerät integriert einen Infrarotempfänger, der von dem Infrarot-Beacon gesendeten URL in unmittelbarem Sichtkontakt empfangen kann (siehe Abbildung 1.1).

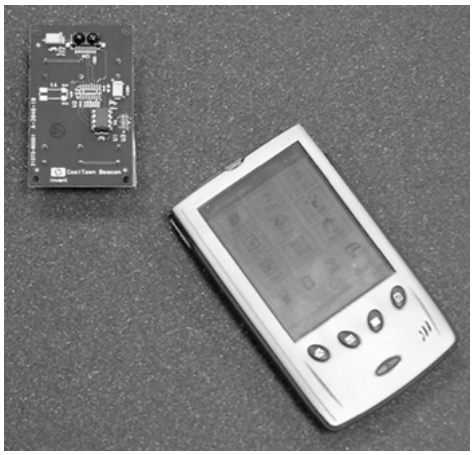


Abbildung 1.1: Mobiles Gerät (Palm PDA) und Infrarot-Beacon des Exponats [KiBa00]

Zusätzlich kann das WLAN-fähiges PDA über das im Museum verfügbares WLAN-Netz drahtlos Internetseiten abrufen und im installierten Webbrowser anzeigen.
Nähert sich nun ein Besucher einem bestimmten Exponat und befindet sich in der begrenzten Reichweite des Infrarotsignals, kann das tragbare Gerät die gesendete URL empfangen und automatisch dessen Inhalt anzeigen (siehe Abbildung 1.2).

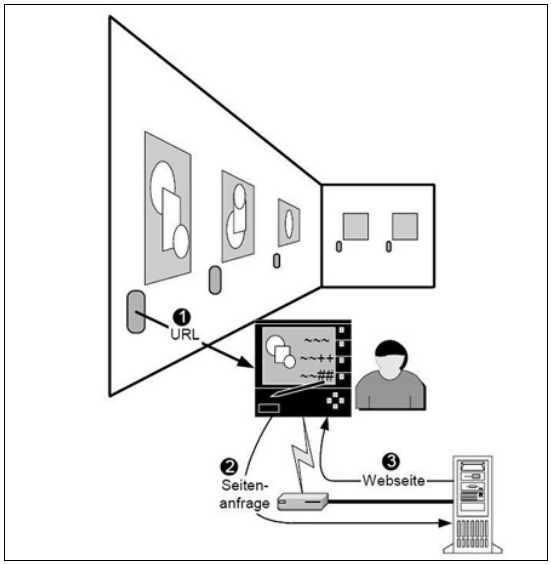


Abbildung 1.2: Datenübertragung im CoolTown-Museumsführer [Rot02]

Der Benutzer kann auch Bookmarks auf die Web-Seiten setzen, um sie später zu lesen. Am Ende eines Rundgangs wird automatisch eine Historie über angesehene Objekte und dazugehörige Web-Seiten erstellt, die der Benutzer als Erinnerung ausdrucken kann.
Der CoolTown-Museumsführer ist ein relativ einfaches Beispiel für die Realisierung eines kontextbezogenen Systems, die nur die Position des Benutzers als Kontextinformation benutzt, um ortsspezifische Informationen abzurufen. Das benutzerfreundliche Verhalten des Systems zeigt sich dadurch, indem es nur solche Informationen zum Anzeigen selektiert, die in der aktuellen Umgebung des Benutzers relevant sind.

1.3 Ziel dieser Arbeit

Gegenstand dieser Diplomarbeit ist es zu untersuchen, ob und inwieweit sich ein *generisches*, *erweiterbares* und *anwendungensunabhängiges* Kontextmodell und Kontext-Management-System für eine Middleware in einem verteilten mobilen System entwickeln lässt.
Der generische Ansatz bedeutet in diesem Zusammenhang, dass das entwickelte Modell von den konkreten Kontextinformationen (wie beispielsweise Ort, Zeit, Temperatur, Geschwindigkeit, usw.) und deren Darstellung abstrahiert: Nicht die Details zu den unterschiedlichen Informationstypen sind im Modell relevant, sondern ihre gemeinsame Eigenschaften.
Das Attribut Erweiterbarkeit impliziert, dass die Semantik des Modells nicht auf bestimmte Kontextinformationstypen beschränkt sein soll. Beispielsweise sollten Informationen über die umgebende Kommunikationsinfrastruktur genauso auf das Modell abbildbar sein wie Aktivitäten des Benutzers oder physikalische Eigenschaften eines Raumes.
Eine Middleware abstrahiert von einer konkreten Anwendung, der sie benutzt. Entsprechend soll das Modell von einem bestimmten Typ einer kontextbezogenen Anwendung unabhängig sein. Es wird davon ausgegangen, dass verschiedene mobile Anwendungen Kontextinformationen in unterschiedlicher Art und Weise benutzen, um sich anzupassen.
Das Kontext-Management-System als zweite Systemkomponente (neben dem Kontextmodell) hat das Ziel das Modell zu verwalten und adäquate Zugriffsmechanismen auf die Modelldaten bereitzustellen.

Das Primärziel zur Integration von Context-Awareness mit dem oben beschriebenen Modell und Management-System setzt die Erreichung folgender Teilziele voraus: Es besteht zunächst die Notwendigkeit zu einer genauen *Definition des Kontextbegriffs* und von *kontextbezogenen Anwendungen* im Rahmen des Mobile Computing. Anschließend sind bisherige kontextbezogene Anwendungen zu untersuchen und nach bestimmten Merkmalen zu sortieren. Aus der *Klassifizierung bisheriger Anwendungen* sollen die unterschiedlichsten Ausprägungen von Kontextinformationen in mobilen Systemen extrahiert werden. Dies führt, wie bei der Klassifizierung von kontextbezogenen Anwendungen, zu einer systematischen *Kategorisierung von Kontextinformationen* nach bestimmten Eigenschaften. Als ein weiteres Teilziel sind bisher entwickelte *Modelle und Management-Systeme* im Bereich Context-Awareness zu analysieren und zu bewerten. Darauf aufbauend lassen sich konkrete *Anforderungen* für eine prototypische Umsetzung des Kontextkonzepts als *Context Service* in der DEMAC-Architektur [Kun05] ableiten.

Informationen, die der *Context Service* zu modellieren und bereitzustellen hat, wären beispielsweise folgende:

- Wo befindet sich das mobile Gerät gerade?
- Welche mobile Geräte und Benutzer sind momentan erreichbar?
- Welche Identität haben die oben genannten Entitäten?
- Wo befinden sich andere erreichbare Geräte?
- Mit welchen Aktivitäten sind diese gerade beschäftigt?
- Auf welche Dienste kann ich in meiner Umgebung zugreifen?
- QoS (Quality of Service) Parameter verfügbarer Kommunikationsverbindungen: Informationen über Bandbreite, Konnektivität, usw.
- Allgemeine Informationen über das mobile Gerät: Art des Gerät, verfügbare Ressourcen (Prozessorleistung, Speicher), Batteriezustand, Bildschirmgröße, usw.

1.4 Eingrenzung des Themas

Obwohl die Integration von Kontextinformationen in den heutigen Informationssystemen relativ rudimentär ist, und damit keineswegs zum „Alltag“ gehört, existiert doch seit einigen Jahren eine fundierte theoretische Grundlage zur Entwicklung kontextbezogener Systeme in der Informatik. Aufbauend auf diesen Grundlagen haben sich inzwischen kontextbezogene Systeme mit unterschiedlichsten Ansätzen und Ausprägungen entwickelt.
In dieser Arbeit geht es nicht um die Entwicklung einer bestimmten Anwendung, dessen Verhalten mit Hilfe von Kontextinformationen zu beeinflussen gilt, sondern, um eine generische Lösung für möglichst viele mobile Anwendungen, die Informationen über ihre Ausführungsumgebung erhalten wollen. Daher soll primär auf das Thema Context-Awareness als ein Charakteristikum einer Middleware im Mobile Computing, die beliebige mobile Anwendungen unterstützt, eingegangen werden.
Die Thematik fokussiert mehr die abstrakten Komponenten einer mobilen Umgebung, deren Eigenschaften und semantische Beziehungen zueinander, als die Bereitstellung von konkreten „Low-Level-Kontextinformationen“ wie beispielsweise Ort, Zeit, Temperatur oder Geschwindigkeit aus logischen oder physikalischen Sensoren. Die Beschaffung der Informationen über die aktuelle Umgebung eines Gerätes oder eines Benutzers nimmt hier

eine sekundäre Stellung ein. Vordergründig stehen die Entwicklung eines abstrakten (Daten-) Modells, in das Kontextinformationen effizient ausgedrückt werden, und einem Management-System, dass dieses Modell verwaltet, im Blickpunkt.

1.5 Vorgehensweise

Das folgende Kapitel behandelt die Grundlagen kontextbezogener Systeme im Mobile Computing. Nach einer allgemeinen Einführung in das Thema Mobile Computing werden die im Rahmen dieser Arbeit essentielle Begriffe definiert. Der Fokus ist darauf aufgerichtet, dass sowohl wichtige Kontextbegriffe bezogen auf das Mobile Computing systematisch geklärt werden, als auch die historische Entwicklung der Begriffsbildung berücksichtigt wird. Es folgen dann einige Beispiele für mobile Anwendungen, welche Umgebungsinformationen nutzen, um sich einer Situation anzupassen. Anschließend werden wichtige Charakteristika von Kontextinformationen, die Kategorisierung von Kontexttypen, sowie die Klassifizierung von unterschiedlichen kontextbezogenen Anwendungen erörtert. Danach sollen verschiedene Ansätze bisher vorhandener Kontextmodelle und Frameworks dargelegt werden.
Aufbauend auf dem Grundverständnis für das Thema, befasst sich das vierte Kapitel mit der Idee eines generischen Konzepts zur Integration von Context-Awareness. Dabei sind zunächst Vorteile und Nachteile bisheriger Konzepte zu diskutieren. Davon ausgehend dann konkrete Anforderungen an das Modell zu stellen. Anschließend werden das Modell selbst, die Komponenten zum Management von Kontextinformationen und deren Semantik beschrieben.
Das im vorherigen Kapitel entwickeltes abstraktes Konzept und Modell bieten die Grundlage für eine prototypische Implementierung des Context Service innerhalb des DEMAC-Framework in der Programmiersprache Java.
Die Validierung des implementierten Context Service anhand einer kontextbezogenen mobilen Anwendung ist Gegenstand des sechsten Kapitels. Eine Zusammenfassung über diese Arbeit und einen Ausblick über zukünftige Entwicklungen gibt das siebte und letzte Kapitel.

2 Grundlagen kontextbezogener mobiler Systeme

Bevor ein eigenes Konzept zur Abbildung und Verwaltung von Kontextinformationen in einer Middleware für mobile Systeme entwickelt werden kann, müssen grundlegende Definitionen, Technologien und Konzepte untersucht werden. Dazu gehören vor allem auch die Beschreibung und Bewertung von Ansätzen anderer Autoren bezüglich der Integration von Context-Awareness in einem mobilen System.

2.1 Einführung ins Mobile Computing

Da im Rahmen dieser Arbeit die Integration von Kontextinformationen ausschließlich mobile Systeme betreffen, ist ein Einblick in das *Mobile Computing* notwendig. Im Folgenden werden wichtige Begriffe im Umfeld des Mobile Computing, Arten der Mobilität, Restriktionen gegenüber der traditionellen Desktop-Computern und Einsatzgebiete von mobilen Geräten und Mobilkommunikation kurz dargestellt. Außerdem ist es im Rahmen dieser Arbeit besonders wichtig zu präzisieren, in welchem technologischen Umfeld der Begriff Context-Awareness sich bewegt.

2.1.1 Einleitung

Die gegenwärtige allgemeine Vorstellung eines Computers reduziert sich meist auf das Bild eines PC. Er besteht aus einem relativ großen Rechner, einem Monitor, einer Maus und einer Tastatur, die üblicherweise ihren festen Platz auf dem Schreibtisch haben und von dort aus nur schwer zu bewegen sind. Diese gängige Vorstellung eines PCs befindet sich, nach Meinung einiger Experten, gerade in einem radikalen Wandel. Nach Ansicht vieler Autoren sei die Zeit dafür reif, dass die heutigen traditionellen PCs ihren Charakter als ein stationäres System ablegen müssen. Tragbar integriert in die Umgebung des mobilen Benutzers sollen sie unabhängig von einem bestimmten Platz verwendet werden können [Rot02].
Gründe für dieses Bedürfnis der Menschen liegt auf der Hand: Sie sind zunehmend sowohl aus beruflichen als auch aus privaten Gründen mobil. Es werden zum Teil weite Entfernungen auf den Weg zur Arbeit, während einer Dienstreise oder in der Freizeit zurückgelegt. Oft wollen die Benutzer während der Reisezeit erreichbar sein, die Reisezeit effektiv nutzen oder gewünschte Informationen unterwegs abrufen. Der Trend bei der Computernutzung (in Zeiten der Globalisierung) geht zur *Anytime* und *Anywhere Computing* [Dol03].
Schon heute werden im Zuge der Entwicklung von leistungsfähigen tragbaren Endgeräten wie Notebooks, Handys, Smartphones, Handhelds und PDAs einerseits und der Verbreitung der drahtlosen Kommunikation andererseits erste Ansätze zur Erfüllung dieser Anforderungen erkennbar. Dass miniaturisierte mobile Computer weltweit untereinander vernetzt sein werden, scheint angesichts der heutigen technischen Möglichkeiten nur noch eine Frage der Zeit zu sein. Der „Handyboom“ im ausgehenden 20. Jahrhundert zeigt wie schnell sich eine mobile Technologie verbreiten kann [Rot02]. Eine Auswahl existierender mobiler Geräte zeigt Abbildung 2.1:

Abbildung 2.1: Mobile Endgeräte [Dol03]

Mobile Computing als Forschungsgebiet beschäftigt sich nicht nur mit mobilen Endgeräten, sondern auch mit der kommunikationstechnischen Infrastruktur in solch einem Umfeld (vgl. Abschnitt 2.1.2, *Mobilkommunikation*) und mobilen Anwendungen, die sich von den bisherigen traditionellen Desktop-Anwendungen in einigen Aspekten unterscheiden (vgl. Abschnitt 2.1.4). Das folgende Szenario, übernommen aus [Rot02], verdeutlicht die unterschiedlichen Aspekte des Mobile Computing in der Informatik:

> *„Eine Mitarbeiterin einer Firma hat den Auftrag, beim Kunden eine Präsentation zu geben. Vor der Reise wurden die Präsentation sowie einige Textdokumente auf ein Notebook überspielt. Während der Reise im Zug überarbeitet sie noch die Präsentation. Am Zielort ruft sie die lokale Städteinformation ab und sucht ein Hotel einer gewünschten Lage und Preiskategorie. Bezahlt wird mit elektronischen Geld. Auf einem elektronischen Stadtplan kann sie sich den Weg zum Kunden darstellen lassen. Dort angekommen, kann sie ihre Präsentation auf dem Videoprojektor des Kunden abspielen lassen. Handzettel werden auf dem lokalen Drucker ausgedruckt. Zur Klärung konkreter Fragen mit dem Kunden werden aktuelle Informationen aus dem Heimnetzwerk geladen. Auf dem Rückweg liest die Mitarbeiterin noch ihre neuen E-Mails. Zu Hause wird das Notebook mit dem Firmennetzwerk verbunden – automatisch werden die geänderten Dokumente und Präsentationen mit dem zentralen Datenbestand abgeglichen."* [Rot02, Seite 3]

2.1.2 Begriffsklärung

Das im vorherigen Abschnitt geschilderte Szenario enthält viele Konzepte, die zum Forschungsgebiet des Mobile Computing gehören. Jörg Roth hat in [Rot02] die für das Mobile Computing essentiellen Begriffe klassifiziert und prägnant erklärt. Die wichtigsten Begriffe, die die Konzepte im oben geschilderten Szenario umfassen, werden im Folgenden erklärt.

Ubiquitous Computing

Im Zusammenhang mit Mobile Computing fällt oft der Begriff *Ubiquitous* (deutsch: *ubiquitär* = „allgegenwärtig", „überall verbreitet" [Dud97, Seite 833]) *Computing*. Er bezeichnet die allgegenwärtige und überall vorhandene Integration und Vernetzung von „intelligenten Gegenständen" in der Umgebung eines Menschen [Wei93].
Entstanden und geprägt ist dieser Begriff durch die Vision von Mark Weiser in einem oft zitierten Artikel in der „Scientific American" 1991 [Wei91]. Bei genauer Betrachtung seines Ansatzes [Wei93] definierte Weiser Ubiquitous Computing folgendermaßen:

> *„... the method of enhancing computer use by making many computers available throughout the physical environment, but making them effectively invisible to the user.“* [Wei93, Seite 1]

Weiser unterscheidet drei verschiedene Phasen der Computernutzung (siehe Grafik in Abbildung 2.2). Die erste Phase ist charakterisiert durch die Benutzung von großen, teueren und für wenige Menschen verfügbaren Mainframe-Rechnern am Anfang der Computerentwicklung. Anschließend folgte die Ära der Personal Computer – die heutigen PCs. Die Computer wurden wesentlich kleiner, billiger und leistungsfähiger. Fast jeder Haushalt (zumindest solche in Industriestaaten) kann sich heute einen „persönlichen Rechner“ leisten. Die dritte Phase, zu deren Schwelle wir uns heute nach Meinung von Weiser befinden, ist die des Ubiquitous Computing: Dabei sind autonome Rechner in vielen „Alltagsdingen“ integriert und kommunizieren mit drahtloser Verbindung untereinander. Rechner sollen dabei für die Menschen nicht explizit als solche wahrgenommen werden. Als unsichtbarer Bestandteil von Gegenständen – weshalb in diesem Zusammenhang auch die Begriffe *Invisible Computing* und *Disapearing Computing* genannt werden - sollten sie die Nutzer bei der Arbeit und in der Freizeit unterstützen, Aufgaben zu erledigen [Wei93].

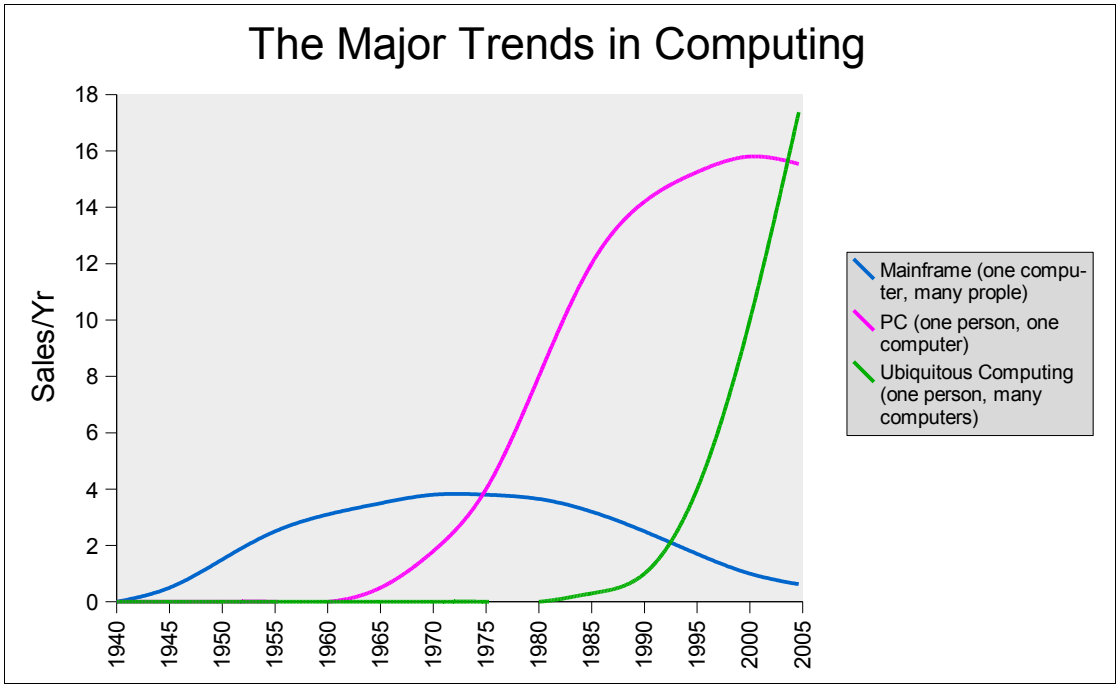


Abbildung 2.2: Trends in der Computernutzung (nach [Wei96])

Ubiquitous Computing wird oft als ein humanistischer Ansatz verstanden, in dem Computer – in der Form wie wir sie heute kennen – nicht mehr sichtbar sind, sondern in der natürliche Umgebung der Menschen interaktiv agieren. Statt selbst Gegenstand der menschlichen Aufmerksamkeit zu sein, sollen Computer unmerklich Tätigkeiten des alltäglichen Leben unterstützen [Rot02].

Pervasive Computing

Pervasive Computing (lat. Pervadere = „Durchdringen“) ist die industrielle Ausprägung des Ubiquitous Computing. Hier steht die durchdringende Vernetzung von „intelligenten Gegenständen“ mit Mittelpunkt.
Von vielen Autoren wird das Pervasive Computing als Synonym für Ubiquitous Computing verwendet. Aber im Gegensatz zu Ubiquitous Computing fokussiert Pervasive Computing nicht die menschliche Integration von Computern in der Umgebung, sondern versucht mit zum Teil heute verfügbarer Technik Geschäftsprozesse und allgemeine Lebensbereiche zu durchdringen. Hansmann et al. fassen wichtige Aspekte des Pervasive Computing zusammen:

> *„'Everywhere at anytime' - This common slogan expresses in a nutshell the goal of Pervasive or Ubiquitous Computing. Both terms describe the visible and mobile front-end for the next generation of integrated IT applications. Pervasive Computing includes flexible and mobile devices like personal digital assistants, mobile phones, pagers, hand-held organizers and home entertainment systems, which will access or provide a rich diversity of applications.“* [HMNS03, Seite 1]

Nomadic Computing

Im Forschungsgebiet des Mobile Computing existiert eine enorme Vielfalt von mobilen Endgeräten, Kommunikationsnetzen und (Betriebssystem-) Plattformen. Diese unterschiedlichen Systeme müssen in der Umgebung des mobilen Benutzers zusammenarbeiten. Das *Nomadic Computing* legt deshalb seinen Schwerpunkt im Wesentlichem – noch mehr als Mobile Computing – *„auf die Mobilität des Anwenders, während Mobile Computing auch Probleme behandelt, die sich am Zielort nach einer Reise ergeben“* [Rot02, Seite 6].
Das Nomadic Computing beschäftigt sich vor allem damit, wie die Heterogenität der Systemkomponenten im Mobile Computing aus Sicht des nomadischen Benutzers *transparent* zu überwinden ist [Kle96].

Ad-hoc Networks und Mobile Networking

Oft existiert in der Umgebung mobiler Teilnehmer keine feste Kommunikationsinfrastruktur, mit dessen Hilfe Daten ausgetauscht werden können. Um ohne eine feste Infrastruktur eine Kommunikation zwischen mehreren mobilen Endgeräten zu ermöglichen, wird spontan (ad-hoc) ein Netz aufgebaut. Ein *Ad-hoc-Netzwerk* bezeichnet also ein drahtloses Netz zwischen zwei oder mehr mobilen Teilnehmern, das ohne aufwendige Konfiguration auskommt [Rot02]. Diese Technik wird beispielsweise bei Bluetooth verwendet, um eine spontane Kopplung zwischen Mobiltelefon und Headsets zu erreichen.
Die geschlossene und kurzfristige Vernetzung von autonomen *mobilen* Teilnehmern wird in der Literatur auch als *MANET* (Mobile Ad-hoc Networks), *Instant Infrastructure* oder *Mobile mesh Networking* bezeichnet. Sie stimmen alle weitgehend mit Begriff Ad-hoc-Netzwerk überein.
Die Ad-hoc Vernetzung schließt streng genommen die zusätzliche Kopplung von mobilen Teilnehmern mit einem Weitverkehrsnetz, wie zum Beispiel das Internet, aus. Diese

Kopplung kann aber innerhalb eines Ad-hoc-Netzwerks notwendig sein, um gewünschte Daten und Dienste abzurufen. Der Begriff *Mobile Networking* bezeichnet den Zusammenschluss von Ad-hoc-Netzwerk mit mobilen Rechnern untereinander und die Vernetzung von mobilen Geräten mit einem stationären Netzwerk [Rot02].

Embedded Networking

Die Vision der eingebetteten (englisch: embedded) Systeme ist es Haushalts-, Konsum- und Unterhaltungsgeräten in naher Zukunft die Möglichkeit zu geben, ähnlich wie beispielsweise Rechner und Drucker, Daten zu verarbeiten und untereinander zu kommunizieren. Kleine integrierte Rechner in verschiedenen elektronischen Geräten sollen so zu Steuerungs- und Kontrollzwecken Daten verarbeiten können. Kommt es zum Zwecke der Kommunikation zwischen diesen Geräten zu einer Vernetzung, so spricht man auch von *Embedded Networking* [Rot02].
Ein klassisches Beispiel, die mit *embedded Systems* in Zusammenhang gebracht wird, ist der „Internet-taugliche" Kühlschrank. Er bestellt automatisch Nahrungsmittel über dem Internet, die vom Benutzer preferiert werden und nicht mehr im Kühlschrank vorhanden sind.

Mobile Communication und Wireless Communication

Während der Begriff Mobile Computing unter anderem auch mobile Anwendungen und Geräte umfasst, beschäftigt sich das *Mobile Communication* (deutsch: Mobilkommunikation) lediglich mit der reinen Kommunikation. Eng verwandt mit Mobil Communication ist der Begriff der *Wireless Communication* (deutsch: drahtlose Kommunikation). Dennoch gibt es signifikante Unterschiede zwischen diesen Begriffen [Rot02].
Die drahtlose Kommunikation fokussiert die Art der Anbindung von mobilen Endgeräten - zum Beispiel über Funk-, Infrarot- oder Bluetooth-Schnittstelle. Die Mobilkommunikation setzt allerdings seinen Schwerpunkt nicht auf die Art der Anbindung, sondern eher auf die Mobilität des Gerätes zwischen verschiedenen Netzwerken. Die folgende Tabelle, übernommen aus [Rot02], zeigt einige Beispiele für mobile und drahtlose Kommunikation.

Tabelle 2.1: Beispiele für mobile und drahtlose Kommunikation (nach [Rot02])

	Nichtmobile Kommunikation	**Mobile Kommunikation**
Drahtgebunde Kommunikation	Workstation in einer Büroumgebung	Notebook im Hotelzimmer, angebunden über Modem
Drahtlose Kommunikation	Workstation im drahtlosen lokalen Netz	Notebook über Mobiltelefon drahtlos angebunden

2.1.3 Arten der Mobilität

Als *Mobilität* wird die Beweglichkeit von jemand oder etwas bezeichnet. Murphy et al. [MPR01] klassifizieren zwei Typen von Mobilität - physikalische und logische Mobilität - während Pandya [Pan00] und Roth [Rot02] zwischen drei verschiedenen Arten der Mobilität in der Mobilkommunikation unterscheiden – Endgerät-, Benutzer- und Dienstmobilität:

Physikalische und logische Mobilität

Die physikalische Mobilität ist der (unbeschwerliche) Transport (Bewegung) von physikalischen Entitäten. Die logische Mobilität dagegen bezeichnet die Beweglichkeit von Programmcode (z.B. bei der Migration von mobiler Programmcode von einem Rechner zum anderen, mobile Softwareagenten, ect.) innerhalb eines verteilten Informationssystems [MPR01].

Endgerätmobilität

Die Endgerätmobilität betrifft die physische Mobilität von tragbaren Endgeräten. Sie tritt dann auf, wenn Endgeräte unabhängig von ihrem Aufenthaltsort vernetzt bleiben. Die drahtlose Kommunikation wird dabei in der Regel vorausgesetzt [Pan00].
Ein Beispiel für ein Kommunikationsnetz das Endgerätmobilität gewährleistet, ist das GSM-Mobilfunknetz. Ein Gebiet, das durch das GSM-Netz abgedeckt ist und wo Funkübertragung möglich ist, erlaubt mobilen Benutzern das Mobiltelefon zu transportieren, ohne dass die Verbindung zum Netz abbricht (siehe Abbildung 2.3 b).
Book et al. [BGHS05] unterteilen die Mobilität des Endgerätes in weitere vier Grade:

- Ein *lokal* funktionierendes Gerät kann sich nicht mit dem Netzwerk verbinden.
- Ein *verteilt* funktionierendes Gerät kann sich mit dem Netzwerk verbinden.
- Ein *mobil* funktionierendes Gerät kann sich mit verschiedenen Netzwerkzugangspunkten verbinden.
- Ein *in Bewegung* funktionierendes Gerät kann sich mit verschiedenen Netzwerkzugangspunkten verbinden, während sein Benutzer es verwendet.

Benutzermobilität

Ein mobiler Benutzer kann verschiedene Geräte in Anspruch nehmen, um bestimmte Dienste der Kommunikationsnetze abzurufen. Hierzu muss ein Identifikationsmerkmal vorhanden sein (z.B. Fingerabdruck, Geheimcode, Chipkarte, ect.), um den jeweiligen Benutzer am Gerät und am benutzen Netz eindeutig zu identifizieren [Rot02].
Beispielsweise kann ein Benutzer die SIM-Karte eines GSM-Netzes benutzen, um von unterschiedlichen Mobiltelefonen aus in dieses Netz zu telefonieren (siehe Abbildung 2.3 c).

Dienstmobilität

Unter Dienstmobilität versteht man die Verwendung von Kommunikationsdiensten unabhängig vom Aufenthaltsort des Benutzers [Pan00]. Ein mobiler Anwender kann beispielsweise aufgrund der Dienstmobilität von jedem Punkt der Erde seine E-Mails abrufen (siehe Abbildung 2.3 d).

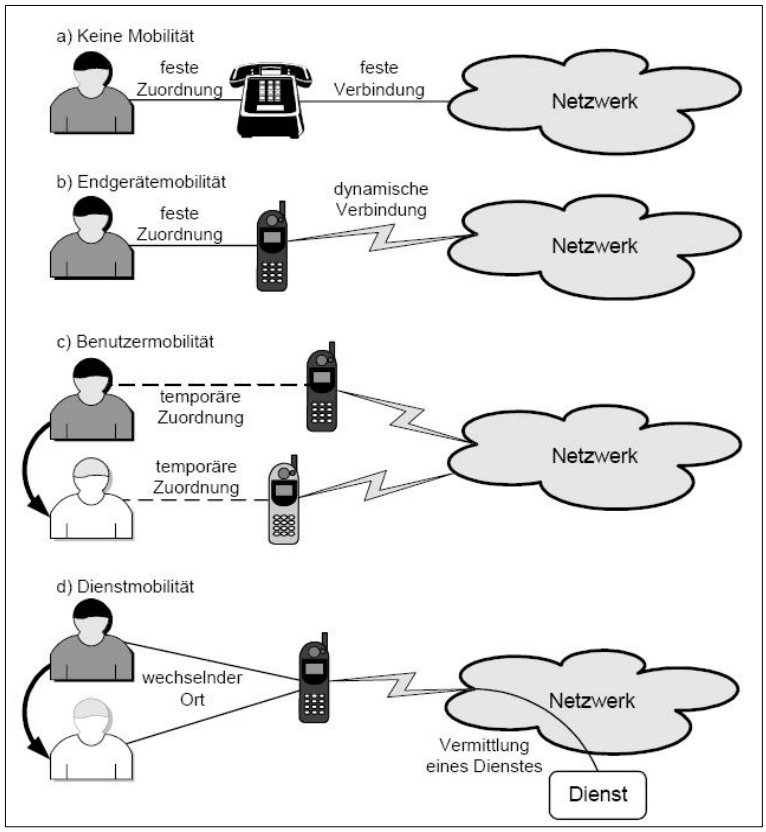


Abbildung 2.3: Arten der Mobilität [Rot02]

2.1.4 Mobile Geräte und Anwendungen

Mobile Endgeräte umfassen ein sehr breites Spektrum von tragbaren Computern, die sich von den herkömmlichen PCs deutlich unterscheiden. Einen detaillierten Übersicht über vorhandene mobile Endgeräte gibt [RuSi03]:

- Mobiltelefon, Smartphone (langsame 16 oder 32bit CPU, 50KB bis 4MB Speicher)
- Personal Digital Assistant (PDA) (schnelle 32bit CPU, 32MB bis 64MB Speicher)
- Tablet PC (schnelle 32 oder 64bit CPU, mindestens 32MB Speicher)
- Notebooks, Laptops (schnelle 32 oder 64bit CPU, mindestens 256MB Speicher)

Ergänzend zu den oben genannten Geräten seien auch Bordcomputer in Fahrzeugen und *Wearable Computers*, die als Teil der Kleidung getragen werden oder in Kleidern integriert sind, erwähnt.
Eine ausführlichere Beschreibung und Klassifizierung der vorhandenen mobilen Geräte nimmt Müller-Wilken in seiner Dissertation [Mül02] vor. Dabei werden mobile Geräte nicht anhand definierter Kriterien klassifiziert – so wie bei Weiser [Wei91, Wei93] und Schiller [Sch00] – sondern Geräteklassen durch Beispiele gebildet.
Roth dagegen klassifiziert mobile Geräte nach ihrer Nutzungsart. Geräte, die keinem bestimmten, festgelegten Zweck zugeordnet sind, können nahezu beliebig mit verschiedenen Anwendungen betrieben und erweitert werden. Sie werden von Roth als *universelle* mobile Geräte bezeichnet. Im Gegensatz dazu sind *Spezialgeräte* zu einem bestimmten Zweck

konstruiert und lassen sich auch technisch kaum erweitern. Zu Verdeutlichung zeigt Tabelle 2.2 aus [Rot02] die Einordnung der mobilen Geräte in diesem Raster:

Tabelle 2.2: Klassifikation mobiler Endgeräte (nach [Rot02])

Kategorie	Universalgerät	Spezialgerät
Mobile Standard-computer	Notebook	Spezielle mobile Computer, z.B. in der Vermessungs-technik, Kartografie und Archäologie
Bordcomputer	-	Bordcomputer in Schiffen Fahr- und Flugzeugen, Computer in Satelliten
Handhelds	PDA	Elektronische Kalender (nicht programmierbar), Lesestift, E-Book, Web-Pad, mobiles Datenerfassungstermin al in der Lagerhaltung, GPS-Empfänger, Mobiltelefon, Pager, Digitalkamera
	Smartphone, Communicator, Mobile Spielkonsole, Programmierbare Taschenrechner	
Wearables	Programmierbares Wearable	Armbanduhr, Pulsmesser
Chipkarten	Smart Card	SIM-Karte, EC-Karte mit Bargeldfunktion, Telefonkarte, Identifikation zur Zeiterfassung, Karte für digitale Unterschrift

Die Zeile mit „Smartphone, Communicator, usw." erstreckt über die Spalte der Universal- und Spezialgeräte. Das hängt damit zusammen, dass diese Produkte abhängig von ihrer Ausführungsform sowohl als Spezialgerät als auch als Universalgerät klassifiziert werden können. Beispielsweise kann eine mobile Spielkonsole als Spezialgerät dienen, indem sie nur mit einer bestimmten Anzahl von Spielen betrieben werden kann, oder als Universalgerät, die eine Programmierschnittstelle zur Entwicklung neuer Spiele bietet [Rot02].
Im Rahmen dieser Arbeit steht die Entwicklung eines Kontextkonzepts für eine Middleware in mobilen Systemen im Vordergrund. Deshalb sind für diese Arbeit programmierbare, mobile Universalgeräte von größerer Bedeutung als nicht-erweiterungsfähige Spezialgeräte, für die eine Middleware meist nicht verwendet wird.

Nach [DiHe95] ist eine mobile Anwendung vor allem dadurch charakterisiert, dass sie *„überall dort, wo sich Menschen zur Erledigung ihrer Aufgaben frei bewegen müssen und dennoch aktuell und flexibel über verschiedenste Informationen verfügen möchten“* zum Einsatz kommt.
Typische Anwendungsbereiche für Anwendungen im Mobile Computing zählt Harbeck in seiner Diplomarbeit [Har04] auf: Dazu gehören vor allem ortsabhängige (z.B. Navigationssysteme und Touristenführer) und allgemeine (z.B. Nachrichten und Aktienkurse) *Informationsdienste*, Systeme für den *elektronischen Handel* (z.B. Mobile Banking oder Mobile Payment) und *Unterhaltungssoftware* (z.B. Multiplayer-Spiele für Handys) [Har04].

2.1.5 Randbedingungen im Mobile Computing

Die Mobilität bringt wesentliche Einschränkungen bei der Verwendung von mobilen Geräten. In diesem Kontext existieren einige signifikante Unterschiede gegenüber traditionellen Desktop-Computern, die entscheidende Auswirkungen auf die Benutzbarkeit von mobilen Endgeräten haben. Es sind nicht nur mobile Geräte selbst von diesen Restriktionen betroffen, sondern auch mobile Anwendungen und die Mobilkommunikation. Konsequenterweise müssen Entwickler von Kommunikationsnetzen für mobile Systeme, mobilen Endgeräten und Anwendungen diese Einschränkungen in der Entwicklung ihrer Systeme berücksichtigen.
Satyanarayanan klassifiziert in [Sat96] einige Restriktionen, welche das Mobile Computing charakterisieren:

Ressourcenknappheit

Verglichen zu stationären Computern sind tragbare Geräte - wie PDAs und Smartphones - mit relativ *geringer Prozessorleistung* und *Speicherkapazität* (sowohl Arbeits- als auch Festplattenspeicher) ausgestattet. Zudem sorgt ein *endlicher Energievorrat* dafür, dass das Gerät nicht beliebig lange mit der Batterieleistung betrieben werden kann.

Sicherheit

Da mobile Geräte einerseits oft persönliche Daten wie beispielsweise Benutzerinformationen, Termine, und Notizen enthalten (*Personal Computing*), und andererseits häufig in verschiedenen (zum Teil unbekannten) Netzen agieren, besteht ein erhöhtes Sicherheitsbedürfnis gegenüber stationären Rechnern. Die im Mobile Computing eingesetzte drahtlose Kommunikation erhöht zudem das Risiko, dass sensitive Daten während der Übertragung abgehört werden [Sat96].
Weiterhin können (teure) tragbare Geräte leichter verloren gehen, entwendet oder beschädigt werden als vergleichsweise große PCs, die im Büro oder zu Hause einfach eingeschlossen sind.

Variierende Kommunikationsbedingungen

Das Pervasive Computing betont die allgegenwärtige Verfügbarkeit von Kommunikationsnetzen und Dienste. Dieser Ansatz führt meist jedoch zu einer enormen Heterogenität der Netzen mit ihren verschiedenen Eigenschaften. So kann beispielsweise die Leistung von Netzen hinsichtlich der *QoS* (Quality of Services) von Verbindungen - abhängig von der aktuellen Umgebung des Benutzers - stark variieren. Beispielsweise kann in einem Café ein

leistungsfähiges WLAN zur Verfügung stehen, während im Freien meistens nur über das kostenpflichtige und langsamere UMTS- und GSM-Netz Daten ausgetauscht werden können. Zudem kann die *Konnektivität* temporär eingeschränkt sein und es damit zu unerwarteten *Verbindungsabbrüchen* kommen, die von mobilen Geräten und Anwendungen berücksichtigt werden müssen.

Heterogenität von Endgeräten

Die Idee des Ubiquitous Computing sieht eine Integration von Computern in alltäglichen Dingen vor. Diese Dinge unterscheiden sich enorm von einander. Schon heute reicht die Bandbreite existierender mobiler Endgeräte von PDAs und Mobiltelefone (siehe Abbildung 2.1) bis hin zu *Wearable Computers*, die meistens verschiedene Ausführungsplattform für mobile Anwendungen bieten. Diese Heterogenität macht es relativ schwierig mobile Anwendungen zu entwickeln, die auf möglichst vielen Geräten lauffähig sind.
Die Einschränkungen in der *Benutzerschnittstelle* (eingeschränkte Eingabe- und Ausgabemöglichkeit) verglichen zu stationären Desktop-Systemen sind ebenso bei der Entwicklung von entsprechenden Anwendungen zu bedenken.

