Universität Hamburg
Fakultät für Mathematik,
Informatik und Naturwissenschaften

Verteilte Systeme und Informationssysteme

Diplomarbeit

Aktivitätsorienterte Kontextadaption für mobile Anwendungen

Hamburg, den 27.7.2009

Jan D.S. Wischweh

mail@wischweh.de
Studiengang Informatik
Matr.-Nr. 5201504

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

Danksagung

Viele unterschiedliche Personen haben dazu beigetragen, dass diese Arbeit in der vorliegenden Form entstehen konnte. Ihnen gebührt mein besonderer Dank. Ich danke

Joachim von der Heydt, für großzügige Unterstützung, die mir erlaubte mich während des letzten halben Jahres voll auf diese Arbeit zu konzentrieren,

dem Arbeitsbereich Verteielte Systeme und Informationssysteme (VSIS) an der Uni Hamburg, insbesondere Winfried Lamerdorf und Dirk Bade, für hervorragende Arbeitsbedingungen und engagierte Betreuung,

Christa, für meine Arbeitswochen im Bonn und viele spannende Gespräche,

Anja, für Hilfe im Umgang mit der deutschen Sprache und langjährige Freundschaft,

Julia, für gemeinsame Marktbesuche mit Wurst und Muffins, ohne welche die Regeneration während anstrengender Phasen kaum gelungen wäre,

allen Studierenden, die trotz zunehmend restriktiver Studienbedingungen, interessengeleitet, kritisch und solidarisch studieren und so dafür sorgen, dass der Universitätsbesuch eine lohnende Erfahrung bleibt.

Inhaltsverzeichnis

1 Einleitung

In diesem Kapitel soll in die vorliegende Arbeit zur „Aktivitätsorientierten Kontextadaption für mobile Anwendungen“ eingeführt werden. Es werden die Gründe dafür geschildert, sich mit diesem Thema zu beschäftigen (Abschnitt 1.1). Darauf aufbauend wird die Fragestellung (Abschnitt 1.2), die Zielsetzung (Abschnitt 1.3) sowie die gewählte Vorgehensweise und Gliederung (Abschnitt 1.4) dieser Arbeit erläutert.

1.1 Motivation

Im Jahr 2008 wurden weltweit mehr als 1.22 Milliarden Mobiltelefone verkauft [Gar09b]. Dabei wächst der Anteil von Smartphones, Telefonen, deren Funktionalität weit über bloßes Telefonieren hinausgeht, stetig [Gar09a]. Den explosionsartig wachsenden Möglichkeiten der Geräte stehen prinzipielle Schwierigkeiten bei deren Bedienung gegenüber: Durch die Mobilität der Geräte und der daraus resultierenden geringen Größe, steht nur wenig Platz für Ein- und Ausgabemöglichkeiten zur Verfügung [LK06].

Eine Strategie, um diese Probleme zu reduzieren, ist das Ausnutzen des jeweiligen Kontextes, in dem Interaktion mit einem mobilen Gerät erfolgt. So könnte ein Gerät, das automatisch erkennt, dass sich der Nutzer in einer Situation in der Störungen unangebracht sind (z.B. in einer Besprechung) befindet, in einen stummen Modus wechseln und den Nutzer nur noch durch ein Vibrationssignal über eingehende Telefonate informieren. Ein Hervorholen des Gerätes und das manuelle Einschalten des Stumm-Modus, bei dem ggf. durch mehrere Menüebenen navigiert werden muss, entfiele. Kontext kann also genutzt werden, um den Umgang mit mobilen Geräten zu vereinfachen.

Im Bereich der Mobiltelefone konzentrierte sich die Forschung zur Nutzung von Kontext bisher vor allem auf die Aspekte der physikalischen Umgebung (Ort [DRD+00], Umweltbedingungen [Sch00] und verfügbare technische Ressourcen [PSGP01]) und zum Teil auf die aktuelle soziale Situation des Nutzers [ORT05]. Aspekte wie Informationen über den Nutzer, seine Aktivitäten und zeitliche Zusammenhänge wurden bisher weniger genutzt [KA04b].

Dabei verspricht die Kenntnis von aktuellen, vergangenen oder zukünftigen Aktivitäten des Nutzers im Verleich zu anderen Bereichen von Kontext besondere Möglichkeiten, die Interaktionsabläufe zwischen Nutzern und mobilen Geräten zu verbessern. Aktivitäten können dabei als zielgerichtete Tätigkeiten verstanden werden, wie z.B. einen Termin vereinbaren oder sich an einen bestimmten Ort bewegen. Diese Aktivitäten lassen sich in einzelne Aktionen zergliedern [KPC06, S.7]. Die Vereinbarung eines Termins besteht beispielsweise aus den Aktionen Telefonieren und eine Eintragung in einen Kalender vornehmen. Bei der Navigation zu einem bestimmten Ort treten die Aktionen „Konsultieren der Karte“, „Inspizieren der Umgebung“ und „eine Teilstrecke zurücklegen“ auf.

Zu den spezifischen Möglichkeiten, die sich aus der Kenntnis von vergangenen, laufenden oder zukünftigen Tätigkeiten ergeben, gehört beispielsweise:

- Durch Aktivitäten werden Beziehungen zwischen Personen, Orten und Objekten hergestellt. Durch Analyse dieser Aktivitäten lässt sich feststellen, welche dieser Entitäten wie miteinander in Verbindung stehen. Dieses lässt sich nutzen um zu bestimmen, welche Personen, Orte oder Objekte in einer bestimmten Situation relevant sein könnten.
- Die Aktivitäten des Nutzer bestimmen, wo gerade sein Aufmerksamkeitsfokus liegt und wie hoch seine aktuelle kognitive Last ist. Diese Kenntnisse sind eine gute Grundlage dafür Elemente der Benutzerschnittstelle dynamisch an die aktuellen Bedürfnisse des Nutzers anzupassen.
- Kenntnisse über übliche Tätigkeitsmuster eines Nutzers lassen sich zur Vorhersage wahrscheinlicher zukünftiger Aktivitäten nutzen.

Diesem Ansatz kommt zugute, dass mit steigendem Funktionsumfang der Geräte und zunehmender Nutzung der Mobiltelefone jenseits des Telefonierens, wie beispielsweise dem Nutzen von *Personal Information Manager Software* (PIM) oder Navigationssystemen, immer mehr Informationen über die Aktivitäten des Nutzers gewonnen werden können.

Aber auch in der durch Mark Weiser geprägten, weiter vorausschauenden Perspektive des *Ubiquitous Computing* spielt Kontext eine große Rolle [Wei91]. Weiser stellte fest, dass Computer immer zahlreicher und tiefer in unseren Alltag eindringen. In der frühen Phase der Computernutzung prägten Großrechner, die von einer Vielzahl von Personen gemeinsam genutzt werden mussten, das Bild. In der zweiten Phase wurde der *Personal Computer* bedeutsam. Das Leitbild war: Ein Computer pro Person. Möglich wurde dies durch eine drastische Miniturisierung und Verbilligung der Bauteile. Diese Miniaturisierung und Verbilligung hält bis heute an und ermöglicht viele kleine Computer pro Person, die immer mehr in alltägliche Gegenstände integriert werden. Dieses stellt die dritte Phase der Computernutzung dar. Der gegenwärtige Stand der Technik entspricht einer Transition von der zweiten in die dritte Phase [WB97].

Diese allgegenwärtigen Computer sollten in der von Mark Weiser als *Ubiqitous Computing* bezeichneten Vision nicht nur physisch immer kleiner werden, sondern trotz ihrer zunehmenden Anzahl immer mehr in den Hintergrund treten. Sie sollten wie selbstverständlich ihre Aufgaben erledigen, ohne dass die Interaktion mit ihnen bewusst wahrgenommen wird. Die Computer sollen *unsichtbar* werden. Weiser formulierte es so:

> „Ubiquitous Computing has its goal in enhancing computer use by making many computers available throughout the physical environment, but making them effectively invisible to the user.“ [Wei93]

Die technologischen Entwicklungsschritte, die zum Erreichen dieses Ziels entscheidend sind, wurden von Strang und Linnhoff-Popien dargestellt (vgl. Abbildung 1.1)

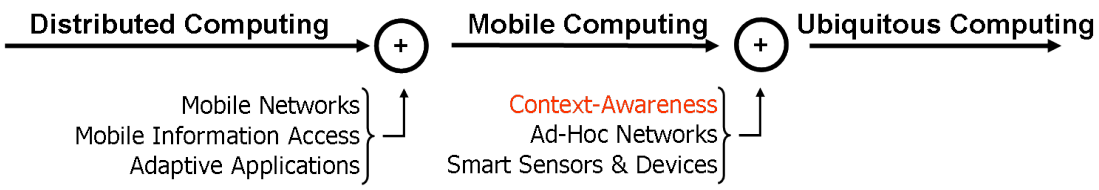

Abbildung 1.1: Entwicklungsschritte zum Ubiquitous Computing [SLP04].

[SLP04]. Mobil verfügbare Netzwerke, mobiler Zugriff auf Informationen und Anwendungen, die sich an technische Bedingungen wie Verbindungsqualität in einem Netzwerk anpassen können, prägen das Bild des *Mobile Computing* heute. Werden diese Möglichkeiten um Geräte angereichert, denen intelligente Sensoren zur Verfügung stehen und die in der Lage sind, spontane Netzwerke zur Kooperation zu bilden sowie ihren Nutzungskontext zu berücksichtigen, lässt sich Ubiquitous Computing realsieren. Die eigenständige Vernetzung der Geräte ist notwendig, damit intelligente Geräte nicht als isolierte Werkzeuge fungieren, sondern Daten austauschen und kooperieren können. Sensoren können explizite Eingaben durch den Nutzer überflüssig machen und tragen somit zum Verschwinden von Schnittstellen bei. Aber erst die Berücksichtigung des Kontexts lässt solche Geräte auch in den verschiedensten Situationen sinnvolles Verhalten an den Tag legen.

Auch für den Bereich des Ubiqitous Computing ist der bisher wenig untersuchte Aktivitätskontexts von besonderem Interesse. Denn letzlich sind es die Aktivitäten und Ziele des Nutzers, die bestimmen, was für die Handlungen eines Nutzers von Belang ist. Oder in den Worten von Paul Dourish formuliert:

> „[...] Context arises from the activity. Context isn't just 'there', but is actively produced, maintained and enacted in the course of the activity at hand. " [Dou04]

Die Berücksichtigung von Aktivitätskontext ist somit kurzfristig von Interesse, um konkrete Nutzungsschwierigkeiten beim Umgang mit mobilen Geräten zu reduzieren, und langfristig, um dem Ziel des Ubiqitous Computing näher zu kommen. Daher befasst sich diese Arbeit damit, wie sich dieser Bereich von Kontext zur Verbesserung der Interaktion mit heutigen mobilen Geräten nutzen lässt und will Perspektiven für die Nutzung von Aktivitätskontext entwickeln.

1.2 Fragestellung

Ausgehend von der Fragestellung „Wie lässt sich der Aktivitätskontext zur Laufzeit erfassen und zur Verbesserung der Interaktion mit mobilen Geräten nutzen?". Diese Frage umfasst folgende Teilfragen:

- Wie können Aktivitäten erfasst und modelliert werden?

- Welche Schlüsse können aus diesen Aktivitäten gezogen werden?
- Wie lassen sich diese Informationen sinnvoll nutzen?

Die vorliegende Arbeit will dabei einen Ansatz entwickeln, der zwar die langfristige Vision des Ubiqitous Computing im Hinterkopf behält, sich aber auf die Umsetzbarkeit für heutige mobile Geräte fokussiert und versucht, die Bedienung dieser Geräte zu vereinfachen.

1.3 Zielsetzung

Ziel der Arbeit soll der Entwurf und die Umsetzung einer Software-Infrastruktur sein, die das Erfassen von Aktivitäten und die Berücksichtigung von Aktivitätskontext in mobilen Anwendungen unterstützt. Zu diesem Zweck ist die Entwicklung eines generischen aktivitätsorientierten Kontextdatenschemas erforderlich. Ein solches Datenschema propagiert eine einheitliche Art und Weise, wie Aktivitäten aufgefasst und dargestellt werden. Es ist die Grundlage für die gemeinsame Nutzung von Kontextinformationen durch unterschiedliche Applikationen und ermöglicht die Unterstützung der Abfrage, Nutzung und des Austausches dieser Informationen mittels einer Middleware. Eine solche Middleware implementiert das entwickelte Konzept, stellt Anwendungen eine Schnittstelle zur Nutzung des Aktivitätskontextes bereit und bietet Möglichkeiten zum Erfassen der auftretenden Nutzeraktivität.

Darauf aufbauend soll eine Anwendung entwickelt werden, die einige Möglichkeiten der Nutzung von Aktivitätskontext exemplarisch aufzeigt und das Zusammenspiel zwischen Anwendung und Middleware illustriert.

1.4 Vorgehensweise

Zunächst soll ein Verständnis dafür entwickelt werden, was Aktivitätskontext ausmacht und wie dieser sich nutzen lässt. Dazu wird in Kapitel 2 ein Beispielszenario entwickelt, das Möglichkeiten zur Nutzung von Aktivitätskontext veranschaulicht und die Aspekte, die beim Umgang mit diesem von Bedeutung sind, aufzeigt.

Zur weiteren Entwicklung des Verständnisses beschäftigen sich die nächsten beiden Kapitel mit den Themen „Aktivitäten“ und „Kontext“ aus theoretischer Sicht. Bezugspunkt für das in dieser Arbeit entwickelte Verständnis von Aktivität ist die *Aktivitätstheorie*. Sie wird in Kapitel 3 kurz umrissen und es werden einige Implikationen der Aktivitätstheorie für diese Arbeit dargestellt. Das darauffolgende Kapitel 4 widmet sich der Frage, was Kontext ausmacht und wie Kontext zur Verbesserung von Interaktion mit mobilen Geräten berücksichtigt werden kann, wobei auch besondere Potentiale von Aktivitätskontext erarbeitet werden.

Die beiden folgenden Kapitel beschäftigen sich mit dem praktischen Forschungsstand zur Modellierung von Kontext (Kapitel 5) und den bisher vorgeschlagenen Software-Infrastrukturen zum Umgang mit Kontext (Kapitel 6). Dabei werden die bisherigen Arbeiten zu diesen Themen insbesondere für ihre Eignung zum Umgang mit Aktivitätskontext untersucht.

Der Ansatz dieser Arbeit wird in den abschließenden Kapiteln dargestellt. Dabei wird zunächst in Kapitel 7 ein generisches Kontextdatenschema für Aktivitätskontext entworfen. Anschließend werden in Kapitel 8 Konzepte für eine Middleware entwickelt, die dieses Schema implementiert und den Umgang mit Aktivitätsdaten vereinfacht, aber auch zum Umgang mit weiteren Kontextinformationen geeignet ist. Die Implementation dieser Middleware und einer Anwendung, die das Beispielszenario aus Kapitel 2, realisiert wird schließlich in Kapitel 9 geschildert.

Den Schluß bildet das Kapitel 10, in dem die Ergebnisse dieser Arbeit noch einmal zusammenfassend dargestellt sind und offene Fragen sowie weiterer Forschungsbedarf hervorgehoben werden.

2 Beispielszenario: Auswahlassistenz

Der Nutzen und die Möglichkeiten im Umgang mit aktivitätsorientiertem Kontext sollen in diesem Kapitel an Beispielen verdeutlichet werden. Das zunächst etwas ausführlicher geschilderte Beispielszenario (Abschnitt 2.1) diente als orientierende Vision bei der Entwicklung dieser Arbeit und bildet die Grundlage für die darauf aufbauend entwickelte Demoanwendung. Der Schilderung des Szenarios folgt eine Analyse (Abschnitt 2.2), die untersucht, welche Aspekte sich aus dem Szenario für die weitere Arbeit ergeben. Um zu verdeutlichen, dass die Nutzungsmöglichkeiten von Aktivitätskontext noch viel breiter sind und auch in ganz andere Richtungen gehen können, werden noch zwei weitere Beispielszenarien (Abschnitt 2.3) gegeben, die im Rahmen dieser Arbeit zwar nicht realisiert, jedoch konzeptionell mitbedacht wurden.

2.1 Szenario: Alice verabredet sich mit Bob

Personal-Information-Managment-Anwendungen (PIM) wie Kalender und Adressbücher gehören heute fest zur Grundausstattung von Mobiltelefonen. Navigationssysteme sind zunehmend Teil moderner Mobiltelefone. Da sich die vorliegende Arbeit an realen Einsatzmöglichkeiten für aktivitätsorientierten Kontext auf heutigen Geräten orientiert, wird ein Beispielszenario aus diesem Bereich als Leifaden für die Entwicklung dieser Arbeit dienen.

Angenommen, Alice telefoniert mit Hilfe ihres Mobiltelefons mit Bob (siehe Abbildung 2.1a). Anschließend öffnet Alice ihren digitalen Terminkalender auf ihrem Mobiltelefon (b) und wählt den Menüpunkt „Neuen Termin eintragen“. In dieser Situation ist es recht wahrscheinlich, dass Alice einen Termin mit Bob in den Kalender eintragen möchte, den Sie soeben verabredet hat. Um den Eingabeaufwand zu verringern, könnte die Kalenderanwendung in der Maske für einen neuen Termineintrag Bob als eine Person vorschlagen, die diesem Termin zugefügt werden kann (c). Wählt Alice nun Bob über diese Schnellauswahl aus, könnte der Kalender Orte für das Treffen vorschlagen, die entweder direkt mit Bob in Verbindung stehen, beispielsweise seinen Arbeitsplatz oder seine Privatadresse oder Orte, an denen sich Alice bereits öfters mit Bob in der Vergangenheit getroffen hat, wie z.B. das Restaurant „Chez Charles“ (d). Hat Alice nun das „Chez Charles“ als Treffpunkt ausgewählt, muss sie nur noch das Datum des Treffens eintragen. Einige Tage später rückt das Treffen näher und Alice möchte zum Restaurant fahren. Sie öffnet das in ihr Mobiltelefon integrierte Navigationssystem. Da das Treffen mit Bob bei „Chez Charles“ für die nächste Stunde registriert ist, kann ihr das Navigationssystem direkt anbieten, eine Route von ihrem aktuellen Standpunkt zum Restaurant zu berechnen und anzuzeigen.

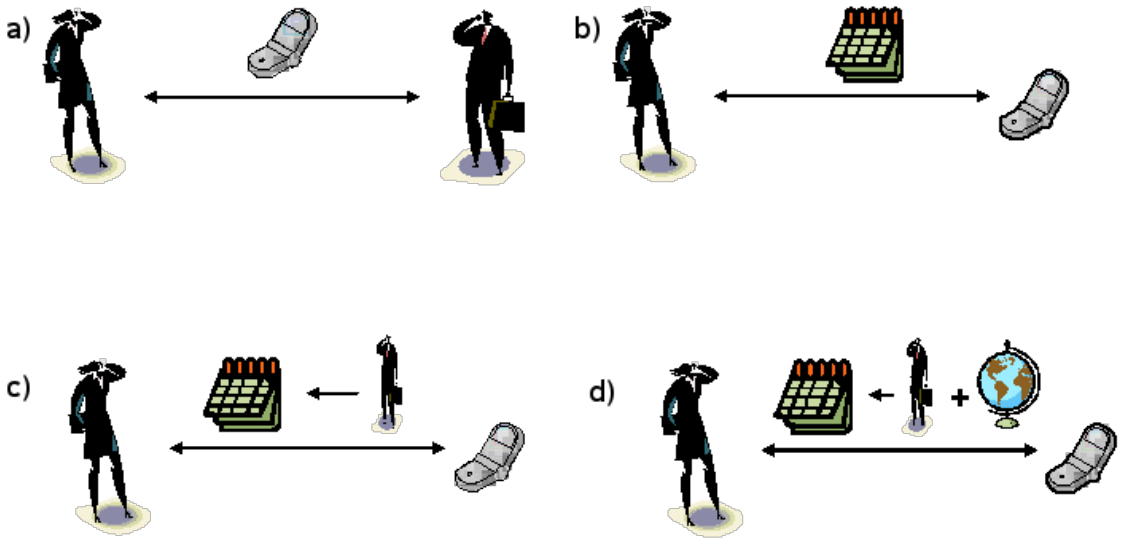

Abbildung 2.1: Beispielszenario - Alice telefoniert mit Bob
Alice telefoniert mit Bob (a). Anschließend öffnet sie Ihren Kalender (b). Es ist wahrscheinlich, dass sie ein Treffen mit Bob eintragen wird (c). Der Treffpunkt ist vermutlich ein Ort, an dem sich beide häufig treffen oder der mit Bob in Verbindung steht (d).

Als weitergehende Vision, ließe sich auch denken, dass nicht alle genutzten Komponenten auf dem Mobiltelefon liegen. Der verwendete Kalender könnte sowohl über das Telefon, als auch über Alice Arbeitsrechner nutzbar sein. Statt des in das Mobiltelefon integrierten Navigationssystems könnte Alice auch das Navigationssystem in ihrem Auto nutzen wollen.

2.2 Analyse des Szenarios

Neben verschiedenen Stellen, an denen aktivitätsorientierte Kontextinformationen zur Verringerung des Eingabeaufwandes genutzt werden können, verdeutlich das in Abschnitt 2.1 geschilderte Szenario einige wichtige Aspekte, die bei der Nutzung des Aktivitätskontextes eine Rolle spielen.

2.2.1 Zeitliche Zusammenhänge

So wird beispielsweise deutlich, dass neben den aktuellen Tätigkeiten der Nutzerin sowohl vergangene wie zukünftige Aktivitäten von Bedeutung sind. Vergangene Tätigkeiten bilden den persistenten Kontext der Anwendung (siehe. 4.1.3.3) und können analysiert werden, um Orte zu bestimmen, an denen sich Alice gerne mit Bob trifft. Informationen über zukünftige bzw. geplante Tätigkeiten können Basis für eine Schätzung sein, welche Orte aktuell für Alice von Belang sein könnten.

2.2.2 Beziehungen ersten und zweiten Grades

In dem geschilderten Szenario werden Beziehungen ersten und zweiten Grades genutzt. Zunächst wird vom Kalender Bob als Person vorgeschlagen. Da dies aufgrund der Tatsache geschieht, dass die Beiden eben erst miteinander telefoniert haben, also direkt miteinander in Verbindung stehen, kann man davon sprechen das hier eine Beziehung ersten Grades genutzt wurde. Wenn das System nun Bobs Arbeitsplatz als Treffpunkt vorschlägt, wird eine Beziehung zweiten Grades ausgenutzt, da Bobs Arbeitsplatz nur indirekt über Bob mit Alice in Verbindung steht. Theoretisch könnten natürlich auch Beziehungen höherer Grade noch genutzt werden, allerdings dürfte dann die Zahl der Entitäten, zu denen Verbindungen gefunden werden exponentiell wachsen, während die Wahrscheinlichkeit, dass eine bestimmte Entität tatsächlich von Belang ist, drastisch sinkt.

Beziehungen zweiten Grades sind jedoch nützlich, um die Arten der Entitäten, die vorgeschlagen werden können zu erweitern. Würden für das Szenario nur Beziehungen ersten Grades benutzt, gäbe es im Beispiel keine Grundlage, einen Ort für das Treffen von Bob und Alice vorzuschlagen.

2.2.3 Einschränkungen der Entitätstypen

Am Beispiel kann verdeutlicht werden, dass bestimmte Anwendungen jeweils nur an Informationen zu bestimmten Entitätstypen interessiert sind. Für einen Kalendereintrag sind Personen und Orte von hoher Wichtigkeit. Es wäre vorstellbar, dass es sinnvoll sein kann, darüber hinaus bestimmte Dokumente mit einem Termin zu verknüpfen. So könnte Alice etwa ein Papier, das mit Bob besprochen werden soll, mit dem Termin verknüpfen wollen. Für die Navigation hingegen sind ausschließlich Orte von Belang. Für einen dritten Fall etwa, z.B. für einen E-Mail-Client, könnten hingegen Personen und Dokumente von Interesse sein. Das genutzte Softwarewerkzeug schränkt also die Entitäten, die für dieses von Interesse sind, anhand ihres Types ein. Die für das Besipielszenario bedeutsamen Entitätstypen sind Personen und Orte.

2.2.4 Verzahnung unterschiedlicher Softwarewerkzeuge

Das Szenario zeigt, wie durch die Nutzung von Kontextinformationen eine Verzahnung von unterschiedlichen Anwendungen entsteht. Dabei potentiert sich der Nutzen von Kontextinformationen je mehr Anwendungen diese nutzen und bereitstellen.

Die Vorschläge der Kalenderanwendung im Szenario basieren auf Informationen über vergangene Telefonate, bisherige Treffen mit Bob und den zu Bob hinterlegten Adressen. Also aus drei verschiedenen Softwarewerkzeugen: eines zum Telefonieren, eines zum Verwalten von Terminen und ein weiteres zum Verwalten von Adressen. Durch den Eintrag eines zukünftigen Termins entstehen neue Kontextinformationen, die durch eine weitere Anwendung, das Navigationssystem, genutzt werden können. Dabei ist es nicht notwendig, dass die einzelnen Anwendungen direkte Kenntnis von einander besit-

zen. Durch eine Middleware, welche die Kontextinformationen vermittelt (vgl. Kapitel 6) können Anwendungen Kontextinformtationen nutzen, die durch andere Anwendungen entstanden sind, auch ohne Kenntnis von deren Existenz oder gar deren Aufbau zu haben.

Je mehr Anwendungen Kontextinformationen bereitstellen und nutzen umso größer wird insgesamt das Potential der Kontextnutzung. So wird im Szenario die Eingabe von Informationen über Kenntnis des Kontexts vereinfacht. Gleichzeitig können die eingegebenen Informationen von verschiedenen Programmen genutzt werden. Wäre die Eingabe des Ortes für das Treffen umständlich, etwa weil Alice eine kleine reduzierte Tastatur verwenden muss um eine vollständige Adresse einzugeben und entstünde daraus kein Nutzen, etwa weil der Zielort im Navigationssystem ohnehin noch einmal eingegeben werden muss und Alice sich notfalls auch an den Treffpunkt erinnern kann, würde sie wahrscheinlich auf die Angabe des Ortes vollständig verzichten.

2.2.5 Informationsfluss in zwei Richtungen

Aus Sicht der Kalenderanwendung wurden Kontextinformationen zur Beantwortung der Fragen „Welche Personen sind aktuell relevant?“ und „Welche Orte stehen in Verbindung mit Bob?“ genutzt. Die Anwendung sollte zusätzlich noch die Orte berücksichtigen, die aktuell relevant für Alice sind, etwa da Alice gerade zu diesen Informationen abgerufen hat. Im Beispielszenario wurde dabei davon ausgegangen, dass es keine solchen Orte gibt.

In umgekehrter Richtung muss die Kalenderanwendung die neu erhaltenen Informationen über das zukünftige Treffen mit Bob bei „Chez Charles“ so zur Verfügung stellen, dass diese von anderen Anwendungen, wie etwa dem Navigationssystem, genutzt werden kann. Es findet nicht nur eine Nutzung von Kontextdaten statt, sondern durch die Verwendung der Softwarewerkzeuge entsteht Kontext, der verfügbar gemacht werden muss.

2.2.6 Verwandschaft zum Ubiqitous Computing

Die Erweiterung des Szenarios, bei der auf unterschiedliche Geräte verteilte Anwendungen den Aktivitätskontext nutzen, rückt dieses Szenario in die Nähe von kontextadaptiven Ubiqitous-Computing-Szenarien und zeigt, wie fließend diese Übergänge sind. Dadurch, dass nicht für eine komplexe monolithische Anwendung Kontextinformationen bereitgestellt und genutzt werden, sondern diese für eine Reihe unterschiedlicher Werkzeuge bereitgehalten werden, sind die Grenzen fließend. Schließlich sind bereits die lokal auf dem Mobiltelefon laufenden Anwendungen unabhängig von einander realisiert.

Ein Ubiqitous-Computing-Szenario zu implementieren, würde den Rahmen dieser Arbeit sprengen. Die aufbauend auf dem einfachen Szenario entwickelten Konzepte sollten sich jedoch weitgehend übertragen lassen.

2.3 Weitere Szenarien

Es lassen sich auch anders gelagerte Szenarien für die Nutzung von Aktivitätskontext finden, die verdeutlichen, dass Aktivitätskontext auf sehr unterschiedliche Art- und Weise genutzt werden kann.

2.3.1 Aktivitätskontextabhängige Hilfe

Ein kontextabhängiges Hilfssystem, wie das in Abschnitt 3.1.3 beschriebene Lumière-System [HBH+98], lässt sich auch auf den Bereich der mobilen Geräten übertragen. Allerdings ist davon auszugehen, dass auf mobilen Geräten weniger eine komplexe monolithische Software genutzt wird, als vielmehr eine Vielzahl ineinandergreifender kleinerer Software-Werkzeuge[1], so dass ein entsprechendes Hilfesystem anwendungsübergreifend realisert werden sollte.

Ein entsprechendes System könnte dann beispielsweise darauf reagieren, wenn ein Nutzer per Textmitteilung eine Telefonnummer erhalten hat und diese Telefonnummer manuell in das Adressbuch des Telefons eingibt. Sollte eine Funktion zum automatischen Übertragen der Nummer existieren, könnte das Hilfesystem den Nutzer auf diese Funktion hinweisen.

2.3.2 Beobachtung des Aufmerksamkeitsfokus'

Durch Beobachtung der Aktivitäten des Nutzers lassen sich Annahmen über seinen Aufmerksamkeitsfokus anstellen. So ist es möglich, erhaltene Nachrichten oder Warnhinweise zu einem geeigneten Zeitpunkt oder auf einem geeigneten Kanal zu signalisieren. Es könnte ein, durch eine Kalenderanwendung versandter Hinweis auf einen bevorstehenden Termin zu einem Zeitpunkt übermittelt werden, an dem der Nutzer nicht durch ein Telefonat oder eine andere Aktivität, die eine hohe Aufmerksamkeit erfordert, belastet ist. Das Navigieren, z.B. mittels eines Navigationssystems, kann eine hohe kognitive Belastung darstellen. Während einer solchen Aktivität könnten Mitteilungen von geringer Priorität über einen unaufdringliches optisches Signal angezeigt werden, während auf wichtige Hinweise mit einem akustischen Signal hingewiesen werden könnte.

2.4 Zusammenfassung

Es wurden anhand dreier unterschiedlicher Szenarien die Nutzungsmöglichkeiten für Aktivitätskontext illustriert. Dabei kann Aktivitätskontext auf unterschiedliche Art und Weise genutzt werden. Im ersten Beispiel, dem Vorschlagen von Personen und Orten für Kalendereinträge, werden die Beziehungen, die durch vergangene und zukünftige Aktivitäten zwischen Entitäten hergestellt werden, für Prognosen der Relevanz von bestimmten Entitäten genutzt. Im zweiten Szenario werden die Aktivitätsabläufe analysiert, um

[1] Neuere Plattformen für mobile Geräte unterstützen eindeutig diesen Trend. Vgl. Abschnitt 9.1.7.1

aufgabenbezogene Hilfe anzubieten und im dritten Beispiel wird die augenblickliche Aktivität herangezogen, um die kognitive Last des Nutzers zu bestimmen.

Die Analyse des ersten Szenarios hat gezeigt, wie Aktivitäten Beziehungen zwischen Entitäten herstellen und wie dies zur Erleichterung bei der Eingabe von Informationen führen kann. Um dies zu realisieren, ist ein Umgang mit und eine Analyse von vergangenen und zukünftigen Ereignissen notwendig. Dies muss so geschehen, dass eine Vielzahl unabhängiger Software-Werkzeuge diese Informationen nicht nur nutzen, sondern auch zu ihnen beitragen können.

3 Aktivitäten

In diesem Abschnitt soll das Verständnis von menschlichen Aktivitäten dargestellt werden, auf das diese Arbeit im weiteren Bezug nimmt. Hierzu werden zunächst einige Perspektiven auf Aktivitäten in der Informatik skizziert (Abschnitt 3.1). Dann wird der für diese Arbeit gewählte theoretische Bezugsrahmen, die in der Psychologie entwickelte Aktivitätstheorie, die auch eine zunehmende Rezeption in der Informatik erfährt, umrissen (Abschnitt 3.2) und einige Konsequenzen dieser Theorie für die vorliegende Arbeit dargestellt (Abschnitt 3.3).

3.1 Aktivitäten in der Informatik

Aktivitäten spielen in der Informatik an vielen Stellen eine große Rolle, denn letztlich geht es der Informatik darum, technische Werkzeuge zu schaffen, die Menschen beim Erfüllen ihrer Aufgaben und Bedürfnisse unterstützen. Einige Bereiche der Informatik, bei denen Aktivitäten eine besondere Rolle spielen, sollen kurz angerissen werden (Abschnitte 3.1.1ff). Anschließend wird der eigene Ansatz dagegen abgegrenzt (Abschnitt 3.1.4).