2.2 Definitionen

Im Zuge der Entwicklung von kontextbezogen Systemen entstanden zahlreiche Begriffe und Definitionen. Die genaue Erläuterung dieser Begriffe mit der historischen Begriffsfindung ist für das weitere Verständnis dieser Arbeit maßgebend. Die Autoren, die die Fachbegriffe geprägt haben, fokussieren jeweils unterschiedliche Aspekte. Diese Aspekte und die zum Teil unterschiedlichen Ansätzen der Autoren werden im folgenden Abschnitt dargelegt.

2.2.1 Kontext im Allgemeinen

Eine allgemeine Definition des Terminus Kontext liefert das „Merriam-Websters Online Dictionary“ [Web05]:

> *„The interrelated conditions in which something exists or occurs.“* [Web05, siehe Link]

Die allgemeine Bedeutung des Wortes Kontext sieht Kontext als eine wechselseitige Beziehung zwischen verschiedenen Bedingungen einer Situation, in der sich jemand oder etwas befindet, oder in dem etwas passiert. Für das Duden Fremdwörterbuch [Dud97] ist Kontext *„der umgebende, inhaltliche (Gedanken-, Sinn-, Sach-, Situations-) Zusammenhang“* [Dud97, Seite 442].
Aus der eben genannten Definition geht hervor, dass im Rahmen des Terminus Kontext die Begriffe *Bedingung*, *Zusammenhang* und *Umgebung* eine wichtige Rolle spielen. Trotzdem ist die allgemeine Definition des Kontexts zu wage, um in der Informatik bzw. im Mobile Computing, unzweideutig verwendet zu werden. Aufgrund dessen haben sich Fachautoren damit befasst, die Semantik des Begriffs Kontext im Umfeld eines mobilen Benutzers und einer mobilen Anwendung zu erörtern.

2.2.2 Kontext im Mobile Computing

Um die Unschärfe des allgemeinen Kontextbegriffs zu beseitigen, bedarf es einer neuen Definition. Dies erkannten auch die ersten Entwickler von kontextbezogenen Systemen. Da der allgemeine Kontextbegriff zu abstrakt war, versuchten einige Autoren den Kontext

zunächst durch die *Aufzählung* enthaltener Attribute zu definieren. Ein Beispiel dafür ist der Guideline „ISO 13407“ [ISO99]. Er versucht unter anderem in standardisierter Form Entwicklern von interaktiven Systemen Richtlinien vorzugeben, die festlegen welche Kontextelemente bei der Entwicklung von kontextsensitiven Systemen relevant sind. Dazu zählen beispielsweise die Aufgaben und signifikante Attribute des Benutzers, und die technische und physikalische Umgebung [ISO99]. Jedoch ist der „ISO-13407-Standard“ mehr ein Leitfaden, der die einzelnen Kontextelemente angibt, und weniger eine allgemeine Definition. Auch Brown listet in seiner Definition signifikante Kontextinformationen auf [Bro96]: *„der Ort, Identitäten von Personen in der Umgebung des Benutzers, Tageszeit, Jahreszeit, Temperatur, usw.“*.
Der erste Versuch den Kontextbegriff etwas abstrakter zu fassen, stammen von Schilit et al. [SAW94].

> *"Three important aspects of context are: where you are, who you are with, and what resources are nearby".* [SAW94, Seite 1]

Dennoch beschränkt der Ansatz von Schilit et al. den Kontext auf die drei wichtigen Merkmale und liefert daher keine Definition, die den Begriff vollständig umfasst. Diese Definition wird als *benutzerzentrierte* Definition klassifiziert, da sie die Aspekte des Kontexbegriffs im Vordergrund stellt, die aus Sicht des Benutzers einer mobilen Anwendung zu berücksichtigen sind [BHS05].
Trotzdem mussten die Definitionen des Kontexts durch Aufzählen von Elementen bzw. Aspekten des Kontextbegriffs immer wieder revidiert oder erweitert werden, da neue Klassen von kontextbezogenen Anwendungen Typen von Kontextinformationen verwendeten, die der Kontextbegriff bis dahin nicht umfasste. Dey et al. weissen auf das Problem der Definitionen hin, Kontextelemente einfach aufzulisten. Sie seien relativ schwierig anwendbar, da es unklar sei, ob ein bestimmter Begriff der nicht aufgelistet ist zum Kontext gehöre oder nicht [DeAb99].
Ein weiterer Versuch den Kontextbegriff im Sinne des Mobile Computing abstrakter zu umfassen, machten Lieberman et al. [LiSe00]. Sein Ansatz gilt als rein *systemzentriert* [BHS05]. Sie hat im Gegensatz zu Schilit nicht den Benutzer im Blickpunkt, sondern das technische System, das die Kontextinformationen verarbeitet. Danach ist der Kontext eine Ansammlung von Daten, die die Berechnung eines Systems beeinflussen, ausgenommen sind dabei jedoch die expliziten Ein- und Ausgabedaten:

> *"Context can be considered to be everything that affects the computation except the explicit input and output."* [LiSe00, Seite 618]

Lieberman et al. abstrahieren dabei von einzelnen Typen von Kontextinformationen und sehen mehr den Einfluss dieser Informationen auf die Berechnung als signifikant an (siehe Abbildung 2.4). Als explizite Eingabe des Benutzers sind nach Lieberman et al. nur solche Informationen gekennzeichnet, die direkt vom Benutzer durch die Benutzerschnittstelle eingegeben werden.
Die wohl systematischste und am weitesten verbreitete Definition eines einheitlichen Kontextbegriffes findet sich bei [DeAb99]. Dey und Abowd untersuchten dabei vorherigen Definitionen von Kontext, die meist durch Aufzählungen von Beispielen oder Synonymen beschrieben waren, und verglichen sie miteinander. Sie kamen zu dem Schluss, dass zwar die

meisten bisherigen Definitionen jeweils einen elementaren Aspekt des Kontextbegriffs berücksichtigten, aber dennoch nicht den Begriff vollständig umfassten.

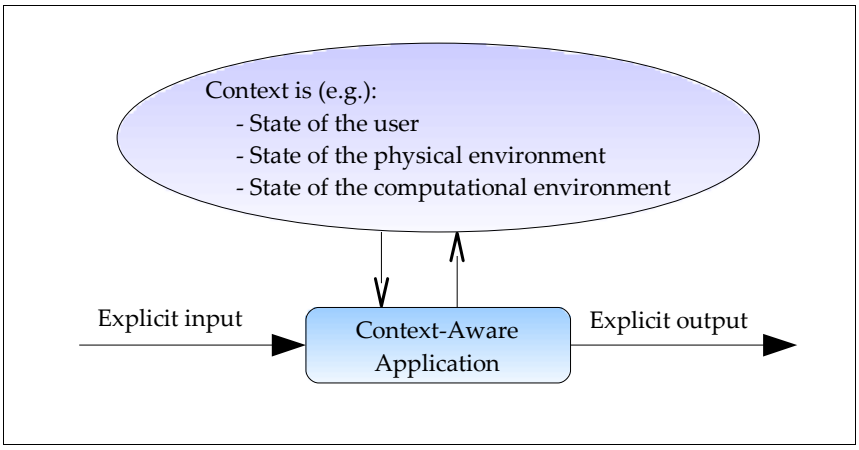


Abbildung 2.4: Kontext als implizite Ein- und Ausgabe (nach [LiSe00])

Die Defizite, die sich aus vorherigen Definitionen ergaben, seien nach Meinung von Dey und Abowd durch eine abstrakte Beschreibung zu eliminieren:

> *„Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.“* [DeAb99, Seite 3]

Danach kann es sich beim Kontext um praktisch jede Art von Information aus dem Umfeld einer Anwendung handeln. Diese Definition lässt im Gegensatz zu [LiSe00] sowohl implizite, als auch explizite Informationen zu. Das heißt, es ist irrelevant ob das System die Information selbst ermittelt, oder der Benutzer diese durch eine Eingabe selbst angibt. Beide Informationstypen können als Kontextinformationen verwendet werden.
Die Einführung von abstrakten Entitäten ist ein zweiter Aspekt der Definition von Dey. Diese können beliebige (abstrakte) Objekte sein, welche für die Interaktion zwischen Benutzer und Anwendung relevant sind. Das bedeutet, es wird von einer konkreten technischen, sozialen oder physikalischen Umgebung abstrahiert.
Entitäten selbst inkludieren den Benutzer und die Anwendung. Diese Abstraktion impliziert, dass sowohl der Benutzer als auch die Anwendung abstrakte Entitäten sind, die zum Kontext gehören können.
Aufgrund des generischen Ansatzes dieser Definition, wird im weiteren Verlauf dieser Arbeit der Terminus Kontext im Sinne von Dey und Abowd verstanden.
Kontext ist also nach diesem Ansatz die Menge *aller* Informationen bezüglich der Situation einer Entität, die für das Verhalten einer Anwendung relevant ist. Einzelne Informationen aus dieser Menge - wie beispielsweise Ort, Zeit, Aktivität, Temperatur oder Lichtverhältnis – werden als *Kontextinformation* bezeichnet.

2.2.3 Context-Awareness

Nach den verschiedenen Ansätzen zur Definition des Kontexts im vorherigen Abschnitt, wird nun der im Zusammenhang von Mobile Computing häufig verwendeter Begriff *Context-Awareness* erörtert.
Mobile Anwendungen, die Kontextinformationen benutzen, um auf die aktuelle Ausführungsumgebung reagieren zu können, werden als *kontextbezogen* oder *kontextsensitiv* (englisch: „*context-aware*“) bezeichnet. Als Synonym für Context-Awareness wird häufig der Begriff *Context-Aware Computing* verwendet [BHS05, DeAb99, SAW94, ChKo00].
Ähnlich wie beim Begriff des Kontexts existieren auch hier unterschiedliche Definitionen und Kategorisierungen sowohl von Kontextsensitivität selbst, als auch von den Funktionen, die kontextsensitive Anwendungen unterstützen können. Eine Klassifizierung von kontextsensitiven Anwendungen folgt im Abschnitt 2.4.3. Hier geht es lediglich um die Darlegung und Analyse der Definitionen.
Analog zu seine Definition des Kontexts (vgl. Abschnitt 2.2.2) beschreiben Schilit et al. in dem Artikel von 1994 [SAW94] den Begriff Context-Awareness als:

> *„... the ability to adapts according to the location of use, the collection of nearby people, hosts, and accessible devices, as well as to changes to such things over time.“* [SAW94, Seite 1]

Konsequenterweise verfolgen die Autoren so wie bei ihrer Definition des Kontexts den benutzerzentrierten Ansatz. Der Benutzer steht bei der Kontextsensitivität von (mobilen) Systemen im Vordergrund. Die folgenden drei Aspekte seien aus Sicht des Benutzers von Bedeutung. Der Ort an dem er sich befindet, welche andere Menschen und Geräte ihn umgeben, und welche Veränderungen bezügliche dieser Elemente im Laufe der Zeit auftreten.
Dey et al. analysieren parallel zu ihrer allgemeinen Definition des Kontexts (vgl. Abschnitt 2.2.2) verschiedene frühere Beschreibungen der Context-Awareness, wonach Anwendungen Kontextinformationen entweder direkt nutzen oder sich indirekt daran anpassen können [DeAb99]. Darauf aufbauend leitet er eine eigene, verallgemeinernde Definition der Context-Awareness ab, die die spezielleren vorherigen Definitionen auf einer abstrakteren Ebene zusammenfasst:

> *„A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task.“* [DeAb99, Seite 6]

Ähnlich wie Dey spezifizieren Becker et al. Context-Awareness mit der Definition einer *kontextbezogenen Anwendung* [RBB03]:

> *„Eine Anwendung ist kontextbezogen (context-aware), wenn ihr Verhalten durch Kontextinformationen beeinflusst wird.“* [RBB03, Seite 3]

Für das Verhalten einer Anwendung kann danach eine oder mehrere Entitäten relevant sein. Kontextsensitive Anwendungen greifen auf die Kontextinformationen dieser Entitäten zu, um entweder sie den Benutzer zu präsentieren oder adaptiv zu reagieren.

Einen formalen Ansatz zur Beschreibung von kontextsensitiven Anwendungen wählen Schmidt et al. [ScGe01]. Zu diesem Zweck definiert sie eine kontextsensitive Anwendung A mit einem Tripel:

$$A = (\Omega, \Theta, \Re)$$

$$\Omega = \{k_1, k_2, ..., k_n)$$

$$\Theta = \{f_1, f_2, ..., f_m)$$

$$\Re: P(\Omega) \rightarrow P(\Theta)$$

Das Tripel A besteht aus den Komponenten Ω, Θ und ℜ. Ω stellt die Menge der Kontexte k_i dar, die für die Anwendung A relevant sind. Die Menge Θ enthält die Funktionen, die zu der Anwendung A gehören. Die eigentliche Kontextsensitivität ist durch die Relation ℜ gegeben. Dabei wird die Potenzmenge der Kontextmenge auf die Potenzmenge der Funktionsmenge abgebildet. Somit wird eine kontextbezogene Anwendung dadurch beschrieben, indem festgelegt wird in welchem Kontext oder in welcher Kombination von Kontexten welche Funktionen ausgeführt werden [ScGe01].
Schmidts versucht zwar durch seinen formalen Ansatz eine abstrakte Beschreibung der kontextbezogenen Anwendungen zu geben, allerdings geht aus der obigen Definition nicht hervor *in welcher Form* Kontexte die Funktionen der Anwendung beeinflussen. Die Relation ℜ nimmt lediglich eine Zuordnung von Kontexte zu Funktionen. Entscheidende semantische Informationen über die Art der Auswirkungen von Kontexte auf Funktionen fehlen hier.

2.2.4 Situation

Eine *Situation* bezeichnet – im Allgemeinen – *„die Gesamtheit der augenblickliche Umstände oder Verhältnisse“* [Bro93, 20. Band, Seite 337]. Verglichen mit der allgemeinen Definition von Kontext scheint der Situationsbegriff ein Synonym für Kontext - und umgekehrt.
Auch im Bereich des Mobile Computing verwenden einige Autoren die Begriffe Situation und *Situation-Awareness* als Synonyme für Kontext bzw. Context-Awareness [ScTh94, WJH97, HNB97]. Im Gegensatz dazu besteht für Dey [Dey01] und Schmidt et al. [ScGe01] semantische Unterschiede zwischen Situation und Kontext.
Ausgehend von seiner Kontextdefinition (vgl. Abschnitt 2.2.2) beschreibt Dey eine *Situation* als „*die Menge der Zustände von Entitäten*“ und eine *Situationsabstraktion* als „*die Beschreibung der Zustände relevanter Entitäten*“ [Dey01]. Hallberg [HSS04] interpretiert den Zusammenhang zwischen Kontextinformation, Kontext und Situation im Sinne von Dey:

> *„With these definitions in mind the connection between location, context, and situation is that location is a type of context, as is time, weather, mood, and so on, while a situation is a collection of contexts.“* [HSS04, Seite 1]

Nach Schmidt et al. sind kontextsensitive Anwendungen dadurch charakterisiert, dass sie situationsabhängige Informationen während der Verarbeitung berücksichtigen. Eine konkrete Situation lasse sich aber *„im Allgemeinen nicht vollständig und objektiv beschreiben“*. Für die Nutzung des Kontexts in Anwendungen genügt es, Situationen durch ihre charakteristischen Merkmale hinreichend genau zu beschreiben. Für Schmidt et al. entsteht die Beschreibung

des Kontexts aus der Beschreibung der „charakteristischen Merkmale“ einer Situation. Diese charakteristischen Merkmale heben sich von anderen Merkmalen einer Situation dadurch ab, indem sie relevante Informationen beinhalten, die für eine mobile Anwendung von Bedeutung sind [ScGe01].
Die Wahrnehmung einer Situation ist ein komplexer Vorgang. Um eine „reproduzierbare Erfassung von Situationen“ in technischen (mobilen) Systemen trotzdem zu ermöglichen, schlagen Schmidt et al. eine systematische Vorgehensweise vor:

> ***1**. Verbindliche Festlegung der Situationen, die für die Anwendung relevant ist.*
> ***2.** Verbindliche Festlegung der charakteristischen Merkmale einer Situation.*
> ***3.** Für jedes Merkmal und für jede Situation wird der Wert (die Beschreibung) des Merkmals festgelegt, welcher die Situation indiziert.*
>
> *„Die Werte von Merkmalen können Skalare, Vektoren oder auch allgemeine Beschreibungen sein. Durch diese Vorgehensweise wird die Situation auf den Merkmalsraum reduziert, d.h. im Designprozess findet eine Abstraktion statt. Durch die Verwendung von Merkmalen wird die zu erfassende Datenmenge wesentlich reduziert.“* [ScGe01, Seite 5]

2.3 Beispiele für Context-Awareness im Mobile Computing

Im folgenden Abschnitt sollen exemplarisch einige kontextsensitive Anwendungen betrachtet werden, die im Rahmen der Forschung auf dem Gebiet der mobilen, kontextbezogenen Systeme bereits realisiert wurden.
Anhand der folgenden Beispiele wird geschilderten wie Kontextinformationen in verschiedenen Anwendungsbereichen verwendet werden. Mobile Anwendungen in der *Arbeitsumgebung* [WHFG92, HHSW+99], in *Navigationssystemen* [LKAA96, AAHL+97, BKW02], in der *Feldforschung* [Pas98], als *Gedächtnisunterstützung* [Pas97, HePi98] oder zur Verbesserung der *Mensch-Maschine-Kommunikation* [BKW02, BGS99, ScGe01] machen zunehmend von Kontextinformationen Gebrauch, um Anwendungsdaten kontextbezogen zu präsentieren und/oder adaptiv zu reagieren.
Die Beschreibungen der Anwendungen sollen einen ersten Eindruck darüber vermitteln, welche Anwendungen in welcher Form Kontextinformationen nutzen. Doch nicht nur die Form der Nutzung von Kontextinformation ist von Bedeutung, sondern auch welche Arten von Kontextinformationen benutzt werden. Eine entsprechende Klassifizierung der Kontextinformationen und der beschriebenen kontextsensitiven Anwendung folgt im Abschnitt 2.4.

2.3.1 Navigation

Die ersten kontextsensitiven mobilen Anwendungen stammen aus dem Bereich der Navigationssysteme. Dabei wurden bereits realisierte Navigationsanwendungen durch die Nutzung von kontextsensitiven Informationen so erweitert, dass aus Sicht des Benutzers hilfreiche Informationen über die Umgebung des Benutzers dargestellt werden konnte. Zwei prototypische, adaptive Navigationssysteme werden im Folgenden beschrieben.

CyberGuide

Eines der ersten kontextsensitiven Systeme wurde Anfang der 90er Jahre an der „Georgia Institute of Technology“ entwickelt. Die Entwickler des *CyberGuide-Systems* [LKAA96] stellten sich die Frage wie Positionsinformationen eines Benutzers von einer mobilen Anwendung sinnvoll genutzt werden kann. Um diese Frage zu klären, wurde ein lokationsbasierter Führer (tour guide) für Besucher des „Graphics, Visualization and Usability Center“ (GVU-Center) der „Georgia Institute of Technology“ realisiert.
Dabei erhält jeder Besucher beim Eintreten des GVU-Centers ein mobiles Endgerät („Apple MessagePad“). Auf einer Karte kann der Besucher sehen in welchem Raum er sich befindet und welche Objekte ihn umgeben (Abbildung 2.5, links).
Zu jedem Objekt im GVU-Center kann der Benutzer detaillierte Informationen abrufen und auf dem Display des mobilen Gerätes lesen (Abbildung 2.5, rechts). Neben dem Abrufen lokationssensitiver Informationen stellt das System auch automatisch ein Tagebuch der besuchten Orte zusammen, die der Benutzer am Ende der Tour ausdrucken kann.

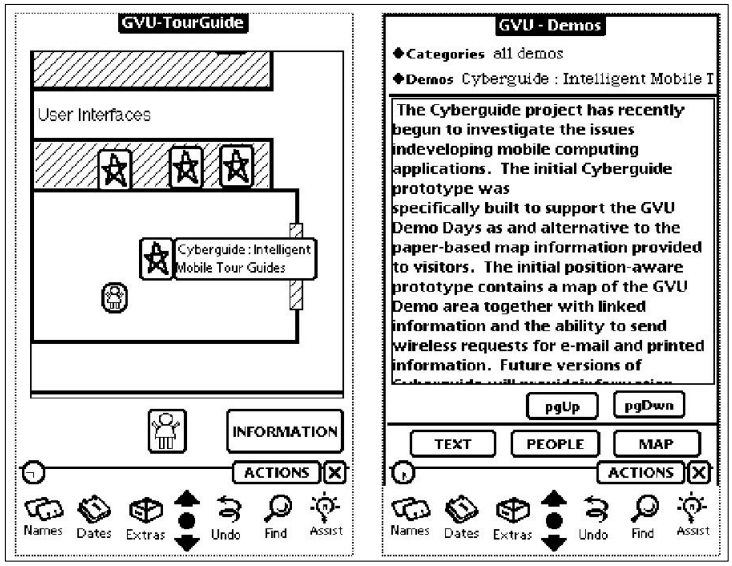


Abbildung 2.5: Kartenansicht (links) und Informationen über ein umgebendes Objekt (rechts) [LKAA96]

Der CyberGuide-Führer für Innenräume basierte dabei auf Infrarotpositionierung. Dazu wurden in den Innenräume des GVU-Center *Infrarot-Beacons* installiert und die mobilen Geräte mit einem Infrarotempfänger ausgestattet (Abbildung 2.6). Ein zentraler Server, der über ein WLAN mit den mobilen Geräten vernetzt ist, berechnet die aktuelle Position eines Benutzers und teilte sie dem jeweiligen Client (mobiles Endgerät) mit. Auf Anforderung eines Clients sendete der Server entsprechende ortsbezogene Anwendungsdaten (beispielsweise Informationen zu einem Objekt) [LKAA96].
Die Entwickler des CyberGuide-System setzten sich zum Ziel eine dynamische Architektur zu entwerfen, um die Erweiterbarkeit des Systems zu gewährleisten. Die Systemarchitektur sollte nicht nur auf eine ortsbezogene Anwendung in dem relativ kleinen GVU-Center zugeschnitten sein, sondern so flexibel sein, dass es mit möglichst geringem Aufwand für den Außenbereich erweitert werden kann.

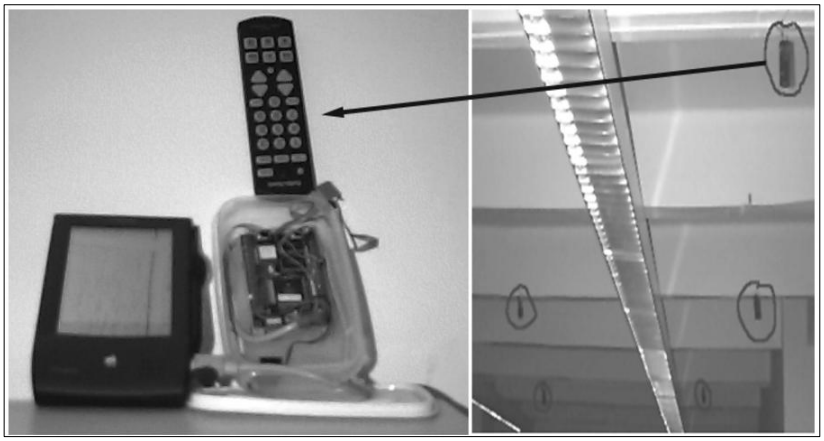


Abbildung 2.6: Infrarotfähiges mobiles Gerät (links) und Infrarot-Beacons in den Innenräumen (rechts) [LKAA96]

Aus diesem Grund wurde eine Komponentenarchitektur bevorzugt, die aus den folgenden Modulen besteht:

1. **Mapping modul**: Diese Komponente enthält Informationen über die physikalische Umgebung des Benutzers, wie Objekte in einem Gebäude oder Sehenswürdigkeiten in der Umgebung. Das Kartenmodul sorgt aber auch dafür, dass die physikalischen Objekte innerhalb einer Karte auf dem mobilen Gerät benutzerfreundlich angezeigt werden.
2. **Information modul**: Das Informationsmodul ermöglicht einem Client (mobiles Gerät) den Zugriff auf detaillierte Informationen über die umgebende Objekte im Gebäude. Dies betrifft beispielsweise die Beschreibung eines Objekts oder die Identität von Personen, die mit einem bestimmten Exponat eines Museums in Beziehung stehen. Das Modul sollte aber auch in der Lage sein als eine Art Expertensystem konkrete Fragen des Benutzers zu beantworten (z.B. „Wer arbeitet in diesem Gebäude?“, „Welcher Künstler hat dieses Gemälde gemalt?“ oder „Welche andere Vorführung ist in Zusammenhang mit dem angesehenen Exponat interessant?“).
3. **Positioning modul**: Damit das Informationsmodul ermitteln kann auf welche physikalische Objekte sich bestimmte Fragen beziehen (z.B. „Wie heißt das Gemälde vor dem ich gerade stehe?“), muss das Positionierungsmodul den Aufenthaltsort und die Orientierung des Benutzers berechnen, um die umgebenden Objekte zu ermitteln. Auch das Kartenmodul benötigt Positionsinformationen, um die Lokation des Benutzers genau auf der Karte abzubilden [LKAA96].
4. **Communication modul**: Zusätzlich kann ein einfacher Nachrichtenaustausch zwischen Museumsbesuchern erforderlich sein. Der Benutzer möchte beispielsweise Fragen an den Museumsausteller senden, wenn dieser gerade nicht in der Nähe ist. Das Modul ermöglicht zusätzlich das Versenden und Empfangen von (Broadcast-) Nachrichten von und an alle mobilen Benutzer (z.B. „Der Reisebus fährt in 15 Minuten ab!“).

Später wurde das CyberGuide-System für die Benutzung im Freien als Touristenführer erweitert [AAHL+97]. Es liefert einem Touristen – ähnlich wie das System für Innenräume - Informationen in Abhängigkeit von seinem aktuellen Aufenthaltsort. So kann er zum Beispiel Hintergrundinformationen zu der Sehenswürdigkeit abrufen, bei der er sich momentan

befindet, sich zu einem bestimmten Ort navigieren lassen oder Notizen auf einer interaktiven Landkarte hinterlassen (Abbildung 2.7, links).

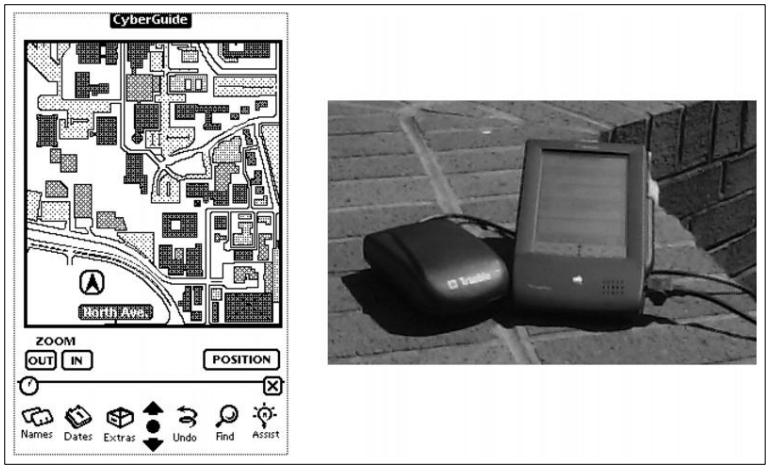


Abbildung 2.7: CyberGuide-Karte für den Außenbereich (links), mobile Gerät mit GPS-Empfänger (rechts) [AAHL+97]

Außerdem kann mit dem Wissen über die bisherigen Aufenthaltsorte automatisch ein Reisetagebuch geführt werden und es können weitere Ziele, die den Benutzer interessieren könnten, vorgeschlagen werden.
Für die Positionsbestimmung im Freien wurde GPS verwendet. Dazu musste das „Apple MessagePad“ mit einem GPS-Empfänger ausgerüstet werden (Abbildung 2.7, rechts). Die modulare Architektur des Systems blieb aufgrund seiner Flexibilität unverändert. Lediglich die Module zur Positionierung (GPS- statt Infrarotpositionierung) und zur Kartendarstellung (Ausschnitt einer Stadtkarte statt einer Karte für Innenräume) wurden ausgetauscht [AAHL+97].

REAL-Projekt

Die Forscher der Universität Saarbrücken entwickelten im Rahmen des REAL-Projekts ein adaptives Navigationssystem für Fußgänger [BKW02]. Das Teilprojekt IRREAL realisierte zunächst eine Navigationsanwendung für Innenräume, die je nach Situation des Benutzers (Stehen oder Laufen), Ausgabemöglichkeiten des mobilen Endgerätes (Display-Größe, Sprachausgabe) und Qualität der vorhandenen Positionsinformationen angepasste Benutzerausgaben erzeugte.
Im Gegensatz zum CyberGuide-System wurden nicht nur *Position* und *Orientierung* sondern auch *Geschwindigkeit*, *Eigenschaften des mobilen Gerätes* und die *Qualität* der errechneten Positions- und Orientierungsinformationen als Kontextinformationen verwendet.
Abbildung 2.8 zeigt die verschiedenen Präsentationsformen des IRREAL-Systems. Bewegt sich der Benutzer während der Navigation zu seinem Ziel, dann navigiert ihn das System nur mit einem Pfeil. Bleibt er ein Moment stehen, erscheint ein Fragezeichen (Abbildung 2.8 A). Bei dessen Aktivierung zeigt die Anwendung welche Räumlichkeiten und Objekte ihn gerade umgeben (Abbildung 2.8 B). Weitere Details zu den einzelnen Objekten und Räumlichkeiten erhält der Benutzer durch Aktivierung der entsprechenden Symbole (Abbildung 2.8 C) [BKW02].

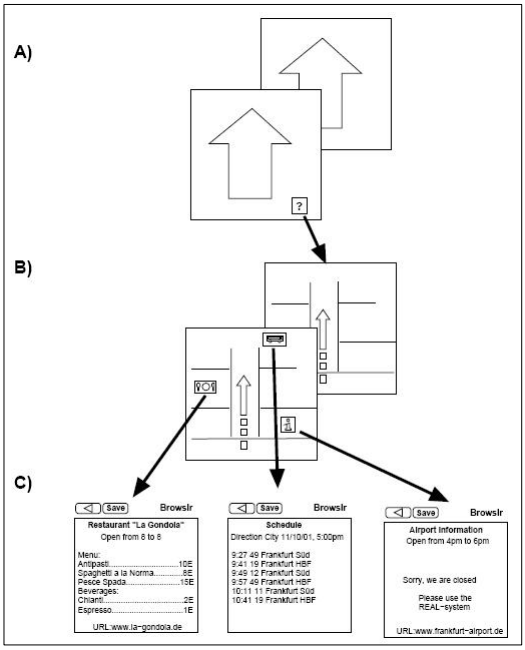


Abbildung 2.8: Präsentationsgraph des IRREAL-Systems [BKW02]

Eine besondere Eigenschaft des REAL-Systems ist es die Qualität von Kontextinformationen, wie beispielsweise Position und Orientierung, selbst als Kontextinformation zur Darstellung der Navigationsdaten zu nutzen. Dies ist deshalb notwendig, da die Verfügbarkeit von kontextsensitiven Informationen abhängig von der aktuellen Umgebung eingeschränkt sein kann und somit auch ihre Nutzung durch eine mobile Anwendung. Die Daten zur Positionsbestimmung und Orientierung des Benutzers sind im REAL-System keineswegs immer und überall mit hundertprozentiger Genauigkeit verfügbar. Aus diesem Grund verhält sich das REAL-System adaptiv bezüglich der Qualität solcher Informationen [BKW02].

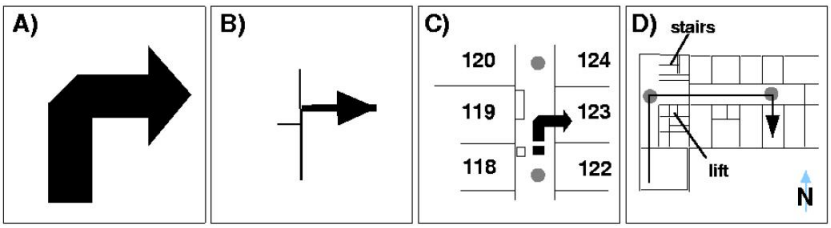


Abbildung 2.9: Grafische Anpassung der Darstellung abhängig von der Verfügbarkeit der Kontextinformationen [BKW02]

Sind hinreichend Positions- und Orientierungsinformationen eines Benutzers bekannt, so reicht ein einfacher Pfeil, um den Benutzer durch die Räumlichkeiten eines Gebäudes zu navigieren (Abbildung 2.9 A). Wenn die Daten zur Orientierung des Benutzers teilweise fehlen oder zu ungenau sind, dann kann ein einfacher Pfeil irreführend sein, da nicht genau bekannt ist in welcher Richtung der Anwender blickt. Existiert eine Ungenauigkeit von +/-90 Grad bei Ermittlung der Orientierung werden deshalb die Flure der Umgebung zusätzlich abgebildet, damit der Benutzer sich selbst zurecht findet kann (Abbildung 2.9 B) [BKW02].

Temporär können sowohl Informationen zur Position als auch zur Orientierung zu ungenau sein oder gar in einzelnen Bereichen des Gebäudes nicht vorhanden sein (Abbildung 2.9 C: keine Kontextinformationen zwischen den grauen Punkten). Aus diesem Grund werden in solchen Situation mehr Details zu den Räumlichkeiten, wie beispielsweise Darstellung und Identifikation von umgebenden Räumen, angezeigt [BKW02].
Tritt der Fall ein, dass alle Kontextinformationen für längere Zeit nicht verfügbar sind, werden nicht wie beim vorherigen Fall (Abbildung 2.9 C) nur umgebende Objekte zwischen zwei Entscheidungspunkten (sog. „decision points“) angezeigt, sondern auch signifikante Objekte (sog. „landmarks“) wie Haustreppen und Fahrstühle (Abbildung 2.9 D). Die dargestellten Details der Navigationskarte steigen also umgekehrt proportional zur Verfügbarkeit und Genauigkeit der Kontextinformationen.
Ähnlich wie das CyberGuide-System erweiterte das Teilprojekt ARREAL das REAL-System (mit Hilfe von GPS-Positionierung) für die Navigation im Freien. Auch hier wurde aus der Kombination von Genauigkeit der Ortungsdaten und Geschwindigkeit des Benutzers die Kartenansicht auf dem mobilen Gerät dynamisch angepasst (Abbildung 2.10).

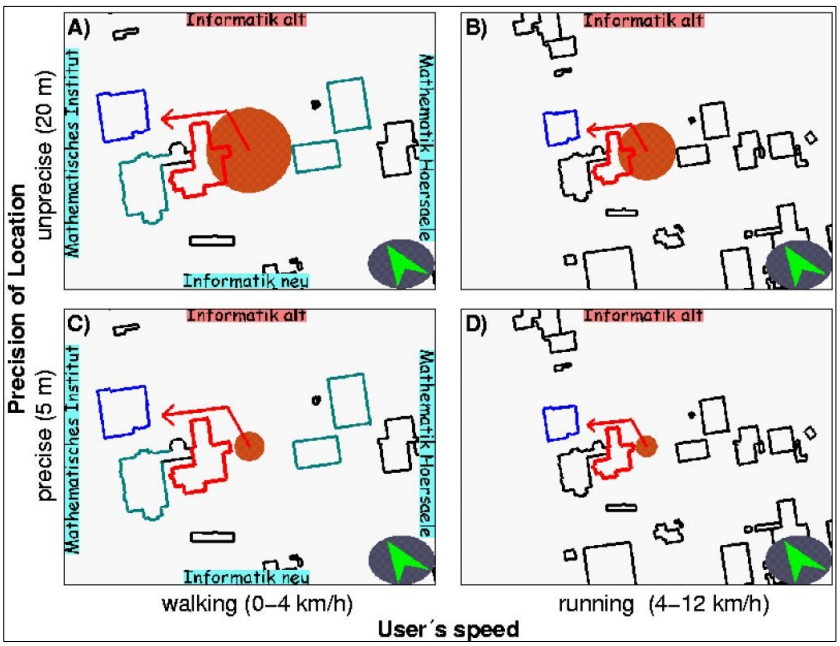


Abbildung 2.10: Kontextsensitive Kartendarstellung im ARREAL-Teilprojekt (roter Kreis: potentieller Aufenthaltsort) [BKW02].

2.3.2 Arbeitsumgebung

Auch im Büroumfeld können kontextsensitive mobile Informationssysteme Angestellten einer Organisation bei der Erledigung ihrer Aufgaben unterstützen. In diesem Zusammenhang sind meist der aktuelle Aufenthaltsort und die Aktivität eines Mitarbeiters von Interesse. Solche Informationen können genutzt werden, um beispielsweise eingehende Anrufe automatisch an das nächstgelegene Telefon weiterzuleiten oder unterschiedliche Ausgabemöglichkeiten in der momentanen Umgebung zu verwenden (z.B. automatische Erkennung eines Beamers).

Active Badge Location System

Die Firma „Olivetti Research Lab“ entwickelte Anfang der 90er mit dem *Active Badge Location System* das erste kommerzielle kontextsensitive System [WHFG92]. Das System sollte einen/eine Telefonrezeptionist/in eines Unternehmens oder eines Krankenhauses bei der Weiterleitung eingehender Telefonate unterstützen. Zu diesem Zweck wurde jeder Mitarbeiter der Organisation mit einem „Active Badge“, einem tragbaren kleinen Anstecker (Abbildung 2.11), ausgestattet. Abbildung 2.11 zeigt die vier Generationen der Active-Badge-Anstecker.

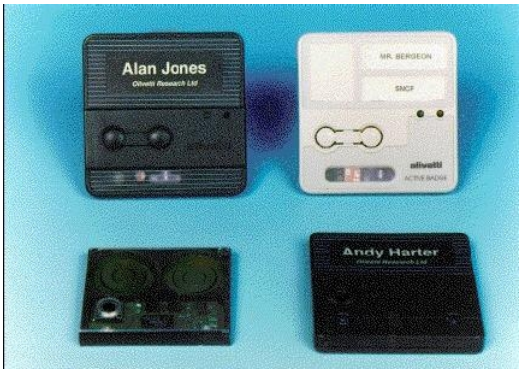
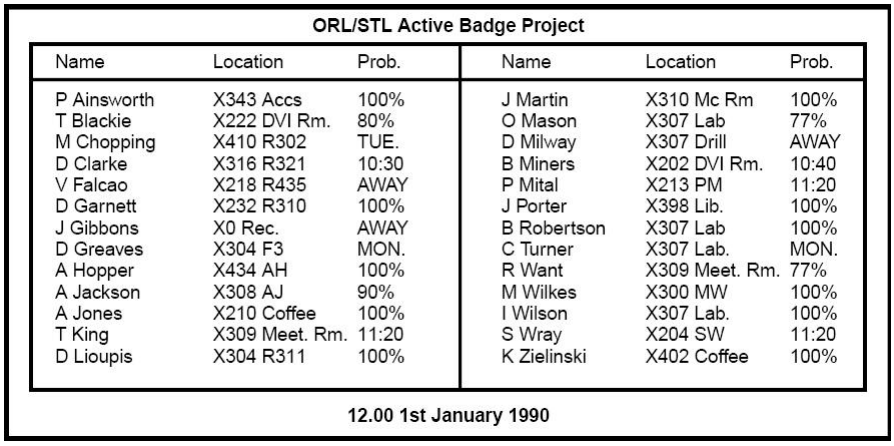


Abbildung 2.11: Generationen der Active-Badge-Anstecker [WHFG92]

Das Besondere an den Active-Badges-Anhängern ist, dass er für jeden Mitarbeiter alle 15 Sekunden ein eindeutiges Infrarotsignal aussendet. Diese Signale werden von Sensoren empfangen, die innerhalb des Gebäudes verteilt sind. Es muss nicht unbedingt eine direkte Verbindung zwischen Sender und Empfänger bestehen, da aufgrund der eingesetzten Infrarottechnologie Signale von Wänden und anderen Gegenständen reflektiert werden können. Die Sensoren leiten die empfangenen Signale anhand eines internen Kommunikationsnetzes an einen zentralen Server weiter, der die Lokalität des Mitarbeiters (als Kontextinformation) innerhalb des Gebäudes berechnet. Da der Computer des Mitarbeiters an der Rezeption mit dem Server verbunden ist, sieht er auf der graphischen Benutzerschnittstelle einer Client-Anwendung welcher Mitarbeiter sich gerade wo aufhält (Abbildung 2.12) und leitet die eingehende Anrufe entsprechend in die Nähe des Mitarbeiters weiter. Es besteht zusätzlich die Option Anrufe automatisch, d.h. ohne manuelle Weiterleitung des Rezeptionisten, an den entsprechenden Aufenthaltsraum des Mitarbeiters weiterzuleiten [WHFG92].
Abbildung 2.12 zeigt neben den Namen des Angestellten, die Telefondurchwahl des nächstgelegenen Telefons und einer Kurzbeschreibung des Ortes an. Die dritte Spalte zeigt mit welcher Wahrscheinlichkeit ein Mitarbeiter sich an einem bestimmten Ort befindet. Ist die Wahrscheinlichkeit kleiner als 100% so bewegt er sich innerhalb der Gebäude. Ist die Person länger als fünf Minuten nicht lokalisiert, so wird die letzte Uhrzeit der Ortung angegeben. Wurde er mehr als 24 Stunden nicht geortet, so wird der letzte Tag der Lokalisierung angegeben. Bei Nicht-Lokalisierung von mehr als einer Woche erscheint das Wort „AWAY“ in der Wahrscheinlichkeitsspalte, um anzudeuten, dass der Mitarbeiter nicht mehr im Gebäude ist.



ORL/STL Active Badge Project

Name	Location	Prob.	Name	Location	Prob.
P Ainsworth	X343 Accs	100%	J Martin	X310 Mc Rm	100%
T Blackie	X222 DVI Rm.	80%	O Mason	X307 Lab	77%
M Chopping	X410 R302	TUE.	D Milway	X307 Drill	AWAY
D Clarke	X316 R321	10:30	B Miners	X202 DVI Rm.	10:40
V Falcao	X218 R435	AWAY	P Mital	X213 PM	11:20
D Garnett	X232 R310	100%	J Porter	X398 Lib.	100%
J Gibbons	X0 Rec.	AWAY	B Robertson	X307 Lab	100%
D Greaves	X304 F3	MON.	C Turner	X307 Lab.	MON.
A Hopper	X434 AH	100%	R Want	X309 Meet. Rm.	77%
A Jackson	X308 AJ	90%	M Wilkes	X300 MW	100%
A Jones	X210 Coffee	100%	I Wilson	X307 Lab.	100%
T King	X309 Meet. Rm.	11:20	S Wray	X204 SW	11:20
D Lioupis	X304 R311	100%	K Zielinski	X402 Coffee	100%

12.00 1st January 1990

Abbildung 2.12: Client-Display zeigt Aufenthaltsorte von Mitarbeitern [WHFG92]

Zusätzlich zu der graphischen Benutzerschnittstelle, ermöglicht die Client-Anwendung das Ausführen von Kommandos über eine Eingabezeile. Tabelle 2.3 gibt eine Übersicht über ausführbare Befehle.

Tabelle 2.3: Kommandos des Active Badge Systems [WHFG92]

Kommando	Beschreibung
FIND(name)	Gibt den aktuelle Aufenthaltsort einer Person. Falls die Person sich in letzter Zeit innerhalb der Gebäude bewegt hat, wird eine Liste der Aufenthaltsort der letzten fünf Minuten, zusammen mit den Wahrscheinlichkeiten die Person dort anzutreffen, angegeben.
WITH(name)	Ortet eine Person und gibt an welche andere Personen sich in seiner unmittelbaren Umgebung befinden.
LOOK(location)	Anhand einer angegebenen Position wird eine Liste der Personen geliefert, die sich in der unmittelbaren Umgebung dieses Ortes befinden.
NOTIFY(name)	Sobald eine Person lokalisiert ist, ertönt ein Alarmsignal.
HISTORY(name)	Erzeugt einen kurzen Bericht über die Aufenthaltsorte einer Person während der letzten Stunde.

ParcTab System

Zur Forschung im Bereich des Ubiquitous Computing realisierten Schilit und andere Forscher am „Xerox Palo Alto Research Center“ (Xerox PARC) das *ParcTab System* [SAGT+93]. Das System basierte auf kleinen, mobilen (PDA-ähnlichen) Geräte, den sog. ParcTabs, die mit Hilfe eines berührungsempfindlichen Displays und drei Tasten bedient wurden (siehe Abbildung 2.13). Die ParcTabs waren über ein infrarotbasiertes, zellulares Netzwerk mit einem zentralen Server verbunden. In jedem Raum befanden sich Empfängerstationen, die eine ständige Verbindung zum Netz sicherstellten [SAGT+93].

Abbildung 2.13: Xerox ParcTab-Computer [SAGT+93]

Die ParcTabs wurden im Büro als eine Art persönlicher mobiler Assistent benutzt. Zu diesem Zweck wurden verschiedene mobile Anwendungen entwickelt, von denen einige kontextsensitiv waren:

- Die Angestellten wurden innerhalb der Gebäude lokalisiert und ihre Position auf einem Lageplan angezeigt.
- Informationen über den Raum, in dem der Benutzer sich gerade befindet, konnten sowohl aufgerufen werden als auch automatisch beim Betreten des Raumes angezeigt werden. So konnte beispielsweise der Benutzer Ausleihbedingungen beim Eintreten in die Bibliothek erfahren.
- Das ParcTab konnte als Fernbedienung abhängig von Raum, in dem sich der Benutzer gerade befindet, verwendet werden.
- Nahgelegene Ressourcen wie Drucker wurden automatisch lokalisiert und ihre Entfernung zur aktuellen Position angezeigt.
- Das Verzeichnis eines UNIX-Dateisystems wurde mit den Räumen assoziiert. Der Benutzer hatte beim Betreten des Raumes Zugriff auf entsprechende Verzeichnisse und konnte Notizen in Form von Dateien hinterlassen. Andere Benutzer könnten diese Notizen lesen, wenn sie sich in dem Raum befanden.

Conference Assistant

Einer der komplexesten kontextsensitiven Anwendungen wurde von Dey und Salber 1999 am „Georgia Institute of Technology“ entwickelt [DSAF99]. Der *Conference Assistant* unterstützt, als eine prototypische mobile Applikation, Teilnehmer einer Konferenz. Der Benutzer kann im Programm seine Interessen angeben. Diese werden mit der Agenda verglichen und entsprechende Vorträge und Präsentationen angezeigt. Nährt sich der Benutzer einem Konferenzraum, so erhält er automatisch weitere Informationen zum Vortragenden und eine Kurzbeschreibung des behandelten Themas. Während eines Vortrags empfängt die Applikation automatisch die gerade angezeigte Vortragsfolie und kann entsprechende Notizen vermerken. Dabei wird alles mit einem Zeitstempel versehen und kann später wieder aufgerufen werden. Aufkommende Fragen zu bestimmten Folien können ebenfalls notiert werden und am Ende des Vortrags an das Endgerät des Vortragenden gesendet werden [DSAF99].

Die Anwendung ist insofern komplex als sie verschiedene Formen von Kontextinformation verwendet, um auf die Umgebung des Benutzers zu reagieren. Der Conference Assistant benutzt Kontexttypen wie *Identität*, *Aktivität*, *Aufenthaltsort* und *Zeit* in Form von Forschungsinteressen des Konferenzteilnehmers, Verlauf der Vorträge, aktuellem Standort des Benutzers und Veranstaltungskalenderdaten.
Der Conference Assistant basiert auf einem von Dey entwickelten Framework zur Entwicklung von kontextsensitiven Anwendungen (Context Toolkit, vgl. Abschnitt 3.2.1). Die Lokalisierung der Benutzer erfolgt mit Hilfe eines Ortungssystems der Firma „PinPoint", die auf Radiowellen basiert. Die mobile Anwendung ist auf verschiedene Plattformen (z.B. Windows Notebooks und Windows CE Handhelds) ausführbar. Die Kommunikation zwischen den mobilen Komponenten und dem Netzwerk erfolgt über WLAN [DSAF99].

2.3.3 Mensch-Maschine-Interaktion

Durch Kontextinformationen können nicht nur zusätzliche Informationen zur Situation eines Benutzers präsentiert, sondern auch die Mensch-Maschine-Interaktion verbessert werden. Einige Beispiele, wie Kontextsensitivität im Mobile Computing die Benutzerfreundlichkeit von Anwendungen verbessert, werden im Folgenden geschildert.

Context Call

Eine wichtig Kategorie von Kontextinformationen ist der *soziale Kontext.* Durch Verfügbarkeit von Informationen, die Auskunft darüber geben welche Personen uns umgeben, welche Beziehung wir zu ihnen haben oder an welcher gesellschaftlichen Veranstaltung wir gerade teilnehmen, können mobile Geräte (sozial) angemessen reagieren.
Ein einfaches Beispiel für eine mobile Applikation ist der *Context Call* [STM02]. Die auf WAP-basierende Anwendung hat das Ziel potentiellen Anrufern seinen Kontext mitzuteilen. Abhängig von dieser Information kann der Anrufer entscheiden, ob es passend ist die Person anzurufen. Sinn der Anwendung ist es, dass der Benutzer weder wichtige Anrufe verpasst (wenn beispielsweise das Handy ausgeschaltet ist) noch dass er durch unpassende Anrufe gestört wird (z.B. während einer Besprechung) [STM02].

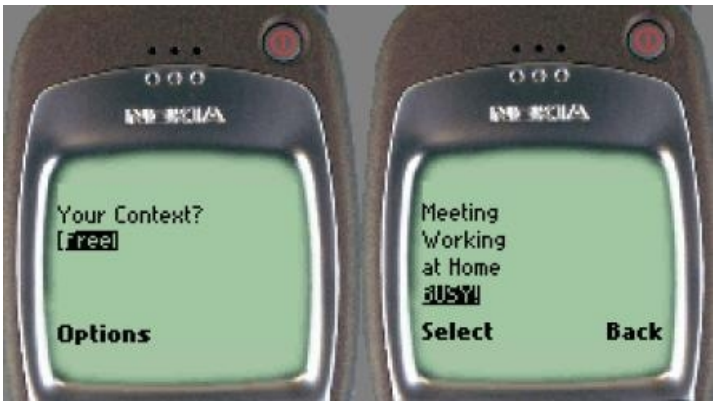


Abbildung 2.14: Kontextwahlmenü der Context-Call-Anwendung [STM02]

Durch eine einfache Benutzerschnittstelle (siehe Abbildung 2.14) kann der Handynutzer seinen aktuellen Kontext auswählen. Diese Kontextinformation wird also nicht automatisch ermittelt, sondern manuell eingegeben. Versucht ein anderer Handynutzer via Context Call

die Person anzurufen, erscheint zuerst der Kontext, in dem sie sich befindet (Abbildung 2.15). Der Benutzer kann nun entscheiden, ob er anrufen, den Rufvorgang abbrechen oder eine Nachricht hinterlassen möchte [STM02].

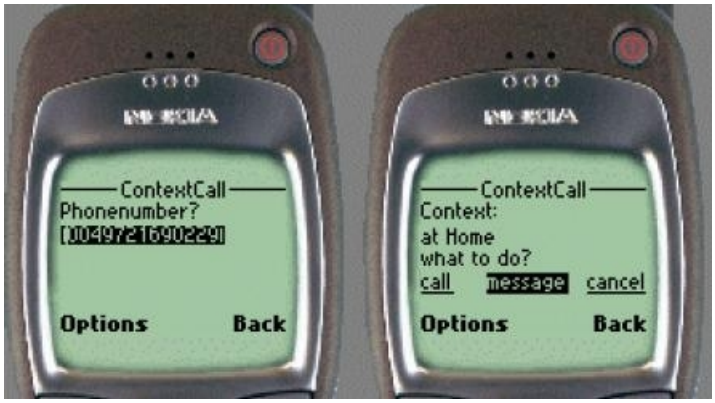


Abbildung 2.15: Kontextanzeigemenü des Context Call [STM02]

Neben der Verbesserung der Benutzerschnittstellen durch Verfügbarkeit und Nutzung von sozialen Kontextinformationen, wird der *Verteilungsaspekt* im Context-Aware Computing hervorgehoben. Wie Context Call zeigt, müssen mobile Geräte auch in einer verteilten Umgebung interagieren können. In Zusammenhang mit Context-Awareness bedeutet dies, dass nicht nur der eigene lokale Kontext das Verhalten einer mobilen Anwendung beeinflussen kann, sondern auch der Kontext von (anderen) involvierten Teilnehmern eines verteilten Systems.

Orientierungssensitiver PDA

Forscher der Universität Karlsruhe entwickelten im Jahre 1999 einen orientierungssensitiven PDA [BGS98]. Zu diesem Zweck wurde ein *Apple Newton MessagePad* mit einem Orientierungssensor ausgestattet, die aus zwei Quecksilberschaltern (Abbildung 2.16, obenrechts) besteht. Das Betriebssystem des PDA unterstützt eine Funktion, die es ermöglicht das Display in alle vier Richtungen zu schalten [BGS98]. Das bedeutet, der Benutzer kann den Bildschirm sowohl vertikal als auch horizontal ablesen. Damit kann anhand der ermittelten Orientierung (als Kontextinformation) des PDAs das Display automatisch richtig (d.h. für den Benutzer in lesbarer Form) ausgerichtet werden (Abbildung 2.16).
Bei diesem System besteht der Vorteil, dass eine manuelle Eingabe zur Ausrichtung des Bildschirms überflüssig wird. Diese Funktion, die im Betriebssystem integriert wurde, erhöht nicht nur den Komfort des Systems, sondern erweitert auch die Ausgabemöglichkeit für verschiedene Anwendungen.
Einige moderne Digitalkameras benutzen die Orientierungsinformation, um die anzuzeigende Bilddatei automatisch richtig auszurichten. Dabei dient die Orientierung der Kamera zum Zeitpunkt der Aufnahme als Kontextinformation.

Abbildung 2.16: Orientierungssensibler PDA [BGS98]

Kontextsensitives Mobiltelefon

Dass Kontext mehr als nur Positionsinformationen [BGS98] sind, beweist das TEA-Projekt [TEA06]. In Zusammenarbeit mit der Universität Karlsruhe entwickelten Forscher ein kontextsensitives Mobiltelefon, das das Ruftonprofil automatisch dem Kontext anpasst. Dazu wurde ein handelsübliches Handy mit einem Sensorboard ausgestattet, das eine Photodiode, ein Mikrofon, einen Beschleunigungsmesser, einen Temperatursensor und einen Berührungsmelder beinhaltet (siehe Abbildung 2.17).

Abbildung 2.17: Prototyp eines Mobiltelefons mit Kontextbezug [TEA06]

Anhand der installierten Sensoren kann das Mobiltelefon bestimmte Situationen bezüglich der Nutzung des Mobiltelefons automatisch erkennen und entsprechend reagieren. In [ScGe01] ist die Analyse über die Benutzung von Mobiltelefonen folgendermaßen beschrieben:

- Hat der Benutzer das Telefon in der Hand, kann er die Benachrichtigung über einen ankommenden Anruf sehr leicht erkennen. Es reicht ein kurzes akustisches Signal, um einen Anruf zu signalisieren.
- In einer formalen Besprechung wird es häufig als unhöflich betrachtet, wenn Teilnehmer akustisch über ankommende Anrufe benachrichtigt werden. Hingegen stört eine subtile visuelle Anrufanzeige meist nicht.
- Befindet sich das Mobiltelefon in einer Tasche am Körper des Benutzers, so werden akustische Signale stark gedämpft, und es geschieht leicht, dass ein Anruf überhört wird. Hingegen ist in solch einer Situation ein Vibrationsalarm sehr effektiv.
- Wenn der Benutzer, nachdem ein ankommender Anruf signalisiert wird, zum Telefon greift, ist die Aufgabe der Benachrichtigung bereits erfüllt. Beim Erkennen der Mikroaktivität „Telefon greifen“ kann die Benachrichtigung eingestellt werden [SeGe01.

2.3.4 Gedächtnisunterstützung

Menschen können sich leichter an etwas erinnern, wenn sie es mit einer bestimmten Situation verknüpfen können. Zum Beispiel wann oder wo man etwas erlebt hat. Aus diesem Grund existieren auch im Mobile Computing kontextsensitive Systeme, die Informationen zu bestimmten Objekte mit einem Kontext verknüpfen.

Stick-e Notes

Ein besonderes Konzept des Kontextbezugs in mobilen Systemen ist die Erweiterung von Gegenständen der realen Welt durch zusätzliche Informationen (auch als Augmentierung bezeichnet). Dabei werden reale Objekte der Welt mit ergänzenden Informationen annotiert. Der Benutzer erhält Zugriff auf Informationen (virtuelle Objekte), die mit realen Objekten verknüpft sind.
Ein kontextbezogenes System, dass dieses Konzept realisiert hat, ist das Stick-e Notes System [Pas97]. Dabei dienten sog. Stick-e Notes zur Annotation der realen Objekte. Stick-e Notes sind vergleichbar mit den kleinen gelben Zetteln, die an bestimmten Gegenständen angeklebt sind, um jemanden an Irgendetwas zu erinnern, das mit diesem Gegenstand in Zusammenhang steht. Bei Stick-e Notes handelt es sich allerdings um Nachrichten oder Aktionen, die in Zusammenhang mit einem mobilen Gerät angezeigt bzw. ausgeführt werden, sobald der Kontext eintritt, mit dem das Stick-e Note verknüpft ist. Der Kontext mit dem ein Stick-e Note verknüpft sein kann, ist beispielsweise die Lokation einer Person, eines Objekts oder einer Kombination aus beiden. Zum Beispiel kann ein PDA automatisch das Adressbuch des Benutzers anzeigen, sobald das PDA in die Nähe des Telefons gehalten wird.
Zunächst wurde das Stick-e-Notes-Konzept nur zur Realisierung eines bestimmten mobilen Systems genutzt. Später wurde das Konzept so verallgemeinert, dass ein Framework zur einfachen Entwicklung von kontextsensitive Anwendungen entstand [Bro06].

StartleCam System

Zur Archivierung visueller Eindrücke entstand 1998 an der „MIT Media Lab“ das *StartleCam System* [HePi98]. Ein tragbares System mit einer Digitalkamera (siehe Abbildung 2.18), die sowohl bewusst als auch unbewusst vom Benutzer gesteuert wird, speichert seine visuellen Eindrücke. Die Kamera nimmt pro Sekunde ein Bild auf. Die letzten fünf Aufnahmen werden

auf dem tragbaren Computer zwischengespeichert oder drahtlos an einem Webserver übertragen.

Abbildung 2.18: StartleCam System [HePi98]

Ein Biosensor an dem Finger des Benutzers misst die Leitfähigkeit der Haut. Wird beim Zusehen eines Objekts ein Reiz ausgelöst und mit Hilfe des Biosensors gemessen, leitet das System daraus ab, dass es sich dabei um einen besonderen visuellen Eindruck handeln muss. Folglich werden die entsprechenden zwischengespeicherten Bilder mit einem Zeitstempel versehen und endgültig gespeichert. Das StartleCam System wurde zur aktiven Unterstützung des Erinnerungsvermögens, zur Analyse des Stresslevels und entsprechende Reaktionen, und zur Hilfe bei Gefahrensituationen entwickelt. Es zeigt, dass sogar biologische Informationen über den Benutzer (wie die Leitfähigkeit der Haut) als Kontextinformationen verwendet werden können [HePi98].

2.4 Kontextklassifizierung

Neben eher abstrakte Definitionen von Kontext, Context-Awareness und Kontextinformationen (vgl. Abschnitt 2.2.2) können zusätzliche Kategorisierungen auf unterschiedlichen Abstraktionsebenen helfen, einzelne Kontexttypen zu identifizieren und in eine systematische Ordnung zu bringen. Die Kategorisierung auf verschiedenen Abstraktionsebenen zeigt einzelne Eigenschaften und Dimensionen von Kontextinformationen. Diese müssen nicht nur bei der Entwicklung einer kontextsensitiven Anwendung, sondern insbesondere bei der Modellierung einer Middleware für mobile kontextsensitive Anwendungen berücksichtigt werden.
Nach der Kategorisierung von Kontextinformationen, folgt im letzten Abschnitt (2.4.4) eine Klassifizierung von kontextsensitiven Anwendungen. Sie soll einen Überblick darüber geben in welche Formen mobile Anwendungen situationsabhängige Informationen nutzen. Auch in diesem Zusammenhang müssen entsprechende Aspekte zur Art der Nutzung von Kontextinformationen im Entwurf von kontextsensitiven Middleware-Systeme einbezogen werden.

2.4.1 Primär- und Sekundärkontext

Mobile Anwendungen verwenden bestimmte Kontextinformationen häufiger als andere. Für Dey et al. [DeAb99] beziehen manche Kontextinformationen einen höheren Stellenwert. Diese sind *Ort*, *Identität*, *Aktivität* und *Zeit*. Diese Auflistung von wichtigen Kontextinformationen unterscheidet sich von Ryans Definition des Kontexts [RPM97] insofern, als dass Aktivität statt *Umgebung* benutzt wird. Für Dey et al. ist Umgebung ein Synonym für Kontext und daher irreführend bei der Festlegung von häufig verwendeten Kontextinformationen. Die vermehrte Verwendung von Ortungsinformationen kann als direkte Folge der erhöhten Mobilität der Benutzer bzw. des Gerätes (vgl. Abschnitt 2.1.3) gesehen werden. Die Relevanz von Identität und Aktivität in diesem Zusammenhang zeigt, dass die Personalisierung in mobilen Anwendungen durch die erhöhte Mobilität zusätzlich verstärkt wird [Bro05].
Die Intention von Dey et al. liegt nicht nur darin Kontextinformationen danach zu trennen wie oft und wie selten sie benutzt werden, sondern auch auf die Abhängigkeiten zwischen ihnen aufmerksam zu machen:

> *„Location, identity, time, and activity are the primary context types for characterizing the situation of a particular entity. These context types not only answer the questions of who, what, when, and where, but also act as indices into other sources of contextual information.“* [DeAb99, Seite 5]

Dem zu Folge ist der *Primärkontext* ein Kontexttyp, der zur Auswahl und Einschränkung von weiteren Kontextinformationen verwendet wird. Er dient als eine Art Globalindex für so genannte *Sekundärkontexte* derselben Entität. Ein Primärkontext referenziert alle relevanten Sekundärkontexte, die den Primärkontext der Entität logisch untergeordnet sind. Diese Hierarchiebildung hat einen praktischen Wert für mobile Anwendungen. Sie kann als Filtermechanismus genutzt werden, um aus einer großen Menge von (sekundären) Kontextinformationen nur solche für die Verarbeitung zu isolieren, die im aktuellen Primärkontext relevant sind.
In [DeAb99] werden einige Beispiele skizziert wie aus einem Primärkontext sekundäre Kontextinformationen abgeleitet werden. So kann über die Identität (Primärkontext) einer Person (Entität) auf zugehörige sekundäre Informationen wie beispielsweise Telefonnummer, Adresse, Geburtsdatum oder Namen der Freunde und ihre Beziehungen zu ihnen referenziert werden. Von Positionsinformationen (Ort) einer Entität kann hergeleitet werden, welche andere Objekte oder Personen sie umgeben oder welche Aktivitäten an diesem Ort gerade stattfinden.

2.4.2 Low- und High-Level-Kontexttypen

Neben der Differenzierung von Kontextinformationen auf primärer und sekundärer Ebene, unterscheiden Chen et al. [ChKo00] die Messung von Kontext auf niedriger Ebene und ihre Berechnung auf höherer Ebene. In [ChKo00] beschreiben die beiden Autoren *Low-Level-Kontext* als Kontextinformationen, die direkt mit physikalischen oder logischen Sensoren gemessen werden. Während physikalische Sensoren Informationen über die physikalische Bedingungen der Umwelt liefern, stammen die Informationen aus logischen Sensoren aus dem mobilen Gerät selbst (wie beispielsweise die Systemzeit).
Low-Level-Kontextdaten zeichnen sich vor allem durch ihre relativ einfache Struktur aus. Dazu gehören beispielsweise die Helligkeit, die mit Photodioden gemessen wird, Schallwellen, die mit einem Mikrofon registriert werden und die Temperatur, die mit einem

Thermometer erfasst wird (siehe Abbildung 2.17). Zur Messung der physikalischen Bedingungen müssen die Sensoren nicht notwendigerweise im mobilen Gerät integriert sein, wie es beispielsweise im kontextsensitive Mobiltelefon (vgl. Abschnitt 2.2.8) der Fall ist. Sensoren, die außerhalb des mobilen Gerätes installiert sind (z.B. in Räumen oder im Freien) können entsprechend der Vision des Pervasive Computing (vgl. Abschnitt 2.1.2) zusammen mit dem mobilen Gerät vernetzt sein und so Daten liefern, die für den Kontext des Benutzers oder Gerätes von Bedeutung sind.
In [Fuc02] identifiziert Fuchs weitere Arten von Low-Level-Kontextinformationen und wie diese gemessen werden. Die folgende Kategorisierung von Low-Level-Kontexttypen basiert weitgehend auf die Ausführungen in [Fuc02]:

- **Lokalisierung:** Besonders wichtig ist die Ermittlung der aktuellen Position des Benutzers und eines Objekts, da aufgrund der Mobilität Standortinformationen in mobilen Systemen häufig verwendet werden.
  Die Vielzahl der existierenden Ortungsmethoden im Mobile Computing kann generell in zwei grobe Kategorien eingeteilt werden: „network based“ und „handset based“. Letztere nimmt die Ortung des mobilen Gerätes ohne Hilfe eines Kommunikationsnetzwerks (z.B. GSM oder UMTS) vor. Zu dieser Kategorie zählt z.B. die satellitengestützte Positionsbestimmung mittels GPS.
  Bei der netzwerkbasierenden Ortungsmethode ermittelt das Kommunikationsnetz selbst die Position des mobilen Gerätes und kann sie bei Bedarf (und zu einem bestimmten Preis) zur Verfügung stellen. Hier gibt es zahlreiche Methoden, die zur Anwendung kommen. Die COO-Methode (Cell of Origin) benutzt die aktuelle Mobilfunkzelle als Ausgangspunkt der Ortung. Eine komplexere Methode, wie das TDOA (Time Difference of Arrival), misst die eingehende Mobilfunksignale an mindestens drei unterschiedliche Mobilfunkmasten und berechnet daraus den Standort des mobilen Gerätes.
- **Zeit**: Die Zeit als Kontextinformationen ist sehr einfach feststellbar (beispielsweise durch die interne Systemuhr). Doch nicht nur die Uhrzeit selbst, sondern auch Informationen wie Jahreszeit, Wochentag, Wochenende, Arbeitszeit oder Freizeit können für eine mobile Anwendung von Bedeutung sein.
- **Optische Kontextinformationen:** Zu diesem Typ zählen Helligkeit, Wellenlänge und Art des Lichts (Sonnenlicht oder künstlich erzeugtes Licht). Mit Kameras besteht die Möglichkeit Bewegungen zu erfassen, Objekte und Markierungen zu erkennen, Personen zu identifizieren und menschliche Gesten zu interpretieren.
- **Akustische Kontextinformation**: Der Schall wird mit dem Mikrofon gemessen. Es liefert Informationen über Lautstärke, Hintergrundsgeräuche und Grundfrequenz. Durch aufwändigere Verfahren können sogar anhand der Akustik personen-spezifische Stimmen identifiziert werden.
- **Biologische Kontextinformationen:** Dazu zählen vor allem Pulsfrequenz, Haut-widerstand und Blutdruck eines Benutzers. Sie werden mit Biosensoren gemessen.
- **Weitere Low-Level-Kontextinformationen:** Es gibt eine große Bandbreite von physikalischen Bedingungen, die mit immer neuen Sensortypen gemessen werden. Einige weitere Sensorarten, die in prototypische, kontextsensitive Systeme verwendet werden, sind Sensoren zur Messung von Orientierung, Berührung, Beschleunigung oder Luftdruck.

Aus der Kombination und Verarbeitung von mehreren Low-Level-Kontextinformationen können komplexere Kontextinformationen gewonnen werden, die in direkter Form nicht von Sensoren bezogen werden können. Zum Beispiel kann aus dem aktuellen Aufenthaltsort des Benutzers, der Uhrzeit und dem Kalendereintrag geschlossen werden, dass der Benutzer eine Besprechung mit seinen Kollegen führt. Diese Art von Kontext bezeichnen Autoren als *High-Level-Kontext* [ChKo00, Fuc02]. Neben der Tätigkeit eines Benutzers zählen Gefühlszustand und das soziale Umfeld ebenfalls zum High-Level-Kontext.
Andere Methoden zur Erkennung von Kontextinformationen auf höherer Ebene sind die Berücksichtigung von statistischen Auswertungen historischer Kontextinformationen und die Mustererkennung durch die Methoden der Künstlichen Intelligenz [Fuc02]. Chen et al. [ChKo00] weisen aber auf die grundsätzliche Problematik hin, die mit High-Level-Kontext zusammenhängt. Die Gewinnung von solchen Informationen kann sehr kompliziert und fehlerhaft sein, wenn unvorhergesehene Ereignisse oder Doppeldeutigkeiten im Kontextzusammenhang nicht beachtet worden sind. Außerdem können solche Systeme auf Ablehnung bei Benutzern stoßen, da aus ihrer Sicht nicht nachvollziehbar ist, wie die High-Level-Kontextinformationen ermittelt werden [Fuc02]. Ein Beispiel für ein solches System wäre die Erkennung von Gefühlszuständen des Benutzers anhand biologischer Sensoren.

2.4.3 Weitere Kategorisierungsformen von Kontextinformationen

Weitere Arten von Kategorisierungsformen teilen Kontexttypen in verschiedene Bereiche ein, um einen besseren Überblick über die vorhandene Informationsarten zu gewinnen. Dabei ähneln sich die verschiedenen Ansätze einander, weil sie versuchen durch ihre allgemeine Klassifikation möglichst den gesamten Informationsraum bezüglich Kontext abzudecken. Nur der Schwerpunkt und die Methodik der Differenzierung sind unterschiedlich. Tatsächlich nutzen kontextsensitive, mobile Anwendungen aufgrund ihrer speziellen Funktion und den vorhandenen Randbedingungen im Mobile Computing (vgl. Abschnitt 2.1.5) nur ein Bruchteil an Kontextinformationen, die von den meisten Klassifizierungen berücksichtigt werden.

Schilits Kontextkategorien

Ausgangspunkt für die Klassifikation von Schilit et al. ist seine benutzerzentrierte Definition von Kontext (vgl. Abschnitt 2.2.2). Analog dazu teilen die Autoren den Kontextinformationsraum in drei Bereiche (Environments) ein, die jeweils Informationenmengen bestimmter Kontextdomänen abdecken [SAW93]:

- **Computing Environment:** Die „technische Umgebung“ bezeichnet die rechen- und kommunikationstechnisch verfügbaren Ressourcen, welche einem Benutzer zu einem bestimmten Zeitpunkt zugänglich sind. Dazu gehören Geräte in der Umgebung des Benutzers, wie etwa Drucker oder Ein- und Ausgabekonsolen, aber auch netzwerkspezifische Informationen: z.B. die Kapazität oder die Kosten für die Nutzung eines Netzwerkdienstes.
- **User Environment** bezeichnet den Aufenthaltsort des Benutzers und die Anwesenheit anderer Personen in seiner Nähe. Schilit macht hier keine Unterscheidung zwischen Low-Level- und High-Level-Kontext. Positionsinformationen gehören hierzu genau so wie Angaben zu umgebende Personen und die soziale Situation, in der sich der Benutzer befindet.

- **Physical Environment**: Zu diesem Bereich gehören hauptsächlich (außer Positionsinformationen) Low-Level-Kontexttypen. Diese sind meist optische, akustische, biologische und andere Informationen (vgl. Abschnitt 2.4.2), die aus physikalische Sensoren bezogen werden.

Hierarchischer Ansatz

Ein *hierarchischer* Ansatz, der zum Teil der Vorgehensweise von Schmidt ähnelt, stammt von Beigle et al. [BGS98]. Auf der ersten Hierarchieebene unterscheidet er zwischen „menschliche Faktoren“ und Informationen über die „physikalische Umgebung“ (siehe Abbildung 2.19). Die zweite Hierarchieebene wird gegliedert durch die Kategorien: „human factors“, „user“ (z.B. Gewohnheiten, physiologische Eigenschaften), „social environment“ (z.B. Nähe zu anderen Personen, Beziehungen) und „task“ (z.B. Aktivitäten, Aufgaben). Die „Physical-Environment-Ebene“ ist in „location“ (z.B. relative und absolute Ortsangaben), „infrastructure“ (z.B. vorhandene technische Infrastruktur) und „conditions“ (physikalische Bedingungen wie Temperatur und Lautstärke) gegliedert.
Der Ansatz von Beigle et al. stellt auch die Grundlage für die Spezifikation eines Kontextmodells (vgl. Abschnitt 3.1) dar. Aus diesem Grund wurden für bestimmte Kontextinformationen weitere Eigenschaften festgelegt. Dabei erhalten Kontextinformationen einen eindeutigen Namen. Jedes Element in der Kontexthierarchie besitzt bestimmte Charakteristika. Ein Charakteristikum selbst besteht aus einem (Kontext-)Wert und einer Einheit. Für den Kontextwert gilt außerdem ein festgelegter Wertebereich.

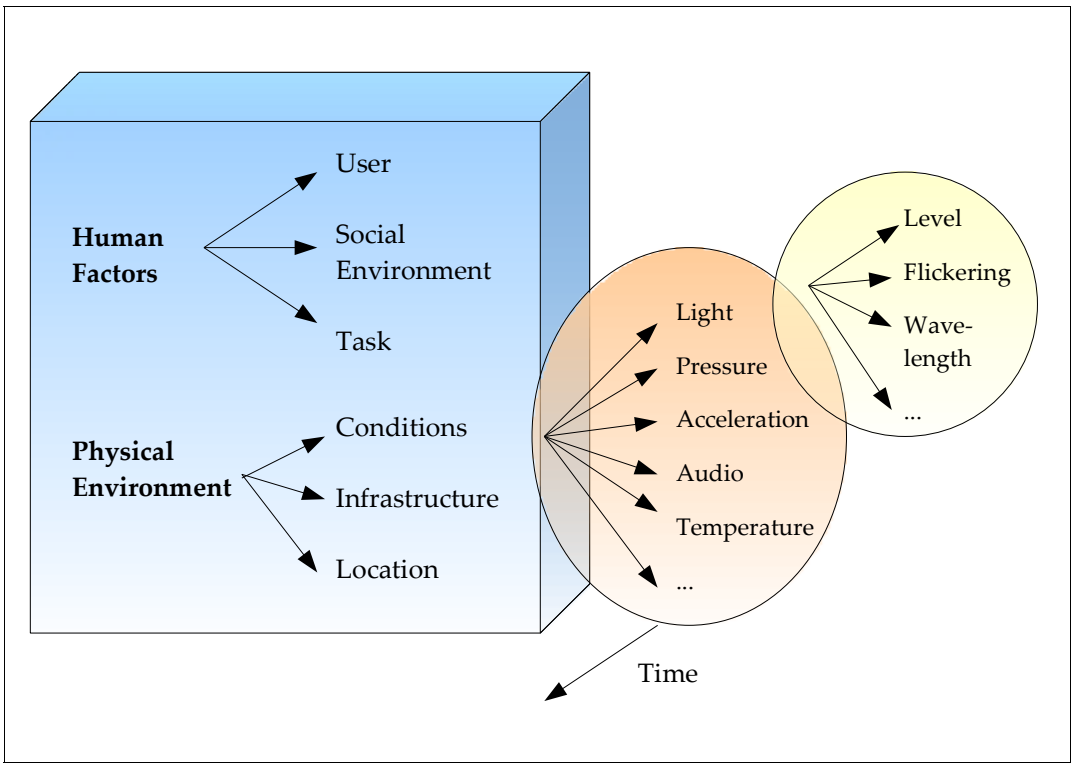


Abbildung 2.19: Hierarchische Klassifzierung (nach [BGS98])

3D-Kontext-Klassifizierung

Das *dreidimensionale Modell* von Schmidt et al. [SATT+99] unterteilt drei wichtige Aspekte des Kontextbegriffs in drei gleichgestellte Dimensionen (Abbildung 2.20). Im Gegensatz zum hierarchischen Ansatz von Beigle et al. [BGS98] ist die Aktivität nicht mehr der Kategorie „human factors" untergeordnet, sondern umfasst jegliche Art von Aktivitäten, die für eine mobile Anwendung von Bedeutung sein kann. Dies schließt beispielsweise nicht nur die aktuelle Tätigkeit des Benutzers ein, sondern auch Prozesse, die das mobile Gerät gerade ausführt.

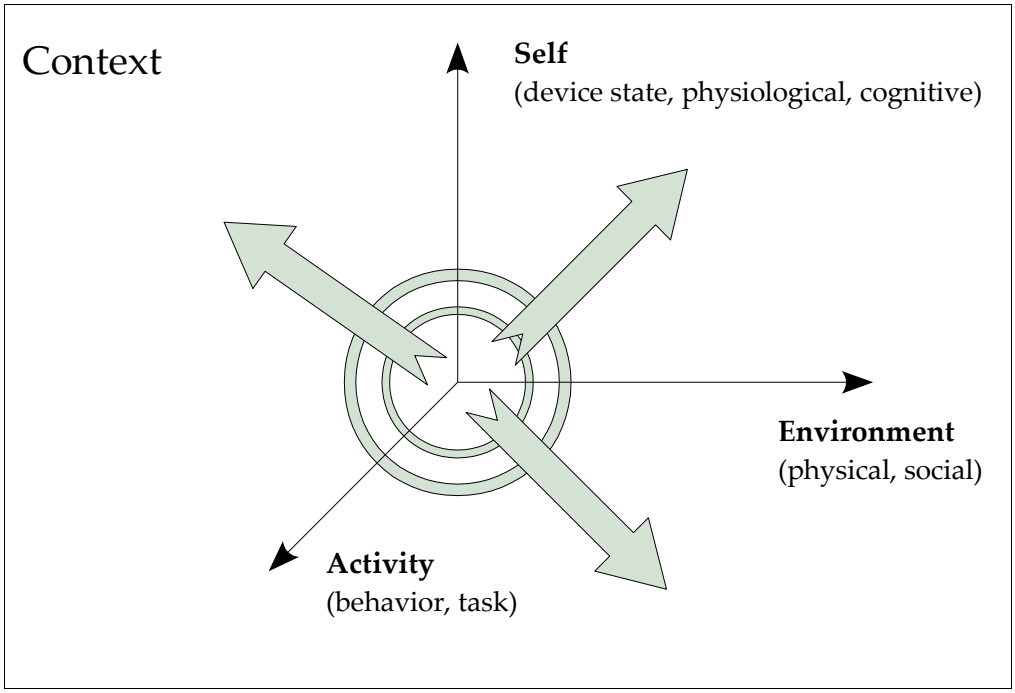


Abbildung 2.20: 3D-Kontext-Klassifizierung (nach [SATT+99])

Eigener Ansatz

Da in Zukunft entsprechend der Vision des Ubiquitous und Pervasive Computing zahlreiche neue kontextsensitive Anwendungen entstehen können, werden wahrscheinlich auch bisher nicht verwendete Arten von Kontextinformationen eingesetzt werden. Um diese zukünftige Entwicklungen in einer Kategorisierung einzubeziehen, bedarf es eines *generischen* Ansatzes zur Klassifizierung, die möglichst alle Formen von Kontextinformationen in entsprechende Klassen einteilt.