3.1.1 Software-Ergonomie

Die Software-Ergonomie befasst sich mit der benutzergerechten Gestaltung von Software-Systemen. Der Methodenschatz der Software-Ergonomie besteht im Wesentlichen aus einer Reihe von Gestaltungsrichtlinien, Methoden zur Ermittlung von Gestaltungsanforderungen und zur empirischen Überprüfung der Gebrauchstauglichkeit von Software. Somit ist ein wichtiges Feld die empirische Analyse von Nutzerverhalten in konkreten Software-Systemen durch Experten. Ziel ist es, die Benutzbarkeit von Software zur Erfüllung von bestimmten Aufgaben und Bedürfnissen zu optimieren. In der Software-Ergonomie werden also menschliche Aktivitäten beim Umgang mit Softwareartefakten durch Experten beobachtet, analysiert und nach Möglichkeit sinnvoll gestaltet. [Kuu96]

3.1.2 Agentensysteme

Während Software-Ergonomie also Tätigkeiten im Zusammenhang mit Software analysiert und gestaltet, wird im Bereich der agentenbasierten Systeme versucht Software zu schaffen, die selbständig Ziele verfolgt und versucht, diese durch geeignete Aktionen zu realisieren. Agenten sind in diesem Sinn Akteure, die Pläne zur Verfolgung eines vorgegebenen Ziels verfolgen. Hierfür bedienen sie sich einem domänenspezifischen, formalen Modell von Zielen, Teilzielen und Aktivitäten zur Realisierung dieser Ziele [Bra07].

3.1.3 User Modelling

Während in der Software-Ergonomie also Nutzerverhalten von außen beobachtet und analysiert wird, realisieren Agentensysteme eigenes Verhalten aufgrund von vorgegebenen Zielen. Im Bereich des *User Modelling* wird versucht, ein formales Modell der Nutzer zu entwickeln um Informationen über den Nutzer zu gewinnen und zu verwalten, um es einem System zu ermöglichen, sich zur Laufzeit an den Nutzer anzupassen. Ein zentraler Bereich hierbei ist das automatisierte Erfassen von Nutzeraktionen, um auf dieser Basis auf die augenblicklichen Ziele und Bedürfnisse des Nutzers zu schließen [JÖ7]. Hierzu wird üblichwerweise ein formales, domänenspezifisches Modell über die Zusammenhänge von Aktionen und Zielen verwendet. Das wohl berühmteste Beispiel für User Modelling ist das Lumière-Projekt [HBH+98]. Das Projekt ist die Grundlage für das Hilfesystem in der MS Office Produktreihe. Die Aktionen der Nutzer, innerhalb der Office-Produkte von Microsoft werden beobachtet und analysiert. Werden Benutzungsschwierigkeiten oder suboptimale Bedienstrategien erkannt, wendet sich das System in personifizierter Form[1] an den Benutzer und bietet diesem Hilfe zu seiner vermuteten Tätigkeit an. Als formale Grundlage dient ein Bay'sches Netz, welches die Wahrscheinlichkeiten modelliert, dass beobachtete Aktionen und Aktionssequenzen in Zusammenhang zu bestimmten Zielen stehen. Im User Modelling wird also ein domänenspezifisches Modell von menschlichen Aktivitäten erstellt und versucht, dieses in Kombination mit der Beobachtung des Nutzerverhaltens für Mutmaßungen über die aktuellen Bedürfnisse eines Nutzers einzusetzen.

3.1.4 Verwendung in dieser Arbeit

Die Verwendung von Aktivitäten in dieser Arbeit kommt von den bisher geschilderten Ansätzen dem User Modelling am nächsten. Im Gegensatz zum User Modelling wird jedoch auf domänenspezifische Aufgabenmodelle verzichtet. Diese sind zum einen sehr aufwendig zu erstellen und zu verwalten und zum anderen stets domänenspezifisch beschränkt[2]. Stattdessen wird untersucht, wie eine abstrakte und allgemeine Beobachtung von Aktivitäten des Nutzers realisiert und genutzt werden kann. Die so gesammelten Daten können zwar auch im User Modelling eingesetzt werden, aber das ist nicht mehr Gegenstand dieser Arbeit.

3.2 Aktivitätstheorie

Aktivitätstheorie wurde als Bezugsrahmen für die Forschung zu Mensch-Maschine-Interaktion im Allgemeinen [Kuu96] [KN96] und für kontextadaptive Systeme im Besonderen [KPC06] [KA04b] vorgeschlagen. Wesentlicher Grund hierfür ist der materialistische Charakter der Theorie. Aktivitätstheorie versucht den Ablauf von Aktivitäten in

[1] als Hund oder Büroklammer

[2] Weitere Gründe zur Skepsis gegenüber der Ableitbarkeit von Zielen des Nutzers sind in 3.3.3 erläutert.

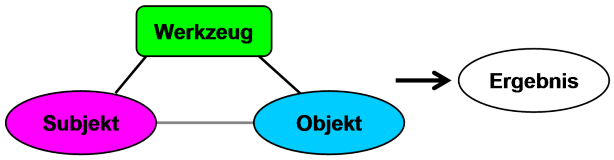

Abbildung 3.1: Vermittlung von Aktivitäten durch Werkzeuge in der Aktivitätstheorie.

Bezug auf äußere Bedingungen und Ziele der Akteure zu verstehen. Zudem bietet Aktivitätstheorie einen einheitlichen begrifflichen Rahmen zum Beschreiben von menschlichen Aktivitäten und ihrer Umstände [KA04a, S. 3]. Aktivitätstheorie geht auf die russische Psychologie der 20er und 30er Jahre zurück. Ihre grundlegenden Ideen wurden von Lew Semjonowitsch Wygotski und später Alexander Romanowitsch Lurija, Alexej Nikolajewitsch Leontjew entwickelt. Ausgehend von einem marx'schen materialistischen Weltbild, ist die angenommene Einheit von Bewusstsein und Handlung Grundlage für die Entwicklung der Theorie. Spätere, wesentliche Weiterentwicklungen stammen von Engström, der als finnischer Wissenschaftler der erste war, der außerhalb des Ostblocks zu Aktivitätstheorie publizierte [Eng87]. Ansätze aus der amerikanischen Psychologie, die Parallelen zu aktivitätstheoretischen Überlegungen haben, sind John Deweys *Pragmatismus* und George Herbert Meads *symbolischer Interaktionismus* [EM99] [Kuu96]. Aktivitätstheorie ist kein geschlossenes Theoriegebäude, sondern mehr eine wissenschaftlich-philosophisch begründete Art über Aktivitäten zu denken. Zu den grundlegenden Prinzipien dabei gehören: Das Prinzip der Objektorientiertheit (Abschnitt 3.2.1), die Vermittlung von Aktivitäten durch Werkzeuge und soziale Bedingungen (Abschnitt 3.2.2), die hierarchische Struktur von Aktivitäten (Abschnitt 3.2.3), das Konzept von Internalisierung und Externalisierung (Abschnitt 3.2.4) und die kontinuierliche Entwicklung von Aktivitäten (Abschnitt 3.2.5).

3.2.1 Objektorientiertheit

In der Aktivitätstheorie wird davon ausgegangen, dass Objekte nicht nur objektive physische Eigenschaften, sondern auch sozial-kulturelle Eigenschaften transportieren, die für objektbezogenes Handeln eine ebenso wichtige Rolle spielen. Die Eigenschaft, dass ein bestimmtes Objekt ein Buch ist und man darin lesen kann, ist ebenso objektiv, wie dessen physischen Eigenschaften, z.B. seine Farbe. Menschen leben demnach in einer Umwelt, die durch ihre Gegenstände Bedeutung direkt transportiert (auf die Implikationen dieser Annahme wird in Abschnitt 3.3.4 detaillierter eingegangen) [KN96]. Gegenstände transportieren demnach auch kulturelles Wissen über die möglichen Umgangsformen mit ihnen.

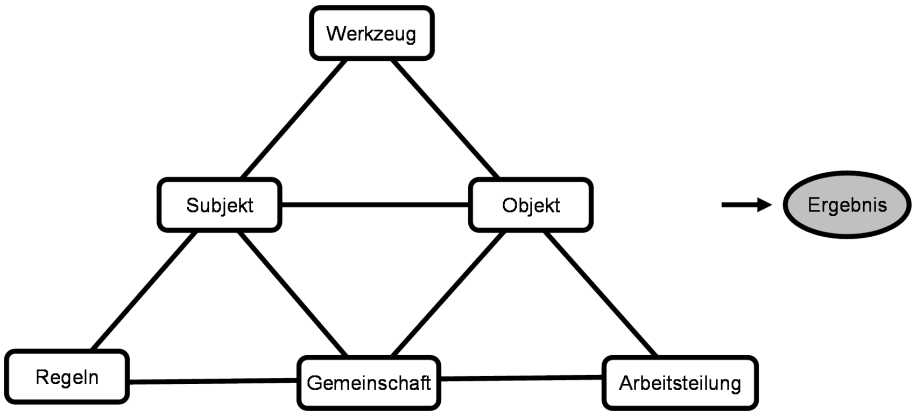

Abbildung 3.2: Soziale Einflüsse nach der sog. erweiterten Aktivitätstheorie. [KPC06].

3.2.2 Vermittlung durch Werkzeuge und soziale Bedingungen

Aktivitäten werden durch Werkzeuge vermittelt. Diese Werkzeuge tragen kulturelles Wissen in sich, das zur Sozialisation genauso bedeutend ist wie formale Erziehung. Werkzeuge vermitteln Aktivitäten und Beeinflussen deren Form (vgl. Abbildung 3.1). Dabei können stoffliche (z.B. physische Dinge wie ein Hammer) und nicht stoffliche Werkzeuge (z.B. Sprache, Mathematik) unterschieden werden. Nicht stoffliche Werkzeuge wirken primär auf den Akteur und dessen Handlungspläne und mentale Modelle, während stoffliche Werkzeuge direkt auf ein Objekt wirken. In beiden Fällen wird die Aktivität durch das verwendete Werkzeug beeinflusst. Engström entwickelte eine erweiterte Aktivitätstheorie in der als weitere Komponenten die Gemeinschaft innerhalb der eine Aktivität stattfindet, Regeln und die bestimmende Arbeitsteilung als Bedingungen aufgenommen wurden, um die sozialen Faktoren die Aktivitäten vermitteln zu erfassen (vgl. Abbildung 3.2) [KN96] [Eng99].

3.2.3 Hierarchischer Aufbau von Aktivitäten

Aktivitäten sind hierarchisch aufgebaut (siehe Abbildung 3.3). Sie gliedern sich in Aktivitäten, Aktionen und Operationen, denen Motive, Ziele und Bedingungen entsprechen. Auf der obersten Ebene stehen Aktivitäten, die einem Motiv folgen. Das Motiv ist der eigentliche Zweck der Aktivität und der Handelnde ist sich in der Regel seiner Aktivität und seines Motivs bewusst. Aktivitäten werden durch eine Reihe von Aktionen realisiert, mit denen (Teil-)Ziele verfolgt werden. Auch diese Ebene findet bewusst statt. Aktionen selber werden durch eine Folge von Operationen erreicht. Diese Operationen sind vom Handelnden bereits soweit internalisiert, dass sie sich unbewusst vollziehen. Die Operationen verfolgen die Bedingungen der Ziele. Aktivitäten können sich über einen längeren Zeitraum erstrecken, dabei werden oft mehrere Aktivitäten simultan verfolgt. Aktionen hingegen finden in einem kurzen, abgeschlossenen Zeitraum statt, üblichwerweise wird

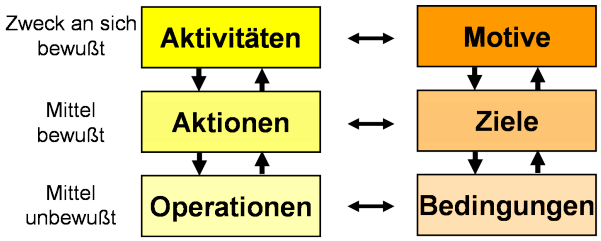

Abbildung 3.3: Hierarchischer Aufbau von Aktivitäten und Motiven nach der Aktivitätstheorie [KPC06].

eine Aktion zur Zeit verfolgt. Operationen brauchen als routinierte, unbewusste Handlungen oft nur kürzeste Zeiträume.

Die Übergänge zwischen den Ebenen können fließend sein, so können routiniert ausgeführte Aktionen zu Operationen werden. In umgekehrter Richtung kann es passieren, dass eine Operation nicht wie gewohnt zum Erfolg führt. In dem Fall wird von einer *Blockierung* gesprochen, der Handelnde wird sich typischerweise der Operation bewusst und muss sich Alternativen zum Erreichen seines Ziels überlegen [KN96] [KPC06]. Die Kenntnis der Aktivität ist die minimale Ebene, um Handlungen verstehen zu können. Leontjev illustriert dies am Beispiel von Jägern, die sich in Buschklopfer und Fänger aufteilen. Das Verhalten der Buschklopfer, die Lärm erzeugen, um ein Tier in die Arme der Fänger zu treiben, macht isoliert betrachtet kaum Sinn und kann nur unter Kenntnis der gesamten Aktivität verstanden werden [Kuu96].

3.2.4 Internalisierung und Externalisierung

Internalisierung und Externalisierung beschreibt die Vorgänge beim Erlangen von Kompetenz in einer Aktivität. Zunächst werden von Lernenden sozial erarbeitete Vorgehensweisen internalisiert und übernommen. Wenn die Aufgaben jedoch wachsen und zunehmend schwieriger werden, können die internalisierten Handlungsweisen externalisiert werden. Dadurch werden diese wieder explizit, können z.B. in Form von Text dargestellt werden und so von den Handelnden oder einer Gemeinschaft reflektiert und weiterentwickelt werden.

3.2.5 Kontinuierliche Entwicklung

Wie sich schon im obigen Teilabschnitt andeutet, betont Aktivitätstheorie, dass sich der Ablauf von Aktivitäten nicht allein durch Ziel und Werkzeug festlegt, sondern einer Entwicklung unterliegt. Es wird von Akteuren in unterschiedlichen Entwicklungsschritten eine Praxis entwickelt, die immer weiter fortgeschrieben wird und sich immer wieder verändern und entwickeln kann[3] [Eng99] [KN96].

[3] vgl. auch die Betonung der Entwicklung vom Praxis im Begriff *Embodied Interaction* von Dourish [Dou04]

Im weiteren Verlauf der Arbeit lehnt sich das Verständnis von Aktivitäten an die Aktivitätstheorie an. Bei der Verwendung des Begriffs „Aktivität“ in dieser Arbeit wird, zumindest dort, wo nicht anders betont, nicht im strengen Sinn der Aktivitätsthorie zwischen Aktivitäten, Aktionen und Operationen unterschiedenen, sondern der Begriff im Sinne eines Oberbegriffs über diese Ebenen verwendet. Die Verwendung des Wortes entspricht also, zur Förderung der Lesbarkeit, den umgangssprachlichen Verständnis.

3.3 Konsequenzen von Aktivitätstheorie für diese Arbeit

Vor allem aus der Objektorientiertheit (vgl. Abschnitt 3.2.1), der Vermittlung durch Werkzeuge (vgl. Abschnitt 3.2.2) und dem hierarchischen Aufbau von Beziehungen (vgl. Abschnitt 3.2.3) lassen sich Implikationen für diese Arbeit ableiten[4].

3.3.1 Gleichzeitigkeit von Aktivitäten

Aktivitätstheorie stellt fest, dass Menschen üblicherweise einer Vielzahl von Aktivitäten parallel nachgehen. Dies ist für den Bereich des mobile Computing von besonderem Interesse, da mobile Geräte vielmehr in alltägliche und unterschiedliche Aktivitäten eingebunden sind und spontaner benutzt werden, als schreibtisch-gebundene Computer (vgl. Abschnitt 4.2.1). Zudem erschwert es die Beobachtung von Aktivitäten.

3.3.2 Was ist beobachtbar?

Nutzen wir Aktivitätstheorie um ein Verständnis darüber zu gewinnen, welchen Bereich von Aktivitäten wir überhaupt automatisch durch ein Softwaresystem erfassen können, so lassen sich einige Feststellungen machen.

Zunächst dürfte klar sein, dass sich der Bereich von Zielen, Motiven und Bedingungen nicht direkt beobachten lässt (vgl. Abschnitt 3.2.3). Aber auch die oberste Ebene der Aktivitäten ist kaum direkt zu erfassen, da sich diese langfristig vollzieht und in der Regel nur ein Teil der zugehörigen Aktionen mit Hilfe von Softwareartefakten unternommen wird.

Andererseits lässt sich aber die Vermittlung von Aktivitäten durch Werkzeuge für deren Beobachtung ausnutzen. Denn insoweit es sich um Computerwerkzeuge handelt, sind dieser der automatischen Beobachtung prinzipiell gut zugänglich. Handlungen, die mit traditionellen Werkzeugen unternommen werden, sind aber kaum noch und nur indirekt automatisch feststellbar und nicht stoffliche Werkzeuge lassen sich überhaupt nicht beobachten (siehe Abbildung 3.4).

Somit sind lediglich Aktionen zugänglich, die mit Computerwerkzeugen ausgeführt werden. Allerdings nimmt die Anzahl solcher Werkzeuge oder von Werkzeugen, die mit

[4] Die anderen beiden Bereichen sind z.B. interessant für den Bereich der Mensch-Maschine-Interaktion oder kooperatives Lernen, da sie einer statischen Auffassung von Nutzer und Nutzerverhalten wiedersprechen und sich mit der Entwicklung von Kompetenz und Praxis befassen [Kuu96].

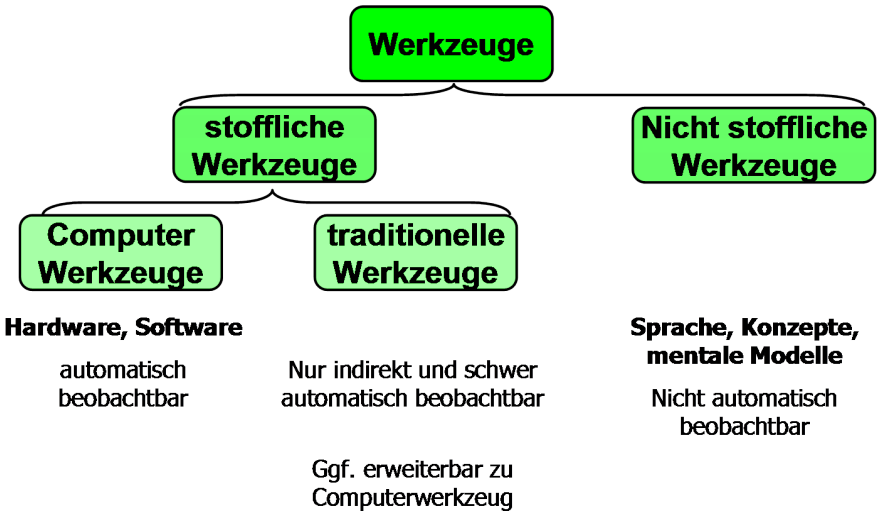

Abbildung 3.4: Werkzeuge und ihre Beobachtbarkeit.

Computertechnologie verwoben sind, stetig zu. Nehmen wir Mark Weisers Vision des Ubiqitous Computing ernst (vgl. 1.1), so ist davon auszugehen, dass zukünftige Werkzeuge zu einem großteil Computerwerkzeuge sind.

3.3.3 Schwierigkeiten bei der Ableitung von Zielen und Motiven

Für die Ableitung von Zielen und Motiven des Nutzers, ergibt sich allerdings eine eher pessimistische Perspektive. Die Beobachtung von Aktivitäten ist lückenhaft, da sich nur ein Teilbereich der Aktionen erfassen lässt. Zudem verfolgt ein Nutzer eine Reihe von Aktivitäten parallel und seine Aktionen können somit im Zusammenhang mit verschiedenen Aktivitäten stehen und lassen sich nicht eindeutig zuordnen. Zudem sind Aktivitäten einer sich stets verändernden Praxis unterworfen. All dies macht das automatische Schließen auf Motive und Ziele des Nutzers zu einer spekulativen Angelegenheit.

3.3.4 Exkurs: Unzulänglichkeit kognitiver Ansätze

Im Bereich der Mensch-Maschine-Interaktion haben sich Forscher auf Aktivitätstheorie bezogen, da der vorherrschende kognitive Ansatz in den 80er Jahren zunehmend als ungenügend angesehen wurde [Kuu96]. Der kognitive Ansatz begreift den Menschen als signalverarbeitendes System. Demnach versteht der Mensch seine Umwelt über die Interpretation von Signalen (siehe Abbildung 3.5). Die Forschung in der Mensch-Maschine-Interaktion, war demnach vorrangig damit beschäftigt, die Signale für den kognitiven Apparat des Menschen zu optimieren. Und vollzog sich sehr laborbezogen, ohne Berücksichtigung von Kontexten [Kuu96]. Die Perspektive der Aktivitätstheorie stellt hierzu einen Gegensatz dar, da sie die Bedeutung des Umfeldes für Handlungen hervorhebt und betont, dass Bedeutungen durch die Artefakte direkt transportiert werden können (vgl. Abbildung 3.7). Im Vorgriff auf Kapitel 4 kann festgestellt werden, dass es hier eine

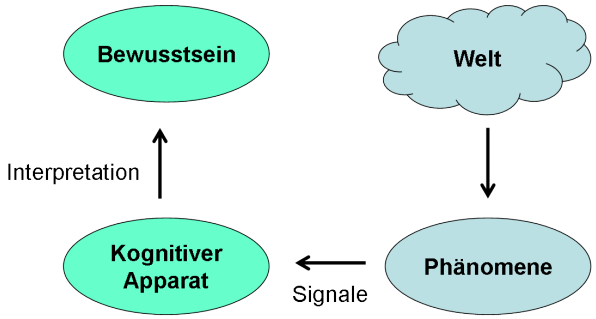

Abbildung 3.5: Paradigma kognitiv geprägter Ansätze.

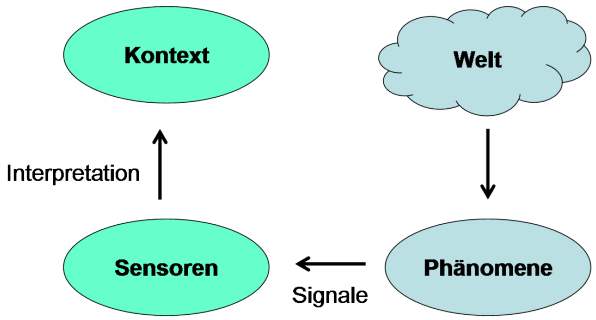

Abbildung 3.6: Traditionelles Verständnis von Context-Awareness.

interessante Parallele zum vorherrschenden Verständnis von kontextadaptiven Systemen gibt. Kontextadaptive Systeme werden überwiegend als Systeme verstanden, die Phänomene der Welt über Sensoren erfassen, diese Sensordaten interpretieren und verarbeiten und so zu einer Repräsentation von Kontext gelangen, wie es in Abbildung 3.6 skizziert ist. Diese Art des Verständnisses wird z.B. sehr deutlich in den Begriffen High-Level und Low-Level-Kontext bei Chen und Kotz (vgl. Abschnitt 4.1.3.3) [CK00].

Demgegenüber steht ein Verständnis von Kontext, nach dem Kontext aus Aktivitäten entsteht (z.B. bei Dourish vgl. Abschnitt 4.1.4 [Dou04]) und Bedeutungen in den Artefakten selber transportiert werden (vgl. Abbildung 3.7). Das heißt natürlich nicht, dass es grundsätzlich keinen Sinn macht Sensordaten zu erheben, aber diese Arbeit plädiert dafür diese eingeschränkte auf Verarbeitung von Sensordaten fixierte Sichtweise als allgemeines Paradigma für Context Awareness aufzugegeben.

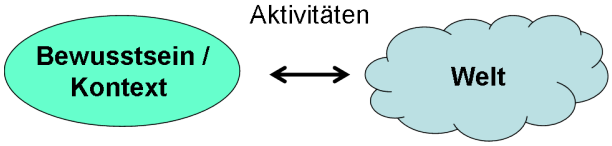

Abbildung 3.7: Aktivitätstheorie umgeht die Fixierung auf Signalverarbeitung.

Aus dieser Perspektive können praktische Konsequenzen gezogen werden: Demnach sollte weniger versucht werden ein System zu schaffen, das Kontext indirekt über Sensoren von außen beobachtet und interpretiert, vielmehr sollte Bedeutung durch die Softwareartefakte selber transportiert werden. Die kulturellen Bedeutungen, die ein Artefakt transportiert, brauchen eine digital repräsentierte Entsprechung in Softwareartefakten. Dies könnte ein wichtiger Schritt hin zur Vision des Ubiqitous Computing sein. Für Aktivitäten bzw. Aktionen heißt dies, dass diese wie im vorigen Anschnitt bereits angedeutet, direkt in den vermittelnden Computerwerkzeugen erfasst werden sollten.

3.4 Zusammenfassung

In der Informatik existieren zahlreiche Bezugspunkte zu Aktivitäten. Ein Bereich, der dem Ansatz dieser Arbeit nahe kommt, ist das User Modelling. Im Gegensatz zum User Modelling verzichtet diese Arbeit jedoch bewusst auf domänenspezifische Aufgabenmodelle und den Versuch, auf die Ziele des Nutzers zu schließen. Als theoretischer Bezugspunkt wurde für diese Arbeit die Aktivitätstheorie gewählt. Insbesondere ihrer Prinzipien der Objektorientiertheit, der hierarchische Aufbau von Aktivitäten und die Vermittlung von Werkzeugen haben Konsequenzen für diese Arbeit. Die wichtigste Konsequenz ist die Erkenntnis, dass Aktivitäten bzw. Aktionen nach Möglichkeit nicht über äußere Sensoren erfasst werden sollten, sondern über die Computerwerkzeuge, die sie vermitteln.

4 Kontext und Kontextadaption

Schilit et al. prägten den Begriff *Context Aware Computing* für mobile Computersysteme, die sich automatisch an ihren Nutzungskontext anpassen [SAW94]. Was dabei als Kontext zu verstehen ist (Abschnitt 4.1) und wie dieser Kontext, insbesondere aktivitätsbezogener Kontext, zur Verbesserung der Gebrauchstauglichkeit von mobilen System genutzt werden kann (Abschnitt 4.2), soll in diesem Kapitel erörtert werden.

4.1 Was ist Kontext?

Um zu einem Begriffsverständnis von Kontext im Sinne des Context Aware Computing zu gelangen, werden in diesem Abschnitt zunächst einige Beispiele für kontextadaptive Systeme angeführt (Abschnitt 4.1.1). Anschließend werden ein paar ausgesuchte Begriffsdefinitionen von Kontext untersucht (Abschnitt 4.1.2). Um das Verständnis zu vertiefen, werden darauf aufbauend verschiedene Teilbereiche und Aspekte von Kontext ausdifferenziert (Abschnitt 4.1.3) und abschließend inhärente Eigenschaften von Kontext, die im Umgang mit diesem zu beachten sind, geschildert (Abschnitt 4.1.4).

4.1.1 Beispiele für Kontextadaption

Im Bereich des Context Aware Computing wurden verschiedene kontextaptive Anwendungen, d.h. Anwendungen, die gewisse Teile ihres Nutzungskontextes automatisch feststellen und sich diesem in einem gewissen Rahmen anpassen können, vorgeschlagen und entwickelt. An dieser Stelle können nur einige Beispiele genannt werden, die zeigen sollen, welche unterschiedlichen Arten von Kontext herangezogen werden können und auf welch vielfältige Art und Weise Kontext genutzt werden kann.

Cheverst et al. haben einen kontextadaptiven Touristenführer entwickelt, der die Position des Nutzers in der Stadt kennt und in der Lage ist, individuelle Routen und Stadtführungen vorzuschlagen, wobei sowohl die Vorlieben des Nutzers als auch die Öffnungszeiten der Attraktionen berücksichtigt werden [CMD99].

Pham et al. haben für ein Szenario in einem Krankenhaus, bei dem Mitarbeiter mit mobilen Geräten arbeiten, eine Architektur entwickelt, die es erlaubt in einem Raum vorhandene Infrastruktur spontan zu nutzen [PSGP01]. So kann beispielsweise ein Arzt eine Krankenakte auf einem mobilen Gerät vorliegen haben, aber zur Anzeige eines auf dem kleinen Gerät gespeicherten Röntgenbildes einen gerade im Raum vorhandenen Monitor zur Besprechung des Bildes mit Kollegen oder Patienten nutzen.

Intelligente Konferenzräume sind ein häufig gebrauchtes Szenario für kontextadaptive Anwendungen. Die Vision ist, dass sich der Raum automatisch an die Bedürfnisse der Nutzer anpassen soll, z.B. automatisch das Licht verdunkelt, den Projektor startet und

Abbildung 4.1: Das *ContextContacts*-Telefonbuch zeigt Informationen zur Situation, in der sich der andere Teilnehmer gerade befindet [ORT05].

bestimmte Folien anzeigt, wenn ein Vortrag beginnt. Darüber hinaus können Informationen automatisch verwaltet werden, z.B. können Vortragsfolien und Teilnehmerdaten den Teilnehmern einer Besprechung automatisch zur Verfügung gestellt werden. Der Raum kann sich auch aktiv Verhalten und etwa die Vortragsfolien vom Arbeitsplatz des Vortragenden selbstständig anfordern [Che04, S. 2f].

Oulasvirta et al. stellten mit *ContextContacts* ein Adressbuch für Mobiltelefone vor, das in der Lage ist, Informationen zur Situation der Kontakte des Nutzer abzufragen und darzustellen [ORT05]. Ein Screenshot dieser Anwendung ist in Abbildung 4.1 dargestellt. Es wird angezeigt, in welchem Modus sich das Telefon des anderen Teilnehmers befindet, ob Vibrationsalarm aktiviert ist, wie viele und welche Art von Personen sich in der Nähe befinden und wann die andere Person sein Telefon zuletzt benutzt hat. Auf Basis dieser Informationen kann der Nutzer dieses Adressbuches entscheiden, ob er den anderen Teilnehmer sofort anrufen möchte, lieber eine Nachricht sendet oder auf eine günstigere Gelegenheit zur Kommunikation wartet.

Das von Apple herausgebrachte iPhone ist ein Beispiel für einen kontextadaptiven Mechanismus, der bereits im Markt eingesetzt wird [Job07]. Das Gerät ist in der Lage, über einen Neigungssensor die Ausrichtung seines Displays (horizontal oder vertikal) zu erkennen und die Darstellung anzupassen, in dem der Inhalt ggf. um 90 Grad rotiert wird und die neuen Dimensionen berücksichtigt werden (siehe Abbildung 4.2).

Die genannten Beispiele zeigen ein breites Spektrum in der Nutzung von Kontext. Die Möglichkeiten reichen von der Erfassung der sozialen Situation wie bei ContextContacts, bis zur automatischen Anpassung an lokal vorhandene Hardware wie bei Pham et al. Es können physikalische Eigenschaften, wie bei der Ausrichtung des iPhones oder Infor-

Abbildung 4.2: Der Browser des iPhones passt sich an die Ausrichtung des Gerätes an [Tec09].

mationen über die Umgebung wie bei einem kontextadaptiven Touristenführer genutzt werden.

Das Spektrum der Nutzung von Kontext reicht vom Anzeigen von kontextrelevanten Informationen (ContextContacts) über transparente Anpassung an die Bedingungen (Rotation beim iPhone oder Nutzung lokaler Hardware (Pham)) bis hin zu aktivem Verhalten (Chen). Im den folgenden Abschnitten wird näher bestimmt, was Kontext ist und worin die Gemeinsamkeiten bei der Nutzung bestehen und welche Arten von Kontext es überhaupt gibt.

4.1.2 Definitionen von Kontext

Im Bereich der Forschung zu Kontextadaption gibt es keine allgemein akzeptierte kanonische Definition des Begriffs Kontext [KA04b] [ZLO07]. Die Definition, auf die am häufigsten Bezug genommen wird, ist die Definition von Dey und Abowd:

> „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.“ [DA99]

Diese Definition betont, dass prinzipiell jede Art von Information Kontext sein kann. Darüber hinaus führt sie ein abstraktes Konzept von Entitäten ein, das sowohl Entitäten

technischer, als auch sozialer oder physikalischer Natur umfasst [Tur06]. Es wird betont, dass es bei Kontext um die Beeinflussung von Interaktion durch Entitäten geht.

Diese Definition wird in einigen Punkten von unterschiedlichen Autoren kritisiert. So wenden verschiedene Autoren ein, dass diese Definition zu offen sei, da in diesem Sinn prinzipiell jede Information Kontext sein könne [ZLO07] [Hen03, S. 14] [Win01a]. Winograd merkt an, dass dies ein prinzipielles Problem sei, da die Informationen, die zum Kontext gehören, vom Kontext abhängig seien [Win01a] und befüwortet die Verwendung des Begriffs „Situation Awareness“ und ein damit verbundenes pragmatischeres Verständnis vor. Demnach sei es besser, sich auf die typischen Aspekte einer Situation zu konzentrieren, welche die Benutzung von Software beeinflussen, als ein all gemeingültiges Konzept von Kontext zu verfolgen. Ähnlich argumentieren Kofod-Petersen und Cassens [KPC06] sowie Brézillion und Pomerol [BP99, Abschnitt III.1].