Der Ansatz von Schilit ist zu sehr auf die Umgebung konzentriert. Für ihn ist Kontext in erster Linie Umgebungsinformationen. Er unterscheidet in diesem Zusammenhang nur zwischen verschiedene Arten von Umgebungen. Beigles Kontexthierarchie unterscheidet zwar Kontextinformationen auf verschiedene Ebenen und spaltet sie sukzessiv auf, jedoch fehlen zentrale Kontextelemente wie *Aktivität* und Eigenschaften des mobilen Gerätes. Bei dem 3D-Kontextmodell findet das Kontextelement *Zeit* keine Zuordnung. Die Zeit ist zum einen für andere Kontextinformationen von Bedeutung, da sie beispielsweise als Stempel die

zeitliche Dimension – und damit auch den Gültigkeitszeitraum – von Low-Level-Kontext wiedergibt, und zum anderen selbst als Kontextinformation benutzt werden kann.
Um die genannten Defizite der obigen Ansätze entgegen zu wirken, werden die vier *Dimensionen* des Kontext festgelegt: *Umgebung*, *Aktivität*, *Identität* und *Zeit* (siehe Abbildung 2.21). Dabei wird zunächst von zusätzlichen Details zu den Dimensionen abstrahiert. Zum Beispiel gibt die Dimension Umgebung nicht an, um welche Form von Umgebung es sich handelt. Auch die Begriffe Aktivität und Identität lassen es offen, ob sie den Benutzer oder dem mobilen Gerät betreffend gemeint sind. Die Dimensionen identifizieren zentrale Elemente, die ein Kontext charakterisiert: Die Umgebung gibt das Umfeld des Kontext an. Also damit auch *wo* etwas stattfindet. Die Aktivität subsumiert Informationen darüber welche Entitäten mit *was* gerade beschäftigt sind. Um zu erfahren *wer* (z.B. Objekte oder Personen) in den Kontext involviert ist, müssen Entitäten identifiziert werden. Die Zeit beantwortet die *Wann*-Frage des Kontexts.
Eine zweite Ebene unterteilt die einzelnen Kontextdimensionen in unterschiedliche *Subdimensionen*. Die physikalische und soziale Umgebung mit vorhandener Kommunikationsinfrastruktur sind Elemente der Dimension Umgebung. Bei Aktivität und Identität scheint die Kategorisierung für das mobile Gerät und dem mobilen Benutzer sinnvoll. Die dargestellten Subdimensionen sind nur einige Ausprägungsformen der Dimensionen. Sie erheben keinen Anspruch auf Vollständigkeit. Andere Subdimensionen, wie beispielsweise Aktivität oder Identität anderer Entitäten, können ergänzt werden, d.h. der entsprechenden Dimension untergeordnet werden. Damit erhält dieser Klassifizierungsansatz einen *dynamischen* und *erweiterbaren* Charakter.

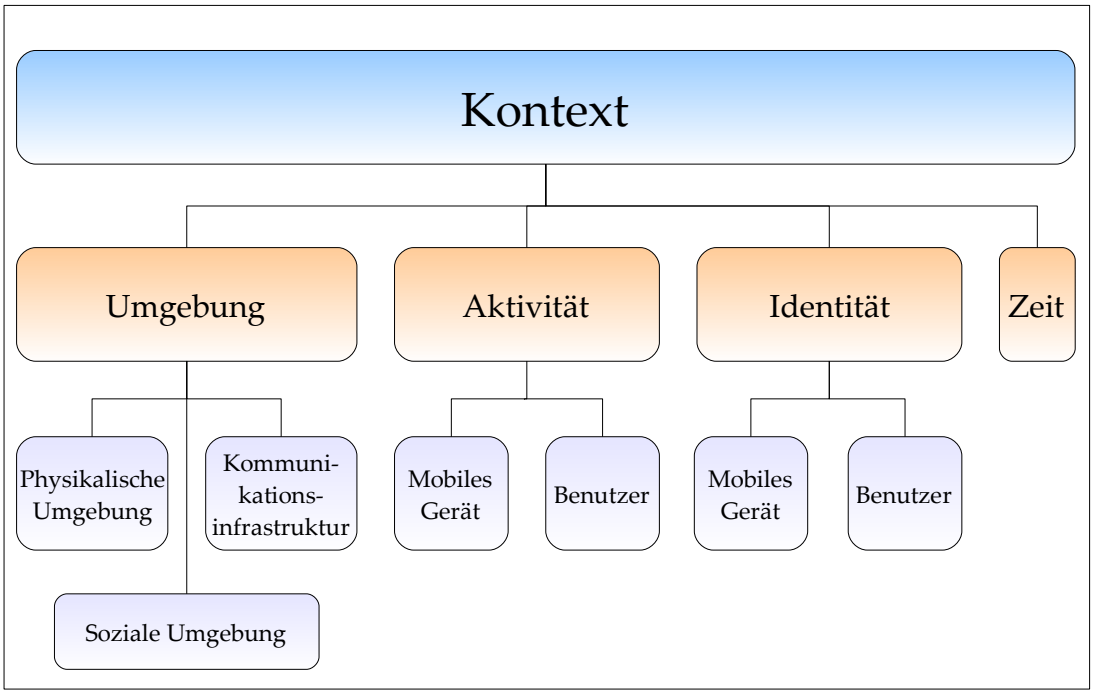


Abbildung 2.21: Kontextklassifizierung

2.4.4 Arten der Kontextadaptivität

Mobile Systeme, die Kontextinformationen einsetzen, um eine bestimmte Funktionalität zu erfüllen, werden als kontextadaptive oder kontextsensitiv bezeichnet. Nach Pascoe [Pas98] besteht die Herausforderung in der Entwicklung solcher Systeme. Also im Besonderen bei der *Konstruktion*, *Präsentation* und *Verarbeitung* von Kontextinformationen.
Die Konstruktion bezeichnet die Ermittlung solcher Informationen, die in der Regel den Einsatz von Sensoren und Softwarekomponenten voraussetzt. Damit kontextbezogene Informationen zwischen einzelne Anwendungskomponenten einerseits und zwischen verschiedenen Geräten in einer verteilten Umgebung andererseits ausgetauscht werden können, ist eine einheitliche Präsentationsform der Kontextinformationen erforderlich. Der Prozess der Verarbeitung behandelt den Aspekt, wie die Kontextinformationen in mobilen Anwendungen eingesetzt werden. Der Verarbeitungsprozess ist sehr anwendungsspezifisch. Jedoch lassen sich Gruppierungen von mobilen kontextsensitiven Anwendungen bilden, die sich hinsichtlich der Form der Kontextnutzung ähneln.
Einer der ersten Versuche kontextsensitive Anwendungen zu klassifizieren, unternahm Schilit [SAW93]. Dabei lassen sich nach Schilit alle kontextsensitive Systeme in vier Klassen einteilen: „*Proximate Selection*“, „*Automatic Reconfiguration*“, „*Contextual Commands*“ und „*Context-triggered actions*“. Zusätzlich werden diese Klassen in folgende orthogonale Dimensionen eingeteilt: *datenbezogen*, *aktionsbezogen*, *automatisch* und *manuell*. Abbildung 2.22 zeigt die Einordnung der Klassen in entsprechende Dimensionen.

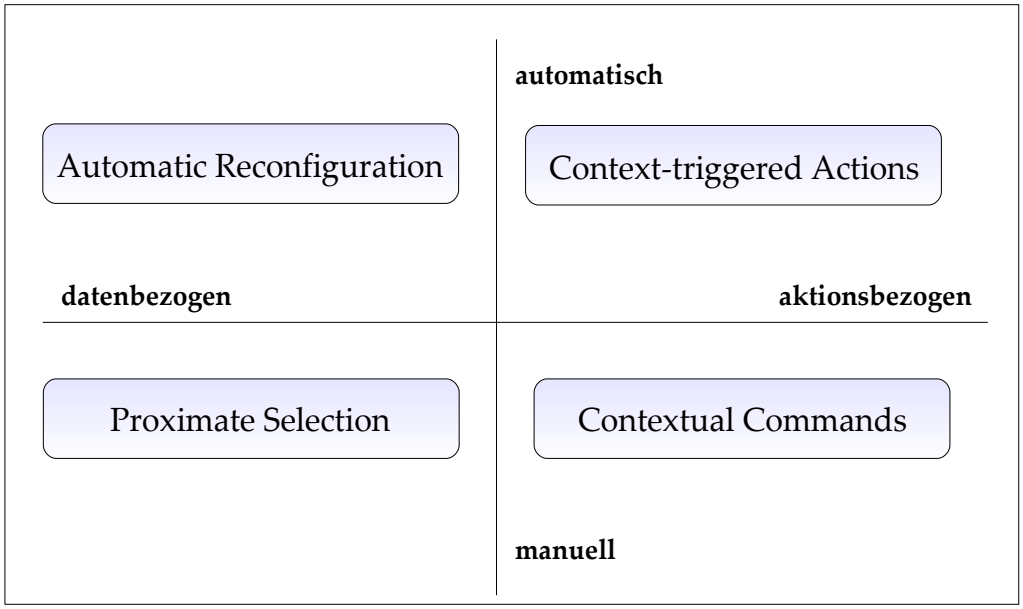


Abbildung 2.22: Schilits Klassifizierung kontextsensitiver Anwendungen

Eine Anwendungsfunktionalität, die durch manuelle Eingabe des Benutzers gesteuert – aber kontextbezogen - Daten zustellt, wird der Gruppe Proximate Selection zugeordnet. Ein Beispiel dafür ist das Anzeigen nächstgelegener Drucker, wenn der Benutzer ein Dokument drucken möchte.

Im Gegensatz zu Proximate Selection werden Anwendungsfälle der Gruppe Automatic Reconfiguration zugeteilt, die bei der Zustellung von Daten automatisch unter Berücksichtigung kontextsensitiver Daten eine Verbindung zu einer vorhandenen Ressource herstellt. Bezogen auf das Druckbeispiel wird ein Druckauftrag automatisch zum nächstgelegenen Drucker gesendet.
Anwendungsaktionen, die manuell vom Benutzer initiiert und im Bezug zum aktuellen Kontext ausgeführt werden, gehören zu Contextual Commands. Hier besteht die Möglichkeit zum einen Dienste kontextsensitiv anzubieten und zum anderen die Ausführung von Diensten zum aktuellen Kontext anzupassen.
Schließlich werden Operationen einer Anwendung, die vollständig autonom und kontextbezogen Aktionen ausführen, als Context-Triggered Actions bezeichnet. Das bedeutet, sobald ein Kontext festgestellt wird, werden Aktionen ohne Beeinflussung des Benutzers umgesetzt. Beispielhaft dafür ist die automatische Umleitung eingehender Anrufe in dem Raum, in dem der Benutzers sich gerade befindet.
Dey [DeAb99] und Becker et al. [RBB03] abstrahieren im Gegensatz zu Schilit von den vier Dimensionen der Klassen kontextsensitiver Anwendungen. Datenbezogene, aktionsbezogene, manuelle oder automatische Operationen werden weniger in den Fokus gestellt als die Art der kontextbezogenen Funktion der Anwendung selbst. Die Beschreibung der folgenden Klassifizierung basiert weitgehend auf Becker et al. [RBB03]:

- **Kontextbezogene Selektion**: Eine mobile Anwendung kann Auswahl von Informationen und Diensten kontextbezogen steuern. Dabei ist gerade aufgrund der Mobilität des Benutzers und des mobilen Gerätes die Identifizierung und Selektion von erreichbaren Diensten im aktuellen Kontext zunehmend von Bedeutung. Becker et al. sehen Informationen über die unmittelbare Umgebung (z.B. Taxi, Busfahrplan, Restaurant in der Nähe, usw.) als besonders interessant für innovative mobile Anwendungen [RBB03].
- **Kontextbezogene Präsentation**: Abhängig von einem bestimmten Kontext kann die Darstellung bestimmter Anwendungsdaten angepasst werden. Beispielsweise können in Bezug auf die Geschwindigkeit eines zu navigierenden Benutzers die anzuzeigenden Navigationselemente angepasst werden oder Ausgabemöglichkeiten für Video- und Audiodateien am aktuellen Ort genutzt werden.
- **Kontextbezogene Aktionen**: Durch die Ermittlung des Ort und der Identität eines Benutzer als Primärkontext können automatisch Aktionen kontextbezogen ausgeführt werden. Diese Klasse von Anwendungsoperationen fasst die beiden Klassen Contextual Commands und Context-triggered Actions der Taxonomie von Schilit zusammen.

Die folgende Tabelle zeigt welche kontextsensitiven Anwendungen, die im Abschnitt 2.3 ausführlich vorgestellten wurden, in welche Klassen (im Sinne von Becker) eingeteilt werden können. Zusätzlich wird der benutzte Primärkontext zur Selektion der dazugehörigen Sekundärkontexte angegeben.

Tabelle 2.4: Klassifikation der vorgestellten Anwendungen

Projekt	Beschreibung	Primärkontext	Merkmale kontext-abhängigen Verhaltens
CyberGuide	Gebäudeführer	Identität, Lokation	Selektion
REAL	Navigations-system	Identität, Lokation	Präsentation
Active Badge Location System	Anrufweiter-leitung	Identität, Lokation	Aktion
ParcTab System	Elektronischer Büroassistent	Identität, Lokation, Zeit	Selektion, Aktion
Conference Assistant	Interaktive Unter-stützung von Konferenz-teilnehmern	Identität, Lokation, Aktivität, Zeit	Selektion, Präsentation, Aktion
Context-Call	Kontextbez. Mobiltelefon	Identität	Selektion, Präsentation
Orientierungs-sensitiver PDA	PDA mit Orientierungs-sensoren	-	Präsentation
Kontextsensitives Mobiltelefon	Mobiltelefon mit phys. Sensoren	Identität	Präsentation, Aktion
Stick-e Notes	Virtuelle Post-Its	Identität, Lokation	Selektion, Aktion
StartleCam System	Archivierung visueller Eindrücke	Identität	-

3 Modelle und Frameworks zur Entwicklung kontextsensitiver mobiler Anwendungen

Die im Abschnitt 2.3 vorgestellte Systeme sind erste Prototypen kontextsensitiver Anwendungen. Sie sind meist ein in sich geschlossenes System, das die benutzten Kontextinformationen als Teil des internen Softwaresystems modellieren und verwalten. Dadurch entsteht ein zusätzlicher *Aufwand* bei der Entwicklung von kontextsensitiven Systemen, da jede Anwendung selbst die zu berücksichtigen Kontexte auf eine geeignete Datenstruktur abbilden und verwalten muss. Neben dem entstanden Aufwand, bedeutet die Kontextsensitivität in solchen Systemen *Heterogenität* und *Inkompatibilität* von Kontextinformationen. Unterschiedliche Anwendungen können in einem mobilen verteilten System aufgrund *fehlender Standards* keine Informationen untereinander austauschen und somit interagieren.
Das folgende Kapitel beschäftigt sich mit den ersten Modellierungsansätzen und Frameworks, die aus der Idee entstanden sind, Kontextinformationen anwendungsunabhängig in einem allgemeinen, erweiterbaren Modell abzubilden. Um die Daten des Kontextmodells zu verwalten, ist zusätzlich eine *Entkopplung* des Kontext-Management (Ermittlung, Auswertung und Verwaltung der Modelldaten) von der Anwendung notwendig. So lassen sich bereits implementierte Komponenten des Frameworks von anderen Anwendungen *wiederwenden*.

3.1 Modellierungsansätze

Um Kontextinformationen, in einer Anwendung benutzen zu können, müssen sie durch ein geeignetes Modell repräsentiert werden. Solch ein Model erlaubt es der Anwendung, benötigte Daten in standardisierter Form zu beziehen. Auf dieser Grundlage können mobile Anwendungen die kontextbezogenen Informationen interpretieren. Das Modell legt also fest wie die einzelnen Informationen in Form von Datenstrukturen ausgedrückt werden und welche Erweiterungsmöglichkeiten bestehen, um sie anwendungsspezifisch anzupassen.
Der heutige Stand der Technik zur Modellierung von Kontextdaten lässt sich nach der verwendeten Datenstruktur klassifizieren. Eine *technologiegetriebene* Klassifizierung der verwendeten Modelle nehmen Strang et. al vor [STLP04]. Sie teilen die Modelle zur Präsentation des Kontexts in sechs verschiedene Klassen ein, die sich durch die verwendete Technik zur Präsentation der Informationen unterscheiden. Anschließend werden die untersuchten Ansätze hinsichtlich festgelegter Kriterien bewertet.

Key-Value Modell

Die einfachste Form zur Modellierung von Kontextinformationen ist eine Strukturierung durch ein Schlüssel-Wert-Paar. Dabei legt ein Dienst zum Bereitstellen von Kontextinformationen den Kontextwert (beispielsweise die aktuelle Position) zu einem entsprechenden Schlüssel als Umgebungsvariable des Systems ab (z.B. als USER_POS). Eine kontextbezogene Anwendung greift über den Schlüssel auf den dazugehörigen Kontextwert zu. Bereits Schilit et al. [SAW93] benutzten diese Methode, um Anwendungen Kontextinformationen zur Verfügung zu stellen. Der Key-Value-Ansatz ist eine einfache Modellierungsmethode und mit relativ wenig Aufwand zu realisieren. Sie eignet sich jedoch

nicht zur Strukturierung komplexer Kontextinformationen, die Anwendungen benötigen, um adaptiv zu reagieren.

Markup Schema Modelle

In diesem Modell werden Kontextdaten von Form von Markup-Tags mit Attributen und Inhalten modelliert, die rekursiv weitere Markup-Tags beinhalten können. Typische Vertreter dieser Modelle werden zur Strukturierung von Profilinformationen eingesetzt, die auf einer Serialisierung in einem Dialekt der „Standard Generic Markup Language“ (SGML) basieren. SGML ist dabei die Ursprache aller Markup-Sprachen (wie zum Beispiel XML).
Markup Schema eignen sich aufgrund ihrer universellen Strukturierbarkeit und starken Verbreitung auf verschiedenen Plattformen zum Austausch von Kontextinformationen. Typische Markup Schema Modelle sind das „Comprehensive Structured Context Profiles“ (CSCP) [HBS02] und die „CC/PP Context Extension“ [IRRH03]. Sie orientieren sich beide an das W3C-Standard „Composite Capability/Preference Profiles” (CC/PP) [W3C] und nutzen XML zur Serialisierung.
CC/PP spezifiziert ein Basisvokabular zur Beschreibung von Geräteeigenschaften und Nutzerpreferenzen. Das Basisvokabular beinhaltet einen Katalog von Beispieldaten. Gerätehersteller und Anwendungsentwickler werden aufgefordert eigene Attribute zu definieren, um ihre Geräte bzw. Anwendungen zu beschreiben. Grund für die Spezifikation von CC/PP war das Abrufen von Internetseiten in einem mobilen Gerät. Dabei wurden Profilinformationen zwischen dem mobilen Client und dem Webserver ausgetauscht, um die angeforderten Inhalte an das Profil des Benutzers und an den Eigenschaften des mobilen Gerätes anzupassen. Die Nutzung des CC/PP Profils erfolgt dem entsprechend nach folgendem Schema: Wenn ein Endgerät eine Anfrage per HTTP an einen Webserver schickt, übermittelt es im HTTP-Header das CC/PP Profil in der Anfrage mit. Der Webserver kann daraufhin die zu verarbeitenden Daten filtern, transformieren und entsprechend den Anforderungen des Clients anpassen. CC/PP Profile werden dabei mit dem „Ressource Description Framework“ (RDF) beschrieben. RDF kann mittels XML serialisiert werden. Quelltext 3.1 zeigt ein einfaches CC/PP Profil, das die Geräteinformationen Bildschirmgröße und verfügbaren Arbeitsspeicher beinhaltet.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:ex="http://example.com/Schema#">

      <rdf:Description rdf:about="http://example.com/HardwareDefaults">
            <rdf:type rdf:resource=
                  "http://example.com/Schema#Hardware"/>
            <ex:displayHight>400</ex:displayHight>
            <ex:displayWidth>600</ex:displayWidth>
            <ex:memoryMb>32</ex:memoryMb>
      </rdf:Description>

</rdf:RDF>
```

Quelltext 3.1: Beispiel für ein CC/PP Profil (nach [W3C05])

Ein einfaches XML-basiertes Protokoll zum Austausch von kontextrelevanten Daten stellt das Projekt *ConteXtML* [Rya99] bereit. Dabei kann ein mobiler Client sowohl seinen eigenen Kontext einem Kontextserver mitteilen als auch benötigte Kontextinformationen anfordern. Im Quellcode 3.2 übermittelt das mobile Gerät mit den Tags `<person>` und `<spatial>` seine Identität bzw. Position an einen Server und fordert mit dem `<require>`-Tag detaillierte Karteninformationen zu seiner Position.