Verschiedene Autoren betonen die Bedeutung von Aktivitäten und Zielen des Nutzers stärker, als dieses in Deys Definition der Fall ist. Für Chen umfasst Kontext das Modellieren der Aktivitäten und Aufgaben von Personen und digitalen Entitäten [Che04, S. 21]. Kofod-Petersen und Cassens halten fest, dass die Aktivitäten der Akteure zu den wichtigsten Merkmalen einer Situation gehören und dass eine kontextadaptive Anwendung daher zu jedem Zeitpunkt über die Aktionen des Nutzers informiert sein sollte [KPC06]. Für Henricksen bezieht sich Kontext auf Informationen, die zum Erfüllen einer Aufgabe von Bedeutung ist [Hen03, S. 14]. Diesen Punkten schließen sich Zimmermann et al. an und sie kommen zu einer modifizierten Form der Dey’schen Definition:

> „Context is any information that can be used to characterize the situation of an entity. Elements for the description of this context information fall into five categories: *individuality*, *activity*, *location*, *time*, and *relations*. The activity predominantly determines the relevancy of context elements in specific situations, and the location and time primarily drive the creation of relations between entities and enable the exchange of context information among entities.“ [ZLO07]

Wobei unter „individuality“ Informationen zur Entität selbst, also deren Attribute, verstanden wird und unter Beziehungen soziale, funktionale (d.h. Benutzungsbeziehungen z.B. Person A sitzt auf Stuhl B) und kompositorische Beziehungen (Aggregationen und Assoziationen z.B. ein Auto umfasst Motor, Karosserie und Navigationssystem bzw. Person A gehört zu Abteilung C) gefasst sind. Die Definition betont ebenfalls die Bedeutung von Aktivitäten, da sich aus diesen die Relevanz der anderen Elemente ergibt und hebt die Beziehungen, die durch Kontext entstehen, hervor.

Zimmermann et al. beschäftigen sich mit dem Problem, wie verschiedene Akteure zu einem gemeinsamen Verständnis einer Situation gelangen können, so dass diese gemeinsam vorliegenden Informationen für ihre weitere Interaktion die Basis bildet. Dieses Verständnis der Verwendung von Kontext entspricht der Art und Weise, wie Kontext bei der

Interpretation von Text durch Menschen benutzt wird. Kontext besteht dort, wo ein gemeinsames Verständnis einer Situation und von Begrifflichkeiten implizit vorausgesetzt wird und dieser als Grundlage für die Interpretation von Kommunikation dient. Aufgrund dieses Verständnisses heben Zimmermann et al. die Bedeutung des Austausches von Kontextinformationen in ihrer Arbeit hervor.

Der Aspekt der Austauschbarkeit von Informationen ist zwar für die vorliegende Arbeit von geringerer Bedeutung, aber wegen der Ausarbeitung des Verständnisses von Aktivitäten, Beziehungen und Entitäten in der Definition von Kontext lehnt sich das dieser Arbeit zugrunde liegende Verständnis von Kontext an die Definition von Zimmermann et al. an.

4.1.3 Kategorisierungen von Kontext

Es wurden zahlreiche Anstrengungen unternommen Kontextarten zu klassifizieren, um so zu einem systematischeren Verständnis von Kontext beizutragen, die nicht alle in dieser Arbeit gewürdigt werden können[1]. Oft wird zwischen persönlichem und Umweltkontext unterschieden [GS01b] [SBG99]. Persönlicher Kontext sind Informationen über den Nutzer. Ein Großteil dieser Information kann als Nutzerprofil verstanden werden, das z.B. die Präferenzen des Nutzers beschreibt und für die Dauer einer Anwendung relativ stabil ist. Umweltkontext umfasst äußere Faktoren wie Tageszeit, Umweltbedingungen oder Verfügbarkeit von Diensten[2]. Eine andere Unterteilung von Kontext in verschiedene Bereiche erfolgte durch Schmidt et al. In dieser Unterteilung wird zwischen den Dimensionen *Selbst*, *Aktivität* und *Umwelt* unterschieden [SAT+99]. Die Dimension „Selbst“ umfasst mit physiologischen und kognitiven Bedingungen des Nutzers nicht nur den Persönlichen Kontext, auch Informationen über die benutzte Hardware fallen in diesen Bereich. Mit der Dimension „Aktivität“werden hier die Bereiche persönlicher Kontext und Umweltkontext um eine Dimension erweitert, welche Verhalten und Aufgaben des Nutzers beinhaltet.

4.1.3.1 Entitätsbezogene Klassifizierung

Zimmermanns Definition (vgl. 4.1.2) teilt Kontext in die Bereiche Individualität, also Informationen zu den Attributen einer Entität, Aktivität, Zeit, Ort und Beziehungen sozialer, funktionaler und kompositorischer Art auf. Da in Zimmermanns Verständnis Kontext immer subjektiv und einer Entität zugeordnet ist, entfallen die Kategorien persönlicher Kontext und Umweltkontext und sind in der Kategorie Individualität aufgehoben. Zimmermann betont, dass die Aktivitäten einer Entität ihre Bedürfnisse im Wesentlichen bestimmen. Der Aktivitätskontext beschreibt nach Zimmermann, was die Entität erreichen will und wie sie dies tun möchte.

[1] Eine vergleichende Übersicht findet sich bei Kaenampornpan und Ay [KA04b].
[2] Sowohl solche in der physischen Welt als auch virtuelle Dienste

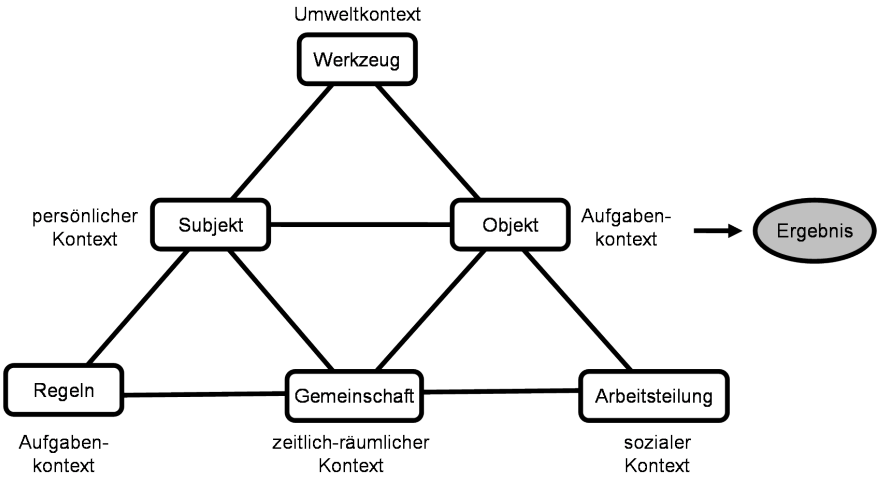

Abbildung 4.3: Zuordnung von Begriffen der Aktivitätstheorie zu Bereichen des Kontextmodells nach Kofod-Petersen/Cassens (2006) [KPC06].

4.1.3.2 Nutzerbezogene Klassifizierung

Kofod-Petersen und Cassens ordnen den Aspekten der erweiterten Aktivitätstheorie (vgl. Abschnitt 3.2) Kontextbereiche zu (siehe Abbildung 4.3) und nutzen diese Zuordnung, um eine eigene Kontexttaxonomie zu entwickeln [KPC06]. Diese Kontexttaxonomie unterteilt den Nutzerkontext, wie in Abbildung 4.4 dargestellt, in persönlichen Kontext, Aufgabenkontext, sozialen Kontext, zeitlich-räumlichen Kontext und Umweltkontext. Der persönliche Kontext umfasst die physiologischen und mentalen Bedingungen des Nutzers, wie z.B. seine Stimmung, körperliche Einschränkungen oder Expertise auf verschiedenen Wissensgebieten. Der Aufgabenkontext beschreibt was der Nutzer tut. Er kann die Ziele, Aufgaben und Aktivitäten des Nutzers umfassen. Dabei können auch Nebenbedingungen, etwa solche die sich aus Regeln und Gesetzten für das Erfüllen von Aufgaben ergeben, mitberücksichtigt werden. Der soziale Kontext umfasst Informationen über die sozialen Aspekte des Nutzers, z.B. die Rollen, die ein Nutzer inne hat. Der zeitlich-räumliche Kontext umfasst die Aspekte aktuelle Zeit, Position des Nutzers und das die Situation bestimmende aktuelle soziale System[3]. Der Umweltkontext schließlich umfasst die Umgebung des Nutzers, wie z.B. physische Dinge, verfügbare Dienste, anwesende Personen und Umweltbedingungen. Somit werden auch in dieser Taxonomie Aktivitäten als ein Teilbereich von Kontext verstanden. Auch hier sind die Bereiche persönlicher Kontext und Umweltkontext vorhanden. Zeitlicher- und räumlicher Kontext wurden mit der sozialen Situation zusammengefasst. Und der allgemein soziale Kontext des Nutzers als eigene Kategorie eingeführt.

Die Definitionen von Zimmermann et al. bzw. Kofod-Petersen/Cassens widersprechen sich dabei nicht. Zimmerman klassifiziert die Arten von Kontextinformationen, die einer einzelnen Entität zugeordnet werden können, während Kofod-Petersen und Cassens

[3] engl.: the community present

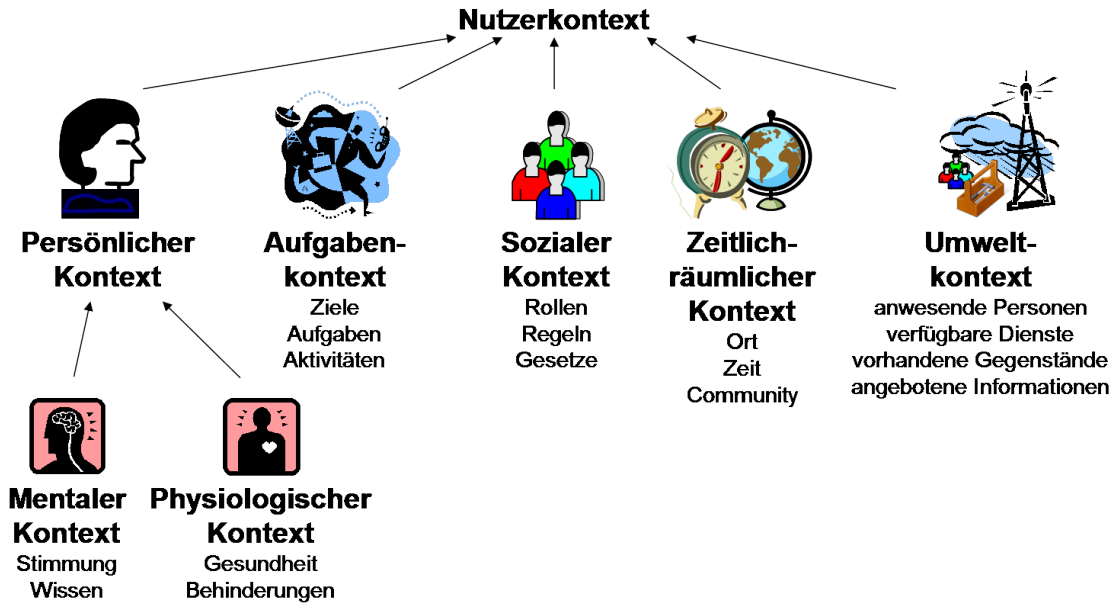

Abbildung 4.4: Eine an der Aktivitätstheorie orientierte Taxonomie von Kontext [KPC06].

die Informationsarten klassifizieren, die zur Bestimmung des Kontextes für einen Nutzer von Bedeutung sind. Diese Informationen können jedoch eine Reihe von Entitäten umfassen.

4.1.3.3 Qualifizierende Unterscheidungsmerkmale

Die beiden zuletzt ausgeführten Definitionen betonen die Subjektivität von Kontextinformationen. In dem einen Fall beziehen sie sich auf je eine Entität, in dem anderen Fall auf den Nutzer und bilden damit einen Gegensatz zu den Ansätzen, die Kontextinformationen als global bzw. objektiv verstehen, wie dies beispielsweise in der Definition durch Dey und Abowd der Fall ist. Die Unterscheidung zwischen *subjektivem* und *objektivem* Kontext betrifft also eher das Verständnis von Kontext als eine einzelne Information.

Ein wichtiges Unterscheidungsmerkmal für Kontextinformationen wurde durch Dey und Abowd gegeben. Die Autoren unterscheiden zwischen *Primär-* und *Sekundärkontext* [DA99]. Unter Primärkontext wird Ort, Identität, Zeit und Aktivität einer Entität verstanden. Diese Informationen beantworten die Frage nach dem Wer, Was, Wann und Wo einer Situation und können herangezogen werden, um weitere Kontextinformationen zu erhalten. So können über die Identität einer Entität (Primärkontext) ihre weiteren Attribute (Sekundärkontext) in Erfahrung gebracht werden oder über ihre Position (Primärkontext) die Entitäten, die sich in der Nähe befinden (Sekundärkontext). Der Primärkontext ist für den Nutzer oft nicht direkt von Belang, sondern indirekt durch die ableitbaren Informationen. Der Nutzer dürfte weniger an den geografischen Positionsdaten seines Standortes interessiert sein, als an Personen, Geschäften und Wegen in seiner Nähe.

Eine Unterscheidung, die in eine ähnliche Richtung geht, ist die Unterscheidung zwischen *Low-* und *High-Level-Kontextdaten* durch Chen und Kotz [CK00]. Low-Level-Kontext bezeichnet Kontext, der direkt durch Hard- oder Softwaresensoren gemessen werden kann, während High-Level-Kontext aus der Verarbeitung und Interpretation von Low-Level-Kontext entsteht.

Lei unterscheidet zwischen *persistentem* und *vergänglichem*[4] Kontext [LSD+02]. Vergänglicher Kontext bezieht sich auf eine konkrete Situation während persistenter Kontext relativ stabiles Hintergrundwissen darstellt. So ist z.B. die Tatsache, dass Alice gerade mit Bob telefoniert vergänglicher Kontext, während das Wissen, dass Alice üblicherweise montagmorgens um 9:00 Uhr mit Bob telefoniert, persistenter Kontext ist.

4.1.4 Eigenheiten von Kontext

Dourish weist auf vier zentrale Eigenheiten von Kontext hin [Dou04]:

- Kontext ist nicht einfach eine Sammlung einer Menge von Informationen, sondern Kontextualität ist eine Eigenschaft einer Information. Eine Information oder ein Objekt kann bezogen auf eine Aktivität kontextuell *relevant* sein oder auch nicht. Kontext ist daher eher eine Aussage über Relevanz von Informationen.

- Was zum Kontext gehört, kann nicht im Voraus bestimmt werden. Die Menge der kontextuellen Merkmale ergibt sich stets *dynamisch*.

- Kontext ist nicht stabil, sondern spezifisch für jede Aktivität oder Aktion. Kontext ist eine fallbezogene Eigenschaft *spezifisch* für jede Kombination aus Umständen, Aktivitäten und Beteiligten.

- Kontext und Aktivitäten können nicht voneinander getrennt werden. *Kontext entsteht erst durch Aktivitäten*.

Dourish weist darauf hin, dass Kontext meist als ein Problem der Repräsentation von Wissen verstanden wird und dass in dieser Sichtweise die eben genannten Eigenschaften nicht nur nicht berücksichtigt werden, sondern, meist aus pragmatischen Gründen, eher implizit mit gegenteiligen Annahmen gearbeitet wird.

Dourishs eher skeptische Sichtweise mahnt davor zu glauben, man müsse nur genügend Wissen adäquat repräsentieren, um Kontext zur Optimierung von Interaktion einsetzen zu können. Vielmehr muss es dem Nutzer möglich sein zu bestimmen, wie ein System an eine Situation angepasst werden soll, damit dieser eine eigene Praxis kontextadequater Handlungen entwickeln kann. Gleichzeitig hebt Dourish die Wichtigkeit, die Kontext für die Interaktion hat, hervor und betont die große Bedeutung von Aktivität für Kontext.

[4] engl.: transient

4.2 Kontext und Benutzbarkeit mobiler Geräte

Usability kann als die Gebrauchstauglichkeit einer Technologie verstanden werden, also die Einfachheit mit der eine bestimmte Technologie für ein bestimmtes Ziel eingesetzt werden kann [CK06]. Zu den typischen Zielen einer guten Benutzbarkeit gehören Sicherheit und Fehlertoleranz, Effektivität (die Möglichkeit eine Aufgabe tatsächlich zu bewältigen), Effizienz bei der Bewältigung von Aufgaben, Zufriedenheit der Nutzer und leichte Erlernbarkeit der Bedienung [Mal07].

Usability kann demnach immer nur im Zusammenhang mit einer Aufgabe oder einem Aufgabenbereich bewertet werden. Aber auch Kontext spielt für die Usability eine große Rolle. Es ist unzweifelhaft, dass der Nutzungskontext großen Einfluss auf die Bedienbarkeit von Software hat [CK06]. So macht es z.B. in einer lauten Fabrikhalle kaum Sinn, ein Gerät zu verwenden, mit dem der Benutzer vorwiegend über akustische Signale interagiert (z.B. ein Diktiergerät), auch wenn es in anderen Kontexten gut für eine Aufgabe (z.B. das Erfassen kurzer Notizen) geeignet ist. Von diesem Umstand sind mobile Geräte besonders betroffen, da sie anders als beispielsweise fest installierte Computer auf einem Schreibtisch nicht über einen relativ stabilen Nutzungskontext verfügen, sondern sich dieser beständig ändert. Hinzu kommt die Tatsache, dass sich für mobile Geräte einige spezifische Nutzungsprobleme aus der Größe und Mobilität der Geräte ergeben.

Wenn aber Kontext so eng mit der Nutzbarkeit von Software und Softwareartefakten verbunden ist, ist es naheliegend diesen Kontext zur Verbesserung der Benutzbarkeit zu berücksichtigen.

Das besondere Problem dabei sind die im vorherigen Abschnitt 4.1.4 erläuterten, dynamischen Eigenschaften von Kontext. Es lassen sich demnach nicht alle denkbaren Situationen im Vorwege bestimmen. Dennoch kann und sollte der Versuch unternommen werden, Einflüsse die Kontext auf die Nutzbarkeit einer (mobilen) Software hat, bereits im Softwaredesignprozess zu erfassen und zu berücksichtigen. Methoden für ein systematischees Vorgehen hierbei sind bisher allerdings kaum entwickelt [PV05].[5]

Sind Einflussfaktoren möglicher Nutzungssituationen auf die Benutzbarkeit von Software identifiziert worden, ist es sinnvoll, diesen Kontext zur Laufzeit zu erfassen und zur Verbesserung der Interaktion mit mobilen Geräten einzusetzen.

In diesem Abschnitt sollen daher kurz häufig auftretende Nutzungsschwierigkeiten bei der Verwendung mobiler Geräte erörtert werden (Abschnitt 4.2.1) und dann auf die Potentiale (Abschnitt 4.2.2), aber auch auf die möglichen Gefahren (Abschnitt 4.2.3) bei der Verwendung von Kontext zur Erleichterung von Interaktion eingegangen werden. Im Weiteren werden schließlich die besonderen Potentiale von Aktivitätskontext (Abschnitt 4.2.4) gesondert hervorgehoben.

[5] Ansätze hierzu finden sich beispielsweise bei Pedell und Vetere [PV05], Kaptelinin et al. [Kap99] und Kaenampornpan [KA04b] (vgl. auch 5.2). Diese Ansätze sind jedoch eher einzelne Werkzeuge, als eine umfassende Methodik zum Umgang mit Kontext im Designprozess.

4.2.1 Nutzungsschwierigkeiten von Mobiltelefonen

Die schlechte Gebrauchstauglichkeit ist heute eines der größten Aktzeptanzprobleme für mobile Anwendungen [RG05]. Neben dem Problem, dass der Nutzungskontext bei der Verwendung mobiler Geräte kaum stabil ist, ergeben sich eine Reihe weiterer Probleme für die Entwicklung gut benutzbarer mobiler Geräte und Anwendungen:

Annahmen über den Nutzer einer mobilen Anwendung oder eines Gerätes können häufig nicht gemacht werden, da die Gruppe der Nutzer mobiler Geräte sehr heterogen ist [LK06]. Mobile Geräte verfügen derzeit über einen sehr schnell wachsenden Funktionsumfang bei gleichzeitigen extrem kurzen Entwicklungs- und Lebenszyklen, so dass traditionelle auf Benutzerbeobachtung und Prototyping bedachte Usability-Methoden in den meisten Fällen kaum angewendet werden[6] [LK06].

Die größten Schwierigkeiten dürften sich jedoch aus den Formfaktoren, vor allem der geringen Größe der Geräte ergeben [LK06]. Den Geräten steht nur eine sehr begrenzte Fläche für visuelle Ausgaben zur Verfügung und eine ebenso kleine Fläche für Eingabegeräte. Durch die zunehmende Verwendung von Touchscreens können zwar beide Flächen fast bis auf die Gerätegröße optimiert werden, das grundsätzliche Problem bleibt jedoch bestehen. Hinzu kommt, dass die Geräte üblicherweise in einer Hand gehalten werden und somit lediglich eine freie Hand, und ggf. noch ein Finger der anderen, für die Bedienung verbleibt. Darüber hinaus sind Layout und Größe von Display und Bedienelementen verschiedener Modelle teilweise stark von einander abweichend, was die Entwicklung von Benutzungsschnittstellen, die für eine große Anzahl von Geräten optimiert sind, zusätzlich erschwert. Zudem ist es, wegen des zunehmend universellen Funktionsumfangs der Geräte, meist nicht praktikabel, das Design der Geräte für eine bestimmte Aufgabe zu spezialisieren.

Ein weiterer Punkt, der die Entwicklung von gut benutzbaren, mobilen Anwendungen erschwert, ist die oft geringe Aufmerksamkeitsspanne, welche die Nutzer dem Gerät widmen [Oul05]. Aufgaben mit mobilen Geräten werden oft nebenbei erledigt, oft wird mehreren Aktivitäten simultan nachgegangen. Man stelle sich etwa einen Benutzer vor, der sich zu Fuß in einer belebten Straße auf dem Weg zu einer Verabredung befindet, dabei die Person, die er treffen möchte telefonisch über eine Verspätung informiert und gleichzeitig das Navigationssystem seines Mobiltelefons zur Wegfindung nutzt. Diese Problematik verschärft die Schwierigkeit, die sich aus dem geringen Platz für Ausgabe und Eingabe weiter, da auch die Möglichkeiten in der zeitlichen Dimension knapp bemessen sind.

Eine Strategie um diesen Problemen entgegenzutreten ist die Verwendung von Kontextinformationen. Durch diese können die Eingabeerfordernisse an den Nutzer und dessen Informationsüberfrachtung reduziert werden.

[6] Dieser Umstand könnte die von Coursaris und Kim beobachtete geringe Anzahl empirischer Studien zu mobile Usability [CK06] erklären.

4.2.2 Potentiale von Kontextadaption

Schmidt beschreibt die Nutzungsmöglichkeiten von Kontext als implizite Mensch-Maschine-Interaktion. Analog zu impliziter Kommunikation zwischen Menschen, die ohne direkte Aussagen auskommt, müssen im Gegensatz zu expliziter Mensch-Maschine-Interaktion keine direkten Anweisungen an ein System formuliert werden, damit es den Bedürfnisse des Nutzers entsprechen kann [Sch00]. Schmidt weist darauf hin, dass in den meisten Fällen eine Mischung von impliziter und expliziter Interaktion erfolgen kann. Die Information, die aus dem Kontext für implizite Interaktion gewonnen werden kann, sollte, wo möglich, genutzt werden. Dort, wo Unklarheiten herrschen oder weiter Informationen benötigt werden, kann auf explizite Kommunikation zurück gegriffen werden.

Die Möglichkeiten, Kontext automatisch zur besseren Anpassung einer Software an die Situation zu nutzen, lassen sich in drei Bereiche gliedern: Anpassung der Interaktion, Anpassung des Inhalts und Anpassung der Präsentation [JÖ7]. Bei der Anpassung der Interaktion kann eine der Situation angemessene Interaktionsmetapher gewählt werden. So können bei multimodalen Interaktionsmöglichkeiten sinnvolle Eingabemodalitäten gewählt werden (z.B. Wechsel von sprachbasierter auf gestenbasierte Eingabe). Der angezeigte Inhalt kann an den Kontext angepasst werden, entweder, indem bestimmte Informationen aus- bzw. eingeblendet werden (z.B. Ausblenden geschlossener Attraktionen in einem Touristenführer) oder nützliche Empfehlungen und Hinweise zur Situation gegeben werden. Die Präsentation der Informationen kann angepasst werden, indem ein geeignetes Ausgabemedium gewählt wird, der Inhalt ggf. an das neue Medium adaptiert wird (z.B. Displayrotation beim iPhone) und indem Umweltbedingungen berücksichtigt werden (z.B. automatisches Angleichen von Lautstärke oder Kontrast).

Cheverst et al. weisen auf weitere Möglichkeiten hin. So können die Erfordernisse zur Spezifikation einer Aufgabe, die eine Software übernehmen soll, kontextabhängig reduziert werden [CDME01]. Auf niedriger Ebene bedeutet dies beispielsweise das Übernehmen bestimmter Attribute aus dem Kontext (z.B. die Position des Nutzers zum Auffinden des nächsten Restaurants) statt deren expliziter Eingabe. Auf höherer Ebene kann versucht werden, weitere Aufgaben komplett aus dem Kontext zu prognostizieren und selbständig auszuführen. Darüberhinaus können die Anforderungen an den Nutzer reduziert werden, wenn Teilaufgaben unter Berücksichtigung des Kontextes selbständig von einer kontextadaptiven Anwendung ausgeführt werden. Die Routenplanung in einem kontextadaptiven Museumsführer, welche nur geöffnete Attraktionen berücksichtigt, ist ein Beispiel für eine solche kontextadaptive Ausführung einer Teilaufgabe. Schmidt nennt weitere Möglichkeiten, so können die Störungen durch Anwendungen reduziert werden, indem Hinweise und Eingabeaufforderungen zu einem passenden Zeitpunkt übermittelt werden oder ganz entfallen, weil diese überflüssig geworden sind [Sch00]. So braucht der Nutzer z.B. nicht an einen Termin erinnert werden, wenn er bereits vor Ort ist. Schmidt unterscheidet zudem zwischen Möglichkeiten der Eingabereduktion durch Einschrän-

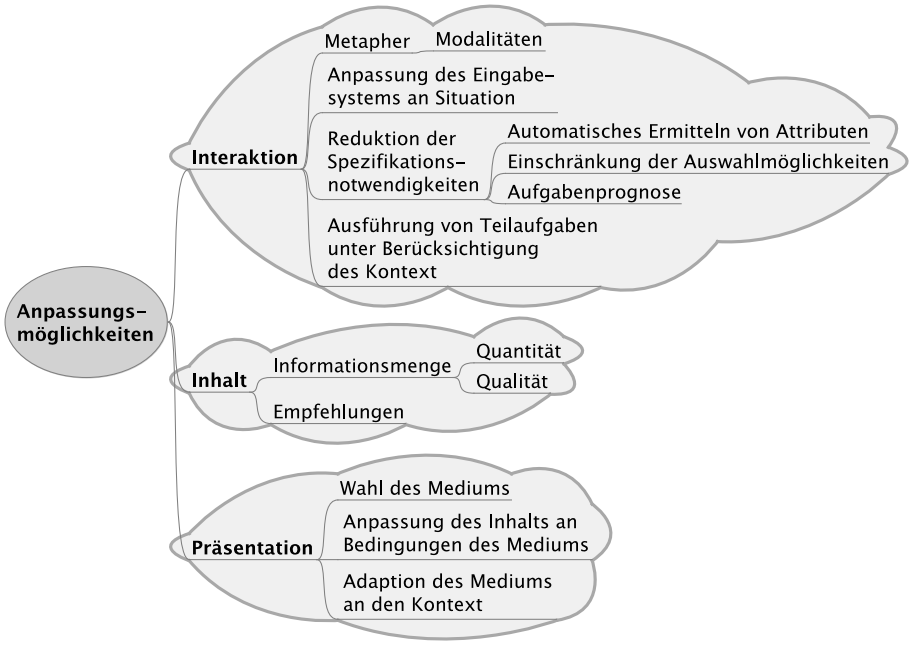

Abbildung 4.5: Möglichkeiten der Nutzung von Kontext.

kung der Auswahlmöglichkeiten auf Basis von Kontext und der komplett automatischen Ermittlung von Attributen. Eine Zusammenfassung der Möglichkeiten ist in Abbildung 4.5 dargestellt.

4.2.3 Gefahren von Kontextadaption

So wünschenswert automatische Kontextadaption in manchen Fällen sein kann, so unangenehme Folgen kann diese unter Umständen auch haben.

Ein eher harmloseres, aber illustratives, Beispiel hierfür ist die automatische Rotation der Anzeige beim iPhone (vgl. 4.1.1). Diese Funktionalität ist in vielen Fällen vorteilhaft, in einem häufigen Fall jedoch nicht: Wenn iPhone-Besitzer ihr Gerät seitlich liegend, etwa zu Hause im Bett, verwenden, passt sich die Ausrichtung des Inhalts der Schwerkraft und nicht der Position des Nutzers an. Der Displayinhalt ist somit um 90 Grad zum Kopf des Nutzers verdreht. Dreht der Nutzer das Gerät nun zurück, passt sich der Inhalt erneut an und ist wieder um 90 Grad gedreht. Eine Deaktivierung dieser Eigenschaft war nicht möglich und konnte nur über eine Software eines Drittanbieters erreicht werden[7] [Tec09].

Cheverst nennt ein anderes drastischeres Beispiel. Ein Anti-Blockiersystem (ABS) ist im Prinzip ein kontextadaptives System, da es das Bremsverhalten eines Autos an die Straßenverhältnisse anpasst und eine Totalblockierung der Räder und den Verlust des Straßengriffs verhindert. Wird das Auto jedoch von einem Fahrer gesteuert, der das fah-

[7] Eine andere Strategie das Problem zu umgehen, ist eine Drehung um weitere 90 Grad in die selbe Richtung, da das Gerät keine 180 Grad Drehung beherrscht und somit in der letzten Konfiguration verblieb.

ren ohne ABS gewohnt ist und manuell das Verhalten des ABS simuliert, kommt es zu längeren Bremswegen, da in der Kombination von Bremsverhalten des Fahrers und ABS die Räder zu wenig blockiert werden [CDME01].

In diesem Beispiel wird klar, wie fatal es sein, kann wenn das mentale Modell, das der Nutzer von einem System hat, nicht zum tatsächlichen Verhalten eines Systems passt. Cheverst nennt weitere Bereiche, in denen kontextadaptives Verhalten zu Problemen führen kann: Wenn System und Nutzer beide Verhalten an den Tag legen um den Kontext zu berücksichtigen, kann es sein, dass sich beide Maßnahmen gegenseitig aushebeln (siehe iPhone) und das System dabei in keinen stabilen Zustand kommt. Generell besteht bei autonomem Verhalten eines Systems immer ein potentieller Konflikt zur Freiheit des Nutzers. Im Falle des iPhones hätte eine einfache Möglichkeit die Anzeigerotation zu unterbinden völlig ausgereicht um das Problem zu verhindern. Als weiteren Punkt nennt Cheverst die Schwierigkeit komplexen, kontextadaptiven Systemen zu vertrauen. Selbst wenn ein solches System in den allermeisten Fällen sinnvolles Verhalten an den Tag legt, ist es aus Nutzersicht sinnvoll diesem System nicht zu vertrauen, wenn es zu gelegentlichen, aber fatalen Fehlern kommt. Eine kontextadaptive Kalenderanwendung, die selbst entscheidet wie und wann sie den Nutzer an Termine erinnert, kann beispielsweise prinzipiell sinnvoll sein. Wenn durch das Verhalten des Systems jedoch nur ein wichtiger Termin verpasst wurde, weil eine unangemessene Entscheidung getroffen wurde, so ist dies nicht akzeptabel.

Malaka nennt typische Usability-Ziele, die zu aktivem Systemverhalten potentiell im Widerspruch stehen [Mal07]:

Steuerbarkeit Der Nutzer will im Allgemeinen die Kontrolle über Software behalten und nicht von dieser kontrolliert werden, eine gewisse Autonomie der Software ist für automatische Anpassung an den Kontext jedoch notwendig.

Unterstützung mentaler Modelle Zum Verständnis von Computersystemen bilden Nutzer mentale Modelle. Komplexes Systemverhalten kann die Ansprüche an den Nutzer verringern, aber auch die Herausbildung adäquater mentaler Modelle erschweren.

Vorhersagbarkeit Bei Systemen, die hochgradig adaptiv und autonom sind, kann es zu dem Problem kommen, dass die Auswirkungen von Handlungen des Nutzers durch diesen kaum noch vorhergesagt werden können.

Transparenz Wenn sich ein System an alle möglichen Faktoren anpasst, ist sein Zustand kaum noch zu erfassen.

Lernförderlichkeit Ein System, das sich in verschiedenen Situationen unterschiedlich verhält, ist schwer zu durchschauen. Mentale Modelle lassen sich nur schwer entwickeln, wenn das System keine Rückmeldung über seinen Zustand und die Gründe für bestimmtes Verhalten angibt.

Einen Königsweg wie mit diesen Konflikten umgegangen werden kann, gibt es sicherlich nicht. In einigen Fällen können Gegenmaßnahmen eingeführt werden, etwa indem ein System so ausgelegt wird, dass es verständliche Informationen über seinen Zustand zugänglich macht [Dou04] und es dem Nutzer eine gewisse Kontrolle belässt, indem es z.B. ermöglicht Nutzerpräferenzen für bestimmte Situationen zu spezifizieren [HI05, Abschnitt 3]. In anderen Fällen muss der Konflikt anwendungsfallabhängig entschieden werden und eine Lösung empirisch auf ihre Angemessenheit hin überprüft werden.

4.2.4 Spezifische Potentiale von aktivitätsbezogenem Kontext

Die bisherige Forschung zu kontextadaptiven Systemen hat sich auf einzelne Teilbereiche von Kontext, vorrangig auf *Location-awareness*, also der Ausnutzung von Positionsdaten [SNE] und technische Bedingungen, wie Verfügbarkeit von Diensten und Netwerkinfrastruktur, konzentriert [Sat96]. In vielen Definitionen von Kontext und seinen Teilbereichen kommt der Bereich Aktivitäten nichtmal vor [KA04b]. Und das obwohl die Aktivitäten und Ziele des Nutzers ein entscheidender Teil von Kontext sind. Denn letztlich bestimmen diese, was überhaupt für den Kontext relevant ist [Dou04] (vgl. 4.1.4). Kenntnisse der Interaktionsvorgänge, also der Aktivitäten des Nutzers mit dem System, sind eine wichtige Voraussetzung für adaptive Systeme [JÖ7].

Unter aktivitätsbezogenem Kontext oder Aktivitätskontext wird in dieser Arbeit Aktivitätskontext im Sinne der von Kofod-Petersen und Cassens vorgeschlagenen Kontexttaxonomie (vgl. 4.1.3) verstanden, also die Aktivitäten und Ziele des Nutzers (und ggf. anderer Personen). Ferner wird davon ausgegangen, dass sich die Ziele des Nutzers im Allgemeinen nicht direkt feststellen lassen. Da sich diese Arbeit nicht mit der Inferenz von Nutzerzielen direkt beschäftigt, geht es also vorrangig um die Frage, welcher direkte Nutzen aus erfassten Aktivitäten gezogen werden kann. Aktivitäten werden als Teil von Primärkontext (vgl. Abschnitt 4.1.3.3) aufgefasst und es soll untersucht werden, welche sekundären Kontextinformationen und Nutzungsmöglichkeiten Aktivitäten bereitstellen.

4.2.4.1 Beziehungen und Relevanz

Zunächst ist festzustellen, dass eine Aktivität keine Entität von Kontext ist, sondern vielmehr eine Beziehung zwischen Entitäten [RG05, S. 116]. Dies ergibt sich auch aus der Betrachtung der Kontextdefinitionen von Zimmermann bzw. Dey (vgl. 4.1.2). Aktivitäten stellen Beziehungen zwischen Entitäten her.

Dabei lassen sich drei Arten von Beziehungen zwischen Entitäten unterscheiden:

- Beziehungen durch gemeinsame Beteiligung an Aktivitäten
- Beziehungen durch zeitliche Nähe
- Beziehungen durch räumliche Nähe

Beziehungen durch gemeinsame Beteiligung an Aktivitäten sind funktionale Beziehungen im Sinn von Zimmermann (vgl. 4.1.2). Beziehungen, die durch zeitliche bzw. räumliche Nähe entstehen, entsprechen nicht zwingend unmittelbaren Beziehungen zwischen Entitäten, können aber aufschlussreich für bestimmte Nutzungsmuster sein.

Aktivitäten bestimmen die Relevanz von Entitäten [Dou04]. Eine Methode, um diese Relevanz festzustellen, ist das Erfassen und die Analyse der durch Aktivitäten hergestellten Beziehungen [GF03]. Die durch Aktivitäten entstandenen Beziehungen, lassen sich somit verwenden, um Prognosen für die Relevanz von Entitäten für den Nutzer oder die Relevanz von Entitäten jeweils für einander zu machen.

4.2.4.2 Aufmerksamkeitsfokus

Die Aktivitäten des Nutzers bestimmen, worauf sich sein Aufmerksamkeit richtet. Durch Kenntnis der Aktivität lässt sich also der Aufmerksamkeitsfokus im Allgemeinen und ggf. die Aufmerksamkeit für bestimmte Entitäten im Speziellen bestimmen.

4.2.4.3 Alternative Aktivitäten bei Blockierungen

Sollten bestimmte Aktivitäten oder Aktionen nicht möglich sein, könnte ein kontextadaptives System automatisch Alternativen vorschlagen. Ruft beispielsweise ein Nutzer eine bestimmte Person an und diese ist nicht erreichbar, könnte das System prüfen, ob diese Person momentan über einen Sofortnachrichtendienst erreichbar ist und ggf. anbieten diesen Kanal zu öffnen oder das Versenden einer E-Mail vorschlagen.

4.2.4.4 Prognose von Aktivitäten und Zielen

Das Beobachten von Aktivitäten kann Grundlage für die Analyse von Tätigkeitsmustern sein und auf dieser Basis Grundlage für die Prognose von zukünftigen Aktivitäten oder aktuellen Zielen des Nutzers sein. Im Bereich des User Modelling, bei dem vom beobachteten Verhalten auf Ziele geschlossen werden soll, ist das Erfassen von Nutzeraktivitäten ein zentrales Problem [HBH+98]. Aber auch ohne explizites Nutzermodell können Muster analysiert und Vorhersagen getroffen werden.

4.3 Zusammenfassung

Die Gebrauchstauglichkeit mobiler Anwendungen wird durch ihren Nutzungskontext beeinflusst. Die Adaption mobiler Anwendungen an den aktuellen Kontext zur Laufzeit, kann helfen, diese Einflüsse zu kompensieren und Eingabe- und Ausgabeschwierigkeiten, die sich aus den Eigenschaften mobiler Geräte ergeben, auszugleichen. Bei den Strategien zur aktiven Anpassung einer Anwendung an eine Situation, muss darauf geachtet werden, dass das System für den Nutzer kontrollierbar, verlässlich und beherrschbar bleibt.

Innerhalb der verschiedenen Aspekte von Kontext stellen Aktivitäten einen zentralen Teil von Kontext dar, der bisher kaum explizit ausgenutzt wurde. Die Möglichkeiten der Nutzung liegen in der Analyse der Beziehungen, die durch Aktivitäten hergestellt werden, der darauf aufbauenden Einschätzung der Relevanz von Entitäten für den Nutzer, der Bestimmung des Aufmerksamkeitsfokus' des Nutzers und der Ableitung von aktuellen Zielen und zukünftigen Tätigkeiten des Nutzers. Sollten bestimmte Aktivitäten des Nutzers nicht erfolgreich zum Ziel führen, können Alternativen vorgeschlagen werden.

5 Kontextmodelle

Es existieren verschiedene Ansätze, Kontextinformationen zu modellieren und formal zu repräsentieren. Um eine leichte Austauschbarkeit von Kontextinformationen zu ermöglichen, ist eine einheitliche Art und Weise anzustreben, wie diese repräsentiert werden. Dieses erhöht außerdem die Wiederverwendbarkeit einzelner Komponenten von kontextadaptiven Systemen, wie z.B. den Sensoren, für Kontextdaten. Große kontextadaptive Systeme mit heterogenen Quellen für Kontextinformationen sind ohne eine solche einheitliche und klar definierte Auffassung von Kontext kaum denkbar.

In diesem Abschnitt sollen verschiedene Ansätze zur Kontextmodellierung vorgestellt und, soweit möglich, miteinander verglichen werden. Da der Begriff Kontextmodell in der Literatur nicht einheitlich verwendet wird, ist hierfür zunächst eine Begriffsklärung notwendig (Abschnitt 5.1). Dabei werden die Begriffe des Kontextdatenschemas und Kontextmodells gegeneinander abgegrenzt und zwischen konzeptueller Ebene und Implementationsebene bei der Kontextmodellierung unterschieden.

Methodische Ansätze bieten Vorgehensweisen und Werkzeuge um für spezifische Anwendungsszenarien zu einem Kontextdatenschema zu gelangen und werden in Abschnitt 5.2 vorgestellt. Sie können konzeptuelle Kontextmodelle sowie Wege zur Erstellung von konzeptuellen Kontextdatenschemata und darauf aufbaueneden Implementationsschemata beinhalten.

Implementationsmodelle bieten konkrete Strukturen an, in denen sich Kontextinformationen im Allgemeinen darstellen und austauschen lassen. Das weitreichendste Unterscheidungskriterium unterschiedlicher Ansätze ist dabei die Datenstruktur, in der die Kontextinformationen erfasst werden. Zur systematischen Betrachtung werden zunächst Bewertungskriterien für Implementationsmodelle erörtert (Abschnitt 5.3.1) und anschließend eine Übersicht über bisherige Ansätze gegeben (Abschnitt 5.3.2ff).

In den weiteren Abschnitten wird auf unterschiedliche Möglichkeiten zum Umgang mit Metadaten (Abschnitt 5.3.9.1), der Erfassung von zeitlichen Abläufen und vergangenen Daten mittels einer Kontexthistory (Abschnitt 5.3.9.2), der Formulierung von Einschränkungen der möglichen Datenwerte und Integritätsbedingungen für den Kontextdatenbestand (Abschnitt 5.3.9.3) und schließlich auf den Umgang mit mehrdeutigen, unvollständigen oder widersprüchlichen Daten (Abschnitt 5.3.9.4) eingegangen. Das Kapitel endet mit einer vergleichenden Bewertung unterschiedlicher Implementationsmodelle anhand der aufgestellten Kriterien (Abschnitt 5.4).

Übersichten über den Forschungsstand finden sich auch bei Chen/Kotz [CK00], Strang/Linnhoff-Popien [SLP04] und Hartmann/Austaller [HA07].

5.1 Begriffsklärung

Während es unzählige Versuche gibt, den Begriff „Kontext“ zu definieren, konnte im Rahmen dieser Arbeit keine explizite Definition des Begriffs „Kontextmodell“ in der Literatur gefunden werden. Dabei wird dieser Begriff in der Literatur bisher leider auf sehr unterschiedliche Art und Weise verwendet.

Einige Autoren verstehen unter dem Begriff das Implementationsschema der kontextrelevanten Daten für eine Anwendung. Also die Angabe von Informationen, ihrer Darstellungsweise und Verarbeitungsregeln für eine bestimmte kontextadaptive Anwendung oder einen Anwendungsbereich. Wird beispielsweise in einer Applikation, die auf das Wetter reagiert, das Wetter durch Temperatur, gemessen in Celsius, und Angabe der relativen Luftfeuchtigkeit bestimmt sowie die Regel aufgestellt ab welcher Temperatur und Luftfeuchtigkeit das Wetter als schwül gilt, so wäre dieses ein Kontextmodell. Diese Verwendung des Begriffs findet sich häufig bei Arbeiten, die konkrete kontextadaptive Systeme beschreiben [HI05].

Andere Autoren bezeichnen mit Kontextmodell die darüberliegende Abstraktionsebene: Die Datenstrukturen und Operationen, die verwendet werden um mit den Kontextdaten zu arbeiten. Würden die eben genannten Daten beispielsweise als Fakten und Konzepte in einer Ontologie hinterlegt sein, würde man in dieser Verwendungsweise des Begriffs von einem ontologischen Kontextmodell sprechen. Dieses Begriffsverständnis liegt meist Arbeiten zugrunde, die unterschiedliche Arbeiten zur Kontextmodellierung vergleichend bewerten [SLP04] [HA07].

Diese unterschiedlichen Begriffsbedeutungen entsprechen dem, was im Gebiet der Datenmodellierung für Datenbanksysteme jeweils mit Datenmodell bzw. Datenbankschema bezeichnet wird [KE04, S. 21f]. Um begriffliche Klarheit zu schaffen und um verschiedene Ebenen der Kontextmodellierung schärfer voneinander zu trennen, wird im Folgenden eine eigene Definition der Begriffe *Kontextdatenschema* (Abschnitt 5.1.1) und *Kontextmodell* (Abschnitt 5.1.2) gegeben, die sich an den entsprechenden Begrifflichkeiten in der Datenbankwelt orientiert.

5.1.1 Kontextdatenschata

Eine kontextadaptive Anwendung muss mit Daten umgehen, die von Hard- oder Softwaresensoren erfasst wurden und die für die Interpretation des Kontextes genutzt werden können. Um diese Daten zu hinterlegen, werden Speicherstrukturen festgelegt, in denen die Sensordaten gespeichert und verarbeitet werden können. Diese Speicherstrukturen stellen eine Abstraktion des beobachteten Weltausschnittes dar. Die Struktur, nach der diese Daten in Datenobjekte gegliedert, abgespeichert und verarbeitet werden können, ist der grundlegende Teil des Kontextdatenschemas.

Zusätzlich zur Struktur der möglichen Datenobjekte spezifiziert ein Kontextdatenschema, welche Beziehungen, wie z.B. „... ist in der Nähe von ...“ oder „... gehört zu ...“, zwi-

schen einzelnen Einheiten von Kontextinformationen erfasst werden und durch welche Strukturen dies geschieht.

Darüberhinaus können notwendige oder optionale Metainformationen, z.B. Zeitstempel, zur Ermittlung der Aktualität von erfassten Sensordaten, festgelegt werden, welche die Verwaltung von Kontextinformationen erleichtern.

Zusätzliche Möglichkeiten können Mechanismen zum Umgang mit unvollständigen oder widersprüchlichen Daten oder Strukturen zur Formulierung von Einschränkungen und Integritätsbedingungen für die Kontextdaten sein.

Für softwaretechnische Prinzipien, wie z.B. der Kapselung und dem Verstecken von Detailinformationen zum Erreichen höherer Abstraktionsgrade, können Konstrukte auf der Ebene des Kontextmodells existieren. [HA07, S.243-246].

Definition 1 (Kontextdatenschema) *Ein Kontextdatenschema ist eine formale Spezifikation von Datenobjekten und deren Beziehungen untereinander, welche den kontextabhängigen Teil des Verhaltens eines kontextadaptiven Systems bestimmen.*[1] *Diese Datenmengen umfassen jene, welche die beobachteten Kontextinformationen repräsentieren und solche, die das Welt- bzw. Domänenwissen bilden, das zur Interpretation der Kontextdaten unmittelbar benötigt wird. Ein Kontextdatenschema umfasst mindestens:*

- *Eine formale Spezifikation der Daten, die zur Verarbeitung von Kontextinformationen erfasst werden und in welchen Strukturen diese als Datenobjekte hinterlegt werden.*

Darüberhinaus kann das Kontextdatenschema weitere Informationen enthalten:

- *Regeln zur Ableitung von weiteren Daten anhand der Datenbasis.*
- *Spezifikationen der Beziehungen, die zwischen Datenobjekten erfasst werden.*
- *Angaben darüber welche Metainformationen zu den Datenobjekten hinterlegt oder generiert werden.*
- *Formulierungen von Integritätsbedingungen für die Datenbasis.*
- *Strategien zum Umgang mit widersprüchlichen oder unvollständigen Daten.*

Analog zu einem Datenbankschema kann ein Kontextdatenschema, als *konzeptuelles Schema* oder als *Implementationsschema*[2] spezifiziert werden. Das konzeptuelle Schema ist ein Zwischenschritt von der Beschreibung des relevanten Kontextes in der realen Welt zu seiner maschinellen Repräsentation. Auf dieser Ebene orientiert sich die Notation an der Lesbarkeit durch Menschen und kann Grundlage für ein Implementationsschema sein.

[1] Der Begriff „Schema“ wird in vergleichbarer Weise auch von Henricksen et al. verwendet [HI05] dort jedoch nicht in Abgrenzung zum Begriff des Kontextmodells wie in dieser Arbeit. Vielmehr wird bei Henricksen der Begriff „Kontextmodell“ synonym zu „Schema“ verwendet.

[2] Im Bereich der Datenbanken ist der Begriff „logisches Schema“ gebräuchlicher [KE04, S. 23f]. Dieser Begriff wird hier jedoch vermieden, um Verwechselungen mit Kontextadaptivensystemenen, die auf formalen Logiken aufbauen (vgl. Abschnitt 5.3.7), zu vermeiden.

Das Implementationsschema hat bereits einen so formalen Grad, dass es direkt im kontextadaptiven System Verwendung findet. In den bisherigen Arbeiten zur Kontextmodellierung wird in der Regel keine starke Unterscheidung zwischen konzeptueller und logischer Ebene gemacht.

Diese Definition wurde in Anlehnung an den Begriff Datenbankschema, wie er im Bereich des Datenbankentwurfs [KE04] benutzt wird, entwickelt. Man könnte also auch davon sprechen, dass das „Kontextdatenschema“ derjenige Teil eines Datenschemas ist, der die Daten, welche als Kontext verstanden werden und solche die zu ihrer Interpretation benötig sind, abbildet.

5.1.2 Kontextmodell

Während ein Kontextdatenschema einen bestimmten Anwendungsbereich umschreibt und abstrakt repräsentiert, bezeichnet ein Kontextmodell eine allgemeine, generische Methode, mit deren Mitteln Kontext modelliert werden kann. Also jene formalen Mittel, in denen ein Kontextdatenschema formuliert wird. Diese Art der Verwendung des Begriffs entspricht der Verwendung in einigen vergleichenden Studien zu Kontextmodellen [SLP04] [HA07].

Definition 2 (Kontextmodell) *Ein Kontextmodell bezeichnet eine Menge von Methoden und formalen Sprachen, die zur Erstellung, Darstellung und Interpretation eines Kontextdatenschemas eingesetzt werden.*

Entsprechend dem vorherigen Abschnitt kann zwischen einem *konzeptuellen Kontextmodell* zur Spezifizierung eines konzeptuellen Datenschemas und einem *Implementationsmodell*[3] zur Formulierung eines Implementationsschemas unterschieden werden.

Charakteristisches Merkmal eines Kontextmodells sind oft die, für die Beschreibung der Struktur der Daten im Implementationsschema verwendeten Formalismen, da diese in engem Zusammenhang mit der erwarteten Struktur von Kontexdaten und ihrer späteren Verwendung stehen. Werden beispielsweise Ontologien zur Kontextmodellierung eingesetzt, impliziert dies ein Verständnis der Verwendung von Kontext, in dem die Kontextnutzung vornehmlich ein Problem der Wissensrepräsentation darstellt.

Bei der Entwicklung von Kontextmodellen und Schemata ist einerseits eine möglichst hohe Generizität und ein entsprechender Abstraktionsgrad anzustreben, um eine optimale Wiederverwendbarkeit zu erreichen. Andererseits ist die für eine bestimmte Gruppe von Anwendungen beste Form eines Kontextmodells stark vom Typ und Umfang des modellierten Kontextes und der geplanten Art und Weise der Verwendung der Kontextinformationen abhängig. Dies führt zu einem gewissen von Ansätzen, das sich allerdings nicht notwendigerweise widerspricht, sondern teilweise durch Möglichkeit zur Bildung von hybriden Modellen oder der Mechanismen zur Überführung unterschiedlicher Kontextmodelle ineinander ergänzt wird.

[3] Im Bereich der Datenbankwelt ist hier der Begriff „Logisches Modell“ gebräuchlicher. Vgl. Fußnote 2

5.2 Methodische Ansätze zur Kontextmodellierung

Das Ziel methodischer Ansätze ist nicht in erster Line ein bestimmtes Kontextmodell oder generisches Kontextdatenschema zu entwickeln. Stattdessen befassen sie sich mit der Erarbeitung von Methoden, Werkzeugen und einem systematischen Verständnis und Vokabular für kontaxtadaptive Anwendungen. Grundlage hierfür sind meist Ansätze, Überlegungen und Theorien aus Philosophie [KPC06], Psychologie [KPC06] [KA04b] [KN96] oder Software-Engineering [HI05].

5.2.1 Context Modeling Language (CML)

Ausgehend von der Kritik, dass die bisherigen Ansätze, die sich auf Schaffung von Infrastrukturen zur Sammlung, Verwaltung und Verbreitung von Kontextdaten konzentrierten, oft wenig natürlich, informal und in ihrer Ausdrucksmächtigkeit eingeschränkt sind, schlagen Henricksen/Indulska (2005) [HI05] beispielsweise einen Ansatz zur grafischen Modellierung von Kontextinformationen als Teil ihrer Entwicklungsmethodik vor. Diese *Context Modeling Language (CML)* genannte, grafische Notation ist eine durch Henricksen et al. entwickelte Erweiterung von *Object Role Modeling (ORM)* [Hal01]. Diese Notation ist in Abbildung 5.1 dargestellt. Es wird zwischen den Konzepten Entitäten und Fakten unterschieden. Entitäten, z.B. Personen, Orte oder Geräte werden ovalförmig dargestellt. Fakten, wie z.B. Person A befindet sich an Ort B, stehen stets in Beziehung zu ein oder mehreren Entitäten und werden durch Rechtecke symbolisiert. Sie können weitere grafische Auszeichnungen erhalten, um bestimmte Eigenschaften zu charakterisieren. So gibt es z.B. eine Notation für Faktentypen, die potentiell zueinander im Widerspruch stehen (Kennzeichnung durch ein *a*), eine weitere Notation für Faktentypen, die einen zeitlichen Verlauf aufweisen und über eine Historie verfügen (gekennzeichnet durch eckige Klammern *[ ]*) sowie die Möglichkeit die Quelle von Faktenwissen zu charakterisieren. Für letzteres stehen die Kennzeichnungen statisch (symbolisiert durch den Buchstaben *s*), auf Beobachtung durch Sensoren beruhend (ausgezeichnet durch eine gezackte Linie) und Schlussfolgerung (Dargestellt durch ein Sternchen *) zur Verfügung. Darüberhinaus können auch elementare Einschränkungen, wie z.B. Einmaligkeitseinschränkungen für bestimmte Werte und Abhängigkeiten, modeliert werden. Strategien wie z.B. mit widersprüchlichen Daten oder der Verletzung von Integritätsbedingungen umgegangen werden soll, lassen sich jedoch nicht formulieren.

5.2.2 Aktivitätstheorie-orientierte Ansätze

Die Aktivitätstheorie (vgl Abschnitt 3.2) wurde von verschiedenen Forschergruppen als methodischer Rahmen für die Entwicklung von kontextadaptiven Anwendungen vorgeschlagen. Kaptelinin, Nardi und Macaulay (1999) haben aufbauend auf der Aktivitätstheorie einen Fragebogen entwickelt, der Entwurf und Evaluation von kontextadaptiven Anwendungen unterstützt [KN96].

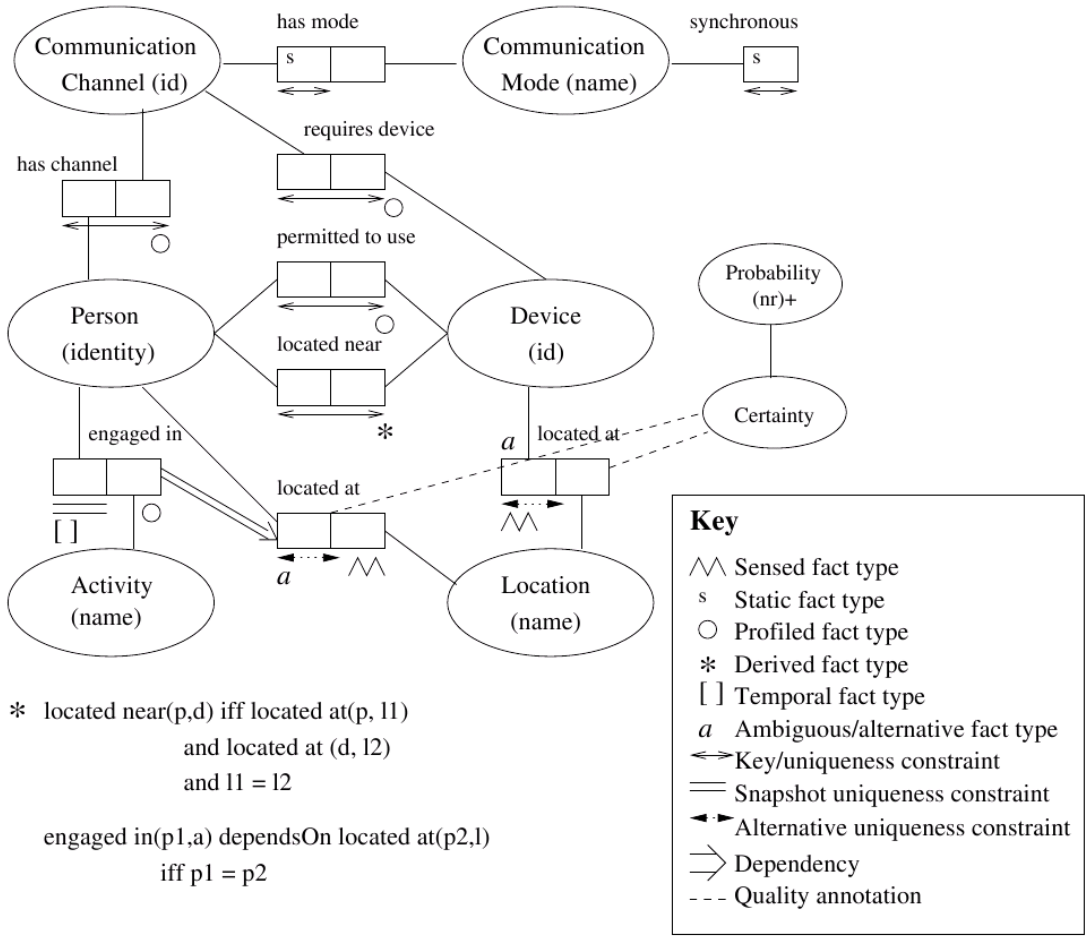

Abbildung 5.1: Modelierung von Kontextdaten mit Hilfe der Context Modeling Language [HI05].

Kaenampornpan und Eamon nutzen die Grundstruktur von Aktivitäten nach der erweiterten Aktivitätstheorie (vgl. 3.2) als Ausgangspunkt, um in einem Top-Down-Entwurf relevante Kontextinformationen identifizieren zu können [KA04b].

Einen ähnlichen Ansatz verfolgen Kofod-Petersen und Cassens. Sie ordnen, wie Bereits in Abschnitt 4.1.3 beschrieben, einzelnen Aspekten dieser Grundstruktur zunächst bestimmte Teilbereiche von Kontext zu (vgl. Abbildung 4.3). Dieses Modell wird von den Autoren in ihrer weiteren Arbeit als Ausgangspunkt für die Entwicklung einer detaillierten Ontologie zur Beschreibung von Kontext genutzt [KPC06].

5.3 Implementationsmodelle

Unterschiedliche Annahmen über die Struktur von Kontextinformationen und die geeignetste Datenstruktur zu ihrer Repräsentation existieren. Welche Art eines Implementationsmodells bevorzugt wird, hängt dabei sowohl von der angenommenen Struktur von Kontextinformationen im Allgemeinen als auch von der geplanten Art und Weise ihrer Verwendung ab. Für eine Anwendung die lediglich einfache Sensordaten verarbeiten muss, reichen einfache Key-Value-Modelle, in denen über bestimmte Schlüssel zugehörige Werte abgefragt werden können, völlig aus. Für Anwendungen, die automatisch komplexere Schlüsse und Ableitungen aus Kontextdaten ziehen möchten, sind hingegen logische oder ontologie-basierte Modelle geeigneter. Für viele Fälle ist es auch möglich, diese Modelle ineinander zu überführen. So lassen sich beispielsweise aus ontologie-basierten Modellen logische Modelle ableiten oder Key-Value-Modelle leicht in Auszeichnungssprachen ausdrücken.

5.3.1 Bewertungskriterien

Ein systematischer Vergleich der bisher eingesetzten Möglichkeiten im Bereich des Ubiquitous Computing wurde von Strang und Linnhoff-Popien unternommen [SLP04]. Die Arbeit unterscheidet zwischen Key-Value-Modellen (vgl. Abschnitt 5.3.2), Modellen basierend auf Auszeichnungssprachen (vgl. Abschnitt 5.3.3), grafischen Modellen, objektorientierten Modellen (vgl. Abschnitt 5.3.4), logischen Modellen (vgl. Abschnitt 5.3.7) und ontologiebasierten Modellen (vgl. Abschnitt 5.3.8). Die Möglichkeiten der Modelle wurden vergleichend bewertet hinsichtlich ihrer Eignung für Verteilung der Kontextdaten in hochdynamischen verteielten System (a), der Möglichkeit Kontextdaten, die nur zum Teil lokal vorliegen, partiell zu validieren (b), möglicher Angaben zur Informationsqualität, wie Aktualität, Genauigkeit oder Umfangs der Informationen zu modellieren (c), der Möglichkeiten zum Umgang mit mehrdeutigen oder widersprüchlichen Informationen (d), ihres Grades der Formalität der Fakten, insbesondere in Hinsicht darauf, dass diese auch von unterschiedlichen Stellen in gleicher Weise interpretiert werden (e) und der Anwendbarkeit in real existierenden Umgebungen (f).

Dieser Vorgehensweise wird im Folgenden nur teilweise gefolgt. Linnhoff-Popien und Strang behandeln dataflow-orientierte Modelle (vgl. Abschnit 5.3.8) als Teil von objektorientierten Modellen, dabei gehen unterschiedliche Eigenschaften dieser Ansätze jedoch verloren. Grafische Modelle sind im Sinne der bereits entwickelten Begrifflichkeit konzeptuelle Modelle (siehe Abschnitt 5.1.2) und Teil eines methodischen Entwicklungsansatzes und daher nicht sinnvoll mit Implementationsmodellen zu vergleichen. Sie können allerdings dazu eingesetzt werden, relationale Implementationsmodelle (vgl. Abschnitt 5.3.6) abzuleiten [HIR03]

Die Bewertung der Anwendbarkeit der Lösungen (f) von Strang und Linnhoff-Popien wurde in Hinblick auf Ubiquitous Computing Szenarien vorgenommen. Da sich die vorliegende Arbeit stärker auf Mobile Computing am heutigen technischen Stand orientiert, sind die Anforderungen an eine effiziente Nutzung der zur Verfügung stehenden Resourcen noch bedeutender. Für eine differenziertere Betrachtung ist es sinnvoll, zwischen den Anforderungen, die an die Infrastruktur gestellt werden (Speicherbedarf, Komplexität der benötigten Softwareinfrastruktur) und der Zugriffseffizienz im Falle vom Zugriff auf größere Datenmengen zu unterscheiden. Die Verteilung von Kontextdaten ist für diese Arbeit nicht entscheidend, kann aber natürlich auch für kontextadaptive Systeme im Bereich des Mobile Computing notwendig sein. Das sehr spezielle Problem der partiellen Validierung von lokal vorliegen Teildaten (b) kann unter die Beurteilung des Ansatzes für Verteilung im Allgemeinen (a) subsumiert werden. Prinzipiell lassen sich Informationen zur Informationsqualität (c) in jedem System darstellen. Interessant ist aber, wie komfortabel, d.h. vorallem transparent, die unterschiedlichen Ansätze mit diesen Informationen umgehen können. Dieses lässt sich jedoch im Punkt Umgang mit unvollständigen und widersprüchlichen Daten berücksichtigen (d). Bei Linnhoff-Popien werden auch die Möglichkeiten der automatischen Entwicklung neuer Fakten durch Inferenz in den unterschiedlichen Ansätzen beschrieben, jedoch nicht als abschließendes Vergleichskriterum behandelt. Da aber gerade dieses einige Ansätze auszeichnet, ist es sinnvoll dieses Kriterium in einen Vergleich aufzunehmen. Verdichtung und Interpretation von Sensordaten zu höherwertigen Informationen ist ein zentrales, wiederkehrendes Problem im Bereich der kontextadaptiven Systeme. Daher ist es sinnvoll, die Eignung der unterschiedlichen Ansätze hierfür zu untersuchen.

Wie bereits in Abschnitt 5.2.1 Erwähnung fand, haben Henricksen et al. darauf hingewiesen, dass sich in vielen Ansätzen Konzepte aus der realen Welt oft nicht auf natürliche, d.h. unmittelbare und direkte Weise in ein Implementationsschema überführen lassen.Daher soll die Natürlichkeit der Abbildung von Konzepten der Realwelt in unterschiedliche Kontextmodelle als weiteres Bewertungskriterium aufgenommen werden.

Für kontextadaptive Systeme ist oft die Kentniss einzelner ggf. verdichteter skalarer Sensorwerte nicht ausreichend, vielmehr sind Kontextdaten präpositional und es können auch Beziehungen von Bedeutung sein [GS01a]. Solch eine Sichtweise auf die Daten ist beispielsweise für die Beantwortung der Frage, welche Gegenstände befinden sich in

der Nähe eines bestimmten Ortes, notwendig. Daher ist es sinnvoll, dass auch der einfache und effiziente Umgang der unterschiedlichen Modelle mit Beziehungen zwischen Entitäten als Beurteilungskriterum berücksichtigt wird.

Zusammenfassend ergeben sich folgende Kriterien:

(K1) Leichtgewichtigkeit Die zur Realisierung dieses Modells benötigte Infrastruktur sollte möglichst geringe Voraussetzungen an vorhandene Hard- und Software machen.

(K2) Zugriffseffizienz Zugriff auf Daten, einschließlich der Suche nach bestimmten Daten, sollte effizient möglich sein.

(K3) Verdichtungsmöglichkeiten Das Modell sollte einfache Mechanismen bereitstellen, automatische Verdichtungen von Low-Level-Kontextdaten zu höherwertigen Daten zu definieren.