```
<context session="new">
    <person role="user" first="Mirwais" last="Turjalei" />
    <spatial proj="UTM" zone="33" datum="2006-02-11 13-34-56">
        <point x="281993" y="4686790" z="205" />
    </spatial>
    <require>
        <map name="contours" />
        <map name="roads" class="minor" />
        <map name="buildings" />
    </require>
</context>
```

Quelltext 3.2: Ein ConteXtML-Protokoll (nach [Rya99])

Objektorientierte Modelle

Ansätze zur objektorientierten Modellierung nutzen allgemeine objektorientierte *Paradigmen*, um Kontextinformationen für mobile Anwendungen bereitzustellen. Aspekte der objektorientierten Programmiersprache wie beispielsweise *Kapslung*, *Wiederverwendbarkeit* und *Vererbung* stellen Methoden dar, um zusammengehörige Kontextinformationen zu Klassen zu kapseln und von Details komplexer Daten zu abstrahieren.
Das im TEA Projekt [SATT+99] entwickeltes Modell kann als typischer Vertreter eines objektorientierten Ansatzes zur Modellierung des Kontextes angesehen werden. Bei Ansätzen dieser Art sind Kontextinformationen in einem Objekt als Statusinformationen gekapselt. Der Zugriff auf diese Informationen ist dabei nur über die Schnittstelle des Objektes möglich. Außerdem wurden verschiedene Typen von Kontextinformationen, gekapselt in entsprechenden Objekten, auf einer Schichtenarchitektur abgebildet. Abbildung 3.1 zeigt den Prozess der Kontextverarbeitung auf den verschiedenen Ebenen.

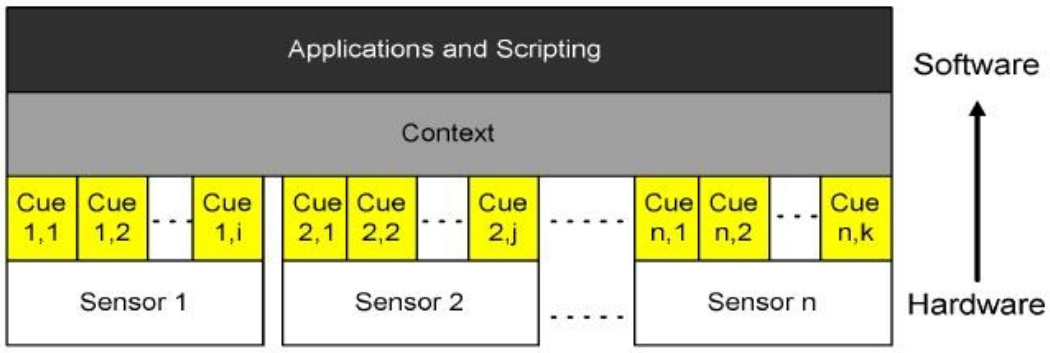


Abbildung 3.1: Architektur des TEA Systems [SATT+99]

So genannte *Cues* stellen Informationen dar, die direkt von logischen oder physikalischen Sensoren (vgl. Abschnitt 2.4.2) bezogen werden. Sie abstrahieren von der vorhandenen Sensorentechnik, mit dessen Hilfe Kontextinformation gemessen und geliefert werden. Jedes Cue erhält seine Informationen aus genau einem Sensor. Allerdings können auch andere Cues auf den selben Sensor zugreifen. Die Funktion des Cues besteht im wesentlichen darin, die von Sensoren ausgehende Datenmenge zu reduzieren oder unter Verwendung von Statistiken zu extrapolieren. Auf der Kontextebene werden Kontextinformationen, die aus verschiedenen Cues stammen, zusammengeführt, um daraus komplexe Kontext-

informationen zu gewinnen. Dieser Prozess wird im Rahmen des TEA Projekts als *Fusionsprozess* bezeichnet. Die Anwendungsebene stellt Mechanismen zum Zugriff auf die Kontextinformationen bereit. Zusätzlich wird einer Anwendung die Möglichkeit gegeben auf die Ereignisse „entering a context", „leaving a context" und „while in a context" zu reagieren.

Logikbasierte Modellierung

Der Ansatz zur logikbasierter Modellierung ist durch einen hohen Grad an formaler Beschreibung von Kontextinformationen gekennzeichnet. Dabei werden Kontexte durch *Fakten*, *Ausdrücke* und *Regeln* in einem Regelbasierten System spezifiziert. In der Regel werden Low-Level-Kontextinformationen durch Hinzufügen neuer Fakten und Logikregeln in das System eingebracht. Mit Hilfe der logischen Regeln wird das kontextuelle Wissen (High-Level-Kontextinformationen) durch logisches Schließen abgeleitet. Diese Vorgehen wird auch als *Reasoning* oder *Inferencing* bezeichnet [McBu97].
Aufgrund des formellen Ansatzes dieser Methode eignen sie sich besonders gut zur Beschreibung von Kontext im Bereich der Künstlichen Intelligenz. Vertreter dieses Konzepts sind z.B. McCarthy und Buvac, die Kontext als abstrakte mathematische Entitäten beschreiben [McBu97]. Sie versuchten ein Formalisierungskonzept anzugeben, das es erlaubt, einfache, allgemein geltende, statische Erscheinungen als Axiome darzustellen, die sich zu einem komplexeren und sich verändernden Kontext abstrahieren lassen.
Üblicherweise bietet die logikbasierte Modellierung kaum Möglichkeiten zur Strukturierung komplexer Kontextinformationen und ist zu ressourcenintensiv für die Verwendung in mobilen Systemen.

Ontologische Modelle

Die Ontologie, als eine Methode der Wissenspräsentation, dient zur formalen Beschreibung eines Systems mit Hilfe von *Konzepten* und *Relationen*. Zusätzlich zu Konzepten und Relation beinhalten Ontologien *Inferenz-* und *Integritätsregeln*. Gefördert durch die Idee des *Semantic Web* [W3CS01] sind verschiedene formale Sprachen, wie beispielsweise RDF-Schema, DAML-OIL und OWL (Web Ontology Language), entstanden.
Die ersten Forscher, die Ontologien benutzen, um kontextuelles Wissen zu modellieren, waren Ötzturk und Aamodt [ÖzAa97]. In ihren psychologischen Studien untersuchten sie den Unterschied zwischen dem Erinnern (recall) und dem Erkennen (recognize) von unterschiedlichen Sachverhalten in Verbindung mit Kontextinformationen. Dabei ergab sich die Notwendigkeit Wissen verschiedener Domänen zu normieren und zu kombinieren [ÖzAa97]. Zur Lösung dieses Problems eigneten sich Ontologien aufgrund ihrer speziellen Fähigkeit zur Wissensmodellierung besonders gut.
Ein weiterer Ansatz zur ontologiebasierter Kontextmodellierung ist der *„Aspect Scale Contextinformation"* (ASC) [Hob02]. Die ASC sieht bei der Verwendung von Ontologien einen einheitlichen Weg vor, die Kernkonzepte des Modells und eine beliebige Anzahl von Subkonzepten sowie Sachverhalten zu spezifizieren. Dieser Ansatz ermöglicht Kontextwissen in einem ubiquitären Informationssystem zu verteilen (context sharing) und wieder zu verwenden [Che04].
Das Modell *CONON* (CONtext ONtology) von Wang et al. [WGPZ04], das auf der Ontologiesprache OWL basiert, entstand aus der gleichen Idee heraus, wie sie auch der ASC zugrunde liegt. Es wurde entwickelt, um Knowledge Sharing, logische Schlussfolgerung und Wiederverwendbarkeit von Wissen zu ermöglichen. Abbildung 3.2 zeigt die Beziehungen zwischen den einzelnen OWL-Klassen im CONON-Model.

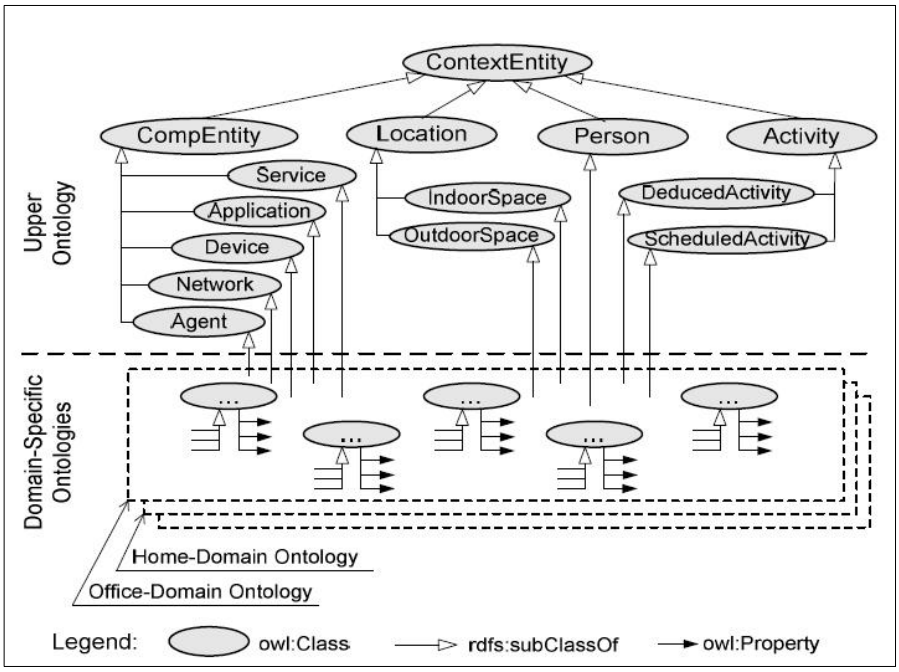


Abbildung 3.2: Das CONON Modell [WGPZ04]

Die Ontologieklassen sind in zwei Schichten eingeteilt. Die Schicht „*Upper Ontology*" beinhaltet die vier Grundkonzepte *Person*, *Activity*, *Computational Entity* (CompEntity) und *Location.* Außerdem werden in dieser Schicht Eigenschaften und Beziehungen der Grundkonzepte, sowie eine Menge von Unterklassen spezifiziert. Die „*Domain-Specific-Ontologies-Schicht"* besteht aus einer Sammlung von bereichsspezifischen Ontologien mit den Besonderheiten der entsprechenden Subdomäne. Im Vergleich hierzu abstrahiert die *„Upper-Ontology-Schicht"* von einer konkreten Domäne. Konzepte der domänspezifischen Ontologies erweitern die allgemeinen Grundkonzepte, um domänspezifisches Wissen zu modellieren.

3.2 Frameworks

Bei der Realisierung kontextsensitiver Anwendung tritt es häufig auf, dass nicht nur heterogene Kontextinformationen ermittelt und in einem geeigneten Modell abgebildet werden müssen, sondern es müssen auch die Modelldaten geeignet *verwaltet* werden, damit eine Anwendung sie effizient nutzen kann. Um die Entwicklung kontextsensitiver Anwendungen zu vereinfachen, ist eine Entkopplung der Kontextmodellierung und Kontext-Management von der konkreten mobilen Anwendung erforderlich. Konsequenterweise entsteht aus dieser Anforderung eine anwendungsunabhängige Systeminfrastruktur einerseits und ein zum Teil anwendungsabhängiges Kontextmodell andererseits. Idealerweise schafft eine geeignete *Middleware* in Form eines Frameworks eine *Abstraktionebene*, die die Ermittlung, Bereitstellung und den Austausch von Kontextinformationen aus der Perspektive der Anwendung *transparent* gestaltet.

Chen unterscheidet in [Che04b] drei verschiedene Methoden, um physikalische Umgebungsinformationen für eine mobile Anwendung bereitzustellen. Die Arten der Kontextbereitstellung fasst er unter dem Begriff „context acquisition" zusammen. Die Methode des „*Direct sensor access*" bezeichnet die direkte Erfassung und Bereitstellung von Low-Level-Kontextinformationen mit Hilfe von Hardware-Sensoren, die im mobilen Gerät integriert sind. Mobile Anwendungen können durch Nutzung einer gerätsspezifischen API benötigte Sensordaten auf niedriger Ebene beziehen. Ansätze der „*Middleware infrastructure*" bieten eine

Architektur, die die technischen Details sensorbezogenen Daten verbergen und höherwertige Kontextinformationen für eine mobile Anwendung liefern. Zusätzlich dazu erlaubt eine Middleware-Infrastruktur die Beobachtung von Kontextinformationen, um Anwendungen die Möglichkeit zu geben auf Kontextänderungen zu reagieren. Die dritte Möglichkeit, um Kontextinformationen zu beziehen, ist nach Chen die Nutzung eines entfernten „*Context server*“, der die Umgebung mobiler Clients beobachtet und auf deren Anfrage Kontextinformationen liefern kann [Chen04b].
Da im Rahmen dieser Arbeit die Integration von Context-Awareness in eine Middleware für *mobile Systeme* im Vordergrund steht, werden in diesem Abschnitt vorhandene Middleware-Infrastrukturen untersucht, die die Entwicklung von kontextsensitiven mobilen Anwendungen unterstützen. Auf die Untersuchung von Infrastruktursysteme, bei denen die Modellierung und das Management von Kontextinformationen vollständig oder zum Teil auf einem zentralen Server stattfinden, wie beispielsweise bei der Nexus-Plattform [DNHB+04], wird verzichtet.

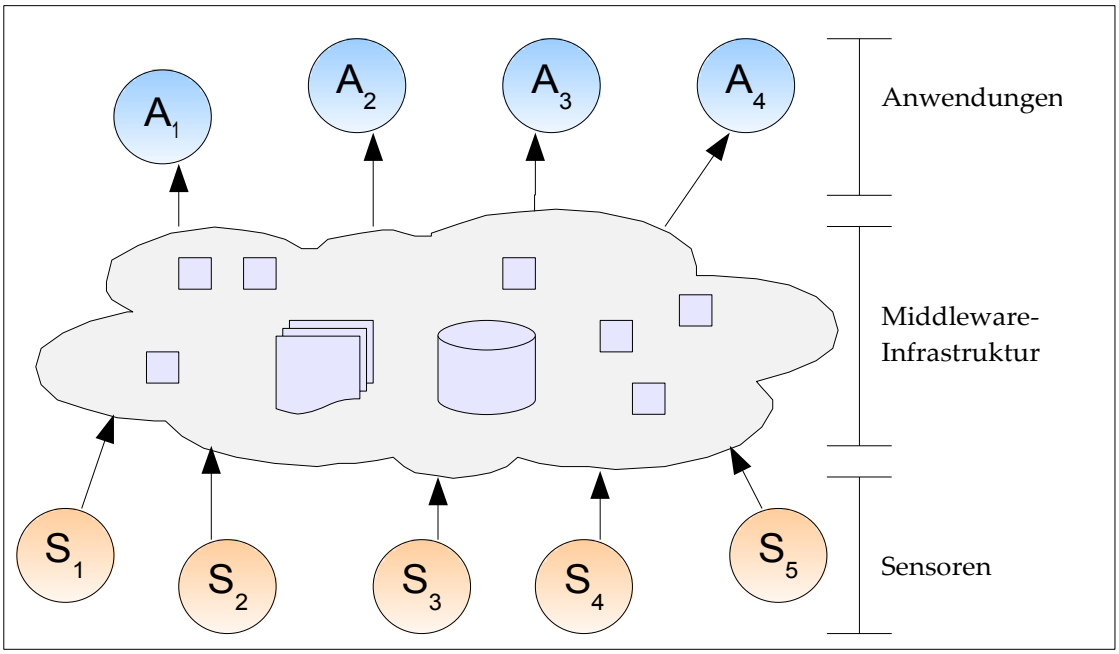


Abbildung 3.3: Grundstruktur einer Middleware für kontextsensitive mobile Anwendungen

Auch vorhandene *ontologische Ansätze* zur Konstruktion von kontextsensitiven Anwendungen, wie zum Beispiel *SOCAM* [WZGP04] oder *CoBrAs* [CFJ03], werden nicht näher untersucht, da sie aufgrund ihrer Komplexität für den Einsatz in mobilen Geräten zu schwergewichtig sind. Im Rahmen dieser Diplomarbeit sind in erster Linie leichtgewichtige Kontext-Management-Systeme von Interesse, die geeignete Werkzeuge für Anwendungsentwickler zur Verfügung stellen und mit der Anwendung auf mobilen Geräten ausgeführt werden.
Im Wesentlichen sind, die im Folgenden beschrieben Frameworks, sensororientierte Systeme. Das bedeutet, sie stellen eine Infrastruktur für mobile Systeme bereit, die durch eine Schicht die Trennung zwischen Anwendungen und Sensoren anstrebt (siehe Abbildung 3.3). Ihre Aufgabe besteht vor allem darin, aus Sensoren Informationen über die physikalischen Bedingungen der Umgebung zu entnehmen, zu verarbeiten und diese der Anwendung über standardisierte Schnittstellen als Kontextinformationen zur Verfügung zu stellen.

3.2.1 Context Toolkit

Mit der Entwicklung eines Frameworks zur Unterstützung von Entwicklern bei der Konstruktion kontextsensitiver Anwendungen begannen Salber, Dey und Abowd 1999 am „Georgia Institute of Technology“. Das *Context Toolkit* [SDA99] bietet eine objektorientierte Architektur, in der über den Ort hinaus, beliebige dynamische Umgebungsinformationen berücksichtigt werden. Das Ziel dieses Ansatzes ist es vor allem sensorbezogene Rohdaten zu filtern und in Objekten zu kapseln, die von der Art und Komplexität eines Sensors abstrahieren, wiederverwendbare Komponenten anbieten, den Austausch von Kontextinformationen in einem Netzwerk ermöglichen, sowie Persistenz und Historisierung kontextsensitiver Daten unterstützen.
Das Besondere an dem Context Toolkit ist die Art wie von konkreten Sensoren abstrahiert wird. Dabei folgen Salber et al. den bereits vorhandenen Abstraktionsansatzes der *GUI Widgets*. Der Begriff GUI Widget ist ein Sammelbegriff für Interaktionselemente in grafischen Benutzeroberflächen, wie beispielsweise Schaltflächen, Textfelder oder Auswahllisten. Sie abstrahieren innerhalb der Anwendung von den unterschiedlichsten Eingabenmöglichkeiten des Benutzers, indem sie zwischen dem Benutzer und der Anwendung agieren, um Benutzereingaben für die Anwendung in einfacher Form zur Verfügung zu stellen. Analog dazu abstrahieren *Context Widgets* als zentrale Komponente im Context Toolkit mit Hilfe von Sensoren von der physikalischen Umgebung des Benutzers [SDA99]. Dadurch wird nach Salber et al. der Zugriff auf Sensordaten wesentlichen vereinfacht, da die Messung und Auswertung von Sensordaten - gekapselt in Context Widgets – gegenüber der Anwendung verborgen bleiben. Abbildung 3.4 zeigt die Analogie zwischen GUI Widgets und Context Widgets. Die dargestellten Pfeile symbolisieren den Zugriff zwischen den einzelnen Elementen. Dabei können Widgets auf anwendungsspezifische Daten zugreifen, um beispielsweise Callbacks auszuführen.

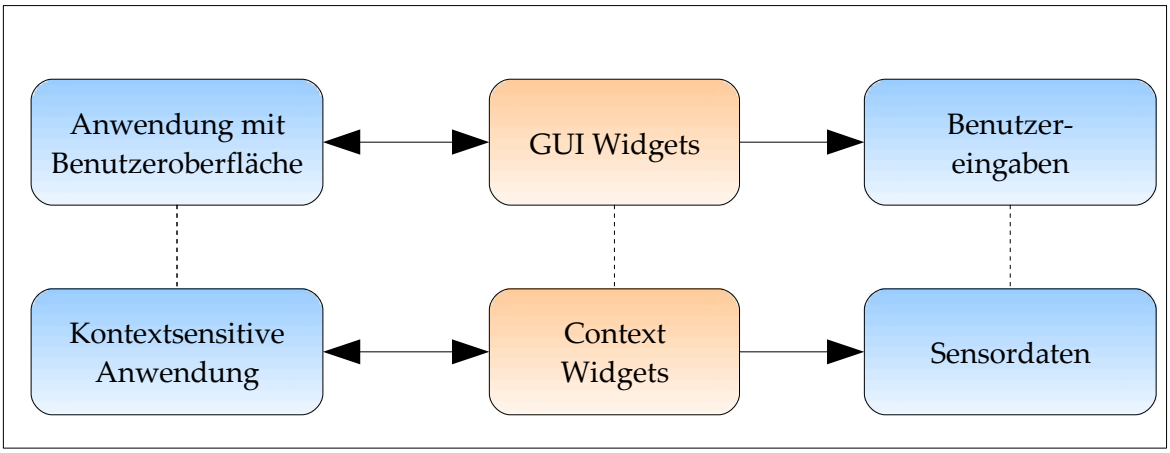


Abbildung 3.4: Beziehung zwischen GUI- und Context-Widgets

Ein anderer Vorteil des Widget-Konzepts ist die Möglichkeit der *Wiederverwendbarkeit* von Komponenten. Ähnlich wie GUI-Widgets-Bibliotheken, wie zum Beispiel Java Swing, in verschiedenen Programmen mit Benutzeroberfläche verwendet werden, eignen sich konstruierte Context Widgets für die Benutzung in verschiedenen kontextbezogenen Anwendungen. Beispielsweise kann ein `GPSLocationWidget`, das Informationen über die Position des Benutzers im Freien liefert, für verschiedene ortsbezogene Anwendungen genutzt werden.
Zusammen mit den Context Widgets bilden *Context Aggregators* und *Context Interpreters* die Hauptkomponenten des Context Toolkits. Ein Aggregator stellt als eine Art Meta-Widget eine beliebige Entität dar. Dies kann eine Person, ein Platz oder ein mobiles Gerät sein, auf die sich eine Menge von verschiedenen Sensorinformationen beziehen [SDA99]. Deshalb aggregieren diese Komponenten eine Menge von Context Widgets (d.h. sie führen mehrere Widgets zu

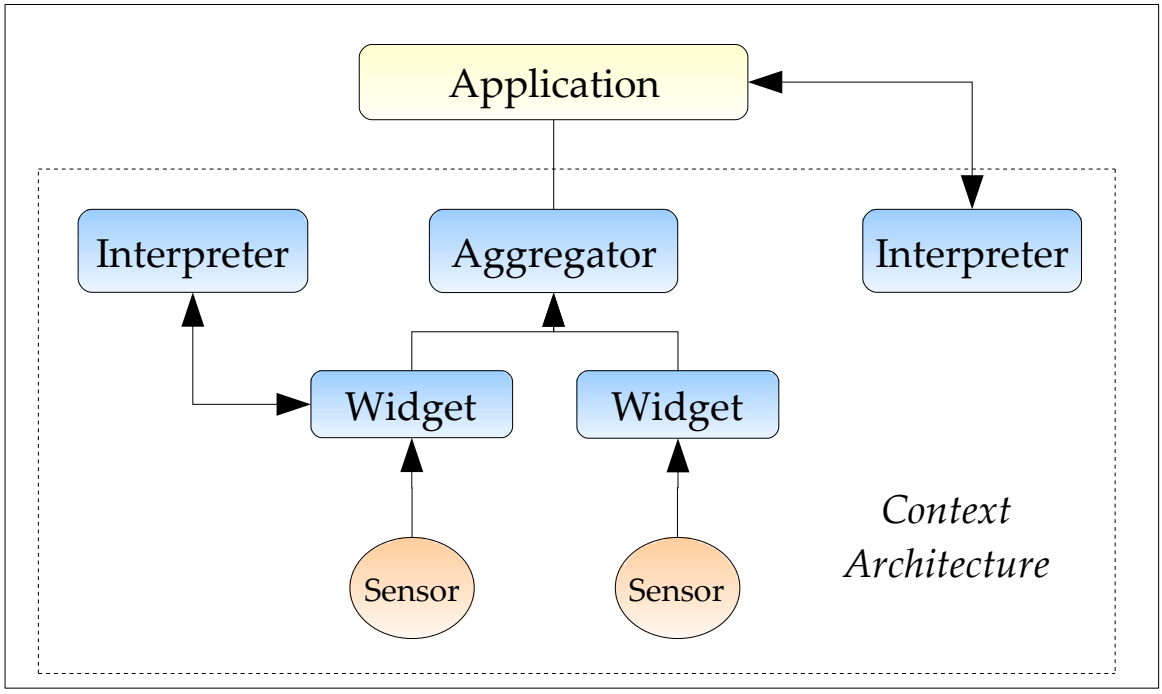


Abbildung 3.5: Architektur des Context Toolkit (nach [SDA99])

einer Entität zusammen) und stellen damit eine Art Proxy für die Anwendung dar, um auf sensorbezogene Kontextinformationen zuzugreifen.
Mit Hilfe des Context Interpreter besteht einerseits die Möglichkeit aus Low-Level- High-Level-Kontextinformationen herzuleiten und zum anderen in Widgets enthaltene Sensorinformationen zwischen ihren verschiedenen Präsentationsformen umzuwandeln. Beispielsweise kann ein Interpreter aus Widget-Daten zur Identität, Position und Lautstärke den Kontext „Meeting“ herleiten [SDA99]. Sensordaten können in unterschiedlichen Präsentationsformen vorliegen, die abhängig vom Anwendungsfall preferiert werden. In diesem Fall besteht die Aufgabe des Interpreters darin, eine Umwandlung zwischen den bevorzugten Darstellungsformen vorzunehmen (zum Beispiel Umrechnung der Temperatur von Fahrenheit in Grad Celsius). Die Beziehungen zwischen den einzelnen Komponenten des Context Toolkit verdeutlicht Abbildung 3.5. Die dargestellten Pfeilen zeigen den Datenfluss zwischen den Architekturkomponenten.
Das Context Toolkit unterstützt zusätzlich die Verwaltung und Kommunikation der oben genannten Komponenten in einer verteilten Umgebung. Widgets, Aggregatoren und Interpreter können mittels einer allgemeinen Kommunikationmöglichkeit (z.B. „XML über HTTP“) Daten austauschen und damit unabhängig von einander in einer verteilen Umgebung existieren. Aus Sicht der Anwendung bleibt die Verteilung der einzelnen Elemente transparent [SDA99].

3.2.2 Hydrogen

Das *Hydrogen* Kontext-Framework [HSPL03] bietet eine kompakte objektorientiere Infrastruktur, die durch eine Drei-Schichten-Architektur gekennzeichnet ist. Es ist ausschließlich für mobile Anwendungen konzipiert. Als Teil der Anwendung wird Hydrogen vollständig innerhalb des mobilen Endgerätes ausgeführt. Aufgrund der existierenden Randbedingungen im Mobile Computing (vgl. Abschnitt 2.1.5) wird einer Architektur für

eine Middleware hinsichtlich ihrer Komplexität und Flexibilität Grenzen gesetzt [HSPL03]. Deshalb stellen die Entwickler dieses Systems Aspekte wie die Verwendung von leichtgewichtige Komponenten (*Lightweightness*), die Robustheit des Systems gegenüber Verbindungsabbrüchen (*Robustness*) und den Austausch von Kontextinformationen in einer verteilten mobilen Umgebung (*Context-Sharing*) in den Vordergrund.
Wie bereits erwähnt, besteht die Architektur aus drei Schichten (siehe Abbildung 3.6). Die *Adaption Layer* besteht aus einer Menge von Verbindungsobjekten (Adaptoren), welche die jeweiligen Sensoren kapseln und Sensordaten an die darüber liegende Schicht liefern. Adaptoren ergänzen Sensordaten mit zusätzliche Informationen, wie beispielsweise Zeitpunkt des Empfangs eines Sensordatums [HSPL03]. Ähnlich wie Context Widgets im Context Toolkit (vgl. Abschnitt 3.2.1) stellen sie wiederverwendbare Komponenten dar, die simultan von mehreren mobilen Anwendungen benutzt werden können.

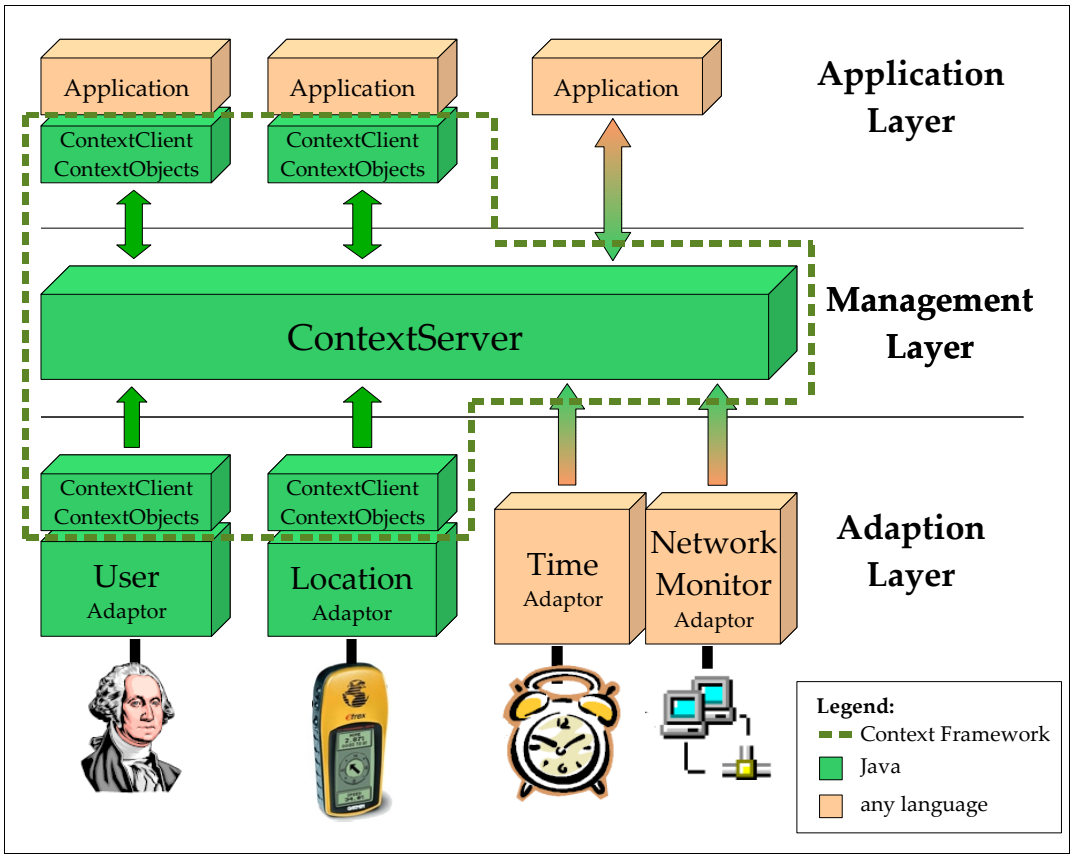


Abbildung 3.6: Architektur des Hydrogen-Frameworks (nach [HSPL03])

Den Kern des Frameworks bildet die *Management Layer*. Sie verwaltet das Objekt `ContextServer`, das alle Informationen über die aktuelle Umgebung des mobilen Gerätes, die aus den Adaptoren stammen, aggregiert [HSPL03]. Auf Anfrage über die spezifizierten Schnittstellen beliefert der `ContextServer` Anwendungen im *Application Layer* mit Kontextinformationen. Zusätzlich dazu kann eine Anwendung durch Zugriff auf die Management Layer sich für Kontextänderungen registrieren lassen. Der implementierte Notification-Mechanismus benachrichtigt in diesem Fall die Anwendung, wenn entsprechende Änderungen vorliegen.
Eine Referenzimplementierung des Hydrogen-Systems existiert für eine J2ME-Plattform (Personal Profil) eines PDAs [HSPL03]. Die Kommunikation zwischen einer Anwendung und

der ausführbaren Java-Klasse `ContextServer` kann am einfachsten über ein vorhandenes `ContextClient`-Objekt erfolgen. Dabei werden die Kontextinformationen über ein XML-basiertes Protokoll ausgetauscht. Eine Nicht-Java-Anwendung muss aber selbst ein Mechanismus implementieren, der für die Kommunikation mit dem `ContextServer` zuständig ist (z.B. XML over TCP/IP) [HSPL03].
Das Hydrogen-Framework bietet auch eine Möglichkeit Kontextinformationen in XML zu serialisieren und mit anderen mobilen Teilnehmern in einem Peer-to-Peer-Netz auszutauschen. So hat eine mobile Anwendung nicht nur Zugriff auf die eigenen *lokalen* Kontextinformationen, sondern auch auf *entfernte* Kontexte anderer mobiler Geräte. Das UML-Klassendiagramm in Abbildung 3.7 zeigt, dass zu einem `Context`-Objekt eine Menge von `ContextObject`-Objekten gehören, die entweder Referenzen auf lokalen Adaptoren darstellen oder Kontextinformationen entfernter Geräte repräsentieren.

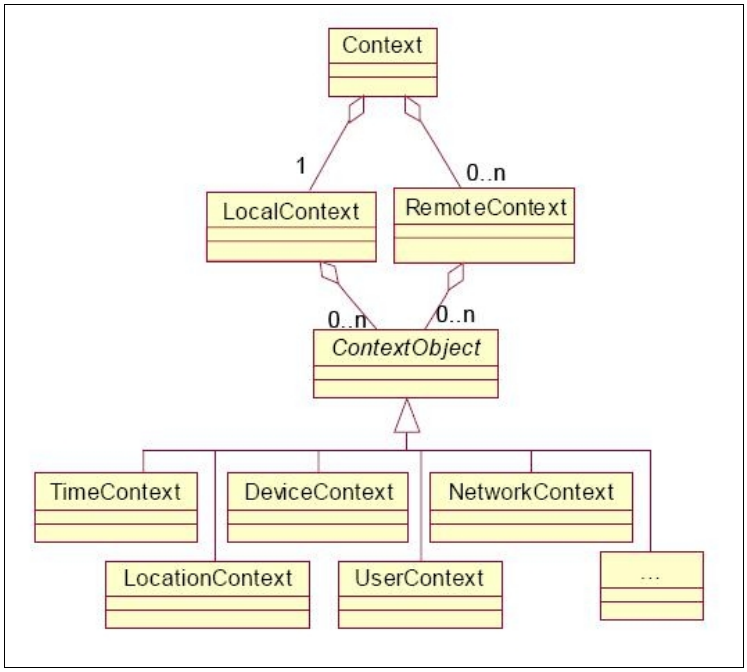


Abbildung 3.7: UML-Klassendiagramm zum Kontext-modell des Hydrogen Framworks [HSPL03]

3.2.3 JCAF – Java Context Awareness Framework

Das *Java Context Awareness Framework* (JCAF) basiert auf der Grundidee durch ein kompaktes, ereignisbasiertes und sicheres System die Entwicklung kontextbezogener Systeme zu unterstützen [Bar05a].
Es besteht aus zwei Teilen. Der erste Teil ist ein Infrastruktursystem (*Context-Awareness Runtime Infrastructure*), das jeweils als ein eigener Prozess auf diversen Knoten eines verteilten Systems ausgeführt wird, um Kontextdienste anzubieten, die miteinander zusammenarbeiten und von Cleints benutzt werden können. Der zweite Teil stellt das *JCAF Application Programmer Interface* (JCAF API) dar. Es bietet eine erweiterbare Programmierschnittstelle für Anwendungsentwickler, um anwendungsspezifische Kontextinformationen zu modellieren, die durch das Infrastruktursystem des JCAF (verteilt) verwaltet und angeboten werden [Bar05a]. Die JCAF API besteht aus einer Menge relativ kompakter Java-Interfaces, die ein

Entwickler einer kontextbezogenen Anwendung seinen speziellen Anforderungen nach implementieren muss.
Die Architektur des Infrastruktursystems ist relativ einfach und kompakt. Context Clients auf der höchsten Ebene bedienen sich der Kontextdienste auf der mittleren Ebene, die die Daten der Sensoren auf der untersten Ebene verarbeiten und von ihnen abstrahieren. Die Kommunikation zwischen Diensten und Clients einerseits und den Diensten untereinander andererseits, erfolgt über Java-RMI [Bar05a]. Kontextdaten sind als serialisierbare Java-Objekte modelliert.
Das UML-Klassendiagramm in Abbildung 3.8 zeigt die wichtigsten definierten Schnittstellen und ihre Beziehungen zu einander. Dabei bietet die `ContextServiceImpl` unter anderem ein Mechanismen für einen `ContextClient`, um auf die Daten von Entitäten und ihre Kontextinformationen zuzugreifen (siehe Abbildung 3.9 für Details zur Modellierung von Entitäten und Kontexte). Der `ContextService` erbt zusätzlich die Methoden der Schnittstellen `EntityListenerHandler`, `ContextClientHandler` und `Transformer-Repository`. Der `TransformerRepository` spezifiziert Methoden zum setzen und zugreifen von Transformatoren (`ContextTransformer`), die für die Umwandlung von Kontextinformationen in äquivalente Formate zuständig sind. Das Interface `Context-ClientHandler` legt Methoden fest, die überprüfen sollen, ob ein `ContextClient` die erforderlichen Zugriffsrechte besitzt, um Entitätsdaten zu lesen oder zu ändern. Die Implementierung eines `EntityListenerHandlers` verwaltet das Hinzufügen und Entfernen von `EntitiyListener`-Objekten, die Zustandsänderungen bei Entitäten beobachten. Pro `ContextService` existiert genau ein `EntityEnvironment`-Objekt, das Informationen zur aktuelle Ausführungsumgebung (zum Beispiel eine Umgebungsvariable des Systems, Version des Java-VM, benutzte RMI-Ports, usw.) des `ContextService` liefert [Bar05b].

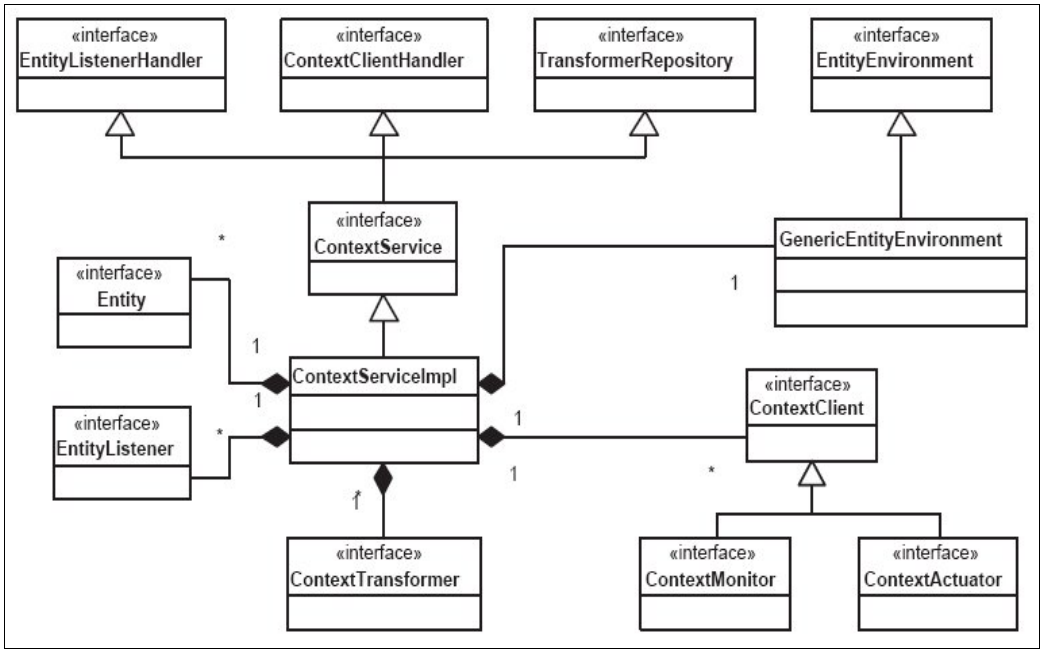


Abbildung 3.8: Management-Komponenten des JCAF als UML-Klassendiagramm [Bar05a]

Das essentielle Grundkonzept zur Kontextmodellierung zeigt das UML-Klassendiagramm im Abbildung 3.9. Hier stehen vor allem die Interfaces `Entity`, `Context` und `ContextItem` im

Mittelpunkt, die von einem Entwickler einer kontextbezogenen Anwendung implementiert werden muss [Bar05a].
Entitäten stellen konkrete Objekte der realen Welt dar. Das können Personen, Orte oder Dinge sein, die mit Hilfe einer `Entity`-Schnittstelle modelliert werden. Der aktuelle Kontext einer Entität wird durch ein `Context`-Interface angegeben. Implementierte Beispiele für einen `Context` kann ein `HospitalContext` oder ein `OfficeContext` sein, welcher durch die Aggregation von einem oder mehrerer `ContextItems` charakterisiert ist. `ContextItems` enthalten elementare Kontextinformationen wie beispielsweise die physikalische Lokalität eines Benutzers oder Aktivität von Personen [Bar05a]. Zusätzlich können `ContextItems` Informationen über die Qualität der ermittelten Kontextinformationen (Beispielsweise Genauigkeit von Ortsangaben) angeben.
Das JCAF wurde ursprünglich zum Experimentieren mit Kontextinformationen und kontextbezogenen Anwendungen entwickelt. Dennoch stellt es für die Integration von Context-Awareness im Mobile Computing einen interessanten Ansatz dar, da es sich mit Hilfe objektorientierter Paradigmen systematisch mit einer schlanken, aber dennoch effizienten, Modellierung von Kontextinformationen auseinander setzt. Zusätzlich dazu berücksichtigt das JCAF den Verteilungsaspekt kontextsensitiver Daten und Erweiterungsmöglichkeit um nützliche Werkzeuge, wie beispielsweise den `ContextTransformer`.
Trotzdem stellt JCAF durch seine entwickelten Funktionsbibliotheken ein Framework dar, das bei der Entwicklung eines Kontext-Management-System helfen soll. Es selbst stellt aber kein Managementsystem für Kontextinformationen dar.

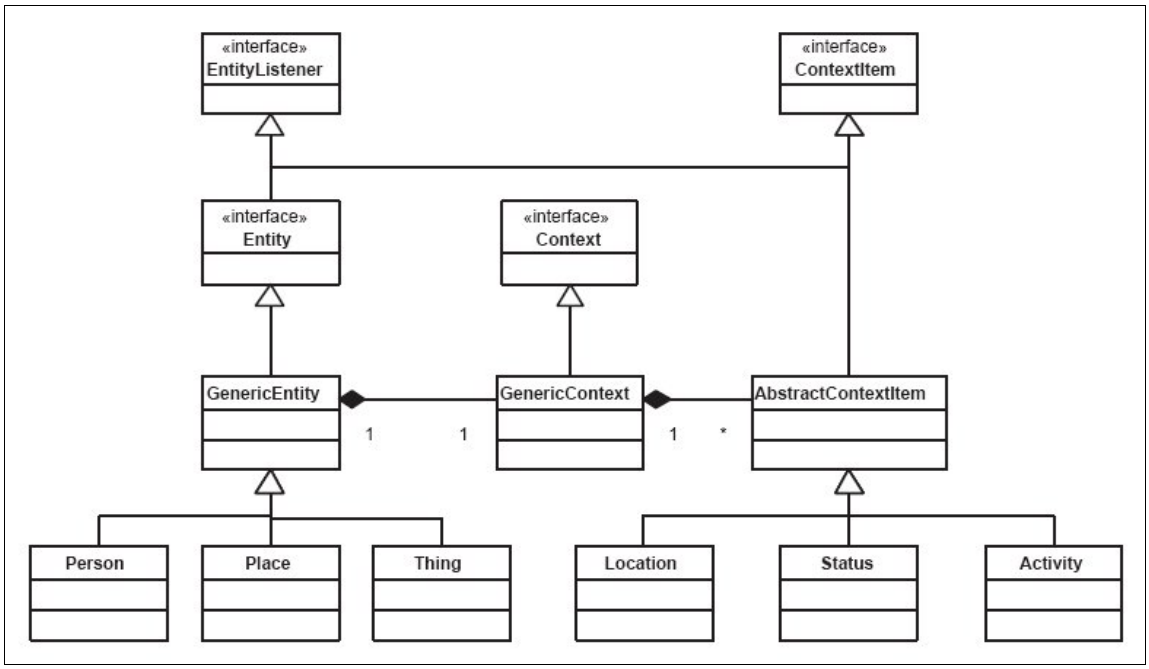


Abbildung 3.9: Kontextmodell des JCAF als UML-Klassendigramm [Bar05a]

4 Konzept für die Integration von Context-Awareness in eine Middleware für mobile Systeme

Wie bereits im vorherigen Kapitel erläutert, existieren erste Ansätze zur Konstruktion von Frameworks, die als Unterstützung bei der Entwicklung von mobilen kontextbezogenen Anwendungen eingesetzt werden können. Diese Ansätze implizieren auch sinnvolle Anforderungen an eine kontextsensitive Infrastruktur, die in diesem Kapitel durch weitere eigene Anforderungen ergänzt werden. Aus den ergänzenden Anforderungen entsteht die Notwendigkeit zu einer eigenen Konzeption einer Middleware für mobile kontextbezogene Systeme, die sämtliche festgelegte Anforderungen erfüllt. Dabei stehen vor allem solche Anforderungen im Vordergrund, die den effizienten Einsatz der Middleware speziell in einem mobilen System garantiert.
Nach der Festlegung der Anforderung werden bisherige Ansätze anhand dieser bewertet. Damit sollen dann Unterschiede zwischen den vorhandenen Ansätzen und dem eigenen Konzept hergeleitet und erfasst werden. Der anschließende Abschnitt dient der Beschreibung der Komponenten der Middleware-Plattform und ihrer Funktionalitäten.

4.1 Ermittlung von Anforderungen an kontextsensitive Middleware-Plattformen

Im Folgenden werden Anforderungen definiert und begründet, die bei der Konzeption einer kontextbezogenen Middleware für mobile Systeme berücksichtigt werden sollten.

(A1) **Bereitstellung von Kontextinformationen**: Die Middleware sollte durch Spezifikation allgemeiner Schnittstellen einen *Zugriffsmechanismus* bereitstellen, die beliebigen mobilen Anwendungen die Möglichkeit gibt, auf lokale und entfernte Kontextinformationen zuzugreifen. Ein Kontextdienst (*Context Service*) sichert den Zugriff der Anwendung auf die Kontextdaten.

(A2) **Trennung zwischen Kontextmodell und Kontext-Management**: Der Context Service sollte die Möglichkeit zur Nutzung eines *Kontextmodells* und eines *Kontext-Management-Systems* bieten. Mit Hilfe des Modells kann die Beziehungen zwischen den elementaren Kontextinformationen anwendungsspezifisch festlegt werden. Insbesondere bestimmt das Modell welche Kontextinformationen (beispielsweise die Lokalität) welcher Entität (zum Beispiel dem Benutzer) zugeordnet werden. Das Kontextmodell ist *konzeptionell getrennt* von Elementen der Middleware, die die Verwaltung des Kontextmodells übernehmen – dem Kontext-Management. Die Verwaltung des Kontextmodells bedeutet in diesem Zusammenhang beispielsweise das Aktualisieren des Kontextmodells mit sensorbezogenen Daten oder den Austausch von Kontextinformationen zwischen mobilen Teilnehmern in einer verteilten mobilen Umgebung.

(A3) **Abbildbarkeit und Abstraktionsgrad**: Das Kontextmodell sollte in der Lage sein, verschiedene Klassen von Kontextinformationen (vgl. dazu Abschnitt 2.4.3 zur Kategorisierung von Kontextinformationen) wie *Umgebungsinformationen*, *Aktivität*, *Identität* und *Zeit* abzubilden. Um dies zu erreichen, sollte das Modell von konkreten Kontextinformationen unterschiedlicher Struktur (wie zum Beispiel Identität des Benutzers, Ortszeit, Temperatur oder Geschwindigkeit) und deren Darstellung *abstrahieren*. Es sollte eher die *gemeinsamen Eigenschaften* solcher Informationen in den Vordergrund stellen und die Details vorerst im Hintergrund belassen.

(A4) **Robustheit und Kompaktheit**: Die Architektur der Middleware sollte durch Robustheit und die Verwendung von *leichtgewichtigen Komponenten* charakterisiert sein. Vor allem ein kompaktes und einfaches Kontextmodell legt dabei die Grundlage dafür, dass die Middleware hinsichtlich der vorhandenen Ressourcenknappheit in mobilen Geräten (vgl. Abschnitt 2.1.5) einsetzbar ist. Unnötige Komplexität beim Entwurf der Architektur sollte unbedingt vermieden werden.

(A5) **Erweiterbarkeit und Wiederverwendbarkeit**: Das Kontextmodell und die Kontext-Management-Komponenten sollten nicht von einer festen Menge bestimmter Kontextinformationstypen abhängig sein. Das Modell sollte um neuen Kontextinformationen, die beispielsweise aus bisher nicht-entwickelten Sensoren bezogen werden, *erweitert* werden können. Auch das Managementsystem der Middleware sollte mit dem erweiterten Kontextmodell umgehen können. Bereits konstruierte Komponenten (zum Beispiel zur Angabe der Geschwindigkeit eines Autos) sollten in einem anderen Zusammenhang (zum Beispiel für die Geschwindigkeit des Benutzers) *wiederverwendet* werden können.

(A6) **Flexibilität und Skarlierbarkeit**: Aufgrund der existierenden Randbedingungen im Mobile Computing ist es sinnvoll, Komponenten der Middleware deaktivieren zu können, wenn diese von der betreffenden Anwendung nicht benutzt werden. Die Middleware sollte daher die Flexibilität besitzen, die auszuführende Anwendung bestimmen zu lassen, welche *Komponenten des Systems zur Nutzung aktiviert* werden sollen. Dies betrifft insbesondere die Komponenten des Kontext-Managements. Verwendet beispielsweise eine Anwendung lediglich den lokalen Kontext des Benutzers, kann die Management-Komponente zum Austausch von Kontextinformationen deaktiviert werden.

(A7) **„Quality of Context" (QoC)**: Insbesondere aus Sensoren ermittelte Kontextinformationen können *ungenau*, *unvollständig* oder *widersprüchlich* sein [BKW02]. Das REAL-Projekt [BKW02] zeigt, dass auch die Qualität der Kontextinformationen zum Kontext gehört und von einer Anwendung sinnvoll verwendet werden kann. Daher sollte die Middleware Angaben zur Qualität der Kontextinformationen berücksichtigen können und entsprechende Schnittstellen für die Anwendung bieten, um den QoC-Parameter zu spezifizieren und abzufragen.

(A8) **High-Level-Kontext und statischer Kontext**: Die Middleware sollte nicht auf die Modellierung und Verwaltung von Kontextinformationen beschränkt sein, die aus Sensoren stammen. Sie sollte *semantische Unterschiede* zwischen Low-Level-Kontext, High-Level-Kontext (vgl. Abschnitt 2.4.2), und einem statischen Kontext machen. Statische Kontextinformationen sind vor allem solche, die durch Eingaben des Benutzers entstehen (z.B. Name oder Geburtstag) oder Informationen darstellen, die sich während der Laufzeit selten bzw. nicht ändern. Eine semantische Unterscheidung zwischen den einzelnen Kontexttypen kann bei der Verwaltung der Kontextinformationen hilfreich sein.

(A9) **Verteilungsaspekt von Kontextinformationen**: In einem mobilen verteilten System kann nicht nur der eigene Kontext relevant sein, sondern auch der Kontext anderer mobiler Teilnehmer. Die Middleware sollte daher ein *Protokoll zum Austausch* von *Kontextinformationen* definieren und Anwendungen einfache Zugriffsmechanismen bieten, um auf den Kontext erreichbarer Geräte in der Umgebung zuzugreifen.

(A10) **Erweiterter Notification-Mechanismus**: Oft rufen Anwendungen nicht nur Informationen zum aktuellen Kontext ab, sondern benötigen einen Mechanismus, um ad-hoc auf *Kontextänderungen* reagieren zu können. Die Middleware sollte daher das Kontextmodell auf Änderungen hin beobachten und die Anwendungen benachrichtigen, wenn von ihnen spezifizierte Änderungen im Kontext eintreten. Dabei ist es besonders wichtig die Anwendung nicht nur bei Änderungen einzelner, voneinander unabhängiger Kontextinformationen zu informieren, sondern auch *Änderungen des Gesamtkontext* zu registrieren. Ein Beispiel dafür wäre die Benachrichtigung der Anwendung, sobald bestimmte Personen sich im selben Raum befinden und der lokale Benutzer sich in einem anderen Raum aufhält.