(K4) Auflösung von Widersprüchen und Unvollständigkeiten Im Modell sollten Möglichkeiten für den Umgang mit wiedersprüchlichen oder unvollständigen Daten vorhanden sein.

(K6) Verteilbarkeit der Kontextdaten Das Modell sollte verteilte Systeme und eine Verteilung der Kontextdaten unterstützen. Idealerweise sollte es möglich sein, Verteilungstransparenz zu realiseren.

(K7) Eindeutige Interpretierbarkeit der Kontextdaten Kontextdaten sollten zwischen verschiedenen heterogenen Systemen austauschbar sein, ohne dass dieses zu Fehlinterpretationen oder Uneindeutigkeiten in den Daten führt. Gemeinsame Standards und Mechanismen, um die Semantik von Daten zu spezifizieren und eindeutig aufzulösen, sind hierfür nötig.

(K8) Erzeugen neuer Fakten durch Inferenz Es ist wünschenswert, dass im Modell, bei Bedarf, Fakten generiert bzw. abgefragt werden können, die sich durch Inferenz aus der bestehenden Faktenmenge ergeben.

(K5) Verwaltung von Beziehungen Der Umgang mit Beziehungen zwischen verschiedenen Entitäten, also deren Abbildung und Abfrage, sollte im Modell untersützt werden.

(K9) Natürlichkeit der Abbildung Die Abbildung von Konzepten der Realwelt in entsprechende Konstrukte des Kontextmodells, sollte möglichst natürlich, d.h. direkt, einfach, unmittelbar und leicht nachvollziehbar sein.

In den folgenden Abschnitten werden nun die bisher eingesetzten und untersuchten Möglichkeiten für Implementationsmodelle geschildert, wobei mit einfachen Key-Value-Modellen begonnen wird und die Möglichkeiten bis zu umfassenden ontologie-basierten Modellen, in aufsteigender Reihenfolge ihrer Komplexität geschildert werden.

5.3.2 Key-Value Modelle

Key-Value-Modelle sind die simpelste Datenstruktur um Kontextdaten verfügbar zu machen. Key-Value-Modelle bilden Schlüssel auf Werte eines elementaren Datentypes oder auf eine Liste solcher Werte ab. Abfragen können mit Hilfe regulärer Ausdrücke, die auf alle bekannten Key-Value-Paare angewendet werden, erfolgen.

Schilit [Sch95] beschreibt wie ein Key-Value-Modell verwendet wird, um in einem adaptiven System Kontextinformationen, wie in diesem Fall die Aufenthaltsorte der Nutzer, zu verwalten. Listing 5.1 zeigt, wie dieses Modell im *PARCTAB* [WSA+96] System eingesetzt wurde, um mobilen Geräten eine eindeutige Identifizierung zuzuordenen und die mit ihnen assoziierten Nutzer anzugeben.

Listing 5.1: Beispiel für ein Key-Value-Modell im *PARCTAB* System

```
1 {Name Badge:802}
2 {PublicKey "R0lGODdhDAMdAc}Yd"}
3 {With {schilit spreitzer theimer}}
```

5.3.3 Auszeichnungssprachen

Auszeichnungssprachen kommen üblicherweise dort zum Einsatz, wo Informationen über Geräte bzw. Dienstgrenzen hinweg ausgetauscht werden sollen. Ein gängiges Beispiel hierfür ist die Bereitstellung von Informationen über Geräteeigenschaften mittels eines in einer Auszeichnungssprache verfassten Profiles. Strang/Linnhoff-Popien [SLP04] geben eine Übersicht zu den verschiedenen Varianten. Diese Auszeichnungssprachen bauen typischerweise auf etablierten Standards wie die Standard Generic Markup Language (SGML) [Int86] bzw. die Extensible Markup Language (XML) [BPSM+08] auf, wodurch für ihre Verarbeitung auf weit entwickelte Werkzeuge zurückgegriffen und eine hohe Kompatibilität gewährleistet werden kann.

Listing 5.2: *Comprehensive Structured Context Profiles (CSCP)* als Beispiel für ein auszeichnungssprachen-basiertes Modell

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <rdf:RDF
3  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
4  xmlns:cscp="context-aware.org/CSCP/CSCPProfileSyntax#"
5  xmlns:dev="context-aware.org/CSCP/DeviceProfileSyntax#"
6  xmlns:net="context-aware.org/CSCP/NetworkProfileSyntax#"
7  xmlns="context-aware.org/CSCP/SessionProfileSyntax#"
8  <SessionProfile rdf:ID="Session">
9  <cscp:defaults rdf:resource=
10 "http://localContext/CSCPProfile/previous#Session"/>
11 <device><dev:DeviceProfile>
```

```
12 <dev:hardware><dev:Hardware>
13 <dev:memory>9216</dev:memory>
14 </dev:Hardware></dev:hardware></dev:DeviceProfile>
15 </device>
16 </SessionProfile>
17 </rdf:RDF>
```

Ein Beispiel für diesen Ansatz ist die Definition von *Comprehensive Structured Context Profiles (CSCP)* durch Held et al. [HBS02]. CSCP bauf auf dem *Resource Description Framework (RDF)* [BG04] Standard auf. Ein Einsatzbeispiel, das illustriert, wie durch Mechanismen der genutzten Standards Semantik verwendeter Tags näher definiert und eine einheitliche Interpretation gewährleistet wird, ist in Listing 5.2 gegeben. Die äußeren Tags definieren die benutzen Standards XML, RDF und die Zeichenkodierung. Die Namespace-Attribute (xmlns) [BHLT06] qualifizieren die inneren Tags und ihre Attribute näher und ermöglichen es so, Mehrdeutigkeiten bei der Interpretation zu vermeiden.

Da Informationen in diesen Sprachen innerhalb von potentiell verschachtelten Elementen dargestellt werden, bilden sie eine hierarchische Form und erlauben somit bereits komplexere Strukturen als Key-Value-Modelle. In beschränktem Umfang können Wertebereiche und Typen für Daten festgelegt werden.

Zusammenhänge, die sich nicht innerhalb einer hierarchischen Struktur darstellen lassen, können jedoch nur umständlich dargestellt und verarbeitet werden.

5.3.4 Objektorientierte Modelle

Objektorientierte Modelle betonen die Prinzipien von Kapselung und Wiederverwendbarkeit. Generell wird der Versuch unternommen, die Komplexität von Kontextinformationen und ihrer Verwaltung innerhalb von Objekten zu verbergen. Nach außen hin werden nur möglichst einfache Schnittstellen angeboten. Abstrakte strukturelle Beziehungen wie Generalisierung, Komposition, Aggregation und Assoziation sind in der objektorientierten Programmierung und dem zugehörigen Methodenschatz elementare Konzepte und können dementsprechend gut umgesetzt werden. Konkretere Beziehungen einzelner Objekte, wie Person A ist in der Nähe von Objekt B, müssen jedoch explizit als Eigenschaften der Objekte modelliert werden.

Diese Technik wurde beispielsweise für den im GUIDE-Projekt entwickelten kontextadaptiven Touristenführer [CMD99] eingesetzt. Zentrale Bausteine der Architektur wie z.B. der Ortungssensor und sehenswürdige Orte, als zentrale Entitäten in diesem System, werden als Objekte modelliert. Die Objekte, welche Entitäten repräsentieren, werden über wichtige Nachrichten, wie z.B. einen Ortswechsel des Nutzers, informiert und können sich gemäß der ihnen innewohnenden Logik dem entsprechenden Kontext anpassen. Da von diesen Entitäten pro Nutzer eine eigene Objektinstanz erzeugt wird, können alle relevanten Kontextinformationen und Verhaltensmuster direkt im Objekt behandelt und für die Außenwelt gekapselt werden.

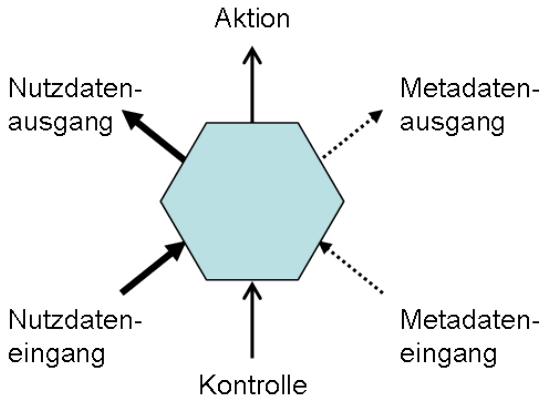

Abbildung 5.2: Nutzdaten, Metainformationen und Kontrolldaten können unterschiedliche Datenflüsse bilden [GS01a].

5.3.5 Dataflow-orientierte Ansätze

Bei Schmidt et al. [SBG99] wird unter Kontext eine Abstraktion von Sensordaten verstanden. Low-Level Sensordaten werden durch Kombinationen und Filtertechniken zu höherwertigen symbolischen Werten kombiniert. Ein bestimmter Kontext wird als gegeben betrachtet, sobald bestimmte Sensordaten für diesen Kontext definierte Wertbereiche einnehmen. Strang/Linnhof-Popienen ordnen diesen Ansatz, wohl wegen dessen Betonung von Kapselung, den objektorienterten zu. Tatsächlich lässt sich jedoch mit einiger Berechtigung von einem dataflowoorientierten Ansatz sprechen, denn im Mittelpunkt einer dataflow-orientierten Betrachtung stehen nicht Entitäten, die als Objekte modelliert werden, sondern Datenströme und ihre einzelnen Verarbeitungsschritte. Wo in einem objektorientierten Ansatz versucht wird Konzepte der Realwelt in Objekte abzubilden, werden in einem dataflow-orientierten Ansatz die einzelnen Schritte der Datenverarbeitung um bestimmte Kontextinformation zu erhalten, in ein System aus datenverarbeitenden Einheiten und Datenflüssen zwischen diesen abgebildet.

Dieser Ansatz ist mit Überlegungen von Salber et al. [SDA99] vergleichbar. Auch diesen Autoren geht es vorrangig um das Erreichen von Wiederverwendbarkeit durch das Prinzip der Kapselung. Sie schlagen vor, Kontextdaten in sogenannten Widgets, deren Name auf ihre Ähnlichkeit mit Prinzipien in der GUI-Entwicklung verweist, zu sammeln. Diese Widgets können ihrerseits kombiniert und zu neuen Widgets zusammengebaut werden. Von einem reinen Dataflow-Modell kann allerdings in diesem Fall nicht gesprochen werden, da Salber et al. auch die Notwendigkeit der Verfügbarkeit von historischen Sensordaten, die in den einzelnen Widgets bei Bedarf bereit gehalten werden, betonen. Man könnte an dieser Stelle daher eher von einem hybriden Ansatz sprechen.

Gray et al. modellieren einen kontextadaptiven Museumsführer, der die Sprache zur Darstellung von Informationen automatisch an den jeweiligen Besucher anpasst, exemplarisch als Beispiel für ein Dataflow-Modell [GS01a]. Dabei wird, wie in Abbildung

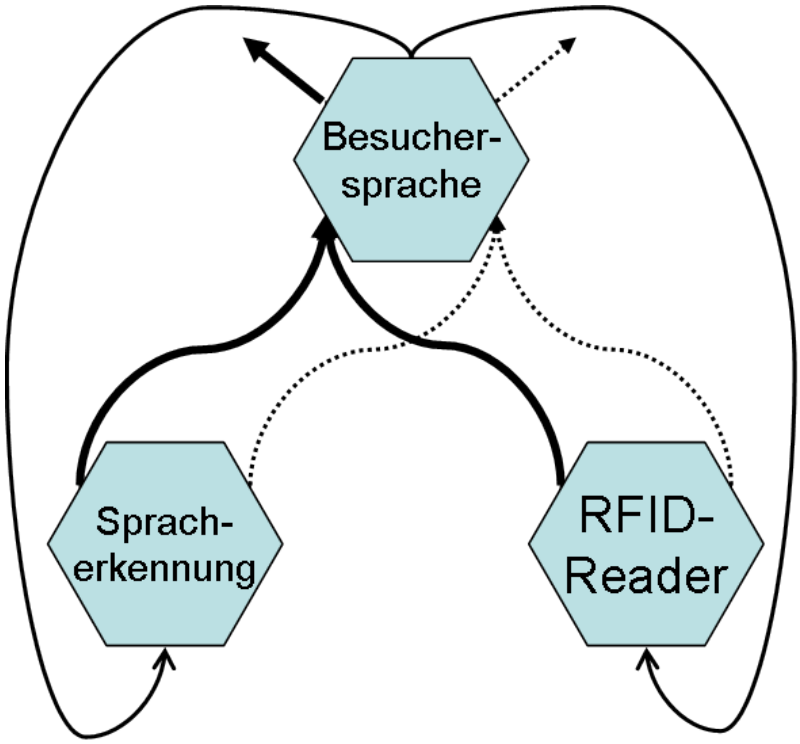

Abbildung 5.3: Sprachauswahl in einem kontextadaptiven Museumsführer als Beispiel für ein Dataflow-Modell [GS01a].

5.2 dargestellt, im Modell zwischen drei unterschiedlichen Arten von Datenflüssen, den Nutzdaten, den Metainformationen und den Steuerungsdaten unterschieden.

Abbildung 5.3 illustriert wie Informationen einer Sprachanalyse-Komponente und eines RFID-Readers genutzt und deren Ergebnisse von einer dritten Komponente verschmolzen werden. Durch dieses Vorgehen können die Schwächen einzelner Komponenten kompensiert werden. Auch Rückkopplungen sind möglich. So können, wie skizziert, Informationen über die erwartete Sprache an die Spracherkennungskomponente rückgemeldet werden, um die Ergebnisse der Spracherkennungskomponente zu verbessern.

5.3.6 Relationale Modelle

Dieses Modell entspricht dem relationalen Datenmodell, welches relationalen Datenbanken zugrunde liegt. Entitäten eines bestimmten Typs werden als Elemente einer entsprechenden Relation dargestellt. Beziehungen zwischen Entitäten werden ebenfalls durch Relationen dargestellt. Zur Realsierung eines in einem relationalen Modell formulierten Schemas kann dementsprechend auf Datenbanksysteme [KE04] zurückgegriffen werden. Die aus dem Datenbankentwurf bekannten Entwurfstechniken lassen sich auf die Entwicklung von kontextadaptiven Systemen übertragen. So ist es möglich, das in Abbildung 5.1 in CML dargestellte konzeptuelle Schema in ein relationales zu überführen [HIR03]. Entitäten werden in einem solchen relationalen Modell dann als Relation

über ihre Attribute dargestellt und Beziehungen zwischen Entitäten als Relationen über die an ihnen beteiligten Entitätstypen.

Dieser Ansatz ist besonders dort von Vorteil, wo mit einer großen Menge von gleichartigen Entitäten und Beziehungen umgegangen werden soll. Darüberhinaus kann ein Großteil der Integritäts- oder Abhängigkeitsbedingungen der Kontextinformationen als Integritätsbedingungen in einem relationalen Datenbanksystem formuliert und dort zur Laufzeit überprüft werden. Abgeleitete Kontextinformationen können über Sichten auf anderen Informationen realisiert und bestimmte Integritätsbedingungen über Datenbank-Constraints überprüft und garantiert werden [HIR03].

5.3.7 Logische Modelle

Logische Modelle bilden ein formales System aus Fakten (so kann z.B. die Aussage auto1 ist rot als rot(„auto1“) formalisiert werden), Termen, die je nach Belegung ihrer Variablen zu Wahrheitswerten evaluiert werden können (z.B.: $A \wedge B$) und Regeln, die gültige Schlussfolgerungen darstellen (z.B.: $A \rightarrow B$: Wenn A gilt folgt daraus, dass auch B gilt).

McCarthy gehört zu den Pionieren im Bereich der Kontextmodellierung mittels Logiken. Er beschäftigte sich früh mit dem Problem, eine formale Repräsentation für kontextabhängige Bedeutungen von Aussagen und kontextabhängiges Weltwissen zu finden. Ausgangspunkt für seine Forschung waren dabei nicht die Anforderungen von Ubiqitous Computing oder kontextadaptiven, mobilen Anwendungen, sondern das Bemühen, die Möglichkeiten menschlichen Verstehens und Schlussfolgerns möglichst weitgehend durch formale Logik nachzubilden [Mcc93].

McCarthy zeigt, wie kontextabhängige Interpretation von Aussagen und Weltwissen in einer formalen Logik repräsentiert werden können. Dabei wird allgemeines und kontextabhängiges Weltwissen über spezielle Regeln ausgedrückt. Eine konkrete Situation oder Aussage kann durch Fakten gegeben werden und höheres Weltwissen kann durch die gegebenen Regeln abgeleitet werden. McCarthy zeigt zwar, wie sich Kontextwechsel formal notieren lassen, wie diese festgestellt werden können bleibt jedoch offen. Daher lässt sich dieser Ansatz zwar nutzen, um Schlussfolgerungsmöglichkeiten und Weltwissen in einem bestimmten Kontext zu modellieren. Ein Modell darüber, wie Beobachtungen in der Welt in eine Repräsentation des aktuellen Kontextes überführt werden können, liegt jedoch nicht vor.

Weitere Ansätze aus dem Feld der künstlichen Intelligenz, die sich mit Schlussfolgerungen unter Berücksichtigung von Kontext befassen, finden sich bei Ghidini/Giunchigala und Akman/Surav. Ghidini/Giunchigala beschäftigen sich mit dem Problem der Repräsentation von partiellem Weltwissen und nötigen Schlussfolgerungen in partiellem Wissen zum erreichen von Zielen [GG01]. Akman/Surav nutzen eine erweiterte Form der Situationstheorie [BP83] um mit partiellem Weltwissen umzugehen. Dieses Weltwis-

sen wird im Sinne der Situationstheorie über Situationen, in denen bestimmte Eigenschaften gelten formalisiert [AS97].

Vertreter der unmittelbaren Forschung zu kontextadaptiven Anwendungen, welche die Verwendung von logischen Modellen vorschlagen, sind Román et al., Bacon et al. sowie Gray und Salber.

Im von Román et al. genutzen Kontextmodell wird Kontext durch vierstellige Prädikate repräsentiert, die in ihrer Struktur natürlichsprachlichen Sätzen, nach dem Muster „Subjekt, Prädikat, Objekt“, zusätzlich einer Angabe des Kontexttypes, ähneln [RHC+02] (siehe Listing 5.3).

Listing 5.3: Fakten im Gaia-Kontextmodell [RHC+02]

```
1 Context(<ContextType>,<Subject>,<Relater>,<Object>)
```

Aus diesen Fakten können höherwertige Informationen mit Hilfe von Ausdrücken der Prädikatenlogik erster Ordnung geschlossen werden (siehe Listing 5.4).

Listing 5.4: Beispiele für Fakten und Schlussfolgerungen in Gaia [RHC+02]

```
1 // Fakt:
2 Context(temperature,room 3221,is,32 C)
3
4 // Schlussfolgerung:
5 Context(Number of people,room 3221,>,4)
6 AND Context(Application,Powerpoint,is,running)
7 ->Context(Social Activity,Room 3221,is,Presentation)
```

Ein weiteres anwendungsorientiertes Besipiels liefern Bacon et al. mit ihrer Realisierung der Repräsentation von Ortsinformationen zur Nutzung in ortsabhängigen, mobilen Multimediaanwendungen in der logikbasierten Programmierisprache Prolog [BBH97].

Gray und Salber beschäftigen sich mit der Modellierung von Sensordaten für kontextadaptive Anwendungen und schlagen logische Formalismen wie die Prädikatenlogik erster Ordnung als geeignete und den Eigenschaften von Sensordaten entsprechende Repräsentation von Sensordaten vor [GS01a].

5.3.8 Ontologien

Ontologien sind ein innerhalb der Forschung zu „künstlicher Intelligenz“ entstandener Ansatz der Wissensrepräsentation, in denen Konzepte, allgemeine Zusammenhänge und einzelne Fakten ausgedrückt werden können [SS04]. Sie spielen u.a. eine zentrale Rolle

bei dem Bemühen, bisher unstrukturierte Informationen im Word Wide Web in maschinenlesbarer Form darzustellen und verarbeitbar zu machen, um so die Vision des *Semantic Web* [BLHL01] zu realisieren.

Innerhalb von Ontologien sind Konzepte mit ihren Attributen (z.B. ein Auto hat vier Räder, ein Kennzeichen und kann zur Fortbewegung genutzt werden) darstellbar. Es können Unterkonzepte, z.B. ein Laster ist ein Auto und hat eine Ladefläche, gebildet werden und Instanzen, z.B. es gibt ein Auto mit Kennzeichen HH-AB-1234 und Beziehungen, z.B. Peter Mustermann fährt das Auto mit dem Kennzeichen HH-AB-1234, definiert werden.

Ontologien sind eng mit logischen Modellen verwandt. Einer ihrer wichtigsten Nutzen ist die Möglichkeit des automatisierten Schließens. So können aufgrund der spezifizierten Konzepte, beispielsweise von explizit angegebenen Fakten implizite Fakten abgeleitet werden. Eine Prüfung der Faktenbasis auf Widersprüche ist ebenfalls möglich.

Ontologien ermöglichen einen leichten Austausch von Informationen. Verschiedene etablierte Sprachen und Standards zur Darstellung von Ontologien und ihrer Serialisierung existieren. So definierte das World Wide Web Consortium (W3C)[4] die *Web Ontology Language (OWL)* [SWM04] als Weiterentwicklung der Ontologien *DARPA Agent Markup Language (DAML)* [Pea02] und von *Ontology Inference Layer (OIL)* [FHH+00] unter Nutzung der Standards RDF und XML.

Ontologien wurden vielfach für den Einsatz in kontextadaptiven Anwendungen vorgeschlagen und genutzt [CFJ03] [SLPF03b].

Chen et al. entwickelten beispielsweise eine Ontologie zur Unterstützung von Präsentationen in „intelligenten Konferenzräumen“ auf Basis von OWL [CFJ03]. Diese Ontologie beschreibt die Entitäten Personen, Softwareagenten, Orte, und Präsentationsereignisse in dieser Domäne. Dabei werden insbesondere Beziehungen zwischen Orten, Rollen, die bestimmte Personen während einer Präsentation einnehmen, und typische Intentionen und Bedürfnisse dieser Personen während der Präsentation modelliert. An diesem Ansatz kann der Nutzen von Ontologien gut an der Lösung des Problems des Umgangs mit symbolischen Orten verdeutlicht werden. Bei der Modellierung von Orten in dieser Ontologie können Beziehungen wie „Ort ist Teil von“ oder „Ort überschneidet sich nicht mit“ spezifiziert werden. So ist es möglich, Räume, wie einen Konferenzraum und eine Cafeteria, zu spezifizieren, die einerseits beide Teil des Hauptgebäudes, andereseits jedoch räumlich voneinander getrennt sind. Ist eine Person nun in der Cafeteria, kann daraus automatisch geschlossen werden, dass sie auch im Hauptgebäude, aber nicht im Konferenzraum, ist.

Strang und Linnhoff-Popien entwickelten mit der *Context Ontology Language (CoOL)* [SLPF03b] eine Ontologie, die sich in zwei Ebenen aufteilt. Die Basiskonzepte sind so gewählt, dass sie sich in die gängigen Ontologiesprachen OWL/DAML+OIL (s.o.) und F-Logic [KLW95] abbilden lassen. Darauf aufbauend existiert eine Erweiterungsebene,

[4] http://www.w3c.org

welche die Ontologie auf die Integration von Service-Frameworks, insbesondere von Web-Services vorbereitet und somit gut geeignet macht für kontextabhängige Service-discovery-Mechanismen. Dementsprechend geben die Autoren als Nutzungsszenario ein System an, welches es einem Touristen in einer fremden Stadt ermöglicht On- und Offlinedienste zu finden und zu nutzen, beispielsweise um einen Ort zu finden, an dem dieser seine digitalen Fotos drucken kann [SLPF03a].

.

5.3.9 Weitere Merkmale von Kontextmodellen

Im vorherigen Abschnitt wurden die unterschiedlichen Datenmodelle erläutert, die einem Kontextmodell zu Grunde liegen können. Im Folgenden werden weitere Aspekte von Kontextmodellen vorgestellt, die Umfang und Möglichkeiten eines Modells beschreiben.

5.3.9.1 Umgang mit Metadaten

Für den adäquaten Umgang mit Kontextdaten können weitere Metadaten erforderlich sein, welche die Daten, beispielsweise durch Angaben zu deren Zuverlässigkeit, näher beschreiben.

Henricksen und Indulska klassifizieren Kontextdaten in statische, explizit angegebene aber änderbare, durch Sensoren erfasste und aus anderen Daten abgeleitete Daten [HI05]. Die explizite Angabe der Klassifikation ist bereits ein Beispiel für die Verwendung von Metadaten. Diese können dazu genutzt werden, den Umgang mit den Daten zu optimieren. So brauchen statische Daten beispielsweise nicht periodisch neu gelesen werden und abgeleitete Daten ändern sich nur wenn sich zugrundeliegenden Daten ändern.

Von besonderem Interesse sind Metadaten jedoch für Sensordaten, da diese ihrer Natur nach unzuverlässig sind und je nach Sensor unterschiedliche Qualität aufweisen können. Metadaten zur Datenqualität erlauben eine Beurteilung ob und in welchem Maße Sensordaten überhaupt noch zu berücksichtigen sind oder, falls mehrere alternative Sensoren vorliegen, welche Daten zu bevorzugen sind.

Eine detaillierte Übersicht über nützliche Metadaten für Sensordaten geben Gray and Salber (vgl. Tabelle 5.1) [GS01a].

5.3.9.2 Kontexthistorie

Für bestimmte Kontextinformationen kann ihr zeitlicher Verlauf von Belang sein, beispielsweise um Prognosen über zukünftiges Verhalten zu stellen oder um eine breitere Datenbasis, als eine aktuelle Momentaufnahme, analysieren zu können.

Kontexmodelle können dies berücksichtigen, indem sie Formalismen zum Umgang mit zeitlichen Abläufen bereitstellen. Dies können Möglichkeiten, zur Angabe in welchem Umfang zeitliche Daten von Interesse sind, sein. Etwa, wie weit eine Historie zu-

Gegenstand	Eigenschaft	Bedeutung
Repräsentation		Welches Datenformat wurde verwendet?
Datenqualität	Abdeckung	Was kann erfasst werden?
	Auflösung	Kleinste messbare Einheit.
	Genauigkeit	Mit wieviel Meßungenauigkeit ist zu rechnen?
	Wiederholbarkeit	Wie stabil ist diese Messung?
	Frequenz	Zeitliche Auflösung der Messung
	Pünktlichkeit	Zeitliche Genauigkeit der Messung
Sensorquelle	Zuverlässigkeit	Fehlerwahrscheinlichkeit des Sensors
	Zudringlichkeit	Werden durch den Sensor Störungen verursacht?
	Security und Privacy	Sicherheitseinschränkungen des Sensors
	Kosten	Welche Kosten entstehen durch die Benutzung?
	Betriebsmodi	Welche Betriebsarten können genutzt werden?
Einstellungen		Sensorenmodifikationen wie Ausrichtung oder Abschaltung

Tabelle 5.1: Metainformationen für Sensordaten nach Gray und Salber [GS01a]

rückgreift oder in welcher Frequenz Daten erwartet werden. Zur Analyse von Sequenzen, könnten Mechanismen bestehen, um Sequenzen zu spezifizieren und abzufragen.

5.3.9.3 Integritätsbedingungen

Kontextmodelle können die Angabe von Integritätsbedingungen ermöglichen. Die Möglichkeiten reichen dabei von Angaben gültiger Wertebereiche für bestimmte Daten, Eindeutigkeit von Angaben (wie z.B., dass eine Person nur an einem Ort gleichzeitig sein kann), bis zu komplexeren Ausdrücken. Sind Integritätsbedingungen angegeben, so muss das Vorgehen bei der Verletzung dieser Bedingungen ebenfalls angeben werden. Die Möglichkeiten reichen von Fehlerbenachrichtigung über automatische Zurückweisung inkonsistenter Werte bis hin zu komplexerer automatischer Fehlerbehebung.

5.3.9.4 Umgang mit Mehrdeutigkeiten, Konflikten und unvollständigen Informationen

Durch den komplexen, dynamischen und oft fehlerbehafteten Charakter von Kontextdaten kann es im Modell zu Mehrdeutigkeiten, Konflikten und unvollständigen Informationen kommen. Strategien, wie mit diesen Phänomenen umzugehen ist, können ebenfalls Teil eines Kontextmodells sein.

Henricksen und Indulska schlagen beispielsweise den Einsatz von dreiwertigen Logiken vor, um mit unsicheren Fakten umzugehen [HI05]. Wird in diesem Formalismus z.B. ein Prädikat `Tempratur` $<$ `0` ausgewertet und für Temperatur liegt keine Angabe vor,

so wird dieses Prädikat nicht zu „wahr“ oder „falsch“, sondern zu einem dritten Wahrheitswert ausgewertet der „möglicherweise wahr“ bedeutet und gemäß dieser Semantik in logischen Formeln ausgewertet wird.

5.4 Bewertung bisheriger Kontextmodelle

Im Folgenden sollen die Stärken und Schwächen der unterschiedlichen Ansätze hervorgehoben werden. Nachdem kurz die Bedeutung der methodischen Ansätze bewertet wird (Abschnitt 5.4.1), liegt der Schwerpunkt des Abschnitts in der vergleichenden Bewertung der Implementationsmodelle (Abschnitt 5.4.2). Im Anschluss an diesen Überblick wird explizit auf den Umgang mit Aktivitäten in den bisherigen Ansätzen eingegangen (Abschnitt 5.4.3). Abschließend wird kurz betrachtet, warum sich die implementationsmodell-unabhängigen Eigenschaften verschiedener Modelle, wie z.B. die genutzen Metadaten, kaum sinnvoll miteinander vergleichen lassen (Abschnitt 5.4.4).

5.4.1 Methodische Ansätze

Die vorgestellten methodischen Ansätze zielen darauf ab, detaillierte Eigenschaften von Kontextinformationen für ein spezifisches gegebenes Szenario oder eine Domäne, systematisch zu erarbeiten. Da der Ansatz dieser Arbeit möglichst allgemeingültig sein soll, lassen sich diese Ansätze nicht unmittelbar anwenden. Stattdessen wurden in den vorherigen Kapiteln systematische Vorüberlegungen angestellt, die Grundlage für die eigene Entwicklung eines Kontextmodells sind.

Die durch Henricksen et al. entwickelte Methode der grafischen Kontextmodellierung mittels der CML wird jedoch übernommen. Sie bietet gute Ausdrucksmöglichkeiten, um ein Kontextdatenschema zu entwickeln. Diese konzeptuelle Modellierungstechnik bereitet zwar die Ableitung eines relationalen Schemas vor, bindet das Kontextmodell aber nicht zwingend an eine relationale Implementation.

5.4.2 Bewertung der Implementationsmodelle

Implementationsmodelle wurden mit unterschiedlichen Szenarien im Blick entworfen und sind somit für unterschiedliche Zwecke verschieden gut geeignet. Eine Bewertung der Implementationsmodelle anhand der in Abschnitt 5.3.1 aufgestellten Kriterien soll in diesem Abschnitt vorgenommen werden.

5.4.2.1 Bewertung von Key-Value-Modellen

Der wesentliche Vorteil von Key-Value-Modellen besteht darin, dass diese minimale Anforderungen an die vorhandene Infrastruktur stellen (K1) . Der direkte Zugriff auf Daten ist effizent, aufgrund der Minimalität der Infrastruktur wird jedoch nicht unbedingt das optimierte Suchen in großen Datenmengen unterstützt (K2) .

Diese Modelle eignen sich nur für einfach strukturierte Daten. Insbesondere Beziehungen zwischen Entitäten müssen einer Entität zugeordnet werden, welche diese in einem Key/Value-Paar verwaltet. Dies ist für eine effiziente Verwaltung von Beziehungen ungünstig (K5) .

Auch Möglichkeiten wie solche zur Verteilung (K6) oder Verdichtung (K3) von Daten, zum Umgang mit Widersprüchlichkeiten (K4), zur Inferenz von neuen Fakten (K8) und zur Unterstützung der Plattformunabhängigkeit der Daten (K7), werden in diesem Ansatz nicht direkt unterstützt. Die Abbildung von Auschnitten der Realwelt in ein Key-Value-Modell dürfte nur in den einfachsten Fällen auf natürliche Art und Weise möglich sein, da es diesen Modellen an Ausdrucksmächtigkeit fehlt (K9).

5.4.2.2 Bewertung von auf Auszeichnungsprachen basierenden Modellen

Auszeichnungssprachen haben eine höhere Ausdrucksmächtigkeit als einfache Key-Value-Modelle und können aufgrund ihres eigenen hierarchischen Aufbaus hierarchische Strukturen gut abbilden. Sie bauen üblicherweise auf etablierten Standards auf, die es ermöglichen Metainformationen zur Semantik einzubinden. Sovereinfachen sie den Austausch von Daten zwischen verschiedenen Systemen (K7). Da sich nahezu alle anderen Modelle mit vertretbarem Aufwand in Auszeichnungssprachen abbilden lassen, sind diese als Kommunikationsmedium zwischen heterogenen Systemen gut geeignet. So zeigt beispielsweise Robinson, wie sich zu diesem Zweck in CML formulierte Modelle in einer Auszeichnungssprache ausdrücken lassen [RHI07]. Ansonsten unterstützen die auf Auszeichnungssprachen basierenden Modelle, ähnlich wie die Key-Value-Modelle, keine weiteren Konzepte zum Umgang mit Kontextdaten.

5.4.2.3 Bewertung objektorientierter Modelle

Objektorientierte Ansätze unterstützen vor allem die Abstraktion und somit die Möglichkeit der Verdichtung von Daten im Entwurf von kontextadaptiven Systemen (K3). Zudem hat objektorientierter Softwareentwurf den Anspruch, Konzepte der realen Welt direkt in softwaretechnische Konzepte umzusetzten, wodurch ein solcher Ansatz die Natürlichkeit der Abbildung begünstigt (K9). Außer der Ausführbarkeit von Code bzw. Kompilat einer objektorientierterten Programmiersprache stellt ein objektorientierter Ansatz keine weiteren infrastrukturellen Ansprüche (K1) , es sei denn, es werden zusätzlich noch komplexere Frameworks zur Entwicklung eingesetzt. Für eine sehr große Anzahl von mobilen Geräten stehen gut entwickelte Virtuelle Maschinen für Java zur Verfügung, welche in der Lage sind Java Bytecode auszuführen [Sun09b], [Sun09f], [Bor08].

Ähnlich wie bei den Key-Value-Modellen müssten Beziehungen in einem objektorientierten Ansatz den einzelnen Objekten zugeordnet werden, was ihre Verwaltung erschwert (K5) . Zwar lassen sich in Objekten prinzipiell unterschiedliche Verhaltensweisen realiseren, eine direkte Unterstützung zur Realisierung von Inferenz (K8) oder dem Umgang mit widersprüchlichen Daten (K4) ist jedoch nicht gegeben.

5.4.2.4 Bewertung dataflow-orientierter Modelle

Grundansatz von dataflow-orientierten Modellen ist es eine Vielzahl unabhängiger Datenströme miteinander zu vereinen und zu höherwertigen Informationen zu verdichten. Die dataflow-orientierten Ansätze machen daher vor allem dort Sinn, wo aus low-level Informationen höherwertige Informationen gewonnen werden sollen (K3) , etwa im Bereich der Sensorfusion.

Prinzipiell eignen sie sich dabei gut zu einer verteilten Realisierung, da in diesem Modell Komponenten und der jeweilige Datenfluß, der zwischen diesen besteht, klar definiert und abgetrennt sind und so einzelne Komponenten relativ leicht auf unabhängigen Systemem residieren können (K6) . Winograd eräutert am Beispiel der Entwicklung von Benutzungsschnittstellen, wie durch einen dataflow-orientierten Ansatz eine Entkopellung von Phänomenen, Eingabegeräten und den die Geräte nutzenden Anwendungen erreicht werden kann [Win01b]. Nimmt man statt herkömmlicher Eingabegeräte Kontextsensoren als Datenquellen an, lassen sich diese Erkenntnisse entsprechend auf den Bereich kontextadaptiver Systeme übertragen. Die Möglichkeit zur dynamischen Rekonfiguration von datafloworienterten Systemen, z.B. um auf veränderliche Umgebungsbedingungen wie den Ausfall von Komponeneten, zu reagieren, begünstigt dabei die Robustheit und Skalierbarkeit in einem verteilten System

Die Anforderungen an die Infrastruktur (K1) sind für einen dataflow-orientierten Ansatz nicht sehr hoch. Es wird vor allem eine Kommunikationsinfrastruktur benötigt, über welche die Datenströme laufen können. In einem verteilten System kann diese auf Basis der ohnehin vorhandenen Netzwerkinfrastruktur realisert werden.

Aufgrund der stromorientierten Sicht auf die Daten ist ein Umgang mit Beziehungen (K5) zwischen Entitäten in diesem Ansatz jedoch schwierig. Dieser Ansatz ist in erster Linie geeignet um Änderungen in der Welt zu erfassen und zu verarbeiten jedoch nicht, um einen komplexeren Zustand der Welt abzubilden. Dieser Umstand beeinträchtigt auch die Natürlichkeit (K9) der Abbildung der Realwelt in das Modell.

Datenflussmodelle können so konfiguriert werden, dass sie unvollständige oder widersprüchliche Daten kompensieren (K4). Inferenz neuer Fakten (K8) wird von diesem Ansatz jedoch nicht direkt unterstützt.

5.4.2.5 Bewertung relationaler Modelle

Aufgrund ihres relationalen Charakters stellen relationale Modelle Beziehungen (K5) in den Mittelpunkt des Datenmodells . Da sie in der Implementation auf etablierte Datenbanksysteme zurückgreifen können, erlauben sie einen effizienten Umgang auch mit einer großen Menge (K2) von Daten und Beziehungen.

Über Sichten lassen sich in Datenbanksystemen Abstraktionsmöglichkeiten und eine Datenverdichtung (K3) realisieren. Allerdings nur wenn diese Verdichtungen mit den Mitteln des Datenbanksystems als Ausdruck über bestehenden Relationen formuliert

werden können. Dieses garantiert jedoch nicht unbedingt eine effiziente Lösung für alle möglichen Anwendungsfälle. So lassen sich zeitliche Verläufe und Muster nur mit einem gewissen Aufwand in einem relationalen Modell darstellen und analysieren.

Die Verteilung von Datenbanken (K6) ist, auch auf mobilen Geräten, prinzipiell möglich [SS07]. Allerdings sind Mechanismen zur plattformübergreifenden einheitlichen Interpretation der Daten (K7) üblicherweise nicht Bestandteil eines Datenbanksystems.

Um ein relationales Modell zu implementieren, sollte ein Datenbankmanagmentsystem als Infrastruktur zur Verfügung stehen. Es existieren Datenbanksysteme, die auf Leichtgewichtigkeit hin optimiert wurden[5]. Teilweise existieren (lokale) Datenbanksysteme als Teil von Entwicklungsplattformen für mobile Geräte (vgl. Abschnitt 9.1).

Um von realweltlichen Konzepten zu einem relationalen Modell zu gelangen, gibt es im Bereich des Datenbankentwurfs eine Vielzahl von etablierten Methoden [Hal01]. Mit der durch Henricksen et al. um die CML entwickelten Methodik existiert ein Ansatz, der spezielle Erweiterungen für kontextadaptive Systeme enthält und versucht einen hohen Grad der Natürlichkeit der Abbildung zu realisieren [HI05].

Zum Umgang mit wiedersprüchlichen oder unnvollständigen Daten (K4) stehen nur begrenzte Mittel zur Verfügung. Zwar können Integritätsbedingungen für die Datenbasis formuliert werden, direkte Mechanismen zum Umgang mit unvollständigen oder widersprüchlichen Daten stehen jedoch ebensowenig zur Verfügung wie solche zum Ableiten von Fakten durch Inferenz (K8).

5.4.2.6 Bewertung logikbasierter Modelle

Logikbasierte Modelle sind in erster Linie dazu geeignet die Inferenz neuer Fakten (K8) aus der vorhandenen Faktenmenge zu ermöglichen.

Eine Verdichtung von Daten (K3) kann über Regeln zu erlaubten Schlussfolgerungen realisiert werden. Da Fakten in logischen Modellen üblicherweise als Relationen realisiert sind, können Beziehungen (K5) ähnlich wie in relationalen Ansätzen, in das Modell abgebildet werden.

Mechanismen zur Plattformunabhängigkeit (K7) und zum Umgang mit verteilten Daten (K6) sind in einem logischen Modell nicht direkt gegeben. Darüberhinaus ist der Ansatz, die Welt als Sammlung von Fakten und Regeln zu modellieren nicht unbedingt eine natürliche (K9) Abbildung.

Ein logikbasiertes Modell stellt gewisse Anforderungen an die Infrastruktur (K1) , da ein Interpreter für die Fakten und Regeln vorhanden sein muss, der eine effiziente Inferenz ermöglicht.

[5] z.B. SQLite: http://www.sqlite.org

5.4.2.7 Bewertung ontologiebasierter Modelle

Da Ontologien mit logikasierten Modellen eng verwandt sind und im Kern selbst ein logisches Modell verwenden, teilen sie mit diesen Modellen die Stärke im Bereich der Inferenz von Fakten (K8) und die prinzipiellen Möglichkeiten zur Verdichtung von Daten (K3) sowie zum Umgang mit Beziehungen (K5).

Dabei weisen Ontologien jedoch in einigen Bereichen deutlich weniger Schwächen auf [SLP04]: Da sie für den Einsatz in komplexeren und verteilten Systemen entwickelt wurden und üblicherweise auf etablierten Standards zum Datentausch aufsetzen, unterstützen sie die Plattformunabhängigkeit der Daten (K7) und die Verteilung der Daten (K6) . Die Wissensrepräsentation ist in Ontologien semantisch reichhaltig und hat den Anspruch, sich stärker an der menschlichen Denkweise zu orientieren, wodurch die Natürlichkeit der Abbildung von Konzepten der Realwelt in das Modell (K9) höher als in reinen logikbasierten Modellen ist.

Um dieses zu realisieren müssen Ontologien jedoch Anforderungen an die Infrastruktur stellen, die über die Anforderungen zur Realisierung eines unmittelbar logikbasierten Modelles hinaus gehen (K1). Robinson et al. weisen auf zwei weitere Schwächen von Ontologien hin [RHI07]: Den meisten Ontologien liegt das RDF-Format zugrunde. Dieses erlaubt allerings nur die Definition von zweistelligen Relationen, wodurch sich mehrstellige Relationen nur indirekt und somit weniger natürlich ausdrücken lassen. Zudem wurde durch die Autoren beobachtet, dass in OWL formulierte Konzepte dazu neigen unnötig komplex und unhandlich zu werden. Diese Nachteile schränken den theoretisch guten Umgang mit Beziehungen (K5) und die Natürlichkeit der Modelle (K9) in der Praxis ein.

5.4.3 Modellierung von Aktivitäten in bisherigen Kontextmodellen

Nach eigenem Erkenntnisstand, liegen keine Arbeiten vor, die sich um ein allgemeines Modell für Aktivitäten bemüht haben. Einige Arbeiten versuchen spezielle Arten von Tätigkeiten zu erkennen und verwenden zu ihrer Darstellung Ad Hoc-Strukturen in ihrem Kontextdatenschema. Ein typisches Beispiel hierfür sind „Intelligente Meetingrooms“, in denen versucht wird zu erkennen ob gerade eine Gruppenaktivität, wie etwa ein Meeting, statt findet um diese Aktivität ggf. entsprechend zu unterstützen. In das Schema abgebildet werden können beispielsweise die daran beteiligten Personen, ihre jeweiligen Rollen und die von ihnen verwendeten Dokumente [CFJ03]. Sie sind auf einen sehr speziellen Anwendungsfall zugeschnitten und lassen die Allgemeingültigkeit vermissen.

Aktivitäten existieren hier nicht als eigenes Konzept im Kontextmodell, sondern eine vorher bestimmte Menge von Aktivitäten wird über andere im Modell vorhandene Konzepte realisiert. Eine Offenheit für neue Aktivitäten ist in diesen Ansätzen nicht gegeben. Nur durch die explizite Erweiterung des Schemas lassen sich einzelne neue Aktivitäten berücksichtigen.

	Key-Value-Modelle	Auszeichnungssprachen	Objektorientierte Modelle	Dataflowoorientierte Modelle	Relationale Modelle	Logikbasierte Modelle	Ontologien
(K1) Leichtgewichtigkeit	++	+	/	−	−	−	−−
(K2) Zugriffseffizienz	/	−	/	/	++	/	/
(K3) Datenverdichtung	−−	−−	++	++	/	+	+
(K4) Kompensation von Wiedersprüchen	−−	−−	/	+	−	−	/
(K5) Beziehungsdaten	−	−	/	−−	++	+	/
(K6) Verteilbarkeit	−	−	−	++	/	−	+
(K7) Eindeutigkeit	−	++	−	−	−	−	++
(K8) Inferenz	−−	−−	−−	−−	−−	++	++
(K9) Natürlichkeit	−	−	++	−	+	/	+

++	Modell hat sehr gute Eigenschaften.
+	Modell hat gute Eigenschaften.
/	Modell hat bedingt gute Eigenschaften.
−	Modell hat schlechte Eigenschaften.
−−	Modell hat sehr schlechte Eigenschaften.

Tabelle 5.2: Bewertung der Kontextmodelle

5.4.4 Weitere Merkmale

Die weiteren Merkmale von Kontextmodellen wie die vorhandenen Metadaten, Historien, Integritätsbedingungen und der Umgang mit Mehrdeutigkeiten lassen sich kaum vergleichend bewerten. Generell sind diese Ausdifferenzierungs- und Erweiterungsmöglichkeiten für Kontextmodelle sinnvoll, aber stark abhängig vom Anwendungsfall. Die Möglichkeiten ergänzen sich eher gegenseitig, als dass sie sich ausschließen. Welche dieser Mechanismen auf welche Art und Weise benötigt werden, lässt sich nur im Zuge der Formulierung des konkreten Kontextdatenschemas und unter Kenntnis des Nutzungsszenarios bestimmen.

5.5 Zusammenfassung

Im Bereich der Kontextmodellierung lassen sich die konzeptionelle und die Implementationsebene unterscheiden. Die Wahl eines Implementationsmodells hat weitreichende Auswirkungen darauf, wie Kontext verstanden wird und auf welche Art und Weise mit Kontextdaten umgegangen wird. Die verschiedenen Ansätze zu Implementationsmodellen wurden in diesem Kapitel beschrieben und vergleichend bewertet. Eine Übersicht über die Stärken und Schwächen der Möglichkeiten ist in Tabelle 5.2 gegeben. Die Übersicht zeigt, dass keiner der Ansätze vollständig über- oder unterlegen ist, sondern eher

spezifische Lösungen für Teilprobleme liefert. Daher überrascht es nicht, dass die hier vorgestellten Ansätze eher idealtypisch sind und in der Praxis hybride Lösungen existieren. So werden beispielsweise logikbasierte Konstrukte auch in dem hier als relational Klassifizierten Ansatz von Henricksen et al. [HI05] und dem als dataflow-orientiert eingeordneten Ansatzes zur Sensorenfusion von Gray und Salber [GS01a] genutzt. Darüberhinaus lassen sich auch automatische Überführungen der Modelle ineinander realisieren, so können beispielsweise in Auszeichnungssprachen formulierte Modelle für den Austausch von Daten zwischen heterogenen Systemen eingesetzt werden [RHI07].

In keinem Kontextmodell oder Kontextdatenschema der untersuchten Ansätze wurde eine allgemeine, generische Modellierung von Aktivitäten vorgenommen.

6 Middleware Architekturen für Context Awareness

Kontextinformationen sind komplex, deren Gewinnung aufwendig und ihre Verwaltung innerhalb eines Kontextmodells nicht trivial. Gleichzeit sind potentiell viele Anwendungen an der Nutzung dieser Informationen interessiert und können, insbesondere im Fall von Aktivitätskontext, selber Kontextinformationen beitragen. Um ihre Nutzung zu vereinfachen ist es sinnvoll, die Verwaltung von Kontextinformationen möglichst weitgehend aus den einzelnen Anwendungen zu entfernen und in einer übergeordneten Ebene zu realisieren.

Dieses kann durch eine Middleware realisiert werden. Ihr kommt die Aufgabe zu, durch geeignete Abstraktionen den Umgang mit Kontextinformation zu erleichtern. Dazu kann sie ein Kontextmodell implementieren und Kontextdatenschemata dieses Modells verwalten, die so von den unterschiedlichen Anwendungen gemeinsam genutzt werden können. Dadurch werden kontextadaptive Anwendungen von den Kontextinformationen entkoppelt. So muss der aufwendige Prozess der Kontextdatengewinnung nur ein einziges Mal ausgeführt werden und aufwändige Prozesse für automatisiertes Schließen brauchen ebenfalls nur einmalig ausgeführt werden. Die Anwendungen werden von der Verwaltung des Kontextschemas entlastet und können sich den gleichen Stand von Kontextinformationen teilen. Darüberhinaus kann die Middleware Routineaufgaben im Umgang mit Kontextdaten, wie z.B. deren persistente Speicherung oder Umrechnungen in andere Formate, übernehmen.

Im Folgenden werden zunächst die Aufgaben von und die Anforderung an eine Middleware zur Nutzung von (Aktivitäts-)Kontext in mobilen Anwendungen erörtert (Abschnitt 6.1). Darauf aufbauend werden bisherige Ansätze vorgestellt (Abschitt 6.2) und schließlich kategorisiert und bewertet (Abschitt 6.3) .

6.1 Anforderungen

Als Maßstab zur Bewertung vorhandener Ansätze und als Vorarbeit für den Entwurf der hier zu entwickelnden Middleware werden in diesem Abschnitt Aufgaben und Anforderungen für eine Middleware zur Nutzung von (Aktivitäts-)Kontext erarbeitet. Dazu werden zunächst die typischen Aufgaben einer Middleware anhand eines Schichtmodelles erläutert (Abschnitt 6.1.1), um anschließend einen Katalog an Anforderungen aufzustellen (Abschnitt 6.1.2).

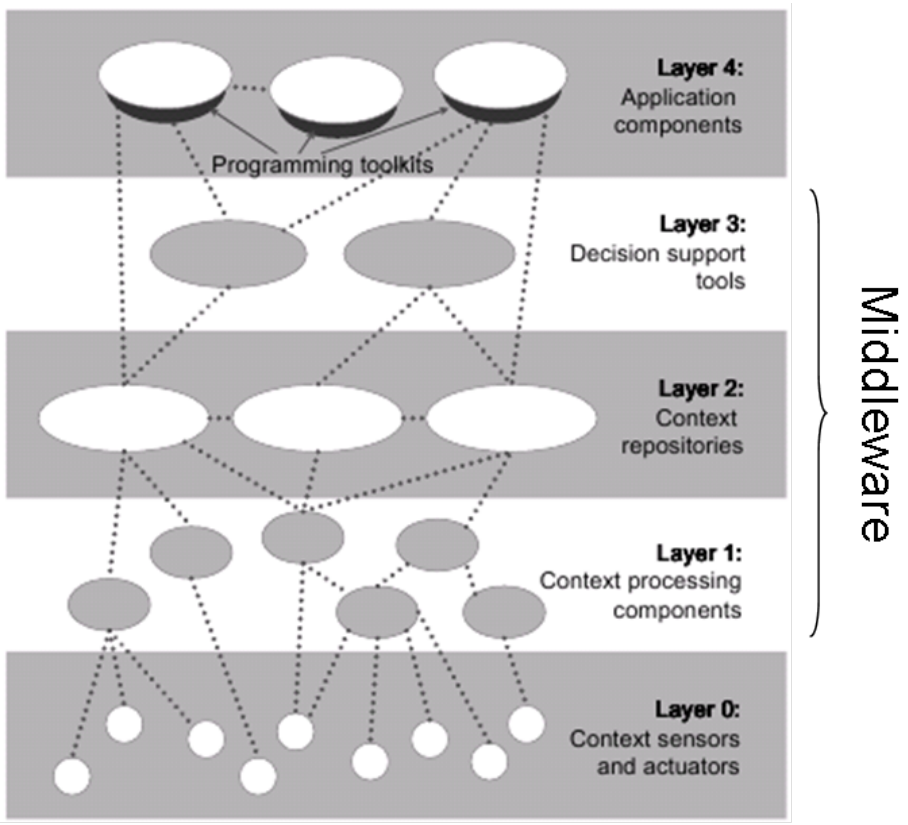

Abbildung 6.1: Typische Schichten in einem kontextadaptiven System [HIM05].

6.1.1 Aufgaben und typische Schichten einer Kontext-Middleware

Nach Henricksen et al. [HIM05] lassen sich kontextadaptive Systeme typischerweise in fünf Schichten aufteilen (vgl. Abbildung 6.1)

Kontextsensoren und Aktuatoren bilden die unterste Schicht 0. In der darüber liegenden Schicht 1 der *Kontextverarbeitungskomponenten* werden in Aufwärtsrichtung rohe Sensordaten zu höherwertigen Daten verarbeitet und abstrahiert sowie in Abwärtsrichtung Aktuatoren der Sensoren abhängig von Aktualiserungswünschen der höheren Schicht eingestellt. Die mittlere Schicht 2 besteht aus *Kontextrepositories*, die für die persistente Speicherung der Daten zuständig sind und einen Zugriff über höherwertige Anfragemechanismen ermöglichen. *Werkzeuge zur Entscheidungsunterstützung* in Schicht 4 helfen, Anwendungen dabei angemessene Aktionen und Anpassungen auf Basis vorhandener Kontextdaten vorzunehmen. Beispiele für Komponenten in dieser Schicht wären ein Modell von Nutzerpräferenzen in bestimmten Situationen oder Komponenten, die aus den Kontextdaten angemessenes Anwendungsverhalten schlussfolgern. Die oberste Schicht 5 bilden schließlich die *Anwendungskomponenten* welche Kontext nutzen. Bereitgestellte Toolkits und APIs zur Nutzung der übrigen Schichten werden ebenfalls zu Schicht 5 gerechnet.

Die Schichten 1-3 können gemeinsam mit Mechanismen zur Kommunikation zwischen den einzelnen Komponenten als der Bereich der Middleware betrachtet werden. Zu den Aufgaben der Middleware gehören somit:

- Kommunikation mit Sensoren
- Verarbeitung von Sensordaten
- Speicherung und Bereithaltung von Kontextdaten
- Unterstütztung von Kontextabhängigen Entscheidungen
- Bereitstellung von Infrastruktur zur Kommunikation zwischen den Komponenten

Die Speicherung und Verarbeitung von Kontextdaten erfolgt dabei idealarweise auf Basis eines Kontextmodells. In diesem Sinne ist es die Aufgabe der Middleware Instanzen von Kontextdatenschemen, zu verwalten und zugänglich zu machen.

6.1.2 Anforderungen an die Middleware

Ausgehend von den Vorüberlegungen zu Aktivitäten (vgl. Kapitel 3), den Überlegungen zu Kontextmodellen (vgl. Kapitel 5) und den typischen Anforderungen an eine Middleware für Kontextadaption (vgl. Abschnitt 6.1.1) können folgende Anforderungen für den eigenen Ansatz zur Unterstützung von aktivitätsorientiertem Kontext aufgestellt werden:

(A1) Abbildbarkeit des Kontextmodells Die Middleware soll ein Kontextmodell implementieren und Instanzen von Implementationsschemata dieses Modells verwalten können. Wünschenswert ist, dass die Middleware mit dem Kontextmodell verzahnt ist und in der Lage die Eigenschaften des Modells auszunutzen und ggf. unerwünschte Eigenschaften des Modells zu kompensieren.