(A11) **Transformation elementarer Kontextinformationen**: Kontextinformationen können *verschiedene Präsentationsformen* haben (beispielsweise Geschwindigkeit in mph oder km/h). Daher sollte die Middleware die Transformation der Daten in äquivalente Darstellungsformen unterstützen, ohne die Semantik des Kontext-modells zu ändern.

(A12) **Sicherheit und Privatsphäre**: Viele Kontextinformationen enthalten persönliche Daten eines Benutzers, die in einer verteilten mobilen Umgebung zwischen Teilnehmern ausgetauscht werden. Eine Anwendung sollte mit Hilfe der Middleware konfigurieren können, welche Kontextinformationen welchen Benutzern zugänglich gemacht werden sollten. Vor dem Hintergrund, dass mobile Geräte, meistens personalisierte Geräte sind, müssen aus *Sicherheitsgründen* und zum Schutz der *Privatsphäre* bestimmte Kontextinformationen vor unbefugten Zugriff geschützt werden.

(A13) **Filtermechanismus**: Ein Middleware muss unter Umständen sehr viele Kontextinformation modellieren und verwalten. Doch nur wenige Informationen werden von einer Anwendung in einer bestimmten Situation genutzt. Um diese für die Verarbeitung zu *isolieren*, sollte ein Filtermechanismus integriert werden. Dies kann zum Beispiel mit Hilfe des Primärkontexts als Filterkriterium erreicht werden (vgl. Abschnitt 2.4.1).

(A14) **Integrationsmöglichkeit in eine vorhandene mobile Middleware**: Das *Konzept* des Kontextsystems sollte so generisch wie möglich entworfen sein. Das bedeutet, möglichst unabhängig von einer bestimmten Programmiersprache oder einem bestimmten Kommunikationprotokoll, mit dem Kontextinformationen ausgetauscht werden. Nur so lässt sich das Konzept der Context-Awareness in eine Middleware für mobile Systeme übernehmen.

4.2 Bewertung bisheriger Middleware

Nach dem die Anforderungen an eine kontextsensitive Middleware für mobile Systeme spezifiziert sind, wird in diesem Abschnitt geprüft, inwieweit die im Kapitel 3 beschriebenen Ansätze, den Anforderungen genügen. Damit werden nicht nur die vorhanden Middleware-Frameworks bewertet, sondern auch geprüft, ob ein eigenes Konzept für eine neue kontextbasierte Middleware entwickelt werden sollte, die die gestellten Anforderungen erfüllen.

Im Folgenden werden die Context-Aware-Frameworks *Context Toolkit* [SDA99], *Hydrogen* [HSPL03] und *JCAF* [Bar05a] mit den aufgestellten Anforderungen aus dem vorherigen Abschnitt bewertet.

Die Tabelle 4.1 zeigt, dass das JCAF bei der Bewertung am besten abschneidet. Dennoch verfolgt das JCAF genau genommen nicht die strenge Trennung zwischen Kontextmodellierung und der Verwaltung des Modell mit einem Kontext-Management-System. Dies beweist die konzeptuelle Entscheidung, den Notification-Mechanismus (siehe A10) als Teil des Kontextmodell zu spezifizieren. Die konsequente Trennung zwischen dem Modell und dem Management einerseits (A2) und der Unabhängigkeit der Modellkomponenten von den Management-Komponenten andererseits bilden die Grundlage für die Skalierbarkeit (A6) und Erweiterbarkeit (A5) des Systems. Gerade die Skalierbarkeit ist hinsichtlich der vorhandenen Randbedingungen im Mobile Computing besonders wichtig. Auch die Erweiterbarkeit der kontextbezogenen Middleware muss aufgrund der Heterogenität der Kontextinformationen extra berücksichtigt werden.

Die Anforderung zum erweiterten Notification-Mechanismus (A10) wird von keinem Framework erfüllt, da sie alle nur einen einfachen Notification-Mechanismus vorsehen, bei dem einzelne Attribute oder Entitäten beobachtet werden, aber Änderungen im Gesamtkontext nicht berücksichtigt werden. Auch ein Filtermechanismus (A13) zur Isolierung und Selektion von relevanten Kontextinformationen ist in keiner Middleware vorhanden.

Die Übernahme der vorhandenen Konzepte zur Integration der Context-Awareness in eine andere mobile Middleware (A14) scheint nur beim Context Toolkit und bei Hydrogen ohne großen Aufwand möglich zu sein.

Tabelle 4.1: Bewertung kontextsensitiver Middleware

	Context Toolkit	**Hydrogen**	**JCAF**
Bereitstellung v. Kontextinformationen	+	++	+
Trennung zwischen Modell u. Management	--	-	+
Abbildbarkeit u. Abstraktionsgrad	+	-	+
Robustheit u. Kompaktheit	+	++	+
Erweiterbarkeit u. Wiederverwendbarkeit	++	+	-
Flexibilität u. Skarlierbarkeit	--	-	-
QoC-Parameter	-	-	++
High-Level-Kontext u. statischer Kontext	+	-	--
Verteilungsaspekt	-	+	+
Erweitertes Notification	-	-	-
Transformation	+	--	+
Sicherheit u. Privatsphäre	--	-	+
Filtermechanismus	-	-	-
Integrationsmöglichkeit	+	+	-

Legende:

++	Die Anforderung wird erfüllt.
+	Die Anforderung wird zum Teil erfüllt.
-	Die Anforderung wird kaum erfüllt.
--	Die Anforderung wird nicht erfüllt.

Die Schnittstellen des Management-Layers sind im Hydrogen-Framework so entworfen, dass auch Anwendungen ohne die Benutzung des `ContextClients` und des `ContextObjects` mit dem `ContextServer` direkt kommunizieren können. Allerdings ist die Kommunikation zwischen den einzelnen Schichten nur über ein TCP/IP-basiertes Protokoll („XML over TCP/IP“) möglich. Ähnliches gilt auch für den Context Toolkit.
Das entwickelte Widget-Konzept ist zwar ein eleganter Weg von Low-Level-Kontextdaten zu abstrahieren und wiederverwendbare Komponenten anzubieten, dennoch liefert sie keine Lösung dafür, wie mobile Geräte untereinander in einer verteilten und mobilen Umgebung Kontextinformationen austauschen sollen. Lediglich einzelne Komponenten können unabhängig voneinander in einer verteilten Umgebung existieren. Dabei ist die Kommunikation zwischen den Komponenten nur über das HTTP-Protokoll möglich („XML über HTTP“).

Das JCAF definiert nur allgemeine Java-Schnittstellen, um bei der Entwicklung eines eigenen Kontextmanagaments zu unterstützen. Trotzdem ist das Konzept des JCAF zu sehr auf die Programmiersprache Java zugeschnitten. Zudem existiert kein allgemeines Protokoll zum Austausch von Kontextinformationen, das unabhängig von Java-RMI ist.

4.3 Architektur für eine kontextsensitive mobile Middleware

Die Komponenten des Kontextmodells und des Kontext-Managements bilden den Kern des Konzepts für eine kontextsensitive Middleware. Die Beschreibung ihrer Semantik und ihre Einordnung in eine *Architektur* zur Integration der Context-Awareness in eine Middleware für mobile Systeme ist Gegenstand dieses Abschnitts.

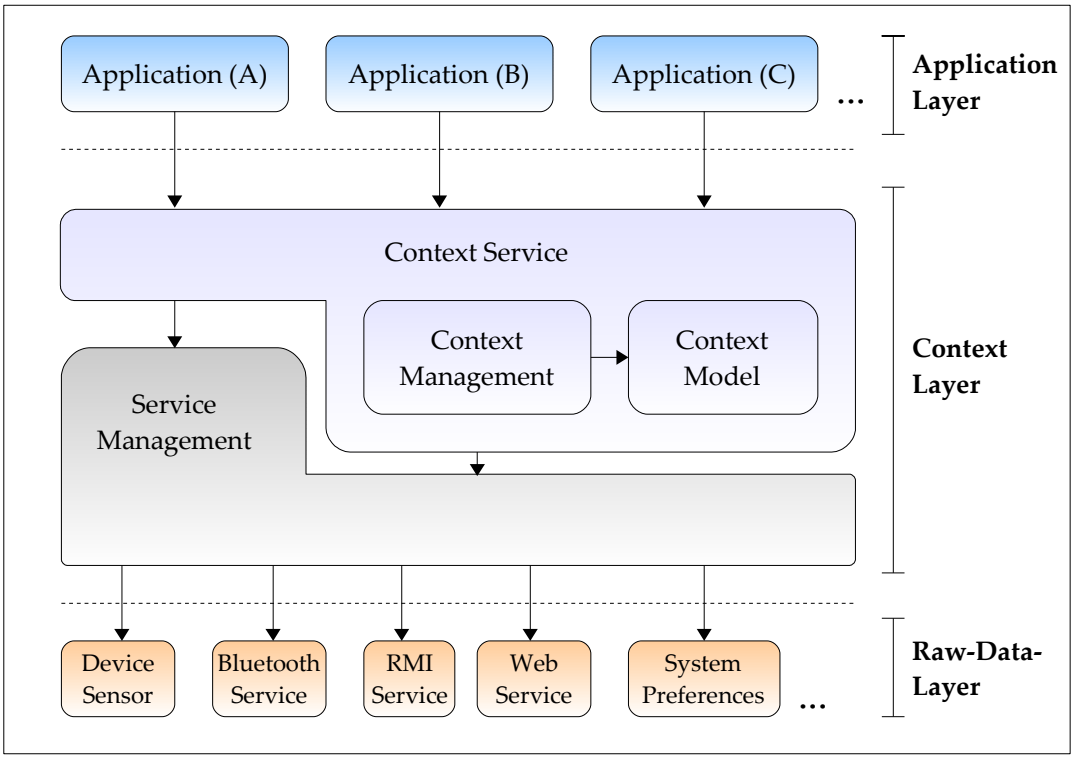


Abbildung 4.1: Architektur zur Integration von Context-Awareness in eine Middleware für mobile Systeme

Abbildung 4.1 zeigt das Grundkonzept einer Drei-Schichten-Architektur. Die *Raw-Data-Layer* als unterste Schicht liegt außerhalb der Middleware. Sie stellt eine Ansammlung *heterogener Dienste* und Komponenten dar, aus denen Basiskontextinformationen in einfacher und unstrukturierter Form bezogen werden. Dazu zählen vor allem geräte- und betriebssystemspezifischer APIs, die die Schnittstelle zu den physikalischen Sensoren des mobilen Gerätes bilden. Auch Web und Bluetooth Services können als Dienst genutzt werden, um Informationen über die Umgebung des Gerätes oder des Benutzers zu erhalten. Ein einfaches Beispiel für die sinnvolle Verwendung von Services wäre die Anbindung einer externen GPS-Maus, die über einen Bluetooth Service Ortsinformationen liefert.
Wie die Komponenten des Raw-Data-Layer zeigen, kann der Kontext mit Hilfe von Daten aus verschiedenen Informationsquellen angegeben werden. Die *Context-Layer* als Teil der Middleware muss zunächst von den unterschiedlichen Dienstarten abstrahieren, damit die vorhandene Heterogenität im Raw-Data-Layer überwunden wird. Aufgabe des *Service*

Managements ist es vor allem, eine abstrakte Dienstbeschreibung zu spezifizieren. Die Beschreibung und Verwaltung homogener Dienste in einer verteilten mobilen Umgebung ermöglicht die Anwendungen auf dem *Application-Layer* eine *transparente* Nutzung von Diensten. Auch die Komponenten der Middleware (wie z.B. der Context Service) können so auf einheitlicher Weise verschiedene Typen von Dienste nutzen.
Im weiteren Verlauf dieser Arbeit wird das Vorhandensein einer Service-Management-Komponente mit Möglichkeiten zur abstrakten Dienstbeschreibung und Dienstverwaltung vorausgesetzt. Hier stehen mehr die Konzepte für das Kontextmodell und das Kontext-Management im Vordergrund.
Das *Context Management* kann durch die Verwendung homogener Dienste des Service Managements das *Context Model* verwalten. Dies betrifft vor allem das Aktualisieren des Kontextmodells mit Rohdaten.
Der *Context Service* kapselt als eine Art *Proxy* alle Zugriffe auf die *Context-Layer*. Eine Anwendung kann beispielsweise über den Context Service Kontextmodelldaten abrufen, Komponenten des Context Management aktivieren und selbst auf Dienste des Service Managements zugreifen.

4.3.1 Semantik der Modellkomponenten

Wie die Raw-Data-Layer der Architektur im vorherigen Abschnitt verdeutlicht, sind Kontextquellen hinsichtlich ihrer Art und Qualität zahlreich und heterogen. Dennoch sollte die Middleware einem Anwendungsentwickler die *Auswahl* und *Komposition* von Kontextinformationen intuitiv und aufwandsarm ermöglichen. Diese Anforderung kann durch ein Kontextmodell erfüllt werden, das einheitliche Schnittstellen und Formate für die Anbindung von unterschiedlichen Kontextquellen anbietet. Die Funktion des Kontextmodell besteht unter anderem darin, einzelnen Kontextinformationen (wie beispielsweise Position, Geschwindigkeit oder Temperatur), die ohne ein Modell direkt aus der entsprechenden Kontextquelle bezogen werden müssen, durch die *Zuordnung zu Entitäten* (wie Personen oder Plätze) eine erweitere Semantik zu geben.
In diesem Abschnitt geht es um die Beschreibung eines geeigneten Kontextmodells, das elementare Kontextinformationen, welche die Service-Management-Komponente liefert, einheitlich integriert und diese einer mobilen Anwendung zur Auswahl und Komposition zur Verfügung stellt. Das UML-Klassendiagramm in Abbildung 4.2 zeigt die einzelnen Modellkomponenten und ihre Beziehungen zu einander. Im Folgenden werden die wichtigsten Elementen des Modells erläutert:

Entitäten: Eine Entität repräsentiert ein einzelnes Objekt aus der physikalischen oder virtuellen Welt, die sich in verschiedenen Situationen befinden kann. Der Kontext dieser Entität kann für eine Anwendung relevant sein. Da der Entitätsbegriff sehr allgemein gefasst ist, sind verschiedene Ausprägungsformen in einer kontextsensitiven Middleware vorstellbar. Beispiele für Objekte, die als Entitäten modelliert werden können, sind:

- Ein *Benutzer* mit dem Kontext Lokalität und Aktivität.
- Das mobile *Gerät*, bei dem die Eigenschaften der Benutzerschnittstelle zum statischen Kontext gehören.
- Ein *Raum* in einem Gebäude, in dem der Kontext Geräuschlautstärke, Lichtverhältnisse und Temperatur wichtige Informationen liefern.

- Eine *Kommunikationsverbindung*, die Kosten und QoS-Parameter als Kontexte hat.
- Ein gesellschaftliches Ereignis wie beispielsweise ein *Meeting*, dass den Vortragenden, das Thema, den Ort, die Zeit und die gerade angezeigte Folie zum Kontext hat.

Eine Entität kann auch andere Entitäten rekursiv kumulieren: Der letzte Anwendungsfall impliziert, dass die Meeting-Entität den Vortragende als eine Entität enthält.

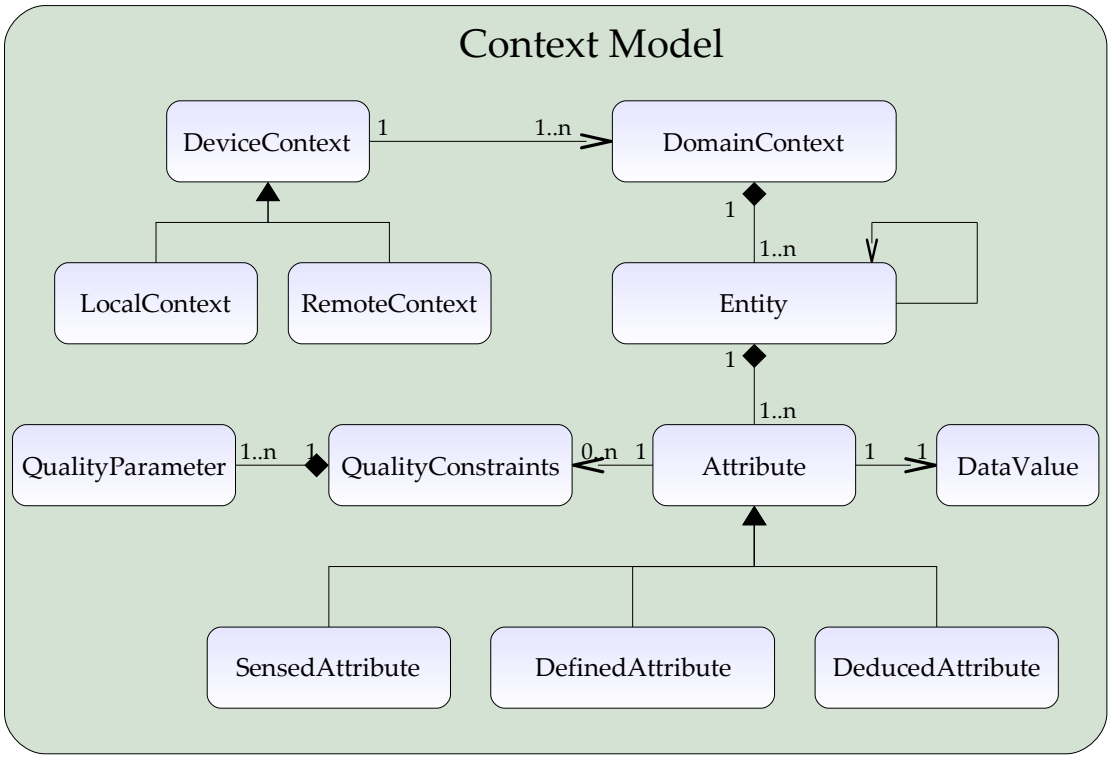


Abbildung 4.2: Ein generisches und erweiterbares Kontextmodell

Attribute: Ein Attribut einer Entität stellt ein Merkmal einer Entität dar, die den Kontext betrifft. Das Attribut kapselt also als Einheit jeweils eine bestimmte Kontextinformation. Die Summe aller Attribute, die einer Entität zugeordnet werden, geben den Kontext der Entität an.
Da die Datenquellen aus denen Kontextinformationen gewonnen werden, heterogen sind und unterschiedliche Charakteristiken haben, wird zwischen drei Typen von Attributen unterschieden:

- **SensedAttribute**: Charakteristisch für diesen Typ ist, dass der Kontextwert sich häufig ändert. Typischerweise stammen Informationen eines SensedAttribute aus logischen oder physikalischen Sensoren. Dennoch sind SensedAttributes nicht auf die Modellierung von Sensordaten beschränkt, sondern setzen in ihrer Semantik den Schwerpunkt auf solche Kontextinformationen, die eine relativ hohe Änderungsfrequenz während der Laufzeit der Anwendung haben. Daher steht hier die Dynamik des Kontextwertes im Vordergrund.

- **DefinedAttribute**: Es existieren auch Kontextinformationen, die im Gegensatz zu den meisten sensorbezogenen Informationen, während der Laufzeit selten oder gar nicht geändert werden. Für diesen Typ von Informationen legt der Anwendungsentwickler den erforderlichen Wert fest und ordnet sie einer Entität zu. Die Gruppe von Attributen, die diese Merkmale besitzen, werden als DefinedAttributes zusammengefasst. Dazu zählen zum Beispiel Art und Modell des mobilen Gerätes, die Identität des Benutzers, Benutzer- und Systemeinstellungen oder die Speicherkapazität eines Endgerätes.
- **DeducedAttribute**: Aus der Kombination und Verarbeitung von mehreren Low-Level-Kontextinformationen können komplexere Kontextinformationen gewonnen werden, die als High-Level-Kontext bezeichnet werden (vgl. Abschnitt 2.4.2). Beispielsweise kann aus dem aktuellen Aufenthaltsort des Benutzers, der Uhrzeit und dem Kalendareintrag geschlossen werden, dass der Benutzer eine Besprechung mit seinen Kollegen führt. Attribute, die Kontextinformationen verschiedener Entitäten benutzen, um einen höherwertigen Kontext (High-Level-Kontext) zu gewinnen, werden mit DeducedAttributes modelliert. Für DeducedAttributes sollte sichergestellt werden, dass entsprechende Zugriffsmechanismen für die benötigten Low-Level-Kontexte vorhanden sind.

Die Komponente *DataValue* gibt den *Wert* der Kontextinformation an. Ein DataValue hat je nach Zuordnung zu einem bestimmten Attribut unterschiedliche Ausprägungen. Ein primitiver *BooleanValue* kann beispielsweise für ein *OrientationAttribute* den Wert für die Orientierung des mobilen Gerätes angeben (horizontal oder vertikal), während ein komplexeres *GPSLocationValue* für ein *LocationAttribute* die Positionsinformationen kapselt (Längengrad, Breitengrad, Höhe über dem Meeresspiegel, usw.).
Wie bereits erwähnt sind Information zur Situation eines Objektes, welche in technischen System ermittelt und verarbeitet werden, oft unvollständig, ungenau oder inkonsistent (vgl. Abschnitt 4.1, A7). Das REAL-Projekt (vgl. 2.3.1) [BKW02] und Buchholz et al. [BKS03] zeigen zudem in welcher Hinsicht die Qualität der vorhandenen Kontextinformation für eine Anwendung von Bedeutung sein kann. Daher ist es sinnvoll Qualitätsangaben auch in der Semantik des Kontextmodell zu berücksichtigen. Die Komponente *QualityConstraints* macht für ein Attribut Qualitätsangaben. Diese bestehen aus einer Menge von Qualitätsparametern (*QualityParameter*). Typische Parameter sind *Standardabweichungen* (z.B. bei GPS-Positionsermittlung ca. 4 Meter), die Zeitpunkt für die Ermittlung der Kontextinformation (*Freshness* und *Up-to-Dateness* der Informationen [BKS03]) und *Genauigkeit.* Buchholz et al. diskutieren in [BKS03] weitere wichtige Qualitätsparameter bezüglich Kontextinformationen.

Domäne: Die Komponente *DomainContext* modelliert ein abgrenzbares Anwendungumfeld, in dem eine endliche Menge von Entitäten relevant sind. Der DomainContext beschreibt damit den Kontext eines in sich abgeschlossenen Bereichs – einer Domäne.
Es wird davon ausgegangen, dass definierte Entitäten nur in einer bestimmten Domäne auftreten können. Beispielsweise sind Räumlichkeiten eines Krankenhauses, die als Entitäten modelliert werden, der Domäne *HospitalDomain* zuzuordnen, während die Entität Schlafzimmer der Domäne *HomeDomain* angehört. Die Gruppierung von Entitäten durch die Spezifikation von Domänen dient nicht nur der einfachen Kategorisierung, sondern bringt vor allem den Vorteil, aus der großen Menge von Daten, die einen Kontext beschreiben, nur solche zu *selektieren*, die gerade von einer Anwendung genutzt werden. Domänen, als Teil

eines *Filtermechanismus*, erleichtern nicht nur das Management von einer großen Menge an Kontextinformationen, sondern dienen auch dazu die Anzahl der Kontextinformationen, die ein mobiles Gerät zu ermitteln und zu verarbeiten hat, möglichst klein zu halten. Das bedeutet, dass die Informationsmenge auf ein bestimmtes Anwendungsumfeld zugeschnitten wird.

LocalContext und RemoteContext: Die Komponente *DeviceContext* aggregiert alle Kontext-informationen eines einzelnen mobilen Teilnehmers. Insbesondere ermöglicht sie den Zugriff auf die aktuelle Domäne, die die dazugehörige Entitäten und ihre Kontextinformationen umfasst.
Es wird zwischen zwei Arten von DeviceContext unterschieden. Der *LocalContext* enthält alle Kontextinformationen des *lokalen* Teilnehmers. Der *RemoteContext* bildet den lokalen Kontext eines *entfernten* mobilen Teilnehmers ab. Der *Gesamtkontext* eines einzelnen Teilnehmers setzt sich aus dem eigenen lokalen Kontext und der Menge der lokalen Kontexte entfernter (und erreichbarer) mobiler Teilnehmer zusammen.

4.3.2 Semantik der Management-Komponenten

Alle Funktionen der Middleware, die der Verwaltung des Kontextmodells betreffen, werden (getrennt vom Kontextmodell) durch Services abgebildet. Sie stellen als autonome Module innerhalb der Context-Layer *Werkzeuge* dar, die dem Anwendungsentwickler angeboten werden, um das Kontextmodell entsprechend der Anforderungen der Anwendung zu bearbeiten.

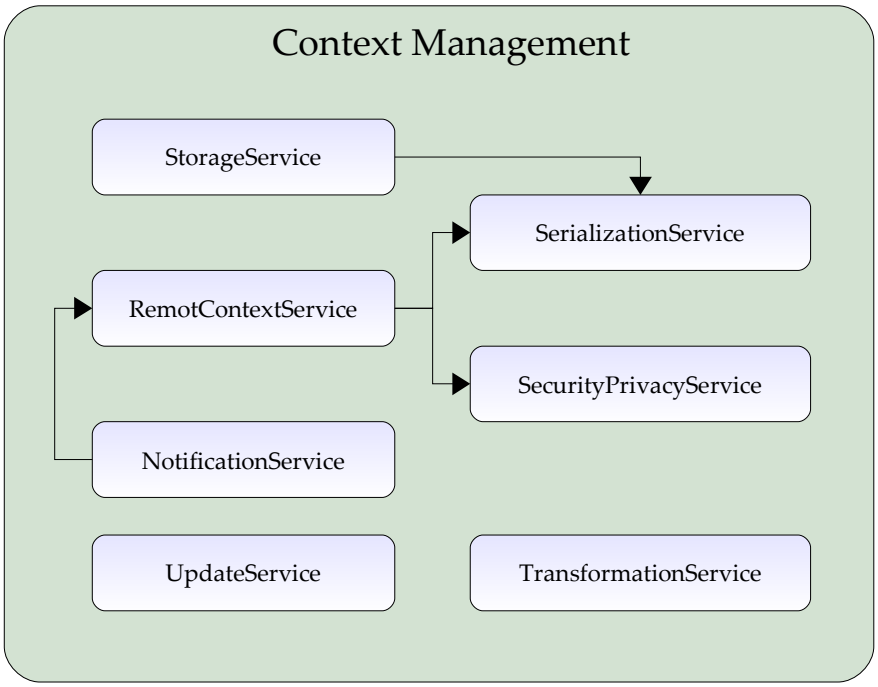


Abbildung 4.3: Komponenten des Kontext-Managements

Die Kontextverwaltungsdienste benötigen technisch einen Zugriff auf das Kontextmodell, um ihren Dienst an dem Modell erbringen zu können. Weiterhin ist durch Spezifikation

adäquater Schnittstellen sichergestellt, dass die Komponenten des Kontext-Managements von Anwendungen im Application-Layer zugreifbar und konfigurierbar sind.
Die wichtigsten Dienste des Kontext-Management-Systems zeigt Abbildung 4.3. Im Folgenden werden diese beschrieben und ihre Notwendigkeit begründet. Die dargestellten Pfeile symbolisieren die Benutzung der Komponenten untereinander.

SerializationService: Der Serialisierungsdienst übernimmt die Aufgabe zur sequentiellen Abbildung des Kontextmodells auf eine alternative Darstellungsform. Eine alternative Darstellungsform wäre beispielsweise das serialisierte Modell im XML-Format. Zusätzlich dazu unterstützt der Service die Umkehrung der Serialisierung (Deserialisierung), um die ursprüngliche Darstellungsform wieder zu erhalten. Der SerializationService wird von anderen Komponenten zur Persistenz oder zum Austausch der Modelldaten benutzt.

SecurityPrivacyService: Kontextinformationen können persönliche Informationen enthalten, die vor unbefugten Zugriff geschützt werden müssen. Zum Schutz der Privatsphäre des Benutzers und aus Sicherheitsgründen bietet der SecurityPrivacyService Funktionen zur Verschlüsselung und Entschlüsselung von kontextsensitiven Daten. Außerdem bietet dieser Dienst für eine Anwendung Konfigurationsmöglichkeiten, damit bestimmt werden kann, welcher Teil der modellierten Kontextdaten welchen anderen mobilen Teilnehmern eines verteilten Systems zur Nutzung weitergegeben werden darf.

StorageService: Es können Anwendungsfälle existieren, in denen historische Kontext-informationen ausgewertet werden müssen, um Prognosen über zukünftige Kontexte machen zu können. Vor diesem Hintergrund muss der aktuelle Zustand des Kontextmodell persistiert werden. Der StorageService bietet eine Funktion, die es mit Hilfe eines Serialisierungsdienstes erlaubt, die Daten des Kontextmodells *lokal* auf dem mobilen Gerät oder unter Verwendung eines Dienstes auf einem *entfernten* Rechner zu speichern und zu laden. Ein Intervall, das bestimmt in welchen zeitlichen Abständen die Daten gespeichert werden sollen, kann von der Anwendung vorgegeben werden.

RemoteContextService: Die Erreichbarkeit und der Kontext anderer mobiler Teilnehmer spielt eine zentrale Rolle bei der Context-Awareness im Mobile Computing. Damit mobile Geräte Kontextinformationen untereinander austauschen können, wird der Dienst RemoteContextService benötigt. Er implementiert einem das Kommunikationssystem-betreffendes, unabhängiges Protokoll, um einerseits den eigenen lokalen Kontext als Dienst zur Verfügung zu stellen und andererseits Kontextdienste anderer mobiler Teilnehmer in Anspruchen zu nehmen, um deren Kontext zu erfahren. Eine Anwendung kann diesen Dienst zeitlich skalieren, indem sie ein Intervall bestimmt, nach dessen Ablauf der Kontext erreich-barer mobiler Geräte aktualisiert wird.

NotificationService: Schilits Klassifizierung [SAW93] teilt kontextsensitive Anwendungen nach ihren unterstützten Funktionalitäten in vier Gruppen ein (vgl. Abschnitt 2.4.4). Dabei reagieren Anwendungen der Gruppe „*Context-triggered Actions*“ und „*Automatic Re-configuration*“ proaktiv auf Änderungen im Kontext. Der NotificationService beobachtet das lokale Kontextmodell und den Kontext entfernter mobiler Teilnehmer. Nach dem Feststellen einer Änderungen kann ein spezifizierter Anwendungscode ausgeführt werden. Dabei muss eine Anwendung die Bedingungen für das Betreten und Verlassen eines Kontextes

spezifizieren. Weiterhin ist durch einen *Callback-Mechanismus* festzulegen welcher Programmcode beim Betreten und welcher beim Verlassen des Kontextes ausgeführt werden soll.

UpdateService: Der UpdateService sorgt für die regelmäßige Aktualisierung des *lokalen* Kontextmodells (*LocalContext*) mit Rohdaten von externen Diensten. Diese externe Dienste werden vom *Service Management* des Context-Layers bereitgestellt. Von der Aktualisierung sind insbesondere die dynamischen *SensedAttributes* der Entitäten betroffen. Analog zu *StorageService* und *RemoteContextService* kann eine Anwendung den Dienst zeitlich skalieren. Durch setzen eines Parameters zur Laufzeit kann die Aktualisierungsrate anwendungsspezifisch geregelt werden.

TransformationService: Kontextinformationen haben verschiedene Formen, in der sie repräsentiert werden können (z.B. Temperatur in Grad Celsius oder Fahrenheit, Position des mobilen Gerätes in Längen-Breiten-Grad oder in symbolischer Form). Die bevorzugte Art der Darstellung ist von den Anforderungen einer Anwendung im Application-Layer abhängig. Der *TransformationService* stellt ein Werkzeug für den Anwendungsentwickler dar, das die *Konvertierung* der Kontextwerte (*DataValues*) in äquivalente Formate übernimmt. Dabei können die Konvertierungsalgorithmen bei einfacher Umwandlung der Kontextwerte als Teil der Middleware implementiert werden (beispielsweise geeignet für das obige „Temperaturbeispiel“) oder bei komplexer Konvertierung (z.B. das obige „Positionsbeispiel“) als externe Dienste ausgelagert werden.

5 Implementierung eines Context Services

Im vorherigen Kapitel wurde ein Konzept für die Integration von Context-Awareness in eine Middleware für mobile Systeme vorgestellt. Dieses Kapitel beschäftigt sich mit der Referenzimplementierung des in Kapitel 4 beschriebenen Konzepts. Die Middleware für mobile Systeme, die es um die Unterstützung von Context-Awareness zu erweitern gilt, ist die DEMAC Middleware („Distributed Environment for Mobility Aware Computing") [Kun05]. Sie wird im Rahmen des gleichnamigen Projektes [DEMAC] stetig weiterentwickelt. Auf Basis der Programmiersprache Java für eine J2ME-Plattform (CDC und Personal Profil 1.0) [J2me06] wird die Implementierung eines *Context Services* innerhalb der DEMAC Middleware erläutert, die Anwendung und Komponenten der Middleware Kontextinformationen bereitstellt.

5.1 Die DEMAC Middleware

Das Ziel des DEMAC Projektes ist die Konzeption und Implementierung einer *prozessorientierten* und *kontextsensitiven* Middleware für mobile Systeme in einer verteilten Umgebung. Sie soll mobile Anwendungen vor allem darin unterstützen langlebige und komplexe Prozesse verteilt auszuführen, bei Bedarf Kontextinformationen zu beziehen, und unabhängig von bestimmten Kommunikationstechnologien auf einheitliche Weise Nachrichten auszutauschen.

Die Systemarchitektur der DEMAC Middleware (siehe Abbildung 5.1) ist durch einen service-orientierten Entwurf gekennzeichnet. Verschiedene Komponenten (Services) bieten ihre Dienste auf hohem Abstraktionsgrad an. Im Folgenden werden die einzelnen Funktionen der Dienstkomponenten erläutert:

Asynchron Transport Service: Der *Asynchron Transport Service* stellt einen Dienst bereit, der das Senden und Empfangen von Nachrichten asynchron über verschiedene Kommunikationsprotokolle ermöglicht. Dabei abstrahiert er von den unterschiedlichen Netzwerktechnologien (wie beispielsweise WLAN, Bluetooth oder IrDA) über die ein mobiles Gerät kommunizieren kann. Der *Transport Service* ist dabei auch in der Lage, automatisch andere mobile Geräte zu finden. Eine interne Adressierung von mobilen Geräte mit Hilfe einer *DeviceHandle* identifiziert und registriert erreichbare mobile Geräte und die von ihnen unterstützte Kommunikationstechnologie. Nutzer des *Transport Services* können nach Erstellen einer DEMAC-Nachricht (*TransportMessage*) und der Angabe des Empfängers (mittels eines *DeviceHandles*) asynchron Nachrichten senden, ohne zu spezifizieren welche konkrete Technologie für die Kommunikation verwendet werden soll. Analog dazu können Geräte, unabhängig von der Kommunikationtechnologie, sich für das Empfangen von Nachrichten beim *Transport Service* registrieren.

Event Service: Eine Sonderform des Nachrichtenaustausches ermöglicht der *Event Service*. Er benutzt den Transport Services, um ereignisbasierte Nachrichten, die z.B. Informationen über Zustandänderungen enthalten können, proaktiv an registrierte Dienstnutzer weiterzugeleitet. Es besteht die Möglichkeit, dass sowohl einzelne mobile Geräte für bestimmte Events registriert werden als auch interne Komponenten untereinander Ereignisse austauschen.

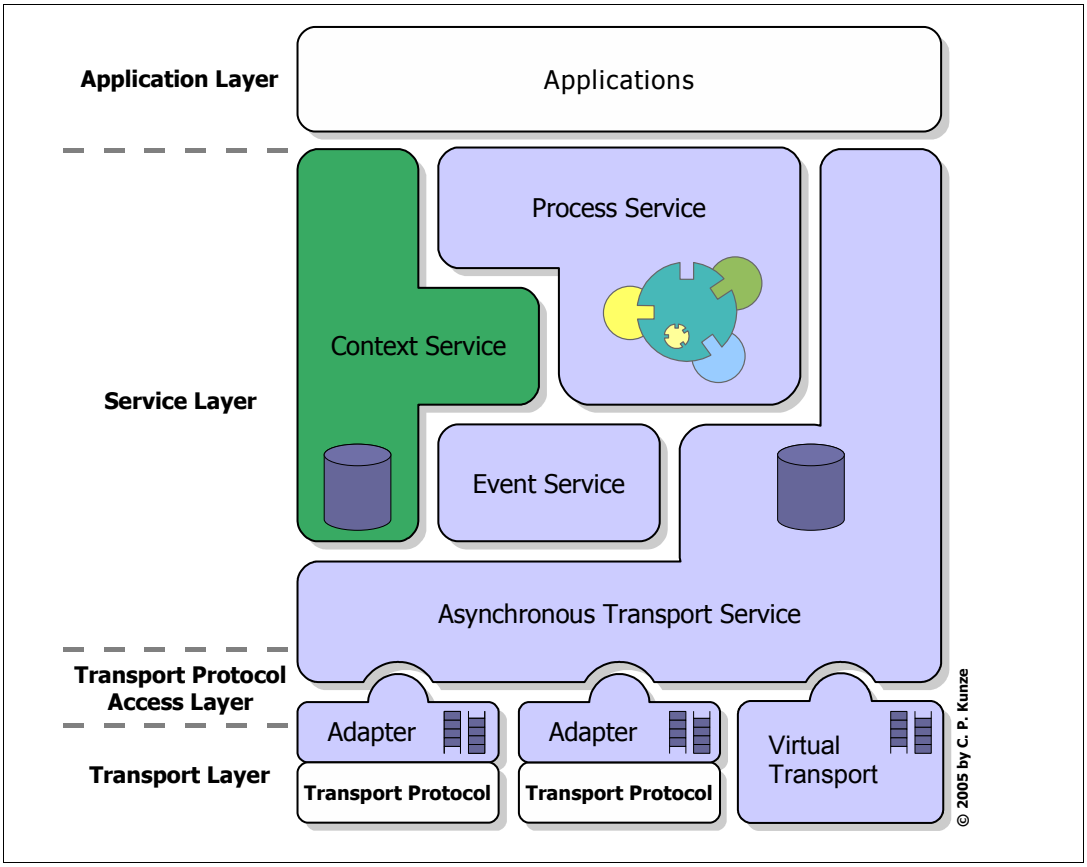


Abbildung 5.1: Die DEMAC-Systemarchitektur (nach [Kun05])

Process Service: Der *Process Service* stellt eine Infrastruktur bereit, die das Definieren und Ausführen von komplexen und langlebigen Prozessen auf mobilen Geräten ermöglicht. Das mobile Gerät kann bei der Benutzung dieses Dienstes ein Teil des Prozesses ausführen und bei Bedarf an Teilnehmer in ihrer unmittelbaren Umgebung zur Weiterbearbeitung delegieren. Die benötigten Dienste und Prozesspartner zur Ausführung des Prozesses werden dabei dynamisch ausgewählt [Zap05]. Ein Teil der erforderlichen Dienste zur Prozessausführung wird aus dem *Context Service* bezogen. Der *Context Service* stellt dabei auch Schnittstellen zur Verfügung, die die vorhandenen Dienste nach Nutzungseigenschaften, wie zum Beispiel Kosten oder Verfügbarkeit, selektieren lässt. Diese Merkmale bei der Benutzung von Diensten werden unter dem Begriff der *nicht-funktionalen Aspekte* zusammengefasst [Zap05].
Im Rahmen des DEMAC-Projekts beschäftigte sich Zaplata (in ihrer Diplomarbeit [Zap05]) mit der Konzeption und Realisierung des *Process Services*. In ihr Arbeit geht es vor allem um den Entwurf einer *Prozessbeschreibungsprache*, die geeignet für mobile Systeme ist, und der Entwicklung einer *verteilten Ausführungsumgebung*, die die Interpretation, Ausführung und Weitergabe der beschriebenen Prozesse ermöglicht.

Context Service: Der *Context Service* implementiert einen Dienst, der den Anwendungen im *Application Layer* und den *Prozess Service* innerhalb der DEMAC *Service Layer* Kontextinformationen bereitstellt. Die Implementierung basiert auf der entwickelten Konzeption im vorherigen Kapitel. Abweichend von der Konzeption der Architektur im Abschnitt 4.3 ist in

der DEMAC Middleware das *Service Management* ein Teil des *Context Service*. Es tritt in Form einer *Service Registry* auf, die als ein verteiltes Dienstverzeichnis heterogene Dienste (der Raw-Data-Layer) in einheitlicher Form verwaltet.

5.2 Implementierung des DEMAC Context Service

In diesem Abschnitt wird die prototypische Implementierung des Context Service im Rahmen des DEMAC Projektes [DEMAC] besprochen. Dabei steht die beispielhafte Umsetzung der entwickelten Konzeption aus Kapitel 4 in den Vordergrund. Für einige Komponenten des Kontextmodell und Kontext-Managements wurden lediglich die Java-Interfaces anlegt.
In den folgenden Abschnitten werden die festgelegten Schnittstellen für den *Context Service*, die Elemente des Kontextmodells und die Dienste des Kontext-Managements erläutert.

5.2.1 Schnittstellen des Context Service

Der Context Service spezifiziert als autonomer Dienst für Softwarekomponenten einfache Schnittstellen, um den lokalen Kontext zu initialisieren, benötigte Dienste des Kontext-Managements zu konfigurieren und den Zugriff auf den Verzeichnisdienst (*Service Registry*) zu ermöglichen.
Context-Awareness im Mobile Computing schließt auch solche Daten als Kontext-informationen ein, die Eigenschaften verfügbarer Dienste beschreiben. Als nicht-funktionale Aspekte solcher Dienste können diese dann zur Konfiguration von Prozessen innerhalb des *Process Services* dienen [Zap05]. Abbildung 5.2 zeigt die Modellierung nicht-funktionaler Aspekte zur *Selektion* von Diensten aus der *Service Registry*.

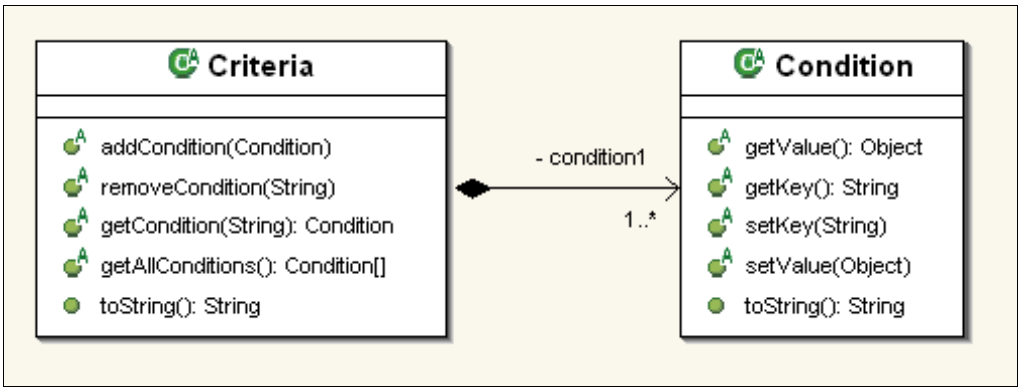


Abbildung 5.2: Kriterien zur Selektion von Diensten als UML-Klassendiagramm

Dabei besteht ein Kriteriumsobjekt zur Auswahl von Diensten aus einer nicht-leeren Menge von Bedingungsobjekten, die anwendungsspezifisch implementiert werden können. Ein `Condition`-Objekt kann anhand eines Key-Attributes angeben, um welche Bedingung es sich handelt (beispielweise `Key=cost`) und welcher Wert ihr zugeordnet ist (z.B. `Value=free`). Der *Context Service* wertet beim Zugriff auf die *Service Registry* das spezifizierte Kriterium aus und ermöglicht nur dann den Zugriff auf den angegebenen Dienst, wenn das Kriterium im aktuellen Kontext erfüllt ist.
Quelltext 5.1 zeigt die Schnittstellen des Context Service als Java-Interface. Alle Dienste der DEMAC Middlewares (siehe Abbildung 5.1) implementieren das Interface `Service`. Es spezifiziert im wesentlichen eine Methode zum Starten und eine zum Beenden des Dienstes.

Die Methoden mit dem Namen `isServiceAvailable` können aufgerufen werden, um die Verfügbarkeit eines `RawDataServices` in der aktuellen Umgebung zu prüfen. `RawDataServices` kapseln in homogener Form alle Dienste, die in der *Service Registry* enthalten sind. Die Identifikation dieser Dienste erfolgt mit einem `ServiceHandle`-Objekt. Mit Hilfe des `Criteria`-Parameters können Angaben zu nicht-funktionalen Aspekten des Dienstes gemacht werden.