(A2) Unterstützung für Aktivitäten Der Umgang mit Aktivitäten wird direkt durch Mechanismen der Middleware unterstützt. Eine Besonderheit von Aktivitätskontext ist, dass Anwendungen nicht nur Kontextdaten nutzen, sondern gleichzeitig auch die primären Quellen für Daten zu Aktivitäten der Nutzerin sind. Daher müssen einfache Mittel für Software-Werkzeuge bereitgestellt werden, um die Middleware über den Verlauf von Aktivitäten zu informieren. Aus Sicht der Anwendungsentwicklung sollte dieses Feedback auch ohne detaillierte Kenntnisse von Kontextmodell oder Implementationsschema möglich sein und nur minimalen Entwicklungsaufwand erfordern, da ansonsten davon auszugehen ist, dass diese Möglichkeit nicht, nur unzureichend oder falsch genutzt wird.

(A3) Einfachheit der Benutzung Für die Bereitstellung und Nutzung von Konextdaten sollen aus Sicht der kontextadaptiven Anwendungen einfache Mechanismen bereitstehen.

(A4) Unterstützung für unterschiedliche Charakteristika von Daten Die Komplexität der Kontextnutzung der Kontextdaten und die verschiedenen Aspekte der Nutzung von Kontext erfordern unterschiedliche Sichtweisen je nach Problemstellung und Art der Daten.

- **(A4.1) Zustands- und ereignisorientierte Sichten** Zur unmittelbaren Reaktion auf den Beginn oder Abschluss von Aktivitäten ist beispielsweise ein zustands- bzw. ereignisorientierter Zugriff bereitzustellen.
- **(A4.2) Relationale Sichten** Viele Kontextdaten, wie z.B. Beziehungen, die durch Aktivitäten hergestellt werden (vgl. Abschnitt 4.2.4.1), sind relationaler Natur und erfordern einen solchen Zugriff.
- **(A4.3) Zeit- und sequenzorientierte Sichten** Für die Analyse von zeitlichen Verläufen und Zusammenhängen sind zeit- bzw. sequenzorientierte Sichten vorteilhaft.
- **(A4.4) Hierarchische Sichten** Die Modellierung von Entitäten und deren Eigenschaften lässt sich oft gut durch eine hierarchische Modellierung erreichen.

(A5) Unterstützung von zeitlichen Verläufen Zeitliche Verläufe sind beim Umgang mit Aktivitäten von besonderer Bedeutung. Dabei ist vor allem bei kontinuierlichen Signalen über die Frequenz, mit der Werte aufgezeichnet werden zu entscheiden. Ja nach Art der Information und der Aufzeichnungsfrequenz können erhebliche Datenmengen auftreten, welche ggf. weitere Mechanismen wie Komprimierung oder Löschung bzw. Auslagerung von Daten mit zu hohem Alter erfordern.

- **(A5) future Unterstüzung von Informationen über die Zukunft** Um mit vom Nutzer geplanten Aktiviäten, z.B. Kalenderdaten oder prognostizierten Aktivitäten und Ereignissen umgehen zu können, ist die Unterstützung der Verwaltung von Informationen über die Zukunft erforderlich.
- **(A5) history Automatische Aufzeichnung zeitlicher Verläufe** Zur Analyse von Mustern und der Nutzung von in der Vergangenheit oder in der Zukunft gültigen Fakten als Kontextinformation ist die Möglichkeit zur automatischen Aufzeichnung von zeitlichen Verläufen der Kontextinformationen hilfreich.

(A6) Toleranz gegen Ausfall von Komponenten Die Middleware sollte möglichst robust auf den Ausfall von einzelnen Komponenten reagieren. Insbesondere das unvorhergesehene Beenden von Sensoren oder Anwendungen, die Kontextdaten beisteuern, darf nicht zur Beeinträchtigung der Funktionalität der Middleware führen.

(A7) Flexibilität und Erweiterbarkeit Zur Unterstützung von zukünftigen Weiterentwicklungen für neue Kontextarten und Anwendungen sollte die Middleware leicht für den Umgang mit neuen Komponenten im Implementationsschema erweitert werden können. Der Änderungsaufwand bei Modifikationen im bisherigen Kontextdatenschema sollte möglichst lokal begrenzt sein.

(A8) Trennung von Schlussfolgerungen und Fakten Fakten und Schlussfolgerungen sollten getrennt von einander realisiert werden. Dies ist nicht nur vorteilhaft im Sinne von der in (A7) geforderten Flexibilität, sondern ermöglicht es, die oftmals aufwendigen Prozesse der Schlussfolgerung effizienter zu verwalten und bedarfsgerecht auszuführen.

(A9) Kompaktheit Generell leiden mobile Geräte unter Resourcenknappheit im Vergleich zu stationären Lösungen. Da die Middleware auf heutigen Mobiltelefonen lauffähig sein soll und den Betrieb von weiteren Anwendugen nicht beeinträchtigen darf, muss sparsam und effizient mit den Betriebsmitteln umgegangen werden. Dazu ist die Leichtgewichtigkeit der Komponenten, eine sparsame Speichernutzung und ein effizientes Kommunikationsverhalten zwischen den Komponenten notwendig.

(A10) Privacy und Security Im Rahmen von Kontextadaption werden potentiell sensitive Informationen über den Nutzer gesammelt. Diese dürfen nur nach Maßgabe des Nutzers an weitere Anwendungen weiter gegeben oder Dritten zugänglich gemacht werden.

(A11) Verteilung Die Kooperation zwischen auf verschiedenen Geräten liegenden Komponenten, z.B. Programmen auf dem Rechner des Nutzers, Werkzeugen in einer Ubiqitous Computing Umgebung oder die Nutzung von Informationen über die Aktivitäten von Freunden oder Kollegen des Nutzers, würde die Möglichkeiten des Erhebens und Nutzens von Kontextinformationen erheblich vergrößern. Das System sollte daher prinzipiell dahingehend erweiterbar sein, verteilte und kooperative Komponenten möglichst transparent zu integrieren.

Welche Ansätze für Kontextarchitekturen bisher entwickelt wurden und wie diese Versuchen Probleme beim Umgang mit Kontextdaten angehen ist Gegenstand des nächsten Abschnitts.

6.2 Existierende Ansätze

Nachdem Kriterien zur Bewertung von Kontextarchitekturen gesammelt wurden, werden im Folgenden eine Reihe von Ansätzen zu Architekturen und Middleware für kontextadaptive Systeme vorgestellt. Aufgrund der Menge von Arbeiten, die zu diesem Thema existieren [Win01a] [HIM05] [Kjæ07] [BD04], können nicht alle Ansätze ausführlich beschrieben werden. Daher wird zunächst im Abschnitt 6.2.1 "Frühe Arbeiten" vor allem das „Context Toolkit“ von Dey, Salber und Abowd als frühe grundlegende Arbeit, welche die weitere Forschung maßgeblich geprägt hat, vorgestellt [SDA99]. Ein Großteil der neueren Arbeiten wird lediglich, nach deren Ansätzen gruppiert in die Bereiche *serviceorientierte Ansätze* (Abschnit 6.2.2), *objektorientierte Ansätze* (Abschnitt 6.2.4) und *agen-*

tenorientierte Ansätze (Abschnitt 6.2.3) zusammengefasst, vorgestellt. Abschließend werden die Ansätze *PACE* von Henricksen et al. (Abschnitt 6.2.5) [HIM05] und *Gaia OS* von Román et al. [RHC+02] ausführlicher vorgestellt, da sie die vorliegende Arbeit stärker beeinflusst haben.

6.2.1 Frühe Arbeiten

Schilits Doktorarbeit über eine Infrastruktur für Context Awareness im Ubiqitous Computing (1995) [Sch95] im Umfeld von XEROX Parc dürfte der erste systematische Versuch sein eine Software Infrastruktur für Context Awareness zu schaffen. Die Arbeit betont bereits die Notwendigkeit von Notifikationsmechanismen und anfrage-basiertem Datenzugriff. Zugleich ist sie ein Vorgriff auf spätere agentenorientierte Ansätze (vgl. 6.2.3), da in diesem Ansatz Agenten dafür vorgesehen sind, die Aufenthaltsorte des ihnen zugeordneten Gerätes bzw. Nutzers zu ermitteln und an die Infrastruktur weiterzuleiten.

Im AROMA-Projekt wurde von Pedersen und Sokoler 1997 ein Ansatz vorgestellt bei dem low-level Kontextdaten von Sensoren zu einem höheren Level abstrahiert werden, um auf die Präsenz von Personen schließen zu können [PS97].

Zusammen mit eigenen Arbeiten von Dey, Abowd und Wood, welche 1998 Möglichkeiten zur Modularisierung in kontextadaptiven Anwendungen berücksichtigten [DAW98], bilden diese Arbeiten Vorläufer für das Context Toolkit [SDA99]

.

6.2.1.1 Context Toolkit

Das von Salber, Dey und Abowd entwickelte *Context Toolkit* versucht den Nutzen der Widget-Metapher aus der GUI-Entwicklung auf die Entwicklung von kontextadaptiven Anwendungen zu übertragen. GUI-Toolkits verbergen die Eigenschaften der verwendeten Interaktionsgeräte und machen diese so leichter nutzbar und austauschbar (a). Die Details einer Interaktion werden vom Widget selbst verwaltet und die eigentliche Anwendung wird nur bei bedeutsamen Zustandsänderungen informiert (b). Darüberhinaus bietet solch ein Ansatz wiederverwendbare Bausteine an (c). Auf die Entwicklung von kontextadaptiven Anwendungen bezogen, lassen sich durch die von Salber, Dey und Abowd vorgeschlagenen *Context Widgets* die Details im Umgang mit Sensoren zur Kontexdatengewinnung verbergen (a), Datenmengen innerhalb von Widgets zu bedeutungsvolleren Daten abstrahieren und verdichten (b) und wiederverwendbare Bausteine für die Anwendungen entwickeln (c).

Ein Context Widget besteht aus einer beliebigen Ansammlung von Generatoren, Interpretern und Servern (vgl. Abbildung 6.2). Generatoren sammeln Daten von Hardware- oder Softwaresensoren. Interpreter sind für die Abstraktion von rohen bzw. low-level Daten in höhere Abstraktionen verantwortlich. Server aggregieren, kombinieren und speichern die Informationen der anderen Komponenten und machen diese verfügbar. Wid-

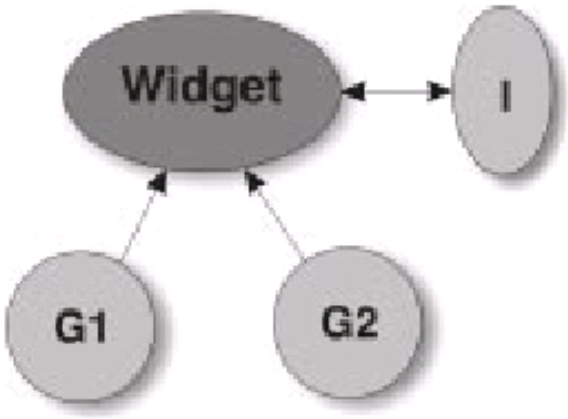

Abbildung 6.2: Ein *Context Widget*, welches Daten aus Generatoren (G) nutzt und die Dienste eines Interpreters (I) zum Verarbeiten der Daten verwendet [SDA99].

gets können ihrerseits auf andere Widgets zugreifen um höherwertige Dienste bereitzustellen. Sowohl Widgets als auch ihre einzelnen Komponenten lassen sich verteilen.

Aus Sicht einer Anwendung hat ein Widget einen Zustand, der durch die Attribute des Widgets und ihrer Werte definiert ist. Diese können direkt abgefragt werden oder über einen Subscribe-Mechanismus bei Änderungen genutzt werden. So ist es möglich die Verarbeitung von Kontextdaten weitgehend aus der Anwendung zu lösen und in Komponenten bereitzustellen, die von mehreren Anwendungen gemeinsam genutzt werden können.

6.2.2 Serviceorientierte Ansätze

Serviceorientierte Ansätze zielen üblicherweise primär darauf ab, den Umgang mit der Hardware und vor allem der Netzwerkstruktur für verteilte Kontextsysteme zu vereinfachen.

Die Arbeit von Lei et al. (2002) beschreibt einen Context Service für Ubiqitous Computing Umgebungen [LSD+02], während die Plattformen *ContextPhone* [ROPT05], *Hyrogen* [HSP+03] und *Capnet* [RDF+05] für den Einsatz auf aktuellen mobilen Geräten wie Mobiltelefonen oder PDAs bestimmt sind. Serviceorientierte Ansätze stellen eine Menge von Diensten, die hilfreich für kontextadaptive Systeme sind, innerhalb einer Servicestruktur bereit. Typische, bereitgestellte Dienste sind Kommunikationsdienste [LSD+02] [ROPT05] [RDF+05], Dienste zur lokalen Aggregation und Speicherung von Kontextdaten [LSD+02] [HSP+03] [RDF+05] und Dienste zum Umgang mit Positionsdaten [LSD+02] [ROPT05] [RDF+05]. Bei Ansätzen, die nicht primär auf einer lokalen Plattform laufen, sondern Verteilung propagieren, sind Mechanismen für Servicediscovery integriert [LSD+02] [RDF+05].

Ein Beispiel für einen serviceorientierten Ansatz, der in begrenztem Umfang Nutzeraktivitäten beobachtet, ist die *ContextPhone*-Plattform [ROPT05] für symbianbasierte mobile Geräte. Sie beobachtet welche Anwendungen aktiv sind und zusätzlich alle Kommunikationskanäle des mobilen Gerätes und bietet Dienste an, die über deren Nutzung informieren.

Die Stärke dienstorientierter Ansätze liegt vorallem im einfacheren Umgang mit komplexerer Hardware und Netzstruktur, der über die angebotenen Dienste möglich wird. Die Idee, kontextadaptive Anwendungen durch eine Komposition von Diensten zu entwickeln, ist in gewisser Hinsicht an die Architektur des Context Toolkits (vgl. 6.2.1.1) angelehnt. All diesen Ansätzen ist gemein, dass die Zusammenarbeit und die Schnittstellen der Dienste nur sehr allgemein spezifiziert sind. Annahmen über ein zugrunde liegendes Kontextmodell werden nicht getroffen und so finden sich hier auch keine Mechanismen zur Unterstützung der Nutzung eines Kontextmodells.

6.2.3 Agentenorientierte Ansätze

In agentenorienterten Ansätzen werden Softwareagenten eingesetzt, um Kontextinformationen zu verwalten. Agenten werden einer oder mehreren Entitäten zugeordnet, z.B. einem Nutzer, einem Gerät oder einem Raum und den darin befindlichen Dingen und verwalten aktiv Kontextinformationen für ihren Zuständigkeitsbereich, d.h. sie akquirieren Daten und können im Bedarfsfall mit anderen Agenten kooperieren und Informationen austauschen. Darüberhinaus können Agenten über das bloße Sammeln von Informationen auch eigenes aktives, kontextabhängiges Verhalten realisieren.

Beispiele für agentenorientierte Ansätze sind die in Abschnitt 6.2.1 erwähnte frühe Arbeit von Schilit [Sch95] sowie der Ansatz von Spreitzer und Theimer, die ebenfalls Agenten in einer Architektur für Lokaliserung einsetzten [ST93] und die jüngere von Chen et al. (2003) entwickelte *Context Broker Architecture (CoBrA)* [CFJ03].

CoBrA ist eine agentenbasierte Architektur, die „Intelligente Konferenzräume“ ermöglichen soll. Sowohl den Räumen als auch den Nutzern sind Agenten zugeordnet. Aufgrund von Kontextinformationen führt der für den Raum zuständige Agent automatische Aktionen aus, wie z.B. die Einrichtung von Lichtverhältnissen, die automatische Aufzeichnung des Meetings oder das Bereitstellen von zugehörigen Daten und Präsentationen. Damit dies erreicht werden kann, findet eine Kooperation mit den Agenten der anwesenden Nutzer statt, so dass sich Agenten untereinander beispielsweise über Terminpläne austauschen und gegenseitig relevante Dateien bereitstellen können. Um die Kommunikation der Agenten untereinander und die gemeinsame Nutzung von Kontextdaten zu ermöglichen, wird ein ontologisches Kontextmodell verwendet, welches die *Web Ontology Language (OWL)* [BHH+04] zur Spezifikation von Kontextdatenschemata verwendet.

Die einzelnen Agenten brauchen jeweils nur einen Teil der verteilten Ontologie kennen, der für ihre Domäne relevant ist. Diese Architektur ermöglicht also die Verteilung

von Ontologien und Kontextdaten. Lokal vorliegende Kontextdaten können durch die vorliegende (Teil-)Ontologie von einem Agenten auf Validität geprüft werden. Da Ontologien in Hinblick auf automatisches Schließen entwickelt wurden, lässt sich auch automatisiertes Schließen in diesem System leicht realiseren. Ontologiebasierte Kontextmodelle (vgl. 5.3.8) lassen sich in dieser Architektur auf direkte und natürliche Weise nutzen.

6.2.4 Objektorientierte Ansätze

In objektorientierten Ansätzen werden Entitäten als Objekte implementiert. Objekte kapseln, wie in der objektorientierten Entwicklung üblich, ihre Daten. Ihr Verhalten, einschließlich kontextabhängiger Aktionen, wird direkt im Objekt bzw. der entsprechenden Klasse spezifiziert. In einer objektorientierten Architektur müssen die Objekte, welche Entititäten repräsentieren, bestimmte Schnittstellen implementieren. Dadurch kann erreicht werden, dass sie durch die Mechanismen der bereitgestellten Infrastruktur verwaltet werden können. So können diese Objekte automatisch beobachtet und bei Änderungen der Eigenschaften von Entititäten, die für sie zum Kontext gehören, automatisch informiert werden. Verteilung der Entititäten kann von der Verwaltungsinfrastruktur für die Objekte transparent realisiert werden. Üblicherweise gehören Mechanismen zur persistenten Speicherung von Entitäts-Objekten zu einer objektorientierten Infrastruktur.

Beispiele für objektorientierte Middlewareinfrastrukturen sind das *Java Context Awareness Framework (JCAF)* [Bar05] und der *DEMAC Context Service* [Tur06].

In JCAF werden Entitäten als Objekte implementiert. Zu diesen Entitäten können weitere Objekte deklariert werden, die für den Kontext der Entität relevant sind. Jedes Entitäts-Objekt wird von einem Manager verwaltet, der das Objekt durch Aufruf einer entsprechenden `onContextChanged()`-Methode informiert, wenn sich Objekte, die als Kontext deklariert wurden, geändert haben. Bei Änderungen im Entitäts-Objekt selber werden diese Änderungen entsprechend an das System propagiert. Benachrichtigungen über Kontextänderungen können sich in diesem System auch an indirekt betroffene Objekte rekursiv fortpflanzen. Wie das Objekt auf Kontextänderungen reagiert, kann dann in der `onContextChanged()`-Methode implementiert werden.

Die *Distributed Environment for Mobility Aware Computing (DEMAC)* [Kun05] soll mobile Anwendungen darin unterstützen, langlebige und komplexe Prozesse verteilt auszuführen. DEMAC folgt einer service-orientierten Architektur. Einer der in DEMAC integrierten Dienste ist ein *Context Service* [Tur06], der ein objektorientiertes Kontextmodell implementiert. Ähnlich wie bei JCAF werden Entitäten als Unterklassen einer abstrakten Entitätsoberklasse definiert und können unterschiedlichen Context-Domänen zugeordnet werden. Zur Darstellung von Attributen einer Entität wird eine Reihe abstrakter Schnittstellen bereitgestellt, die es erlauben, zwischen definierten, abgeleiteten und auf Sensormessungen beruhenden Werten zu unterschieden. Die Attribute können über zusätzliche Qualitätsparameter verfügen. Attribute, die auf Sensordaten beruhen, implementieren aktives eigenes Verhalten um den Umgang mit Sensordaten zu realiseren.

6.2.5 PACE-Middleware

Die von Henricksen et al. vorgestellte Middleware des *Pervasive Autonomic Context-Aware Environments (PACE)* Projekts[1] [HIM05] gehört zu den neueren und umfangreicheren Arbeiten auf dem Gebiet der Middleware für kontextadaptive Systeme.

Zu ihren Merkmalen gehört die explizite Integration, Verwaltung und Repräsentation von in CML formalisierten Kontextmodellen in die Middleware, sowie die Unterstützung von kontextabhängigen Userpräferenzen durch ein eigenes Präferenzmodell.

Ähnlich dem Context Toolkit (siehe 6.2.1.1) werden Kontextdaten in verteilten Repositories bereitgehalten. Jedes dieser Repositories implementiert ein oder mehrere in CML spezifizierte Kontextdatenschemata. Die Kommunikation mit den Repositories erfolgt durch Abfragen oder Notifikationsmechanismen, welche durch die Kommunikationsinfrastruktur unterstützt werden. Der Zugriff kann auf einzelne Fakten erfolgen oder über logische Formeln, die auf ihre Erfüllbarkeit in der Faktenmenge des Repositories geprüft werden.

Die modellspezifischen Teile der Infrastruktur können aus vorliegenden CML-Modellen teilweise automatisch generiert werden. Die einzelnen Repositories tragen formale Beschreibungen der Kontextmodelle, welche sie implementieren. Service- genauer Repositorydiscovery wird über diese Modellbeschreibungen realisiert. So kann z.B. der Discoverymechanismus eingesetzt werden, um nach einem Repository mit bestimmten Faktentypen zu suchen.

Kommunikation zwischen den lose gekoppelten Komponenten erfolgt über inhaltbasierte Nachrichtenverteilung, die eine n:m-Kommunikation erlaubt, d.h. eine Nachricht wird an diejenigen Komponenten übermittelt, welche deklariert haben, an dem entsprechenden Inhalt interessiert zu sein. Dadurch ist es möglich, Abfragen und Updates an mehrere Komponenten gleichzeitig zu verschicken.

Zusätzlich zu einem Kontextmodell verwendet die Middleware ein formales Modell zur Darstellung der Präferenzen eines Nutzers in bestimmten Situationen. Dieses ist eng mit dem Kontextmodell verzahnt und erlaubt die kontextabhängige Ermittlung von aktuell gültigen Nutzerpräferenzen durch die Middleware, das Verwalten von kontextabhängigen Präferenzen durch die Nutzer und das Teilen von Präferenzen zwischen verschiedenen Anwendungen.

6.2.6 Gaia OS

Das von den Entwicklern als Betriebssystem für kontaxtadaptive Ubiqitous Computing Anwendungen bezeichnete *Gaia OS*[2] [RHC+02] besteht aus fünf Diensten: Einem Eventmanager, einem Kontextdatenspeicher, einem Dienst, der für das Feststellen der Anwesenheit von Personen, Geräten, Anwendungen und Diensten verantwortlich ist, einem

[1] http://www.itee.uq.edu.au/~pace/index.html
[2] http://gaia.cs.uiuc.edu/

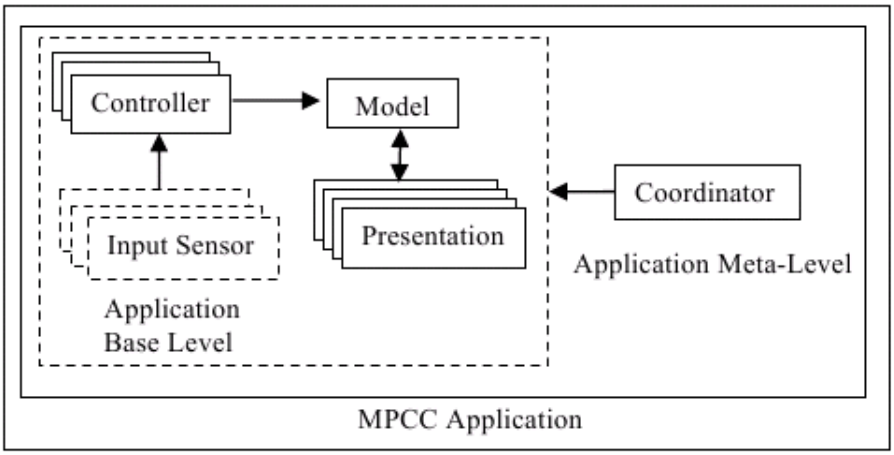

Abbildung 6.3: Schema von Model-Presentation-Controller-Coordinator (MPCC) Anwendugnen: Der Coordinator kann kontextabhängig Controller und Präsentationselemente aktivieren bzw. deaktivieren [Rom03].

Dienst, der die Eigenschaften dieser Entitäten zugänglich macht und einem *Context File System* (s.u.).

Zusätzlicher Teil von Gaia ist ein Application Framework für kontextadaptive Anwendungen [Rom03]. Das Application Framework erweitert das klassische Model-View-Controler Schema [KP88] um eine so genannte Koordinatorkomponente. Der Koordinator kann kontextabhängig einzelne Controller und Views, die hier allgemeiner als Präsentationen verstanden werden, aktivieren oder deaktivieren (siehe Abbildung 6.3).

Anders als die in Abschnitt 6.2.2 vorgestellten Dienste, ist Gaia teilweise eng mit einem Kontextmodell verzahnt und verwendet ein, bereits in Abschnitt 5.3.7 vorgestelltes, logisches Kontextmodell, in dem Kontext durch vierstellige Prädikate modeliert wird.

Die Verzahnung des Modells mit der Middleware lässt sich am eingesetzten *Context File System* [HC02] zeigen: Die Middleware ist in der Lage, eine virtuelle Verzeichnisstruktur in einem Unixdateisystem zu erzeugen, deren Verzeichnisse den Werten, nach dem Muster `/type:/value`, den Prädikaten im logischen Modell entsprechen. Das Verzeichnis `/location:/RM2401/situation:/meeting` entspricht einem Kontext, in dem gilt `location == RM2401 AND situation == meeting`. Dabei werden für bestimmte Attribute, wie beispielsweise „location“ die für den jeweiligen Nutzer geltenden Werte als Standardwert angenommen und ansonsten auf den allgemeingültigen Kontext zurückgegriffen. Dateien können einem Kontext zugeordenet werden, indem sie in ein entsprechendes Verzeichnis kopiert werden. So kann beispielsweise eine Präsentationsdatei für ein Meeting in Raum 2401 in das oben genannte Verzeichnis kopiert werden. Im Verzeichnis `/situation:/meeting` befinden sich nun automatisch alle Dateien zu allen Meetings an sämtlichen bekannten Orten. Über das Schema `/situation:/meeting/current/` könnte eine, einem Nutzer zugeordnete, Anwendung auf alle Dateien für Meetings zugreifen, die dem aktuellen Ort des Nutzers und weiteren Kriterien

wie der zugeordneten Zeit entsprechen, da das Schlüsselwort `current` dem übrigen Kontext des jeweiligen Nutzers entspricht. Diese kontextabhängige, dynamische Form der Addressierung macht es für Anwendungen einfach, auf kontextbezogene Dateien zuzugreifen. Das eigentliche Kontextsystem wird für die nutzende Anwendung transparent.

6.3 Bewertung bisheriger Arbeiten

Nachdem bisherige Ansätze zu Architekturen für Context Awareness im vorherigen Abschnitt vorgestellt wurden, sollen diese hinsichtlich ihres Nutzes in Hinblick auf das Projekt der vorliegenden Arbeit geprüft werden. Dabei wird zunächst auf das Context Toolkit (Abbschnit 6.3.1) eingegangen, um dann wieder zusammenfassend die Gruppen serviceorientierter Ansätze (Abbschnit 6.3.2), agentenorientierter Ansätze (Abschnitt 6.3.3) und objektorientierter Ansätze (Abschnitt 6.3.4) zu behandeln, sowie eine Bewertung der PACE- (Abbschnit 6.3.5) und der Gaia-Architektur (Abbschnit 6.3.6) vorzunehmen. Schließlich wird ein abschließender Überblick gegeben, in dem speziell auf die Eignung der Ansätze für den Umgang mit Aktivitäten (Abschnitt 6.4.4) und zeitlichen Abläufen (Abschnitt 6.4.5) eingegangen wird.

6.3.1 Bewertung des Context Toolkits

Die Stärke des Context Toolkits liegt in seinen Ansätzen zur Verteilung von Komponenten (A11), der Abstraktion von Kontextdaten und der Wiederverwendbarkeit der Bausteine sowie guter Möglichkeiten, aufbauend auf den vorhandenen Komponenten neue zu entwickeln (A7). Hierauf nehmen viele spätere Ansätze starken Bezug. Implizit verwendet es ein Kontextmodell, das dataflow- und key-value-orientiert ist, ohne dass ein solches Kontextmodell explizit formuliert oder ausgearbeitet (A1) würde. Die Architektur stellt keine großen Anforderungen an die Resourcen (A9). Historien für einzelne Werte von Kontextdaten (A5) history sind vorgesehen, ein Umgang mit zukünftigen Daten aufgrund von Prognosen oder Planungen (A5) future jedoch nicht.

Konzepte zum Umgang mit Aktivitäten (A2), zur expliziten Darstellung von Fakten und Schlussfolgerungen (A8) oder zur Unterstützung von Privacy und Security (A10) fehlen.

6.3.2 Bewertung serviceorientierter Architekturen

Serviceorientierte Ansätze erfüllen besonders dort wo Mechanismen für Servicediscovery angeboten werden, das Kriterium der Erweiterkeit (A7) und für die Bereiche, in denen der Ansatz konkrete Dienste anbietet, die Entwicklung (A3). Durch die oftmals sehr losen Vorgaben für weitere Dienste bieten sie jedoch nicht unbedingt Ansatzpunkte für eine strukturierte Weiterentwicklung. Somit ist die Einfachheit und Einheitlichkeit von

späteren Weiterentwicklungen nicht gewährleistet. Durch den Verzicht auf Annahmen zum Kontextmodell (A1) werden Möglichkeiten vertan, den Umgang mit einem Kontextmodell zu erleichtern (A3).

Auffallend viele Ansätze, die für den Einsatz auf mobilen Geräten bestimmt sind, fallen in diesen Bereich und berücksichtigen somit das Problem von Resourcenknappheit in besonderem Ausmaß (A9).

6.3.3 Bewertung agentenorientierter Architekturen

Agentenorientierte Ansätze können ihre volle Stärke dort entfalten, wo ontologische Kontextmodelle oder zumindest Überführungen in ontologische Modelle, existieren. Ihre Stärke liegt in der Möglichkeit zu aktivem Verhalten, im Umgang mit heterogenen Systemen, was teilweise den Umgang mit verschiedenen Datensichten begünstig (A4) und vorallem in der Unterstützung von Planung und Reasoning (A8). Die Planungs- und Schlussfolgerungsmechanismen, die üblichwerweise von Agentensystemen eingesetzt werden, basieren allerdings weitgehend auf Verfahren, die sich die Struktur von Ontologien zu Nutze machen, um auf neue Fakten zu schließen. Verfahren, die auf die Analyse von wiederkehrenden Mustern bauen, werden daher nicht direkt unterstützt.

Auch wenn Implementationen von Agentensystemen für mobile Geräte existieren [Har06] sind Agentensysteme tendenziell komplexere Systeme und tragen weniger zur Resourceneffizienz (A9) bei.

Agenten selbst sind Akteure, die eigenes aktives Verhalten aufweisen. Teilweise ist dieses Verhalten derart, dass ein Agent Aufgaben des Nutzers übernimmt. Insofern gibt es eine Beziehung zu den Nutzeraktivitäten. Im Gegensatz zu spontanen Aktivitäten des Nutzers, verhalten sich Agenten jedoch nach klar definierten Regeln. Ausgangspunkt für ihr Verhalten ist ein explizit angegebenes Ziel oder ein Auftrag, der an sie übertragen wird. Beim Nutzerverhalten ist in der Regel gerade das Ziel nicht bekannt und es können nur seine, meist nicht nach einem strikten Muster ablaufenden, Tätigkeiten beobachtet werden. Insofern lässt sich die Nähe von Agenten zu Aktivitäten (A2) nicht direkt Nutzen. Allerdings könnte, sollten die Ziele des Nutzers durch den Aktivitätskontext erkennbar werden, sich dieses zur Übertragung von Aufgaben an Agenten nutzen lassen.

6.3.4 Bewertung objektorientierter Architekturen

Objektorientierte Modelle erfordern eine Instanziierung der von ihnen verwalteten Entititäten als Objekte. Dies ist nur sinnvoll, wenn die Menge der Entititäten im Vergleich zu den zur Verfügung stehenden Resourcen klein ist. Entweder müssen die Entititäten also hochgradig verteilt sein, wie etwa in einem Ubiqitous Computing Szenario, oder die Menge der Entitäten muss überschaubar klein sein.

Zudem ist die Verteilung und Kapselung der Daten in Objekte für eine effiziente Mustersuche nicht gut geeignet. Die Kriterien Resourceneffizenz (A9) und Verfügbarkeit un-

terschiedlicher Datensichten (A4) werden somit von objektorienteierten Ansätzen unzureichend erfüllt[3].

6.3.5 Bewertung von PACE

Pace ist, als Teil eines jüngeren Forschungsprojekts, eine relativ ausgereifte Arbeit und ein umfassender Ansatz zum Umgang mit Context im Ubiqitous Computing .

Pace nutzt ein relationales Kontextmodell (A1), für das mit der Context Modelling Language eine Sprache und Methodik zum konzeptuellen Entwuf bereitsteht, und das mit der Middleware eng verzahnt ist. So kann die Middleware beispielsweise Nutzerpräferenzen verwalten und ermitteln, welche der Präferenzen auf eine gegebene Situation zutrifft. Die Entwicklungsmethodik orientiert sich an Object Role Modeling aus dem Bereich des Datenbankentwurfs und ermöglicht relativ direkte Abbildungen von Konzepten der Realwelt in ein Kontextdatenschema (A3). Bei der konzeptuellen Modellierierung werden Fakten, Schlußfolgerungen und Abhängigkeiten zwischen diesen explizit gemacht (A8). Da ein relationales Kontextmodell verwendet wird, sind relationale Sichten direkt unterstützt (A4.2) Darüberhinaus existieren Mechanismen zum Erzeugen von Ereignissen bei Änderungen von Daten (A4.1) und Mechanismen um Historien von Daten zu verwalten (A4.3) (A5) history. Der Ansatz sieht vor, dass Kontextdatenschemata, auf verschiedene Komponenten verteilt, verwaltet werden können (A11) und dass weitere Kontextdatenschemata jederzeit zu einen System hinzugefügt werden können (A7). Konzepte für detailierte Zugriffskontrollen auf Kontextdaten zur Gewährleistung von Privacy und Security (A10) existieren [HWMI05].

Da die Architektur jedoch auf Ubiquitous Computing Umgebungen ausgelegt ist und einen relativ umfassenden Ansatz verfolgt, ist sie nicht für resourcenknappheit auf heutigen mobilen Geräten optimiert (A9). Auch in diesem Ansatz fehlen Aktivitäten als konzeptuelles Element (A2).

6.3.6 Bewertung von Gaia

Gaia ist ein weiteres Beispiell dafür, wie ein Kontextmodell mit einer Middleware verzahnt werden kann (A1). Im Beispiel von Gaia wird ein logisches Modell verwendet. Hervorhebenswert ist vorallem das Context File System von Gaia, in dem dieses Modell genutzt wird, um dynamische kontextbasierte Namenschemata zu ermöglichen. Im Fall von Gaia werden dynamische kontextbasierte Namenschemata eingesetzt um Dateien, abhängig vom Kontext, zur Verfügung zu stellen. Aus Sicht der Anwendung, die auf Dateien über dieses Schema zugreift, bleibt die Kontextmiddleware dabei vollständig transparent. Der Zugriff auf kontextabhängige Daten ist über solche Namenschemata