```
public interface ContextService extends Service {

      public boolean isServiceAvailable(ServiceHandle handle)
            throws UnknownServiceException;

      public boolean isServiceAvailable(ServiceHandle handle,
            Criteria criteria) throws UnknownServiceException,
                  ServiceNotReachableForCriteriaException;

      public RawDataService getService(ServiceHandle handle)
            throws UnknownServiceException;

      public RawDataService getService(ServiceHandle handle,
            Criteria criteria) throws UnknownServiceException,
                  ServiceNotReachableForCriteriaException;

      public void initLocalContext(LocalDeviceContext localContext);

      public LocalDeviceContext getLocalDeviceContext();

      public void addManagementService(ContextManagementService
            service);

      public ContextManagementService getManagementService(
            ContextManagementHandle handle);

      public void removeManagementService(ContextManagementHandle
            handle);

}
```

Quelltext 5.1: Schnittstellen des *Context Service* als Java-Interface

Die Methode mit der Signatur `getService(ServiceHandle,Criteria)` liefert nur dann eine Referenz auf einen `RawDataService`, wenn im aktuellen Kontext das spezifizierte Kriterium für den Dienst erfüllt ist.
Die Initialisierung des lokalen Kontext übernimmt die Methode `initLocalContext`. Dabei muss der Nutzer des *Context Service* ein `LocalDeviceContext`-Objekt konstruieren, das das lokale Kontextmodell wiedergibt. Das Kontextmodellobjekt wird dann intern verwaltet. Den aktuellen Zustand des lokalen Kontextmodells kann mit der Methode `getLocalDeviceContext` abgerufen werden.
Ein `ContextManagementService`-Objekt stellt die Implementierung einer Kontext-Management-Komponente (vgl. Abschnitt 4.3.2) dar. Sie werden anhand eines `ContextManagementHandle` eindeutig identifiziert. Der Dienstnutzer kann mit der Methode `addManagementService` ein bereits erzeugtes Kontext-Management-Objekt angeben, das das Kontextmodell bearbeiten soll. Wird ein `ContextManagementService` nicht mehr

benötigt, kann er während der Laufzeit wieder deaktiviert und aus der Liste der Managementdienste entfernt werden.

5.2.2 Kontextmodell

Die Implemenation des Kontextmodells setzt den Entwurf von Java-Interfaces zur Schnittstellendefinition voraus. Konkrete Klassen eines Kontextmodells müssen die definierten Schnittstellen implementieren, um ein anwendungsspezifisches Modell für den *Context Service* bereitzustellen.
Das UML-Klassendiagramm in der Abbildung 5.3 zeigt die Beziehungen zwischen den einzelnen Interfaces. Es gibt zwei Typen eines `DeviceContext`. Beide führen eine Referenz auf das `DeviceHandle`, welches das mobile Gerät angibt, aus dem die Kontextinformationen stammen.

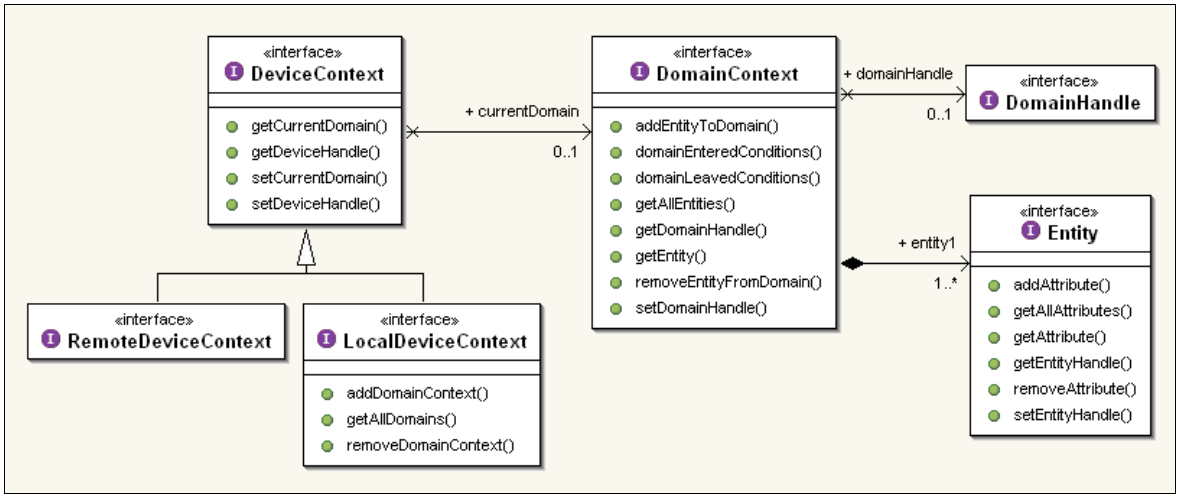


Abbildung 5.3: UML-Klassendiagramm zum implementierten Domänenmodell

Zusätzlich dazu liefert ein `DeviceContext`, die aktuelle (aktive) Domäne des Kontext (`getCurrentDomain`-Methode). Der `LocalDeviceContext` spezifiziert im Gegensatz zu einem `RemoteDeviceContext` weitere Methoden zur Verwaltung des *lokalen* Kontext eines mobilen Gerätes. Dazu gehören vor allem die Angabe anderer – disjunkter – Domänen, in den sich das mobile Gerät befinden kann.
Die Schnittstelle `DomainContext` bietet im wesentlichen den Zugriff auf ihre Entitäten. Zur Identifikation einer Domäne kann ein `DomainHandle` verwendet werden. Eine wichtige Aufgabe eines Anwendungsentwicklers bei der Konstruktion einer Domäne ist die Implementierung der Methoden `domainEnteredConditions` und `domainLeaved-Conditions`. Sie spezifizieren unter welchen Voraussetzungen eine Domäne betreten bzw. verlassen wird. Damit kann also die Gültigkeit der Domäne innerhalb des Kontexts bestimmt werden. Beide Methoden müssen einen Wahrheitswert zurückgeben, die aus der Auswertung der angegebenen Bedingungen entsteht.
Eine `Entity` aggegieren eine Menge von `Attribute`-Objekte. Es wird zwischen drei verschiedene Typen von Attributen (siehe Abbildung 5.4) unterschieden. `Sensed-Attributes` modellieren Kontextinformationen, die einen dynamischen Charakter haben, d.h. der Kontextwert ändert sich während der Laufzeit relativ häufig. Die Quelle für solche Kontextinformationen sind üblicherweise Sensoren. Mit entsprechenden Methoden verweist das `SensedAttribute`-Interface auf ein `RawDataService`, aus dem die rohe Kontextdaten

bezogen werden. Attribute, die einen High-Level-Kontexttypen (vgl. Abschnitt 2.4.2) darstellen, müssen das Interface `DeducedAttribute` implementieren. Darin können die benötigten Referenzen auf die Entitäten (und ihre Kontextinformationen) hergestellt werden, aus dem die „höherwertige“ Kontextinformationen hergeleitet werden. Statische Informationen, die sich während der Laufzeit nicht ändern aber dennoch zum Kontext der Anwendung gehören, werden durch die Implementierung des Interfaces `Defined-Attribute` modelliert.
Ein `Attribute` ist getrennt von dem eigentlichen Kontextwert (`AttributeValue`), der eine konkrete Kontextinformation einer Entität enthält. Eine Beispielimplementierung eines `AttributeValue` wäre ein `SpeedAttributeValue`, der die aktuelle Bewegungsgeschwindigkeit der Entität Benutzer angibt. In solchem Fall könnte die Methode `getUnit` „km/h“ als Einheit des Datum zurückliefern.
Konsequenterweise muss ein `Attribute` durch eine 1-zu-1-Beziehung eine Referenz auf das entsprechende `AttributeValue` abbilden. Mit weitere Methoden eines Attributes kann bestimmt werden, ob die repräsentierten Informationen anderen mobilen Teilnehmern bereitgestellt bzw. zur späteren Weiterverarbeitung persistiert werden sollen.

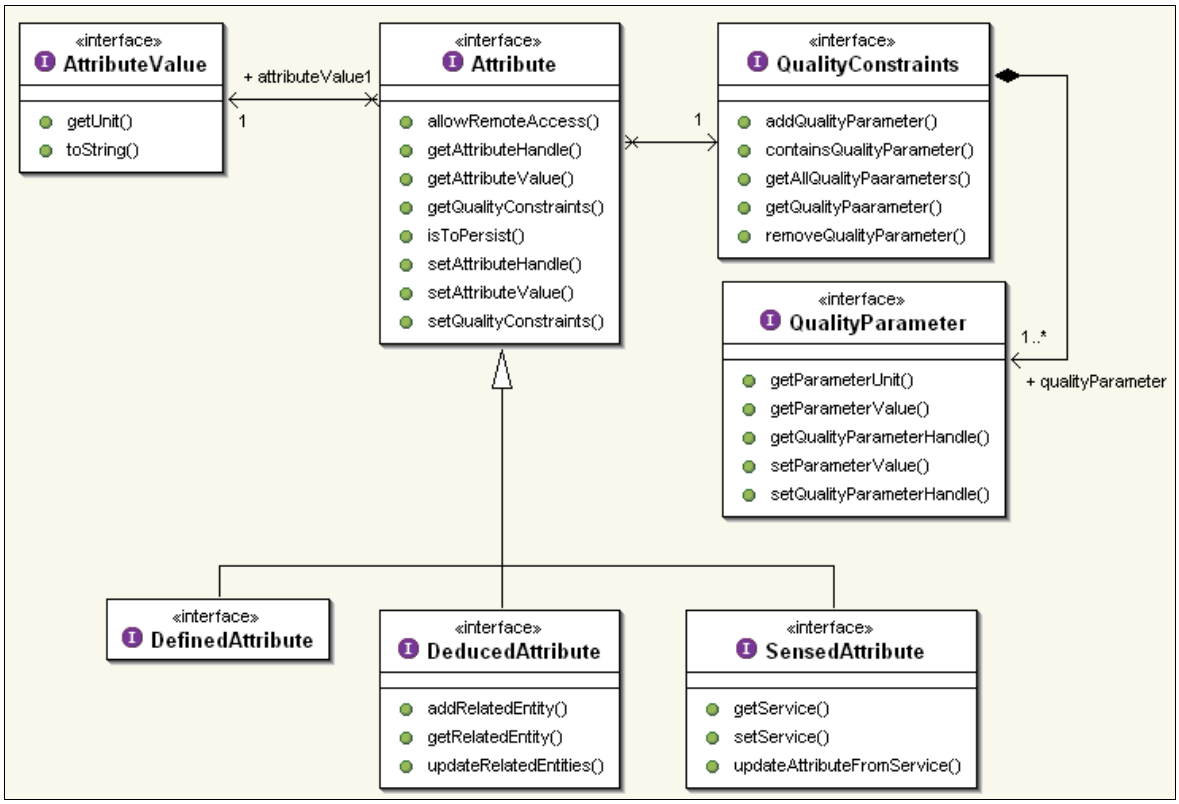


Abbildung 5.4: UML-Klassendiagramm zum implementierten Attributenmodell

Ein wichtiger Aspekt der Kontextpräsentation in einem Informationssystem ist die Modellierung von qualitativen Parametern, die für einen bestimmten Kontextwert gelten (vgl. Abschnitt 4.1, A7). Ein `QualityConstraints` gibt mittels einer 1-zu-1-Beziehung an, welche qualitative Einschränkungen bezüglich eines Attributes zu erwarten sind. Einzelne Qualitätsmerkmale wie Beispielsweise Genauigkeit oder Standardabweichung können durch die Implementierung eines `QualityParameter`-Interfaces modelliert werden.

5.2.3 Kontext-Management

Wie aus dem Quelltext 5.1 zu entnehmen ist, enthält die Implementierung eines `ContextService` eine Menge von Subdiensten, die die Verwaltung und Verarbeitung des Kontextmodells übernehmen. Der `ContextService` bildet somit einen Container für Kontext-Management-Dienste.
Das UML-Klassendiagramm in Abbildung 5.5 verdeutlicht die Beziehungen zwischen den einzelnen Diensten. Dabei werden alle Dienste des Kontext-Managements vom Interface `ContextManagementService` abgeleitet, die selbst die Schnittstellen des Interfaces `Service` erbt. Jeder Dienst kann abhängig von den Anforderungen der jeweiligen Anwendung dynamisch während der Laufzeit gestartet und beendet werden. Weiterhin hat jeder Managementdienst einen Zugriff auf die Objekte des Kontextmodells, um sie zu bearbeiten. Im Rahmen dieser Arbeit wurde nur die in Abbildung 5.5 dargestellten Managementdienste prototypisch implementiert.
Als abstrakte Java-Klasse spezifiziert der `SerializerService` Methoden zum serialisieren und deserialisieren des Kontextmodells. Dabei muss zu einem bestimmten Attributtyp des Modells der dazugehörige `AttributeSerializer` angegeben werden. Die Aufgabe eines `AttributeSerializer` besteht darin, die in einem `Attribute`-Objekt enthaltenen Kontextinformationen in eine äquivalente Darstellungsform zu überführen. Anhand dieses Mappings und der Java-Reflection-API [Krü02] ist beispielsweise ein `XMLSerializer-Service` (abgeleitet von einem `SerializerService`) in der Lage alle Java-Objekte des Kontextmodells in ein XML-Format abzubilden und wieder zu rekonstruieren.

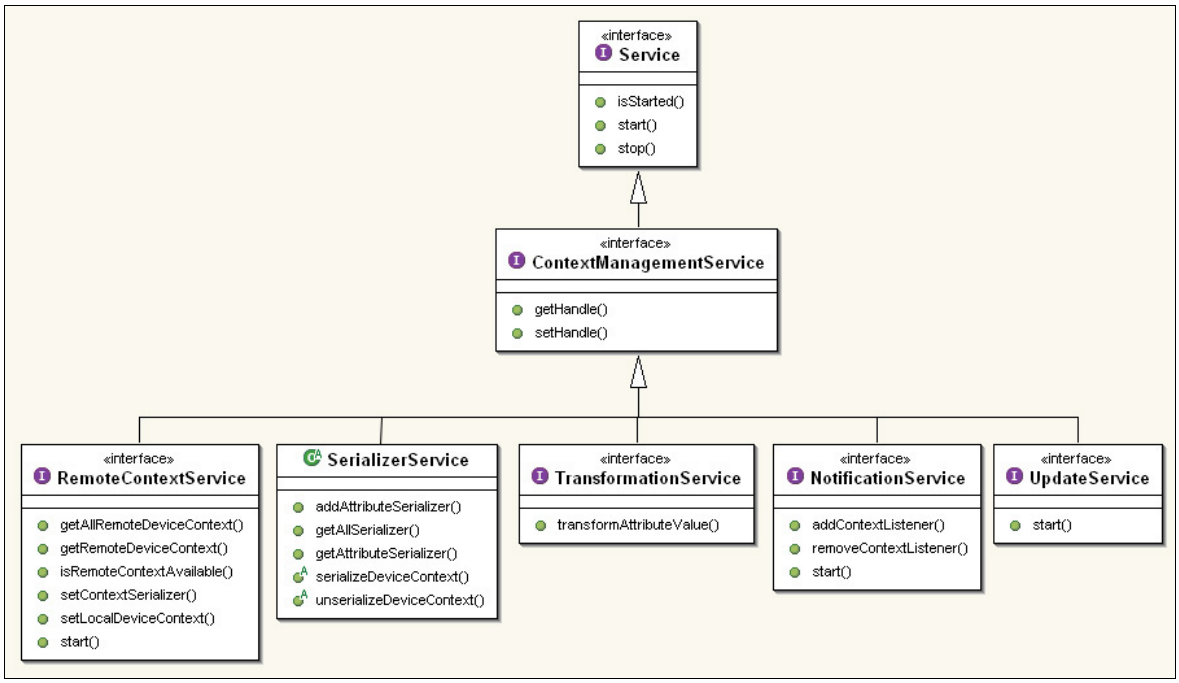


Abbildung 5.5: Dienste des Kontext-Management als UML-Klassendiagramm

Das Interface `TransformationService` spezifiziert eine Methode zur Konvertierung von `AttributeValues` (Kontextwerten) innerhalb des Kontextmodells. Im Gegensatz zum `SerializerService` wird hier nicht das gesamte Modell transformiert, sondern nur einzelne Kontextwerte (`AttributeValues`) in verschiedene Masseinheiten umgerechnet. Ein typischer Anwendungsfall ist beispielsweise die Umrechnung des Kontextwertes

`CelsiusAttribute-Value` in `FahrenheitAttributeValue` für einen Benutzer aus den USA.
Die Aktualisierung des lokalen Kontextmodells mit Kontextinformationen, die sich regelmäßig ändern, ist eine wichtige Verwaltungsaufgabe. Jedes Modell, welches dynamische Kontextwerte enthält, muss ein `UpdateService` starten, um die Modelldaten hinsichtlich ihrer Aktualität konsistent zu halten. Die Methode mit der Signatur `start(long)` startet den Dienst als ein Java-Thread [Krü02], das parallel zu anderen Threads ausgeführt wird. Das Zeitintervall `updateInterval` (in Millisekunden) als Parameter bestimmt die Aktualisierungsrate des Kontextmodells mit unverarbeiteten Kontextinformationen.

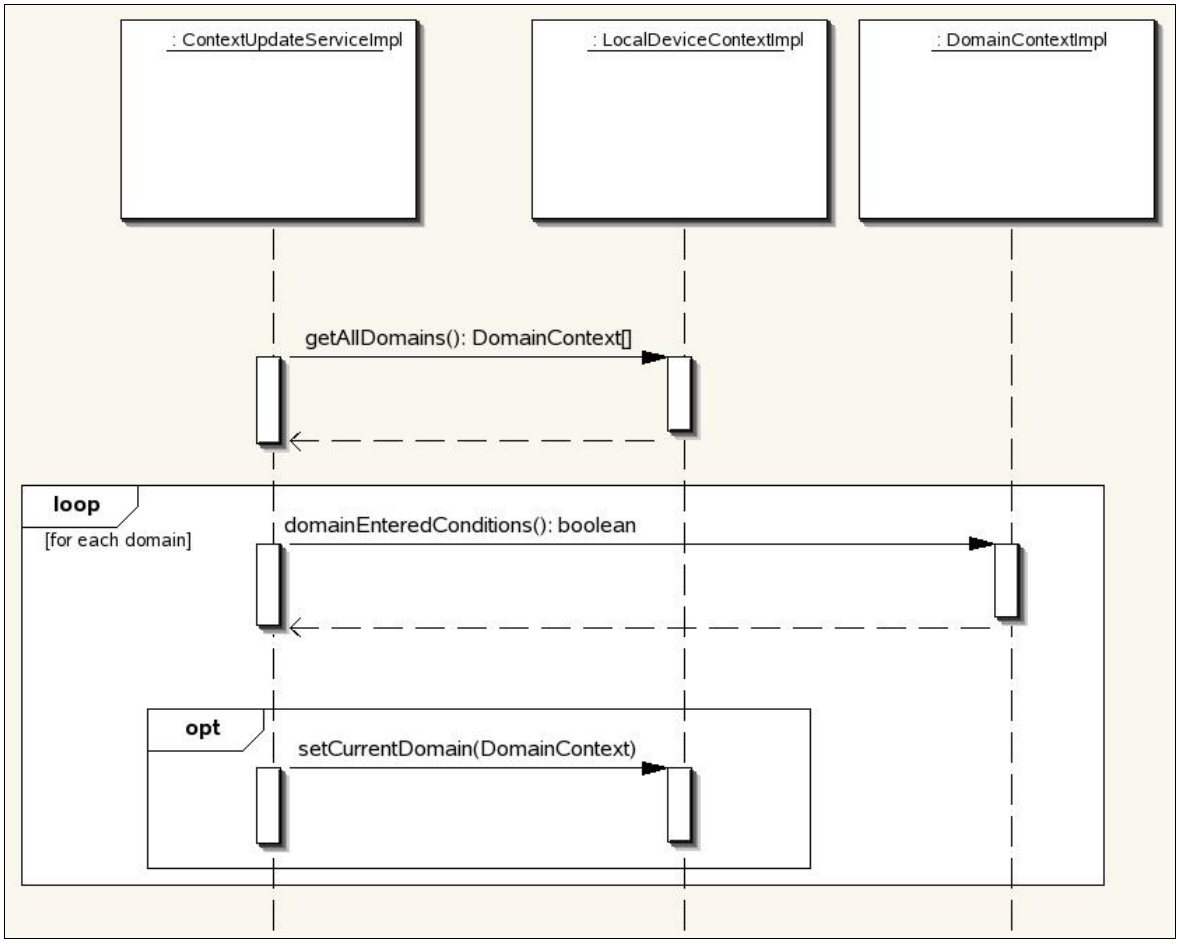


Abbildung 5.6: Ermittlung der gültigen Domäne

Das UML-Sequenzdiagramm in Abbildung 5.6 zeigt wie nach Ablauf des Zeitintervalls die gültige Domäne für das Kontextmodell ermittelt wird. Dazu werden zunächst alle Domänen, die in einem `LocalDeviceContext`-Objekt enthalten sind, abgerufen. Anschließend wird für jedes Domänobjekt anhand der Methode `domainEnteredConditions` die Bedingungen für seine Gültigkeit im aktuellen Kontext geprüft. Liefert diese Methode `true` als Wahrheitswert zurück, dann wird die betreffende Domäne mittels der Methode `setCurrentDomain` als die aktuell gültige Domäne im `LocalDeviceContext` registriert. Wird keine einzige gültige Domäne gefunden, so wird der Aktualisierungsprozess abgebrochen und auf die nächste Überprüfung gewartet.

```
// iterate over all entities of current domain
Entity[] entities = localContext.getCurrentDomain().getAllEntities();
for (int i = 0; i < entities.length; i++) {
      Entity entity = entities[i];
      // iterate over all attributes of current entity
      Attribute[] attributes = entity.getAllAttributes();
      for (int j = 0; j < attributes.length; j++) {
            Attribute attribute = attributes[j];
            // update current SensedAttribute
            if (attribute instanceof SensedAttribute) {
                  ((SensedAttribute) attribute).updateAttributeFromService();
            }
            // refresh relations for current DeducedAttribute
            if (attribute instanceof DeducedAttribute) {
                  ((DeducedAttribute) attribute).updateRelatedEntities();
            }
      }
}
```

Quelltext 5.2: Aktualisierung der Attribute

Ist eine gültige Domäne ermittelt, können die dazugehörigen Entitäten und Attribute sukzessiv mit neuen Kontextinformationen, welche aus den entsprechenden Kontextquellen bezogen werden, aktualisiert (siehe Quelltext 5.2). Die konkrete Aktualisierung der Modellattribute übernimmt jedes `SensedAttribute` für sich selbst. Bei abgeleiteten Attributen (`DeducedAttribute`) müssen die Referenzen auf die Entitäten, die sie zur Ableitung ihres eigenen Wertes benötigen, erneuert werden.

Zum Austausch von Kontextinformationen zwischen mobilen Teilnehmern kann ein `RemoteContextService` benutzt werden. Eine prototypische Implementierung des Dienstes (`RemoteContextServiceImpl`) wurde im Rahmen des DEMAC-Projektes entwickelt. Der `RemoteContextServiceImpl` verwendet zum Senden und Empfangen des lokalen Kontextes den `EventService` und den `TransportService` der DEMAC Middleware (vgl. Abschnitt 5.1).
Mit dem `EventServiceImpl` besteht die Möglichkeit `Events` in eigenem Gerät zu generieren und an erreichbare mobile Geräte zu senden. Events sind kompakte Nachrichten, die ein Ereignis beschreiben und an alle verbundene mobile Teilnehmer gesendet werden. Der `RemoteContextServiceImpl` benutzt diesen Mechanismus, um den lokalen Kontext anderen Geräte zugänglich zu machen. Dazu wurde ein `RemoteContextRequestEvent` konstruiert, das den `EventService` übergeben und in regelmäßigen Abständen an die gefunden mobilen Teilnehmer gesendet wird, um deren Kontextinformationen anzufordern. Wenn mobile Geräte ihren Kontext anderen Teilnehmern mitteilen wollen, müssen sie auf ein `RemoteContextRequestEvent` reagieren.
Das UML-Sequenzdiagramm in Abbildung 5.7 zeigt wie sich der `RemoteContextServiceImpl` eines Gerätes verhält, wenn es aufgefordert wird, seinen Kontext zu versenden: Nach dem der Dienst `RemoteContextServiceImpl` gestartet wurde, erzeugt er ein `RemoteContextRequestListener`-Objekt. Es hat den Zweck speziell auf ein `RemoteContextRequestEvent` zu reagieren. Dazu muss es sich mit der Methode `addEventListener` am `EventService` registrieren. Empfängt nun der `EventServiceImpl` ein `RemoteContextRequestEvent` informiert er alle registrierten Listener, indem er die entsprechenden Callback-Methode (`eventTriggered`) des Listeners aufruft. Durch diesen Aufruf wird der lokale Kontext des mobilen Gerätes mit Hilfe eines `XMLContext-`

`SerializerService` in ein XML-Format überführt und als ein `TransportMessage`-Objekt zum Versenden dem `TransportServiceImpl` übergeben.

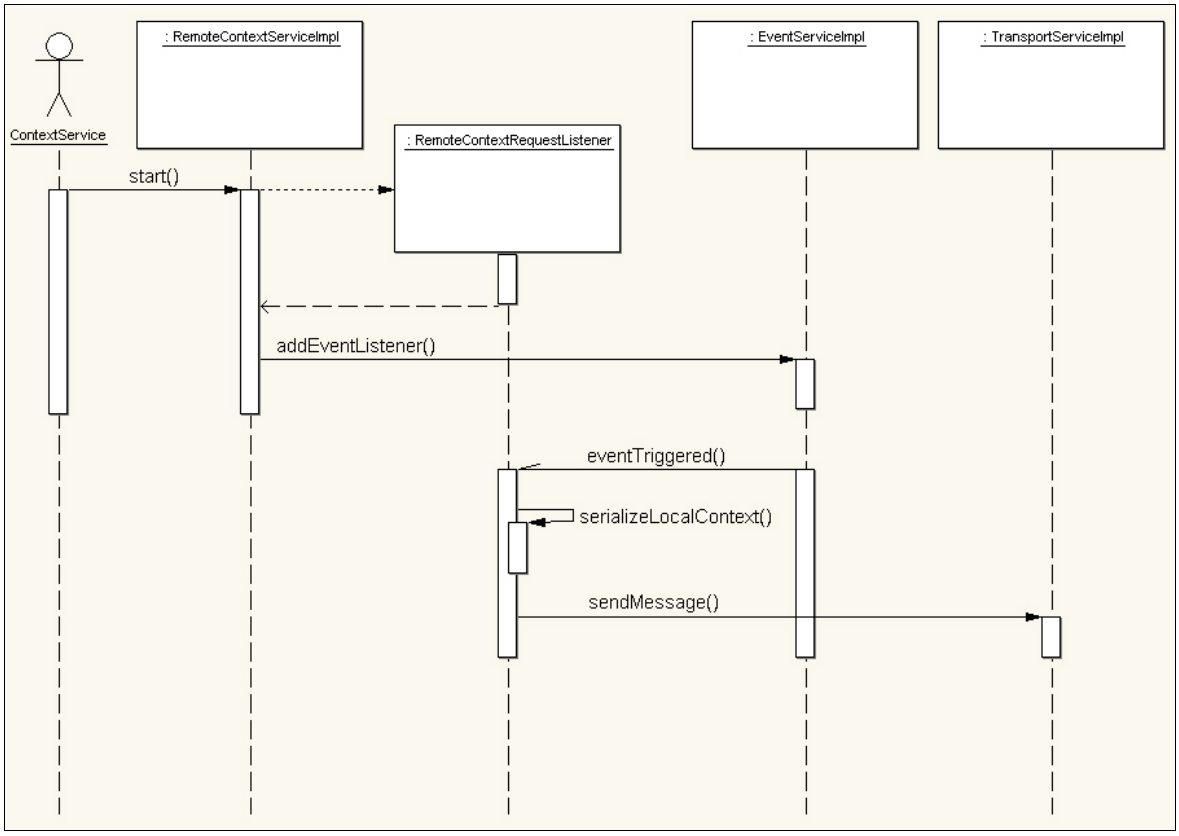


Abbildung 5.7: Senden des lokalen Kontextes als UML-Sequenzdiagramm

Beim Empfang der Nachricht deserialisiert die Gegenseite die Kontextinformationen zu einem `RemoteDeviceContext`-Objekt und fügt sie zu der Liste der verfügbaren entfernten Kontextinformationen (siehe Quelltext 5.3). Ist die Verbindung zu einem mobilen Gerät unterbrochen oder langfristig getrennt, so wird dessen Kontext aus der Liste der verfügbaren Kontextinformationen wieder entfernt.

```
public void messageArrived(MessageHandle handle) {
	if (isRemoteContextMessage()) {
		// get the message from TransportService
		TransportMessage msg = transportService.extractMessage(handle);
		SerializedDeviceContext serializedContext = new
			XMLSerializedDeviceContext(msg.getMessageBody());
		// unserialize remote device context from the message
		RemoteDeviceContext remoteContext = (RemoteDeviceContext)
			serializer.unserializeDeviceContext(serializedContext);
		// put the remote context to the list of available remote contexts
		remoteContexts.put(remoteContext.getDeviceHandle(), remoteContext);
	}
}
```

Quelltext 5.3: Empfang und Deserialisierung eines entfernten Kontextmodells

Der `NotificationService` gibt einer Anwendung die Möglichkeit auf Änderungen im Kontext zu reagieren. Dazu müssen `ContextListener` mit anwendungsspezifischen Eigenschaften konstruiert werden, die anschließend dem `NotificationService` über-

geben werden. Durch einen `ContextListener` kann ein Anwendungsentwickler Bedingungen für die Veränderungen im Kontext angeben. Zusätzlich dazu können Callback-Methoden implementiert werden, die vom `NotificationService` ausgeführt werden, sobald die entsprechenden Bedingungen erfüllt sind. Quelltext 5.4 zeigt die spezifizierten Methoden des `ContextListeners`.

```
public interface ContextListener {

	public boolean contextEnteredConditions();

	public boolean contextLeavedConditions();

	public void notifyContextEntered(ContextEvent event);

	public void notifyContextLeaved(ContextEvent event);

}
```

Quelltext 5.4: Methoden des Interfaces `ContextListener`

Nach dem Starten des Dienstes wird ein neuer Java-Thread erzeugt, der regelmäßig alle `ContextListener`-Objekte überprüft (siehe Quelltext 5.5). Sind die entsprechenden Bedingungen eines Listeners erfüllt, wird ein `ContextEvent` erzeugt, die eine Referenz auf die aktuellen Kontextinformationen enthält. Anschließend werden die Callback-Methoden `notifyContextEntered` bzw. `notifyContextLeaved` mit dem erzeugten `ContextEvent` als Parameter aufgerufen.

```
// iterate over all registered listeners
for (Iterator iter = listeners.iterator(); iter.hasNext();) {
	ContextListener listener = (ContextListener) iter.next();
	RemoteContextService remoteService =
		RemoteContextServiceImpl.getInstance();
	// checks for entering the context of current listener
	if(!listener.isContextEntered() && listener.contextEnteredConditions()) {
		ContextEvent event = new ContextEventImpl();
		event.setLocalDeviceContext(getLocalDeviceContext());
		event.setRemoteDevicesContext(remoteService.getAllRemoteContext());
		listener.setContextEntered(true);
		// notify for entering the context
		listener.notifyContextEntered(event);
	}
	// checks for leaving the context  of current listener
	if(listener.isContextEntered() && listener.contextLeavedConditions()) {
		ContextEvent event = new ContextEventImpl();
		event.setLocalDeviceContext(service.getLocalDeviceContext());
		event.setRemoteDevicesContext(remoteService.getAllRemoteContext());
		listener.setContextEntered(false);
		// notify for leaving the context
		listener.notifyContextLeaved(event);
	}
}
```

Quelltext 5.5: Überprüfung der `ContextListener` durch den `NotificationService`

6 Eine kontextbezogene Anwendungsszenario

Im vorherigen Kapitel wurde die prototypische Implementierung des Kontextdienstes im Rahmen des DEMAC-Projektes vorgestellt. Gegenstand dieses Kapitels ist die Evaluierung des *Context Service* anhand einer kontextsensitiven mobilen Anwendung. Es soll zunächst gezeigt werden wie der DEMAC *Context Service* innerhalb der Anwendungsentwicklung benutzt wird. Außerdem soll geprüft werden, inwieweit die Bereitstellung von Kontextinformationen durch eine kontextsensitive Middleware die Entwicklung von kontextbezogenen Anwendungen unterstützen kann.
Die Anwendung wurde auf Grundlage der *J2ME-Plattform* [J2me06] mit der Konfiguration *CDC* (Connected Limited Device) [Cdc06] und dem J2ME-Profil *Personal Profile* [Per06] entwickelt. Als „J2ME virtual mashine“ kommt die „IBM J9“ zum Einsatz, die auf dem Betriebssystem Windows Mobile 2003 ausgeführt wird. Bei dem benutzten mobilen Gerät handelt es um einen Pocket PC (PDA) der Firma HP (iPAQ 5550).

6.1 Eine mobile kontextsensitive Instante-Messaging-Anwendung

Bereits bevor mit dieser Arbeit begonnen wurde, wurde im Rahmen des DEMAC-Projektes und aufbauend auf die DEMAC Middleware ein einfaches Instante-Messenger-Programm entwickelt. Der *Simple DEMAC Messenger* - kurz SDM – ermöglicht unter anderem das Suchen von anderen mobilen Teilnehmern. Ist ein gewünschter SDM-Teilnehmer (Buddy) gefunden, kann er in die Kontaktliste (Buddy-Liste) einfügt werden. Die Kontaktliste zeigt alle Spitznamen der Kontakte und den dazugehörigen Online-Status an. Bei Erreichbarkeit eines SDM-Teilnehmers kann eine Chat-Sitzung in einem separaten Fenster gestartet werden.
Der SDM wurde im Rahmen dieser Diplomarbeit, um die Nutzung und Bereitstellung von Kontextinformationen erweitert. Diese Erweiterungen sind Teil des kontextbezogenen SDM – dem *Context-Aware Simple DEMAC Messenger* (kurz C-SDM). Zwar setzt er auf dem entwickelten SDM auf, aber dennoch bildet er zusammen mit der DEMAC Middleware eine eigenständige Anwendung. Folgende kontextbezogenen Features sind in C-SDM realisiert:

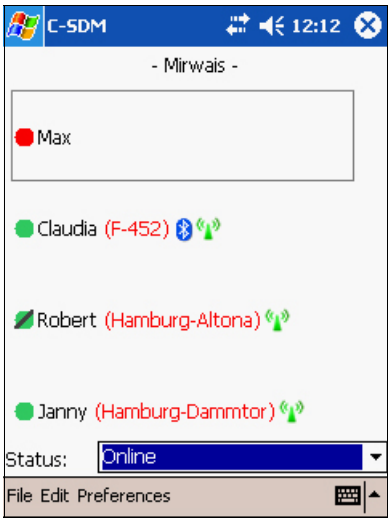


Abbildung 6.1: Buddy-Liste

Kontextbezogene Buddy-Liste: Die Kontaktliste des C-SDM (siehe Abbildung 6.1) zeigt zusätzlich zu den Nicknames den aktuellen Aufenthaltsort des jeweiligen Buddies in roter Schrift. Befindet sich eine Person im Freien so wird die Stadt und der Stadtteil eingeblendet (in Abbildung 6.1 beispielweise bei Robert und Janny). Ist der Benutzer und eine Person aus seiner Kontaktliste in dem selben Gebäude, so wird nur die Raumnummer angezeigt.
Weiterhin zeigen Symbole neben dem Aufenthaltsort die technische Kommunikationsmöglichkeiten mit der Person aus der Buddy-Liste. Das grüne Symbol zeigt, dass zwei C-SDM-Benutzer über eine WLAN-Verbindung mit einander kommunizieren können. Ist zusätzlich ein blaues Symbol eingeblendet, kann im aktuellen Kontext mit einer Bluetooth-Verbindung kommuniziert werden.

Detaillierte Kontextinformationen: Zusätzlich zur Position und Verbindungsmöglichkeiten können Benutzer weitere Kontextinformationen zu einem Buddy abrufen. Dazu wird über einen ausgewählten Buddy der Menüpunkt „Show context information“ aufgerufen (siehe Abbildung 6.2, links). Im darauf folgenden Dialog können detaillierte Kontextinformationen zu dem Buddy eingesehen werden (Abbildung 6.2, rechts). Diese betreffen vor allem die Identifikation des Benutzers und die Eigenschaften eines mobilen Gerätes.

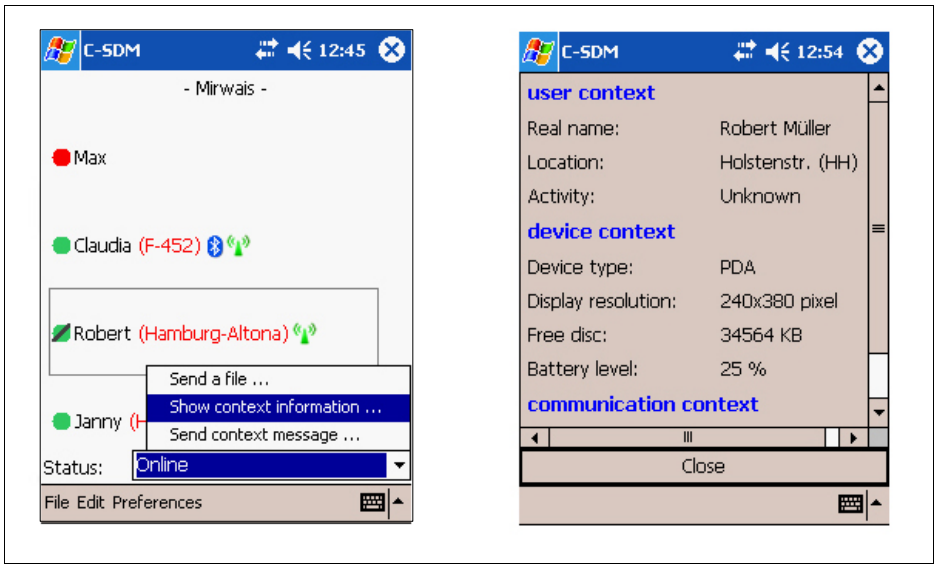


Abbildung 6.2: Detaillierte Kontextinformationen eines Buddies

Ortsbezogene Nachrichten: Im C-SDM werden nicht nur die Kontexte von Benutzern dargestellt sondern auch das Verhalten der Anwendung durch Kontextinformationen aktiv gesteuert.
Ähnlich wie die *Stick-E-Notes* Anwendung (vgl. Abschnitt 2.3.4) bietet der C-SDM die Möglichkeit das Versenden von Nachrichten an einen bestimmten Ort zu knüpfen. Über den Menüpunkt „Send context message“ (siehe Abbildung 6.2, links) kann der Sendevorgang für solch eine Nachricht an einem Buddy initialisiert werden. Die Abbildung 6.3 zeigt die entsprechenden Anwendungsmasken zum Versenden und Empfangen. Dabei muss der Sender einen Raum (mittels einer Auswahlliste) und den Nachrichtentext angeben (Abbildung 6.3, links). Die Nachricht ist nun an diesen Ort gebunden. Sobald der Benutzer den angegebenen Raum betritt, wird die Nachricht automatisch angezeigt (Abbildung 6.3, rechts).

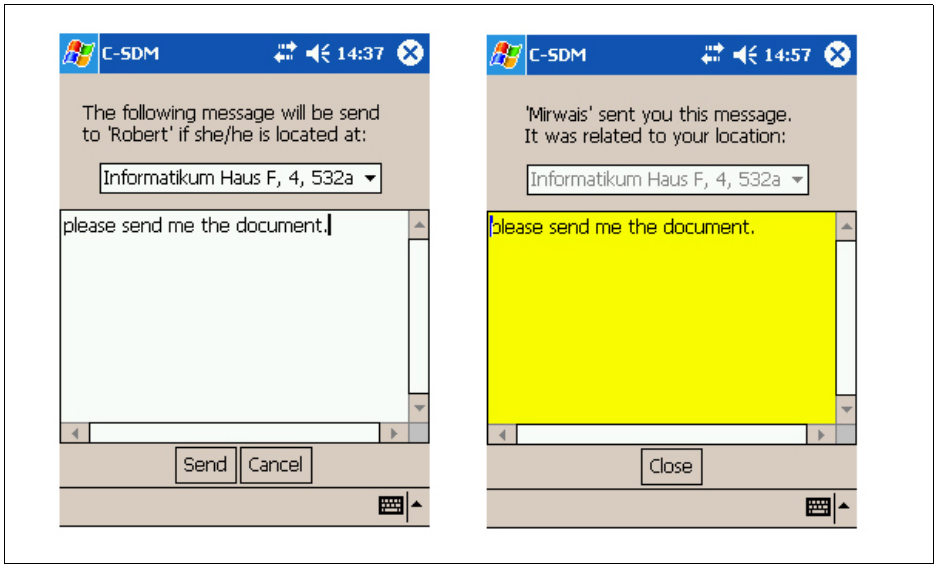


Abbildung 6.3: Masken zum Austausch von ortsbezogenen Nachrichten

Kontextwarnungen beim Dateitransfer: Im Zuge der Weiterentwicklung von SDM zu C-SDM wurde auch ein Feature zum Transfer von Dateien implementiert. Nach dem Selektieren eines Buddies und dem aktivieren des Menüpunktes „Send a file“ (siehe Abbildung 6.2, links) öffnet sich ein Dialog zum Wählen einer lokalen Datei. Bevor jedoch die Datei in *Base64* kodiert wird, um sie als Folge von Textnachrichten versenden zu können, werden die Kontextinformationen des Empfängers und die Eigenschaften der Datei verglichen. Dabei geht es darum zu überprüfen, ob das Versenden der Datei im aktuellen Kontext des Empfängers möglich oder sinnvoll ist. Die Maske in der Abbildung 6.4 zeigt beispielhaft einige Hinweise, die als Folge dieser Prüfung dem Sender angezeigt werden bevor der Sendeprozess beginnt.

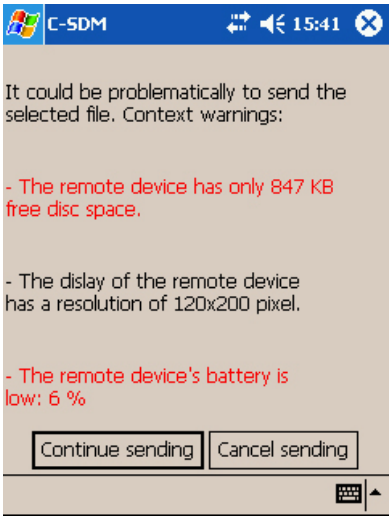


Abbildung 6.4: Kontextwarnungen

Die in roter Schrift angezeigten Hinweise deuten auf besonders kritische Warnungen, die dem Prozess des Dateitransfers gefährden. Dies kann, wie oben angezeigt, den verfügbaren Speicherplatz des empfangenden Gerätes und den Speicherbedarf für die Datei betreffen. Die in schwarzer Schrift zu sehenden Meldungen weisen auf die eventuelle Nicht-Nutzbarkeit

der Datei im aktuellen Kontext hin. Zum Beispiel wird beim Versenden von Bild- und Videodateien auf eine zu geringe Display-Auflösung des mobilen Gerätes aufmerksam gemacht.

6.2 Details zur Implementierung

In diesem Abschnitt werden einige Aspekte zur Implementierung des C-SDM erläutert. Es geht vor allem darum beispielhaft die Benutzung des DEMAC Kontextdienstes im Rahmen der vorgestellten Anwendung zu zeigen. Dies schließt unter anderem die Initialisierung des lokalen Kontextmodells, die Konfigurationen der Managementkomponenten und das Starten des Kontextdienstes ein.
Die Dienste, die als Kontextquelle dienen, um das Modell mit rohe Kontextdaten zu füllen (`RawDataService`, vgl. Abschnitt 5.2.1), werden aufgrund (noch) nicht vorhandener Sensoren/Dienste simuliert. Das bedeutet, dass die Daten, die aus einem `RawDataService` stammen, nicht die tatsächlichen Kontextinformationen wiedergeben, sondern lediglich mit Zufallswerten belegt sind.

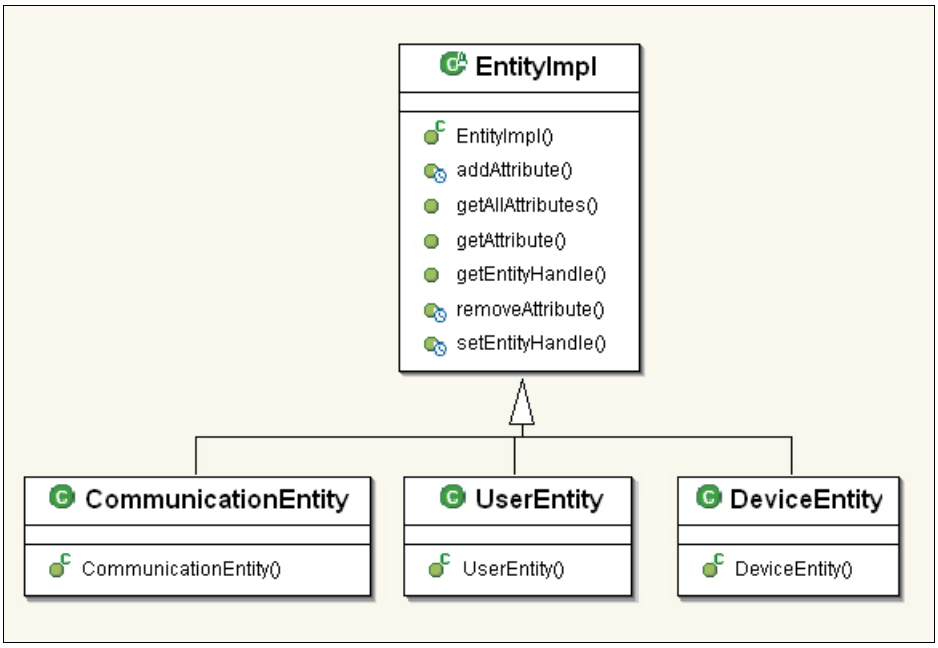


Abbildung 6.5: Modellierte Entitäten im C-SDM

Bevor der `ContextService` verwendet werden kann, muss ein Kontextmodell erzeugt und dem Dienst zur Verwaltung übergeben werden (siehe Schnittstellen des `ContextService` im Abschnitt 5.1.2). Folglich müssen zunächst die Modellkomponenten wie beispielsweise Domänen, Entitäten und Attribute implementiert werden. Das UML-Klassendiagramm in Abbildung 6.5 zeigt die verwendeten Entitäten. Die abstrakte Klasse `EntityImpl` implementiert alle spezifizierten Methoden des Interfaces `Entity` (vgl. Abschnitt 5.2.2). Drei konkreten Klassen stellen alle Entitäten dar, die in der Anwendung benutzt werden. Eine `UserEntity` modelliert Informationen zum Zustand eines Benutzers. Die `DeviceEntity` stellt ein Objekt dar, das die Eigenschaften des mobilen Gerätes wiedergibt. Die kommunikationstechnischen Informationen im aktuellen Kontext liefert eine `CommunicationEntity`. Dazu zählen vor allem Art und Bandbreite einer Verbindung.

Aufgrund der geringen Anzahl der modellierten Entitäten existiert eine einzige Domäne, die alle drei Entitäten aggegiert. Die `DefaultDomain` implementiert Analog zur `EntityImpl` die Methoden der Schnittstelle `Domain` und ist während der Laufzeit der Anwendung stets die einzige aktive Domäne.

Während die Entitätsebene des Kontextmodells in hohem Grad anwendungsspezifisch ist, haben die Komponenten auf der Attributsebene einen generischen Charakter. Im engeren Sinn bedeutet das, dass Attribute (anwendungsunabhängig) für verschiedene Entitäten benutzt werden können, während konkrete Entitäten (und Domänen, in denen sie auftreten) nur solche Objekte repräsentieren, die für eine bestimmte Anwendung relevant sind. Aus diesem Grund können viele Attributsklassen als Teil der Middleware realisiert werden und damit den Aufwand für den Anwendungsentwickler zusätzlich minimieren. Nichtsdestotrotz bleibt die Erweiterungsfähigkeit des Kontextmodells bezüglich der enthaltenen Attribute (außerhalb der Middleware) erhalten.

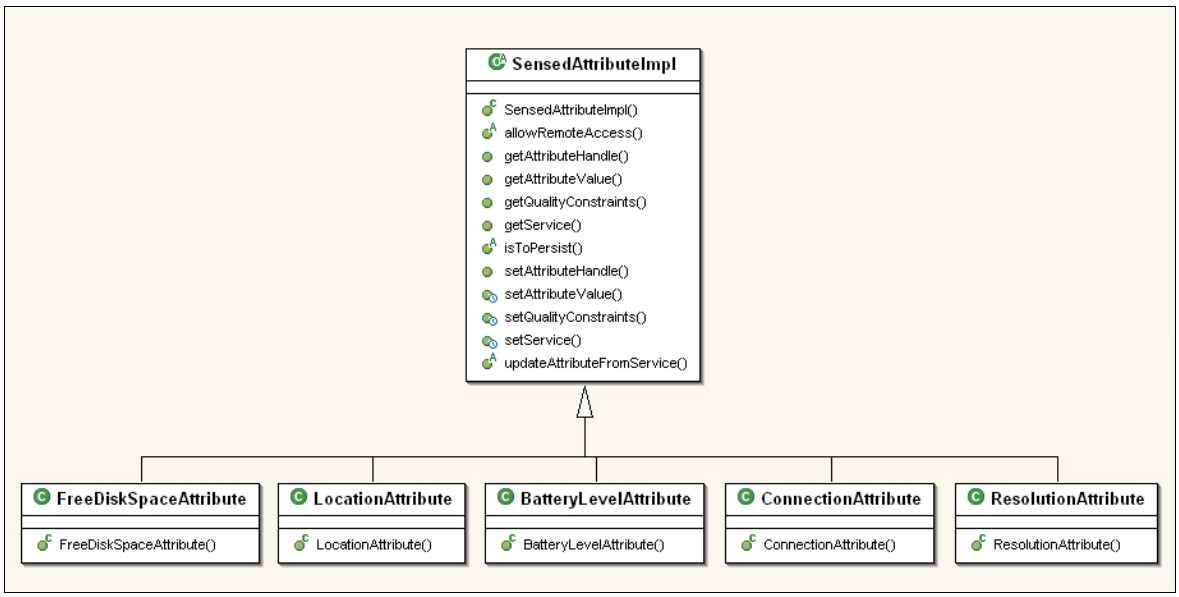


Abbildung 6.6: UML-Klassendiagramm zu den dynamischen Attributen

Alle Informationen, die durch Attributsobjekte gekapselt werden, sind im wesentlichen in der Anwendungsmaske zu den detaillierten Kontextinformationen zu sehen (siehe Abbildung 6.2, rechts). Die Abbildung 6.6 zeigt die verwendeten dynamischen Attribute. Sie erben alle die abstrakte Klasse `SensedAttributeImpl`. Jedes Attribut hat eine Referenz auf einen `RawDataService`, die als Kontextquelle dient. Um zu bestimmen wie die rohen Kontextdaten auf ein Attribut abgebildet werden, müssen alle konkreten dynamischen Attributen die abstrakte Methode `updateAttributeFromService` des `SensedAttributeImpl` implementieren. Der Quelltext 6.1 zeigt beispielhaft die Implementierung der Methode in der Klasse `ResolutionAttribute`.

```
public void updateAttributeFromService() {
	// call the RawDataService and get the current resolution
	DevicePropertiesService service = (DevicePropertiesService) getService();
	long[] resolution = service.getDisplayResolutionAsPixel();
	// create an AttributeValue-Object
	ResolutionValue value = new ResolutionValue();
	value.setWidth(resolution[0]);
	value.setHeight(resolution[1]);
	value.setUnit("pixel");
	// set the ResolutionValue as current attribute value
	setAttributeValue(value);
}
```

Quelltext 6.1: Aktualisierung eines Attributwertes

Nach dem alle Modellklassen implementiert sind, können die entsprechenden Objekte erzeugt werden. Der Quelltext 6.2 zeigt einen Auszug aus dem Initialisierungsprozess des C-SDM. Dabei werden zunächst alle Objekte des Kontextmodells instanziert und anschließend die Objekthierarchie sukzessiv aufgebaut. Das oberste Objekt in der Modellhierarchie bildet ein `LocalDeviceContext`. Dieser wird auch am Ende des Initialisierungsblocks dem `ContextService` übergeben.

```
// get the context service
ContextService contextService = ContextServiceImpl.getService();

// create local context
LocalDeviceContext localContext = new LocalDeviceContextImpl();
localContext.setDeviceHandle(transportService.getLocalDeviceHandle());

// create domain context
DomainContext domainContext = new DefaultDomain();
// create entities
Entity deviceEntity = new DeviceEntity();
...
// create attributes
SensedAttribute resolution = new ResolutionAttribute();
resolution.setService(serviceRegistry.getRowDataService(
	DevicePropertiesServiceImpl.HANDLE));
...
// put all model objects together and set it as local context
deviceEntity.addAttribute(resolution);
...
domainContext.addEntityToDomain(deviceEntity);
localContext.addDomainContext(domainContext);
contextService.initLocalContext(localContext);
```

Quelltext 6.2: Initialisierung des lokalen Kontextmodells

Damit der `ContextService` endgültig gestartet werden kann, müssen noch die benötigten Management-Komponenten instanziert und registriert werden. Im Quelltext 6.3 werden zunächst die Management-Dienste erzeugt und anschließend dem `ContextService` hinzugefügt. Innerhalb der Methode `start` werden dann die hinzugefügten Management-Dienste ebenfalls gestartet.

```
// create management services
UpdateService updateService = new UpdateServiceImpl(localContext);
NotificationService notifyService = new NotificationServiceImpl(localContext);
TransformationService transformService = new TransformationServiceImpl();
SerializerService serialService = new XMLSerializerService(localContext);

RemoteContextService remoteService = new RemoteContextServiceImpl();
remoteService.setContextSerializer(serialService);
remoteService.setLocalDeviceContext(localContext);

// add management service to context service
contextService.addManagementSevice(updateService);
contextService.addManagementSevice(notifyService);
contextService.addManagementSevice(transformService);
contextService.addManagementSevice(serialService);
contextService.addManagementSevice(remoteService);

// finally start the context service
contextService.start();
```

Quelltext 6.3: Initialisierung der Management-Dienste

7 Zusammenfassung und Ausblick

Die Untersuchungen im Rahmen dieser Arbeit haben gezeigt, dass die Nutzung von kontextsensitiven Informationen innerhalb mobiler Anwendungen eine ganz neue Art von innovativer Software hervorbringt, die sich von den bisherigen traditionellen Desktop-Anwendungen unterscheidet. Kontexbezogene mobile Anwendungen können nicht nur den mobilen Benutzer sinnvolle Informationen über die aktuelle Umgebung liefern, sondern mit Hilfe von Kontextinformationen auch ihr Verhalten adaptiv steuern. Der schnelle Fortschritt im Bereich von leistungsfähigen mobilen Geräten, der drahtlosen Kommunikation und der Sensortechnik erleichtern immer mehr die Anbindung von Context-Awareness im Umfeld des Mobile Computing [RBB03].

Um die Entwicklung von mobilen Anwendungen im Allgemeinen effektiv zu unterstützten wird eine Infrastruktur benötigt, die wiederkehrende Aufgaben, die bei der Entwicklung von mobilen Anwendungen auftreten, übernimmt. Im Rahmen des DEMAC-Projektes wird an der Konzeption und Entwicklung einer solchen Infrastruktur für mobile Systeme gearbeitet. Aufgrund immer leistungsfähigeren mobilen Endgeräten (PDAs und Smartphones) ist der Einsatz einer leistungsfähigen Middleware unter Berücksichtigung der existierenden Randbedingungen im Mobile Computing inzwischen möglich.

Auch die Modellierung, Verwaltung und Bereitstellung von kontextsensitiven Daten können als sinnvolle Aufgaben für eine Middleware für mobile Systeme angesehen werden. Verschiedene Anwendungen, die auf der Middleware aufsetzen, können so über einheitliche Schnittstellen Kontextinformationen abrufen und nutzen. Zur Bewerkstelligung der oben genannten Aufgaben wurde im Zuge dieser Arbeit ein Kontextdienst und Kontextmodell entwickelt, die den mobilen Anwendungen auf einheitlicher Weise Zugang zu den Kontextinformationen des System ermöglicht.

Eine besondere Herausforderung bei der Integration von Context-Awareness in eine Middleware für mobile Systeme ist die Erfassung des Kontexts mit Hilfe eines generischen und erweiterbaren Datenmodells. Denn ein Kontext wird durch eine Vielzahl von sehr heterogenen Einzeldaten beschrieben. Diese in einem abstrakten Modell zu integrieren, die nicht auf bestimmte Typen von Kontextinformationen beschränkt ist, erfordert methodisches Vorgehen. Zudem muss das Modell so entworfen sein, dass es nicht von einer bestimmten Art der Anwendung abhängig ist, da eine Middleware von verschiedenen Anwendungen genutzt werden kann.

Obwohl die mobilen Geräte immer leistungsfähiger werden, bestehen weiterhin einige Randbedingungen im Mobile Computing (vgl. Abschnitt 2.1.5), die beim Entwurf des Kontextdienstes einen starken Einfluss ausüben. Der Dienst ist daher so konzipiert, dass eine strenge Trennung zwischen dem Kontextmodell und dem Kontext-Management besteht. Diese Modularisierung ist insofern sinnvoll als damit die Unabhängigkeit, Skalierbarkeit und Erweiterbarkeit beider Komponenten erreicht werden kann. Insbesondere ist die anwendungsabhängige Skalierung des Kontextsystems außerhalb der Middleware entscheidend für einen ressourcenarmen Betrieb des Kontextdienstes.

Ausblickend lässt sich feststellen, dass eine kontextsensitiven Middleware meistens erst dann zum Einsatz kommen kann, wenn die Soft- und Hardwareschicht unterhalb der Middleware entsprechende Quellen für Kontextinformationen zur Verfügung stellen. Die Kontextquellen in Form von physikalischen und logischen Sensoren (vgl. Abschnitt 2.4.2) sind in heutigen mobile Geräte wie PDAs oder Smartphones nur rudimentär integriert. Auch die Allgegen-

wärtigkeit von Diensten im Sinne des Pervasive Computing sind zur Zeit in der Industrie entweder kaum verbreitet oder mit hohen Kosten verbunden [RBB03].
Da Kontextinformationen sehr persönliche Informationen repräsentieren können, müssen sie insbesondere in einer verteilten mobilen Umgebung im Rahmen der Middleware durch entsprechende Maßnahmen gegenüber unerlaubten oder unerwünschten Zugriff geschützt werden. Zum Schutz der Privatsphäre des Benutzers und aus Sicherheitsgründen muss ein *SecurityPrivacyService*, die im Rahmen dieser Arbeit als Kontextsicherheitsdienst vorgesehen ist (vgl. Abschnitt 4.3.2), entsprechende Funktionen für die Anwendungsentwickler bereitstellen. Da diese Diplomarbeit sich in erste Linie mit der Modellierung und Präsentation von Kontext beschäftigt, ist die Entwicklung eines ausführlichen Sicherheitskonzepts innerhalb einer kontextsensitiven Middleware bewusst außen vor gelassen. Zukünftige Arbeiten könnten aufbauend auf dem entwickelten Kontextinfrastruktur ein Sicherheitskonzept entwerfen, die insbesondere beim Austausch der modellierten Kontextinformationen sicherheitstechnischen Schutz bietet.

Literaturverzeichnis

[AAHL+97] Abowd G.D., Atkeson C.G., Hong J., Long S., Kooper R., Pinkerton M.: *Cyberguide: A mobile context-aware tour guide*
Wireless Networks 3(5): 421-433 (1997)
http://citeseer.ist.psu.edu/abowd97cyberguide.html (Abruf: 18.06.2006)

[Bar05a] Bardram J.E.: *The Java Context Awareness Framework (JCAF) - A Service Infrastructure and Programming Framework for Context Aware Applications*
In Pervasive, 2005
http://www.daimi.au.dk/~bardram/jcaf/jcaf.v15.pdf (Abruf: 18.06.2006)

[Bar05b] Bardram J.E.: *Tutorial for the Java Context Awareness Framework (JCAF), version 1.5*
Centre for Pervasive Healthcare Department of Computer Science, University of Aarhus Denmark, 2005
http://www.daimi.au.dk/~bardram/jcaf/jcaf.tutorial.v15.pdf (Abruf: 18.06.2006)

[BBC97] Brown P.J., Bovey J.D., Chen X.: *Context-Aware Applications: From the Laboratory to the Marketplace*
IEEE Personal Communications, 4(5) 58-64. 1997

[BGHS05] Book M., Gruhn V., Hülder M., Schäfer C.: *Der Einfluss verschiedener Mobilitätsgrade auf die Architektur von Informationssystemen*
5. Konferenz Mobile Commerce Technologien und Anwendungen (MCTA2005), Augsburg, Germany 2005
http://ebus.informatik.uni-leipzig.de/papers/paperuploads/Der_Einfluss_verschiedener_Mobilitaetsgrade_auf_die_Architektur_von_InformationssystemenMatthias_Book__Volker_Gruhn__Malte_Huelder__Clemens_Schaefer18344.pdf (Abruf: 18.06.2006)

[BGS98] Beigl M., Gellerson H.-W., Schmidt A.: *There is more to Context than Location*: *Environment Sensing Technologies for Adaptive Mobile User Interfaces*
In Proccedings of the International Workshop on Interactive Applications of Mobile Computing (IMC98), Rostock, Germany, November 1998
http://www.comp.lancs.ac.uk/~albrecht/pubs/pdf/schmidt_cug_elsevier_12-1999-context-is-more-than-location.pdf (Abruf: 18.06.2006)

[BHS05] Bisgaard J., Heise M., Steffensen C.: *How is Context and Context-awareness Defined and Applied? A Survey of Context-awareness*
Department of Computer Science, Aalborg University, 2005
http://www.csconsult.dk/rap/inf7_con.pdf (Abruf: 18.06.2006)

[BKS03] Buchholz T., Küpper A., Schiffers M.: *Quality of Context: What it is and why we need it*
In Proceedings of the Workshop of the HP OpenView University Association 2003 (HPOVUA 2003), Genevy, 2003
http://www.axel-kuepper.de/publications/pub29.html (Abruf: 18.06.2006)

[BKW02] Baus J., Krüger A., Wahlster W.: *A resource-adaptive mobile navigation system*
In Proccedings of International Conference on Intelligent User Interfaces
San Francisco, ACM Press (2002)
http://www.dfki.de/~krueger/PDF/iui2002.pdf (Abruf: 18.06.2006)

[Bro93] *Brockhaus Enzyklopädie*
19.Auflage
F.A. Brockhaus Mannheim, 1993

[Bro05] Broll G.: *Design und Implementierung eines personalisierten Medienportals für mobile Endgeräte*
LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN, Institut für Informatik, 2005
http://www.mobile.ifi.lmu.de/common/Literatur/MNMPub/Fopras/brol05/PDF-Version/brol05.pdf (Abruf: 18.06.2006)

[Bro96] Brown, P.J.: *The stick-e document: a Framework for Creating Context-aware Applications*
In Proceedings of Electronic Publishing, 1996, 259-272.
http://cajun.cs.nott.ac.uk/compsci/epo/papers/volume8/issue2/2point1.pdf (Abruf: 05.05.2006)

[Cdc06] Java Community Process: *JSR-36 Connected Device Configuration 1.0*
http://jcp.org/aboutJava/communityprocess/mrel/jsr036/ (Abruf: 18.06.2006)

[CFJ03] Chen H., Finin T., Joshi A.: *An Intelligent Broker for Context-Aware Systems*
In Adjunct Proceedings of Ubicomp, 2003
http://www.csee.umbc.edu/~hchen4/paper/hc-ubicomp03-poster.pdf (Abruf: 18.06.2006)

[Che04] Chen G.: *Solar: Building a Context Fusion Network for Pervasive*
Doktorarbeit, 2004
http://cmc.cs.dartmouth.edu/papers/chen:thesis.pdf (Abruf: 18.06.2006)

[Che04b] Chen H.: *An Intelligent Broker Architecture for Pervasive Context-Aware Systems*
PhD thesis, University of Maryland, Baltimore County, 2004
http://citeseer.ist.psu.edu/chen03intelligent.html (Abruf: 18.06.2006)

[ChKo00] Chen G., Kotz D.: *A survey of context-aware mobile computing research*
Technical Report TR2000-381, 2000
http://elans.cse.msu.edu/ni/restrict/ChenKotz2000.pdf (Abruf: 18.06.2006)

[DeAb99] Dey A.K., Abowd G.: *Towards a Better Understanding of Context and Context-Awareness*
Workshop on The What, Who, Where, When, and How of Context-Awareness, Conference on Human Factors in Computer Systems. 2000
http://www.csse.monash.edu.au/courseware/cse5610/Students-only/Readings/dey-abowd-99.pdf (Abruf: 18.06.2006)

[DEMAC] *DEMAC-Projekt Homepage*:
http://vsis-www.informatik.uni-hamburg.de/projects/demac/ (Abruf: 18.06.2006)

[Dey01] Dey, A.K.: *Understanding and Using Context*
Personal Ubiquitous Computing, Vol. 5, No. 1. (February 2001), pp. 4-7
http://www.cc.gatech.edu/fce/ctk/pubs/PeTe5-1.pdf (Abruf: 18.06.2006)

[DiHe95] Diehl N., Held A.: *Mobile Computing: Systeme, Kommunikation, Anwendungen* In International Thomson Publishing, 1995

[DNHB+04] Dürr F., Hönle N., Nicklas D., Becker C., Rothermel K.: *Nexus - A Platform for Context-Aware Applications*
In Jörg Roth (ed.): 1. Fachgespräch Ortsbezogene Anwendungen und Dienste der GI
ftp://ftp.informatik.uni-stuttgart.de/pub/library/ncstrl.ustuttgart_fi/INPROC-2004-29/INPROC-2004-29.pdf (Abruf: 18.06.2006)

[Dol03] Dolke S.: *Entwicklung eines mobilen Clients für digitale Bibliotheken*
Universität Rostock, Studienarbeit 2003
http://e-lib.informatik.uni-rostock.de/fulltext/2003/pre-diploma/DolkeSebastian-2003.pdf (Abruf: 19.04.2006)

[DSAF99] Dey K.A., Salber D., Abowd G.D., Futakawa M.: *The Conference Assistant: Combining Context-Awareness with Wearable Computing*
In Proceedings of the third International Symposium on Wearable Computers (ISWC), 1999
http://www.cc.gatech.edu/fce/ctk/pubs/ISWC99.pdf (Abruf: 18.06.2006)

[Dud97] *Duden: Das Fremdwörterbuch*
6. überarbeitete und erweiterte Auflage, 1997

[Fuc02] Fuchs F.: *Kontextsensitive Dienste*
Technische Universität München, 2002
http://www.mobile.ifi.lmu.de/Hauptseminare/ws0102/handouts/fuchs_doc.pdf (Abruf: 18.06.2006)

[Har04] Harbeck M.: *BDI-Agentensysteme auf mobilen Geräten*
Universität Hamburg, Diplomarbeit, 1994
http://vsis-www.informatik.uni-hamburg.de/getDoc.php/thesis/162/harbeck_da.pdf (Abruf: 18.06.2006)

[HBS02] Held A., Buchholz S., Schill A.: *Modeling of context information for pervasive computing applications*
In Proceedings of SCI 2002 / ISAS 2002 (2002)
http://www.rn.inf.tu-dresden.de/scripts_lsrn/veroeffent_print/SCI2002-paper512JH.pdf (Abruf: 18.06.2006)

[HePi98] Healey J, Picard R. W.: *StartleCam: A Cybernetic Wearable Camera*
Second International Symposium on Wearable Computers (ISWC) 1998
http://citeseer.ist.psu.edu/healey98startlecam.html (Abruf: 18.06.2006)

[HHSW+99] Harter A., Hopper A., Steggles P., Ward A., Webster P.: *The anatomy of a context-aware application*
In Proceedings of the fifth annual International Conference on Mobile Computing and Networking, Seattle, WA, ACM Press, pp 59-68, 1999
http://citeseer.ist.psu.edu/harter99anatomy.html (Abruf: 18.06.2006)

[HMNS03] Hansmann U. , Merk L., Nicklous M. S. , Stober T.: *Pervasive Computing*
Springer Verlag, 2003

[HNB97] Hull R., Neaves P., Bedford-Roberts J.: *Towards Situated Computing*
HP Interaction Technology Departement, 1997
http://www.hpl.hp.com/techreports/97/HPL-97-66.pdf (Abruf: 18.06.2006)

[Hob02] Hobbs J. A.: *A DAML Ontology of Time*
2002
http://www.cs.rochester.edu/~ferguson/daml/20020830/daml-time-20020830.xml (Abruf: 18.06.2006)

[HSPL03] Hofer T., Schwinger W., Pichler M., Leonhartsberger G., Altmann J.: *Context-Awareness on Mobile Devices - the Hydrogen Approach*
In Proceeding of the International Hawaiian Conference on System Science, 2003
http://www.hicss.hawaii.edu/HICSS36/HICSSpapers/STMDI02.pdf
(Abruf: 18.06.2006)

[HSS04] Hallberg J., Svensson S., and Synnes K: *Awareness: A survey of research on awareness of context and situation*
Lulea University of Technology, Sweden 2004
http://media.csee.ltu.se/~sarsve/research/Survey_Context_Awareness.pdf
(Abruf: 18.06.2006)

[IRRH03] Indulska J., Robinsona R., Rakotonirainy A., Henricksen K.: *Experiences in using cc/pp in context-aware systems*
In Proceedings of the 4th International Conference on Mobile Data Management, 2003
http://media.csee.ltu.se/~qwazi/reading/papers/ccpp.pdf (Abruf: 18.06.2006)

[ISO99] ISO 13407: *Human centred design processes for interactive systems*
http://www.usabilitynet.org/tools/13407stds.htm (Abruf: 18.06.2006)

[J2me06] Sun Developer Network: *Java ME Documentations*
http://java.sun.com/javame/reference/docs/index.html (Abruf: 18.06.2006)

[KiBa00] Kindberg T., Barton J.: *A Web-Based Nomadic Computing System*
Computer Networks, 2001
http://www.hpl.hp.com/techreports/2000/HPL-2000-110.pdf (Abruf: 02.04.2006)
(CoolTown-Projekt: http://www.cooltown.com) (Abruf: 02.04.2006)

[Kle96] Kleinrock, L.: *Nomadicity: Anytime, anywhere in a disconnected world*
Mobile Networks and Applications 1, 1996
http://www.lk.cs.ucla.edu/PS/paper203.pdf (Abruf: 18.06.2006)

[Krü02] Krüger G.: *Handbuch der Java-Programmierung*
Addison-Wesley, 2002
http://www.javabuch.de (Abruf: 18.06.2006)

[Kun05] Kunze, C. P.: *DEMAC: A Distributed Environment for Mobility Aware Computing*
In Proceedings of the Doctoral Colloquium of PERVASIVE 2005
http://www.pervasive.ifi.lmu.de/adjunct-proceedings/doctoral-colloquium/p115-121.pdf (Abruf: 18.06.2006)

[LiSe00] Lieberman H., Selker T.: *Out of Context: Computer Systems That Adapt To, and Learn From, Context*
IBM Systems Journal 39, 2000
http://www.research.ibm.com/journal/sj/393/part1/lieberman.pdf

(Abruf: 18.06.2006)

[LKAA96] Long S., Kooper R., Abowd G.D., Atkeson C.G.: *Rapid prototyping of mobile context-aware applications: the Cyberguide case study*.
In Proceedings of the second annual International Conference on Mobile Computing and Networking, White Plains, NY, ACM Press, 1996
http://www.cc.gatech.edu/fce/cyberguide/pubs/mobicom96-cyberguide.ps
(Abruf: 18.06.2006)

[McBu97] McCarthy J., Buvac S.: *Formalizing context (expanded notes)*
In Working Papers of the AAAI Fall Symposium on Context in Knowledge Representation and Natural Language, (99–135)
American Association for Artificial Intelligence (1997)
http://www.nbu.bg/cogs/personal/kokinov/COG507/Formalizing%20context.pdf
(Abruf: 18.06.2006)

[MPR01] Murphy A. L., Picco G. P., Roman G.-C.: *LIME: A Middleware for Physical and Logical Mobility*
In Proceedings of the 21st International Conference on Distributed Computer Systems, 2001
http://www.inf.unisi.ch/faculty/murphy/Papers/icdcs01.pdf
(Abruf: 18.06.2006)

[Mül02] Müller-Wilken S.: *Mobile Geräte in verteilten Anwendungsumgebungen: Ein Integrationsansatz zwischen Abstraktion und Migration*
Universität Hamburg, Dissertation, 2002
http://vsis-www.informatik.uni-hamburg.de/getDoc.php/thesis/312/Diss_M%FCller-Wilken.pdf
(Abruf: 18.06.2006)

[ÖzAa97] Ötzturk P., Aamodt A.: *Towards a model of context for case-based diagnostic problem solving*
In Proceedings of the interdisciplinary conference on modeling and using context, 1997
http://www-poleia.lip6.fr/~brezil/Pages2/Publications/CONTEXT-97/18/paper.ps
(Abruf: 18.06.2006)

[Pan00] Pandya R.: *Mobile and personal communication systems and services*
IEEE Press, 2000

[Pas97] Pascoe J.: *The stick-e note architecture: Extending the interface beyond the user*
In Proceedings of the 1997 International Conference on Intelligent User Interfaces, 1997
http://www.cs.kent.ac.uk/pubs/1997/337/content.ps (Abruf: 18.06.2006)

[Pas98] Pascoe J.: *Adding generic contextual capabilities to wearable computers*
In Proceedings of the Second International Symposium on Wearable Computers, IEEE Computer Society Press, 1997

[Per06] Java Community Process: *JSR-62 Personal Profile Specification*
http://jcp.org/aboutJava/communityprocess/final/jsr062/ (Abruf: 18.06.2006)

[RBB03] Rothermel K.; Bauer M.; Becker C.: *Digitale Weltmodelle - Grundlage kontextbezogener Systeme*
Friedemann Mattern: Total vernetzt - Szenarien einer informatisierten Welt Springer-Verlag, 2003

[Rot02] Roth, J.: *Mobile Computing: Grundlagen, Technik, Konzepte*
dpunkt-Verlag, 2002

[RPM97] Ryan N., Pascoe J., Morse D.: *Enhanced Reality Fieldwork: the Context-Aware Archaeological Assistant*
S. (eds.) Computer Applications in Archaeology, 1997

[RuSi03] Rupp S., Siegmund G.: *Java in der Telekommunikation - Grundlagen, Konzepte, Anwendungen*
dpunkt-Verlag, 2003

[Rya99] Ryan N: *ConteXtML: Exchanging Contextual Information between a Mobile Client and the FieldNote Server*
1999
http://www.cs.kent.ac.uk/projects/mobicomp/fnc/ConteXtML.html
(Abruf: 18.06.2006)

[SAGT+93] Schilit B., Adams N., Gold R., Tso M., Want R.. *The PARCTAB mobile computing system*
In Proceedings Fourth Workshop on Workstation Operating Systems , 1993
http://www.ubiq.com/parctab (Abruf: 18.06.2006)

[Sat96] Satyanarayanan M.: *Fundamental Challenges in Mobile Computing*
In Proceedings of the Fifteenth ACM Symposium on Principles of Distributed Computing, ACM, 1996
http://citeseer.ist.psu.edu/satyanarayanan96fundamental.html
(Abruf: 18.06.2006)

[SATT+99] Schmidt A., Aidoo K. A., Takaluoma A., Tuomela U., Van Laerhoven K., Van de Velde W.: *Advanced Interaction in Context*
First International Symposium, HUC '99, Springer Verlag, 1999
http://www.comp.lancs.ac.uk/~albrecht/pubs/pdf/schmidt_huc99_advanced_interaction_context.pdf (Abruf: 18.06.2006)

[SAW94] Schilit B., Adams N., Want R.: *Context-Aware Computing Applications*
In Proceedings of the 1st International Workshop on Mobile Computing Systems and Applications, 1994
http://seattleweb.intel-research.net/people/schilit/wmc-94-schilit.pdf
(Abruf: 18.06.2006)

[ScGe01] Schmidt A., Gellersen H.: *Nutzung von Kontext in ubiqutären Informationssystemen*
Context-Awareness in Ubiquitous Computing, TecO, Universität Karlsruhe, 2001
http://www.comp.lancs.ac.uk/~albrecht/pubs/pdf/gellersen_it_ti_02-2001.pdf
(Abruf: 18.06.2006)

[Sch00] Schiller J.: *Mobilkommunikation – Techniken für das allgegenwärtige Internet*
Networking and Communication Series.
Addison–Wesley, München, Deutschland, 2000.

[ScTh94] Schilit B., Theimer M.: *Disseminating Active Map Information to Mobile Hosts*
IEEE Network, 1994

[SDA99] Salber D., Dey A. K., Abowd G. D.: *The Context Toolkit: Aiding the Development of Context-Enabled Applications*
In Proceeding of the CHI 99 Conference on Human Factors in Computing Systems, Pittsburgh, 1999
http://citeseer.ist.psu.edu/salber99context.html (Abruf: 18.06.2006)

[STLP04] Strang T., Linnhoff-Popien C.: *A context modeling survey*
First International Workshop on Advanced Context Modelling, Reasoning And Management, UbiComp 2004
http://citeseer.ist.psu.edu/strang04context.html (Abruf: 18.06.2006)

[STM02] Schmidt A., Takaluoma A., Mäntyjärvi J.: *Context-Aware Telephony over WAP*
Handheld and Ubiquitous Computing (HUC 2000), 2000
http://www.comp.lancs.ac.uk/~albrecht/pubs/pdf/schmidt_pete_4-2000.pdf
(Abruf: 18.06.2006)

[TEA06] TEA Technology for Enabling Awareness. European Esprit Project 26900, 2006 http://tea.starlab.org (Abruf: 18.06.2006)

[W3C05] W3C: Composite Capabilities / Preferences Profile (CC/PP) http://www.w3.org/Mobile/CCPP (Abruf: 18.06.2006)

[W3CS01] *W3C: Semantic Web* http://www.w3.org/2001/sw (Abruf: 18.06.2006)

[Web05] *Merriam-Webster Online Dictionary* http://www.m-w.com/dictionary/Context (Abruf: 18.06.2006)

[Wei91] Weiser, M.: *The Computer for the 21st Century* Scientific American, 1991 http://www.ubiq.com/hypertext/weiser/SciAmDraft3.html (Abruf: 18.06.2006)

[Wei93] Weiser, M.: *Some Computer Science Issues in Ubiquitous Computing* Scientific American, 1993 http://www.ubiq.com/hypertext/weiser/UbiCACM.html (Abruf: 18.06.2006)

[Wei96] Weiser, M.: *Ubiquitous Computing* http://www.ubiq.com/hypertext/weiser/UbiHome.html (Abruf: 18.06.2006)

[WGPZ04] Wang X. H., Gu T., Pung H. K., Zhang D. Q.: *Ontology Based Context Modeling and Reasoning using OWL* In Proceedings of the 2004 Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS2004) http://www.comp.nus.edu.sg/~gutao/gutao_NUS/CoMoRea2004_gutao.PDF (Abruf: 18.06.2006)

[WHFG92] Want R., Hopper A., Falcao V., Gibbons J.: *The Active Badge Location System* ACM Transactions on Information Systems, 1992 http://sandbox.xerox.com/want/papers/ab-tois-jan92.pdf (Abruf: 18.06.2006)

[WJH97] Ward A., Jones A., Hopper A.: *A New Location Technique for the Active Office* IEEE Personal Communications 4, 1997

[WZGP04] Wang X. H., Zhang D. Q., Gu T., Pung H. K.: *Ontology Based Context Modeling and Reasoning using OWL* In PerCom Workshops, 2004 http://www1.i2r.a-star.edu.sg/~tgu/gutao_NUS/CoMoRea2004_gutao.PDF (Abruf: 18.06.2006)

[Zap05] Zaplata S.: *Prozessintegration in eine Middleware für mobile Systeme* Diplomarbeit, Universität Hamburg, 2005 http://vsis-www.informatik.uni-hamburg.de/getDoc.php/thesis/305/Diplomarbeit_ZA_041005.pdf (Abruf: 18.06.2006)

Abbildungsverzeichnis

Tabellenverzeichnis

Erklärung

Ich versichere, dass ich die vorstehende Arbeit selbstständig und ohne fremde Hilfe angefertigt und mich anderer als der im beigefügten Verzeichnis angegebenen Hilfsmittel nicht bedient habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht.

Außerdem erkläre ich, dass ich mit der Einstellung dieser Diplomarbeit in den Bestand der Bibliotheken der Universität Hamburg einverstanden bin.

Hamburg, den 29.06.2006

Mirwais Turjalei