[3] Beide Probleme könnten gelindert werden, in dem Mechanismen zum mapping von Objekten auf andere Datenstrukturen bereitgestellt werden und diese nur im Bedarfsfall instanziert werden, beispielsweise mit Hilfe eines Objektrelationalen-Datenbank-Mappers [BK04]. Dies erfordert jedoch wiederum komplexere Managmentstrukturen, die nach Kenntnis des Autors in diesen Ansätzen bisher nicht Implementiert wurden. Ohnehin wäre dieser Ansatz eher ein Workaround als eine elegante Lösung

	Context Toolkit	**service-orientiert**	**agenten-orientiert**	**objekt-orientiert**	**PACE**	**Gaia**
(A1) Kontextmodell	b dataflow/ key-value	– –	b Onto-logien	b objekt-orientiert	✓ rela-tional	b logik
(A2) Aktivitäten	–	–	–	–	–	–
(A3) Einfachheit	b	b	b	b	b	b
(A4) Datensichten	b	b	–	–	–	–
(A5) Zeit	b	b	b	b	b	–
(A6) Fehlertoleranz	–	b	b	b	b	b
(A7) Erweiterbarkeit	✓	b	✓	✓	✓	b
(A8) Trennung	✓	✓	✓	✓	✓	✓
(A9) Kompaktheit	b	/	–	–	b	b
(A10) Privacy	–	/	✓	/	✓	–
(A11) Verteilung	✓	✓	✓	✓	✓	b

✓ Anforderung wird vollständig erfüllt.
b Anforderung wird teilweise erfüllt.
/ Anforderung wird von einigen Ansätzen erfüllt.
– Anforderung wird nicht erfüllt

Tabelle 6.1: Bewertung der Kontextarchitekturen

sehr unmittelbar. Im Fall einer Einbindung in das Dateisystem, wie bei Gaia, können sogar ursprünglich nicht kontextadaptive Anwendungen hiervon profitieren. Daher ist dieser Mechanismus ein gutes Beispiel für Einfacheit (A3) im Umgang mit Kontextdaten.

Die Verwendung eines logikbasierten Kontextmodells erlaubt einen guten Umgang mit Inferenz und eine explizite Darstellung von Fakten und Schlußfolgerungen (A8).

Leider ist die Verzahnung mit dem Kontextmodell in Gaia nicht durchgehend und konsistent. Generell leidet der Ansatz auch unter vielen Defiziten der serviceorientierten Ansätze (vgl Abschnitt 6.3.2).

Wie in allen untersuchten Ansätzen, werden auch in diesem Ansatz Aktivitäten konzeptionell nicht weiter berücksichtigt (A2).

6.4 Fazit

Einen vergleichenden Überblick über die Ergebnisse der Untersuchung bietet Tabelle 6.1. Die Ergbenisse werden im Folgenden noch einmal analysiert, wobei auf Gemeinsamkeiten (Abschnitt 6.4.1), die Einsatzfähigkeit auf mobilen Geräten (Abschnitt 6.4.2), Fehlertolerenz (Abschnitt 6.4.3), die Eignung für den Umgang mit Aktivitäten (Abschnitt 6.4.4) und für den Umgang mit zeitlichen Abläufen (Abschnitt 6.4.5) eingegangen wird.

6.4.1 Gemeinsamkeiten

Einige Anforderungen, wie die Erweiterbarkeit und Wiederverwendbarkeit von Komponenten (A7) und Unterstützung von Verteilung (A11), werden im Allgemeinen als grundlegend für Middleware gesehen und in den allermeisten Fällen auf unterschiedliche Art und Weise unterstützt. Durch die meist modularisierte Struktur lässt sich auch eine Trennung von Fakten und Schlussfolgerungen in allen Ansätzen realisieren (A8).

Die Einfachheit der Benutzung in der Anwendungsentwicklung (A3) hängt stark vom Anwendungsfall ab. Eine vielversprechende Idee sind dynamische kontextabhängige Namensschemata, wie sie z.B. in Gaia für ein Dateisystem implementiert wurden.

Der überwiegende Teil der Systeme arbeitet mit expliziten Kontextmodellen, wobei unterschiedliche Kontextmodelle, wie Ontologien, logikbasierte oder relationale Modelle, umgesetzt wurden.

Nahezu alle Änsätze gehen von der Heterogenität der Eingangsdaten durch Sensoren aus. Die meisten Arbeiten bieten sowohl anfragebasierten Zugriff auf Daten als auch Notatifikationsmechanismen für die Benachrichtigung bei Änderungen. Die meisten Anfrageschnittstellen sind jedoch sehr primitiv und ermöglichen nur einfache Abfragen. Auf der anderen Seite des Extrems liegen dienstorientierte Ansätze, die keinerlei Vorgaben für Anfrageschnittstellen machen und dadurch an Einheitlichkeit verlieren. Somit wird von keinem Ansatz ausreichend Unterstützung für verschiedentlich strukturierte Datenmengen (A4) zur Verfügung gestellt.

Privacy Aspekte (A10) wurden von einigen Autoren berücksichtigt in anderen Arbeiten jedoch zunächst vernachlässigt.

6.4.2 Einsatzfähigkeit auf mobilen Geräten

Einige Systeme wurden im Hinblick auf den Einsatz auf heutigen mobilen Geräten wie Mobiltelefonen oder PDAs entwickelt und sind somit für Resourcenknappheit (A9) angemessen optimiert. Diese, im Rahmen dieser Arbeit untersuchten Systeme, fallen alle in den Bereich der serviceorientierten Ansätze und genügen kaum Ansprüchen, die über die Eignung für resourcenknappe Systeme und die Möglichkeit zur verteilten Realiserung (A11) hinausgehen.

6.4.3 Fehlertoleranz

Fehlertoleranz (A6) wird in einigen Arbeiten in begrenztem Umfang durch verschiedene Mechanismen unterstützt. Wie in Abschnitt 6.1 zur Anforderung der Fehlerrobustheit (A6) erläutert wurde, ist in dem Fall, dass Anwendungen auch als Erzeuger von Kontextinformationen dienen, mit deren Ausfall zu rechnen. Die bisherigen Ansätze zur Fehlerrobustheit decken diese Problematik jedoch nicht ausreichend ab. Eine Strategie zur Verbesserung von Fehlertoleranz ist das Management der Dienste durch die Infrastruktur. So ist die Gaia-Middleware (siehe 6.2.6) beispielsweise in der Lage, ausgefallene Dienste

neu zu starten. Und für Ansätze, die Service-Discovery bereitstellen, wäre es möglich, auf alternative Dienste zurück zu greifen, so solche existieren.

Ein anderer Ansatz ist die Ausnutzung von Metadaten und Integritätsbedingungen um ein konsistentes Datenmodell zu gewährleisten. Metadaten wurden in bisherigen Arbeiten zwar innerhalb vieler Kontextmodelle formuliert, deren Ausnutzung bleibt aber den Anwendungen oder einzelnen Aggregations-Komponenten überlassen. PACE (siehe 6.2.5) bietet durch die Möglichkeit Integritätsbedingungen als Teil eines Kontextmodells zu formulieren eine gewisse Unterstützung. Mechanismen zum Umgang mit Verletzungen von Integritätsbedingungen sind jedoch nicht vorhanden und der Fall, dass erwartete Daten ausbleiben, wird nicht abgedeckt.

6.4.4 Unterstützung für Aktivitäten

Einige der Ansätze zu Architekturen für kontextadaptive Anwendungen werden auch zur Beobachtung von Nutzeraktivitäten eingesetzt. So wird beispielsweise von der ContextPhone Plattform (Abschnitt 6.2.2) beobachtet, welche Anwendungen aktiv sind und welche Kommunikationskanäle des Telefons gerade genutzt werden. Es liegt hier jedoch kein systematischer Umgang mit Aktivitäten vor. Die Beobachtung einzelner Arten von Aktivitäten wurde vielmehr ad hoc und jeweils getrennt voneinander realsiert. Dies ist durchaus symptomatisch für den allgemein nicht explizit behandelten Umgang mit Nutzeraktivitäten.

Das Problem wird dadurch verschärft, dass sich wohl alle Ansätze nach dem Schema 1. Sammlung von Sensordaten 2. Verarbeitung der Daten durch die Middleware und 3. Nutzung durch die Anwendungen richten. Der Informationsfluss wird hier primär in eine Richtung gedacht. Dass über die Benutzung von Softwarewerkzeugen Kontext entsteht, wird vernachlässigt. Einfache Möglichkeiten für Anwendungen Kontextdaten an die Middleware zu übermitteln, werden üblicherweise nicht mit berücksichtigt.

6.4.5 Umgang mit zeitlichen Verläufen

Eine Reihe von Ansätzen sehen den Umgang mit historischen Kontextdaten (A5) history vor [Sch95] [SDA99] [HIM05], Mechanismen zur automatischen Aufzeichnung von Kontextdaten wurden ebenfalls realisiert [Tur06]. Detailierter ausgeführte Szenarien zur Nutzung dieser Daten konnten jedoch nicht gefunden werden.

Einige Anwendungszenarien und Implementationen, die im Bereich der Architekturen für Context Awareness vorgestellt wurden, nutzen zwar Kalenderdaten [RDF+05] [LSD+02], doch sie stellen lediglich eine ad hoc Brücke zu einer bestehenden Kalenderanwendung bereit und bieten keine Systematik zum Umgang mit zukünftigen Ereignissen (A5) future an.

6.5 Zusammenfassung

In diesem Kapitel wurden Anforderungen an Middlewarearchitekturen, die kontextadapitive Systeme ermöglichen sollen, erörtert und ein vergleichender Überblick über bestehende Ansätze anhand der erabeiteten Kriterien gegeben. Ein Großteil der Ansätze ließ sich als serviceorientiert, agentenorientiert oder objektorientiert klassifizieren. Die näher untersuchten Ansätze Pace und Gaia fielen jedoch nicht direkt in eine dieser Kategorien.

Die Pace-Middleware, als vielversprechenster Ansatz, unterstützt zwar ein ausgereiftes Kontextmodell und einen erheblichen Teil der Anforderungen, wurde jedoch für ein Ubiqitous Computing Szenario und nicht für den Einsatz auf mobilen Geräten konzeptioniert. Darüberhinaus ist auch hier keine Unterstützung für die spezifischen Probleme von Aktivitätskontext, die Unterstützung von Aktivitäten (A2) und der Umgang mit zukünftigen Ereignissen (A5) future gegeben.

Die Gaia-Architektur verdeutlicht, wie dynamische kontextbasierte Namensschemata den Umgang mit Kontextdaten vereinfachen können.

Keiner der untersuchten Ansätze bietet eine Unterstützung im Umgang mit Aktivitäten (A2) oder ausreichend Unterstüzung für zeitliche Verläufe, insbesondere zukünftiger Ereignisse (A5).

Daher wird in Kapitel 8 eine eigene Middleware entworfen, welche Ansätze für die bisher ungelösten Probleme exemplarisch vorführen soll.

7 Entwicklung des Kontexdatenschemas

In diesem Abschnitt soll ein Kontextdatenschema (vgl. Abschnitt 5.1.1) entwickelt werden, das die verwendeten Daten des Aktivitätskontextes strukturiert und Grundlage für die spätere Implementation ist. Hierfür werden zunächst Anforderungen an ein solches Schema bestimmt (Abschnitt 7.1) und auf dieser Grundlage Überlegungen für die Wahl des Kontextmodells (Abschnitt 7.2) angestellt und schließlich ein Kontextdatenschema für Aktivitätskontext entwickelt (Abschnitt 7.3).

7.1 Anforderungen an Modell und Datenschema

Aufbauend auf den Ergebnissen des Kapitels zu Kontextmodellen (Kapitel 5) und der Analyse des Szenarios (Kapitel 2) werden in diesem Abschnitt die Anforderungen an ein Kontextmodell und Kontextdatenschema formuliert, das Grundlage für die Modellierung von Aktivitätskontext ist. Grundlegende Konzepte, die dabei modelliert werden müssen, sind Entitäten und Aktivitäten (Abschnitt 7.1.1). Die weiteren Anforderungen gliedern sich in Anforderungen zur Generizität (Abschnitt 7.1.2) sowie zum Umgang mit zeitlichen (Abschnitt 7.1.3) und räumlichen (Abschnitt 7.1.4) Informationen.

7.1.1 Entitäten und Aktivitäten als zentrale Konzepte

Wie bereits in Abschnitt 4.2.4.1 festgestellt wurde, ist zwischen Entitäten und Aktivitäten zu unterscheiden [RG05, S. 116]. Das Szenario aus Abschnitt 2.1 zeigt, wie die Beziehungen, die durch Aktivitäten zwischen Entitäten entstehen, genutzt werden können. Diese Art von Beziehungen stehen im Fokus dieser Arbeit und sollten sich daher auf natürliche Weise im Kontextdatenschema abbilden lassen.

Betrachten wir die möglichen Beziehungen, lassen sich Beziehungen unterscheiden, die aufgrund der Eigenschaften von Entitäten zwischen Entitäten entstehen[1], wie z.B. „Dieses Mobiltelefon gehört Bob“, Beziehungen, die aufgrund von Attributen von Entitäten bestehen, wie „Bob befindet sich in der Nähe von Alice“ und Beziehungen, die durch Aktivitäten entstanden sind[2] z.B. “Alice hat sich mit Bob getroffen“. Die ersten beiden Arten sind Beziehungen, die direkt zwischen Entitäten bestehen, die zweiten sind jene Beziehungen, die durch Aktivitäten vermittelt bestehen.

Betrachten wir Entitäten, so wurde in der Analyse des Szenarios (Abschnitt 2.2.3) außerdem die Notwendigkeit erläutert, Entitäten nach ihrem Typ unterscheiden zu können. Aus diesen Bedingungen ergeben sich die Forderungen:

[1] D.h. soziale und kompositorische Beziehungen im Sinne von Zimmermann et al. [ZLO07] (vgl. 4.1.2).
[2] funktionale Beziehungen im Sinne von Zimmermann et al. [ZLO07] (vgl. 4.1.2).

- Es soll zwischen Aktivitäten und Entitäten unterschieden werden.
- Entitäten sollen sich über ihren Typ klassifizieren lassen.
- Beziehungen von Aktivitäten zu Entitäten sollen sich natürlich abbilden lassen.
- Beziehungen zwischen Entitäten sollen sich natürlich Abbilden lassen.

7.1.2 Generizität des Aktivitätskonzeptes

Welche Arten von Aktivitäten zukünftig mit Hilfe von Mobiltelefonen und den auf ihnen laufenden Software-Werkzeugen absolviert werden, lässt sich ebensowenig vorhersagen, wie die Arten von Entitäten, die in Zukunft daran beteiligt sein werden. Es ist lediglich absehbar, dass insgesamt die Möglichkeiten zunehmen werden und in Zukunft ganz neue Arten von Aktivitäten und Arten von daran beteiligten Entitäten möglich werden. Daher sollte der hier entwickelte Ansatz neutral gegenüber den unterschiedlichen Arten von Entitäten und Aktivitäten sein, d.h. es müssen sich beliebige Arten von Entitäten und Aktivitäten, auch solche die zum Zeitpunkt der Entwicklung dieser Arbeit noch nicht vorhersehbar waren, in den eigenen Ansatz abbilden lassen. Ein möglichst abstraktes und generisches Konzept von Aktivitäten und Entitäten ist daher nötig. Daraus ergeben sich folgende Forderungen:

- Die Menge der Entitäten und Entitätstypen ist erweiterbar.
- Die Menge der Aktivitäten ist erweiterbar.
- Das Verständnis von Aktivitäten und Entitäten sollte möglichst abstrakt und allgemeingültig sein.

7.1.3 Umgang mit zeitlichen Informationen

Zeitliche Abläufe spielen bei der Analyse von Aktivitäten eine große Rolle [JÖ07]. Wie an dem Beispielszenario (vgl. Abschnitt 2.2.1) deutlich wird, umfasst dies sowohl Aktivitäten, die in der Vergangenheit stattfanden, als auch solche, die zukünftig geplant sind oder prognostiziert wurden. An einigen Stellen kann es sinnvoll sein, zwischen vergangenen, geplanten und prognostizierten Tätigkeiten zu unterscheiden. Im Allgemeinen ist jedoch, im Sinne einer einfachen und einheitlichen Handhabung, die Gleichbehandlung von Aktivitäten mit unterschiedlichen Modalitäten sinnvoll. Daraus lassen sich folgende Punkte ableiten:

- Aktivitäten sind zeitbehaftet
- Zeitliche Abläufe sollen auf natürliche Weise im Modell abgebildet werden.
- Zukünftige Aktivitäten lassen sich im Modell abbilden.

- Es kann zwischen vergangenen, geplanten und prognostizierten Tätigkeiten unterschieden werden.
- Ob eine Aktivität vergangen, geplant oder prognostiziert ist bleibt im allgemeinen Umgang mit der Aktivität transparent.
- Zeitliche Nähe zwischen Aktivitäten kann effizient und auf natürliche Weise ermittelt werden.

7.1.4 Umgang mit räumlichen Informationen

Neben Beziehungen, die durch zeitliche Nähe hergestellt werden, sind Beziehungen, die durch räumliche Nähe hergestellt werden von Interesse. Analog zu den Anforderungen bei zeitlichen Zusammenhängen ergeben sich folgende Punkte:

- Aktivitäten sind ortsbehaftet
- Räumliche Informationen werden auf natürliche Weise in das Modell abgebildet.
- Räumliche Nähe zwischen Aktivitäten kann effizient und auf natürliche Weise ermittelt werden.

7.2 Wahl des Kontextmodells

Zur Entwicklung eines generischen Kontexdatenschemas im nächsten Abschnitt wird mit der Context Modelling Language [HIR02] zunächst ein methodischer Ansatz zur konzeptionellen Modellierung gewählt (siehe. Abschnitt 5.2). In diesem Ansatz kann die Struktur des Implementationsschemas erarbeitet werden. CML bereitet die Überführung in ein relationales Implementationsmodell vor, macht aber eine relationale Implementierung nicht notwendig.

Wie in Kapitel 5 festgestellt wurde, besitzen die Kontextmodelle jeweils spezifische Stärken und Schwächen (vgl. Tabelle 5.2). Die Wahl eines geeigneten Kontextmodells ist somit abhängig von dessen Einsatzzweck.

Da der Umgang mit durch Aktivitäten entstehenden Beziehungen, und der Umgang mit zeitlichen Verläufen zu den zentralen Problemen beim Umgang mit Aktivitätskontext gehört, ist ein relationales Kontextmodell als Basis für den eigenen Ansatz sinnvoll. In ihm können Beziehungen am unmittelbarsten und effizientesten verwaltet werden. Da bei der Verwaltung und Analyse von vergangenen und zukünftigen Aktivitäten mit größeren Datenmengen zu rechnen ist, ist auch die gute Zugriffseffizienz in einem solchen Implementationsmodell ein Kriterium, das für diese Wahl spricht (siehe Tabelle 5.2).

Der hier modellierte Aktivitätskontext ist jedoch nur ein Teil von Kontext (vgl. Abschnitt 4.1.3). Komplexere kontextadaptive Systeme könnten auch andere Kontextbereiche nutzen wollen, wofür ggf. andere Modelle, wie z.B. Ontologien zur Repräsentation

von allgemeinem Weltwissen und Schlussfolgerungen auf der Wissensbasis, geeigneter sind (vgl. Tabelle 5.2). Da es nicht wünschenswert ist, für jeden dieser Bereiche eine eigene Kontextinfrastruktur zu schaffen, sollte beim Entwurf der Middleware die Möglichkeit berücksichtigt werden, auch andere Kontextmodelle für bestimmte Teilbereiche einzusetzten. Dies würde auf die Möglichkeit hinauslaufen durch Erweiterungen auch ein hybrides Kontextmodell zuzulassen, um von den Vorteilen unterschiedlicher Ansätze profitieren zu können.

Als übergreifendes und verbindendes Modell könnte ein datenflussartiger Ansatz (vgl. Abschnitt 5.3.5) gewählt werden. In einem solchen Ansatz ist vor allem eine gute Unterstützung für die Verteilung der einzelnen Komponenten gegeben. Darüberhinaus ist die Kapselung und Verdichtung der Daten in den einzelnen Komponenten kennzeichnendes Merkmal des Ansatzes (siehe Tabelle 5.2). Eine datenflussartige, übergreifende Struktur ermöglicht es, einzelnen Komponenten intern, ein Kontextmodell ihrer Wahl verwenden, das für ihre Aufgabe angemessen ist und kann zur Verbindung unterschiedlich gearteter Komponenten genutzt werden.

7.3 Ein generisches Kontextdatenschema für Aktivitätskontext

Zur Erläuterung des entwickelten Kontexdatenschemas für aktivitätsorientierten Kontext, werden in diesem Abschnitt zunächst anhand eines CML-Diagramms (vgl. Abschnitt 5.2.1) die Kernkomponenten des Schemas vorgestellt. Anschließend werden einige Details, insbesondere die genutzten Metainformationen (Abschnitt 7.3.4.1) und der Umgang mit Integritätsbedingungen und Konflikten (Abschnitt 7.3.4.2) geschildert. Abschließend wird erörtert, wie der Kern des Modells für komplexere Anwendungsszenarien erweitert werden kann (Abschnitt 7.3.5).

Die Kernkonzepte des Kontexdatenschemas orientieren sich an den Überlegungen zu Aktivitäten aus Kapitel 3, an Zimmermanns Kontextdefinition (vgl. 4.1.2) und den Vorüberlegungen dieses Kapitels. Insgesamt ist der Kernbereich sehr simpel und abstrakt gehalten. Dies entspricht den in Abschnitt 7.1 formulierten Anforderungen und ist gerade die Stärke dieses Ansatzes.

Eine Überblick über die Kernkonzepte des Schemas gibt das CML-Diagramm in Abbildung 7.1 die einzelnen Elemente werden im Folgenden erläutert.

7.3.1 Entitäten

Entitäten sind eine allgemeine Abstraktion von Personen, Orten und digitalen oder realen Objekten. Entitäten können das Objekt von Aktivitäten sein.

Dabei wird von Entitäten nichts weiter gefordert, als dass diese durch einen eindeutigen Bezeichner identifizierbar sein müssen. Welche weiteren Eigenschaften eine Entität aufweist, ist für das Modell unerheblich, so dass die einzelnen Klassen von Entitäten beliebig detailliert oder abstrakt modelliert sein können, solange diese jeweils einen glo-

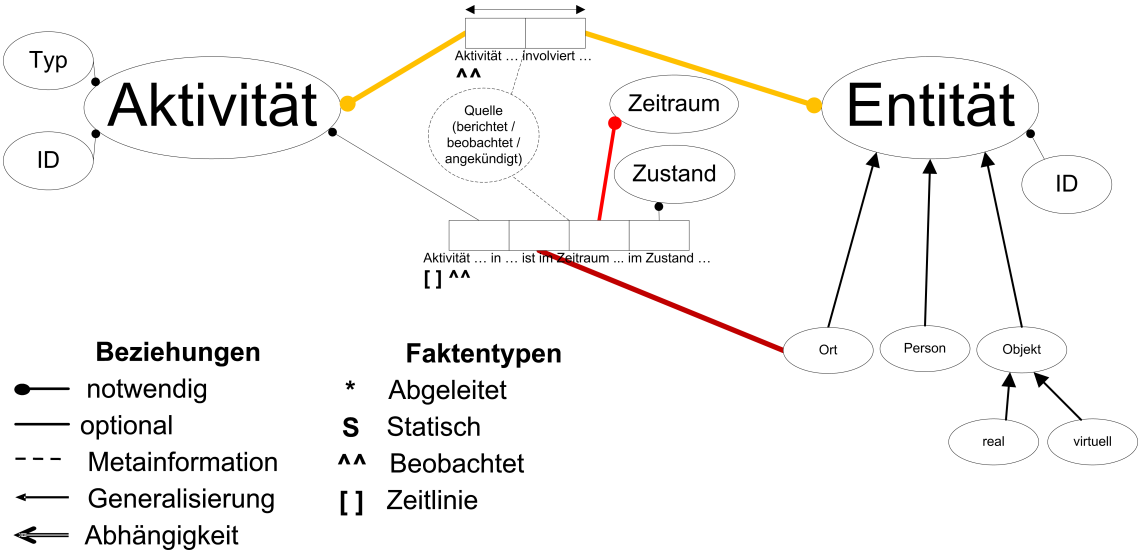

Abbildung 7.1: Die Kernkomponenten des entwickelten Kontexdatenschemas zur Nutzung von Aktivitätskontext bilden Entitäten, Aktivitäten und die Beziehungen, die zwischen diesen bestehen.

balen eindeutigen Bezeichner aufweisen. Die detaillierte Modellierung von Entitäten ist nicht Teil des eigenen Modells, sondern wird den Software-Werkzeugen überlassen, die für den Umgang mit den Entitäten vorgesehen sind.

7.3.2 Aktivitäten

Ähnlich wie Entitäten müssen Aktivitäten über einen eindeutigen Bezeichner verfügen, der diese identifiziert. Zusätzlich erhalten Aktivitäten einen Typ, welcher die Aktivität näher beschreibt.

Ruft Bob beispielsweise Alice an, würde dieses als Aktivität bzw. Aktion mit dem Typ „Telefonat“und den daran beteiligten Entitäten „Bob“ und „Alice“ dargestellt werden. Dieser Ansatz verzichtet bewusst auf die explizite Darstellung der Semantik einer Aktivität, da dieses in den Bereich der Wissensmodellierung fallen würde. Sollten solche Informationen benötigt werden, müsste das Modell um entsprechende Komponenten zum Weltwissen erweitert werden.

Stattdessen liegt der Fokus auf der Erfassung der durch Aktivitäten entstehenden Beziehungen (vgl. Abschnitt 4.2.4.1) frei von detaillierter Semantik.

Allerdings wird im Modell davon ausgegangen, dass Aktivitäten einen Lebenszyklus haben. Wie in Abbildung 7.2 dargestellt, können diese, sobald sie begonnen haben unterbrochen werden, z.B. zu Gunsten anderer Aktivitäten und können auf drei unterschiedliche Arten beendet werden. So ist es möglich, dass Aktivitäten erfolgreich, nicht erfolgreich oder ohne dass über Erfolg oder Misserfolg etwas bekannt ist, beendet werden.

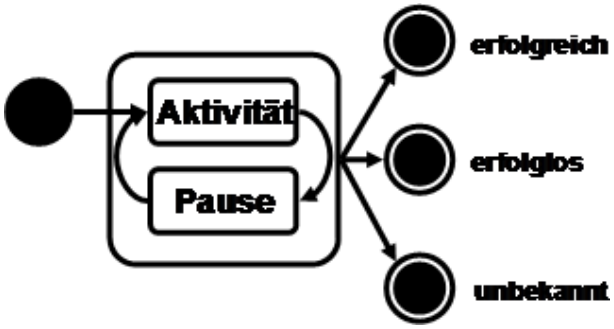

Abbildung 7.2: Lebenszyklus einer Aktivität.

7.3.3 Beziehungen

Auf Aktivitäten wird in zwei Arten von Faktentypen verwiesen. Der erste Faktentyp gibt die Beteiligung von Entitäten an einer Aktivität an (Abbildung 7.1, oben). Pro beteiligter Entität an einer Aktivität existiert an dieser Stelle ein Eintrag. Die Semantik, also die Art der Beteiligung, wird dabei nicht näher spezifiziert.

Der zweite Faktentyp ist verpflichtend und für jede Aktivität mindestens einmal vorhanden. Er verweist auf den Zustand der Aktivität sowie darauf, zu welcher Zeit dieser Zustand eingenommen wurde und an welchem Ort die Zustandsänderung erfolgte. Der erste und zwingende Eintrag für jede Aktivität kennzeichnet ihren Beginn. Für jede Zustandsänderung erfolgt ein weiterer Eintrag. So realisiert dieser Faktentyp den Lebenszyklus der Aktivität.

Die drei Arten von Beziehungen, die durch Aktivitäten zwischen Entitäten hergestellt werden, (vgl. Abschnitt 4.2.4.1) werden durch diese Faktentypen indirekt realisiert. Über den ersten Faktentyp kann die gemeinsame Beteiligung von Entitäten an einer Aktivität ermittelt werden (Abbildung 7.1, gelbe Verbindung).

Mit Hilfe des zweiten Faktentyps kann räumliche Nähe (dunkelrote Verbindung) und zeitliche Nähe (hellrote Verbindung) bei der Verwendung von Entitäten bestimmt werden.

7.3.4 Verfeinerungen

Insbesondere für den Umgang mit Unsicherheiten, die sich beim Umgang mit Aktivitäten ergeben, aber auch für die allgemeine Integrität der Daten, muss das Schema Möglichkeiten bereitstellen. Diese Möglichkeiten entstehen durch geeignete Metadaten (Abschnitt 7.3.4.1) und Integritätsbedingungen (Abschnitt 7.3.4.2).

7.3.4.1 Metainformationen

Aktivitäten werden mit Metainformationen versehen, welche die Art der Quelle für diese Information kennzeichnen. Diese Metainformationen machen es möglich zwischen durch Software-Werkzeugen berichteten, von außen beobachteten[3], angekündigten, etwa aus dem Kalender des Nutzers erhaltenen, prognostizierten oder, z.B. aufgrund von Nebenbedingungen abgeleiteten Informationen zu unterscheiden. Im Fall einer indirekten Quelle, also bei Beobachtung, Ankündigung, Prognose oder Ableitung einer Information, können weitere Metainformationen angegeben werden, welche die Informationsqualität näher kennzeichnen.

Ein Mechanismus um Nebenbedingungen für Aktivitäten zu formulieren, ist die Möglichkeit der Angabe von Timeouts, nach denen automatisch bestimmte Zustände eingenommen werden. So kann z.B. spezifiziert werden, dass, wenn es nach einer Minute keine weiteren Informationen über eine Aktivität gibt, diese als nicht erfolgreich beendet angesehen wird.

7.3.4.2 Integritätsbedingungen und Umgang mit Konflikten

Zu den geforderten Integritätsbedingungen in diesem Modell gehören die Eindeutigkeit der Bezeichner von Entitäten sowie die Existenz mindestens einer Angabe zum Zustand der Aktivität. Darüberhinaus kann eine Aktivität nur einmal beendet werden.

Die im vorherigen Abschnitt erläuterten Timeouts bilden zur Laufzeit, pro Aktivität formulierbare Integritätsbedingungen. Diese erfordern im Gegensatz zu den bisherigen Bedingungen ein aktives Verhalten im Konfliktfall. Sollte ein Timeout überschritten sein, muss automatisch ein Eintrag für den daraus resultierenden Zustandsübergang erfolgen. Die anderen Integritätsbedingungen können bereits durch Nichtannahme gültiger Werte garantiert werden. Beides muss durch die Middleware, die diesen Ansatz implementiert, gewährleistet werden.

7.3.5 Erweiterungen für Anwendungsszenarien

Das bisher geschilderte Schema stellt den minimalen Kern eines Implementationsschemas für Aktivitätsdaten dar. Dieses Schema kann und muss für spezifische Anwendungen erweitert werden. Die Erweiterungsmöglichkeiten liegen vor allem in den Bereichen Weltwissen und Reasoningkomponenten. Abbildung 7.3 zeigt einige Erweiterungsmöglichkeiten.

Um die entwickelte Beispielanwendung zu unterstützen, muss das Modell um eine Komponente erweitert werden. Diese Komponente ist dafür verantwortlich, zu je zwei gegebenen Entitäten, deren jeweils aktuelle Relevanz, d.h. das Maß in dem diese in Verbindung zu einander stehen, zu bestimmen. Dieses Maß bezieht sich jeweils auf zwei Entitäten, seine Ermittlung ist jedoch abhängig von den erfassten Aktivitäten (dargestellt

[3] Gemeint sind beispielsweise über Sensoren erfasste Aktivitäten.

in Abbildung 7.3 oben rechts). Dabei ist denkbar, dass unterschiedliche Heuristiken genutzt werden. So könnte etwa die Beteiligung der Entitäten an gemeinsamen Aktivitäten in einem bestimmtem Zeitraum oder über die Beteiligung an allen bekannten Aktivitäten ermittelt werden.

Für Prognosen von Aktivitäten ist ebenfalls eine Erweiterung des Modells denkbar. Prognosen (Abbildung 7.3 oben links) könnten allein aufgrund beobachteter Muster in den bisherigen Aktivitätsabläufen, oder aber mit Hilfe eines Taskmodells erfolgen, in dem typische Abfolgen von Aktivitäten für bestimmte Zwecke modelliert sind (Abbildung 7.3 unten links).

Auch statische Beziehungen zwischen Entitäten, beispielsweise Person A arbeitet an Platz B, können als Weltwissen modelliert werden (Abbildung 7.3 unten rechts) und in den anderen Komponenten als zusätzliche Grundlage für Schlussfolgerungen dienen.

Die Semantik von Aktivitäten ließe sich als Weltwissen ebenfalls näher spezifizieren (siehe Abbildung 7.3 unten mittig; vgl. Abschnitt 7.3.2).

Zusammenfassend lässt sich sagen, dass der in Abbildung 7.1 beschriebene Kern des Schemas, die dynamisch erfassten Fakten zu Aktivitäten sind. Die Erweiterungsmöglichkeiten des Schemas liegen im Bereich des statisch bereitgestellten Weltwissens und in den auf den anderen Komponenten beruhenden Schlussfolgerungen.

7.4 Zusammenfassung

Die Anforderungen an ein Kontextdatenschema liegen vor allem in der benötigten Abstraktheit und im Umgang mit den, durch Aktivitäten hergestellten, Beziehungen. Da kein allgemeines Modell oder Kontextdatenschema für Aktivitätskontext bekannt ist, wurde ein eigenes Datenschema in einem konzeptuellen Modell entwickelt. Viele Paradigmen der Kontextmodellierung legen sich früh auf ein Implementationsmodell für die Kontextdaten fest. Im Gegensatz dazu wurde in dieser Arbeit zunächst bewusst auf eine Entscheidung über ein Implementationsmodell verzichtet und stattdessen zunächst ein konzeptuelles Kontextdatenschema in CML spezifiziert.

Der Kern des Schemas liegt in der Modellierung der zu den Aktivitäten dynamisch erfassten Fakten. Es wird der Lebenszyklus von Aktivitäten, anhand von Zeit und Ort der Zustandsänderungen erfasst sowie die an Aktivitäten beteiligten Entitäten. Um die nötige Abstraktheit zu gewährleisten, werden Entitäten, Aktivitäten und die durch Aktivitäten hergestellten Beziehungen zwischen Entitäten frei von detaillierterer Semantik modelliert.

Erweiterungen des Modells für komplexere Szenarien sind nötig und möglich. Die Möglichkeiten liegen vor allem im Zufügen von Weltwissen und in der Erweiterung um Fakten, die durch Schlussfolgerung aus der Analyse vorhandener Daten, gewonnen werden.

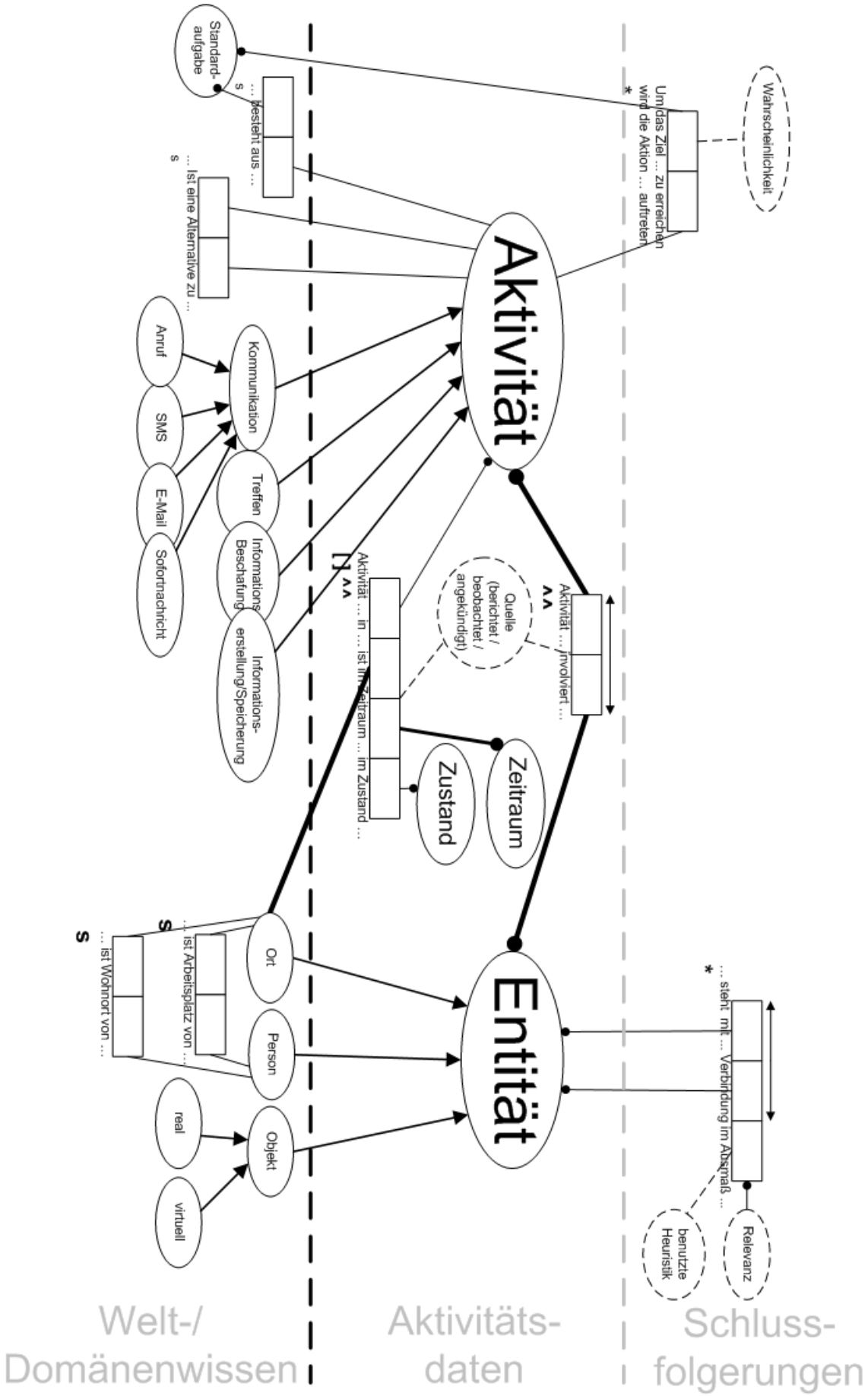

Abbildung 7.3: Erweiterungsmöglichkeiten des Kontextdatenschemas.

8 Kernkonzepte der Middleware

Nachdem verwandte Ansätze Softwareinfrastrukturen für kontextadaptive System herzustellen vorgestellt und bewertet wurden (vgl. Kapitel 6), wird in diesem Abschnitt der eigene Ansatz erläutert. Im Schwerpunkt dieser Arbeit liegen dabei Lösungsansätze, die sich an den Problemen im Umgang mit Aktivitätskontext orientieren. Grundlage dafür ist das entwickelte Kontextdatenschema (vgl. Kapitel 7), das möglichst eng mit der Middleware verzahnt werden soll.

Zunächst wird ein Überblick über die Anforderungen an die Middleware, die zu verwaltenden Bereiche und die Zusammenhänge zwischen diesen gegeben (Abschnitt 8.1). Danach werden einzelne Teilaspekte, Designansätze und Mechanismen der Middleware vorgestellt.

Dabei wird als erstes erläutert, wie Software-Werkzeuge Informationen über Aktivitäten an die Middleware gegeben können (Abschnitt 8.2). Dann wird der modulare Aufbau von Middleware-Komponenten (Abschnitt 8.3) und die für ihre lose Kopplung zuständigen Kommunikationsmechanismen vorgestellt (Abschnitt 8.4). Inhalt des darauffolgenden Abschnitts ist die Adressierung von Entitäten über pfad-basierte Namensschemata und die Möglichkeiten zur Nutzung von kontextabhängigen, dynamischen Namensschemata (Abschnitt 8.5). Danach werden die Mechanismen zum Umgang mit zeitlichen Abläufen und zukünftigen Ereignissen behandelt. Abschließend werden Erweiterungsmöglichkeiten für die Middleware vorgeschlagen (Abschnitt 8.7) und in einem Fazit erläutert, wie durch das Zusammenspiel der vorgestellten Mechanismen die erarbeiteten Anforderungen umgesetzt werden können (Abschnitt 8.8).

8.1 Kernaufgaben der Middleware

Wie bereits im Kapitel Middlewarearchitekturen für Context Awareness bilanziert wurde (siehe Abschnitt 6.4), erfüllen die bisherigen Ansätze, die für aktuelle mobile Geräte entwickelt wurden, kaum Anforderungen, die über einen einfachereren Umgang mit vorhandener Hardware und Kommunikationsinfrastruktur hinausgehen (vgl. Abschnitt 6.4.2). Insbesondere verfügte keiner dieser Ansätze über eine Unterstützung von einem expliziten Kontextmodell oder Kontextdatenschema. Allen Ansätzen, auch denen, die nicht für heutige, mobile Geräte optimiert waren, ist gemeinsam, dass diese kaum den Umgang mit Aktivitäten (vgl. Abschnitt 6.4.4) und zeitlichen Verläufen unterstützen (vgl. Abschnitt 6.4.5). Auch im Bereich der Fehlertoleranz sind diese Ansätze bisher kaum ausgereift (vgl. Abschnitt 6.4.3).

Neben den grundlegenden Aufgaben der Middleware, dem vereinfachten Umgang mit vorhandener Hard- und Software sowie der Kommunikationsinfrastruktur soll der hier entwickelte Ansatz helfen, diese Lücken zu schließen. Hierfür soll eine Instanz des in

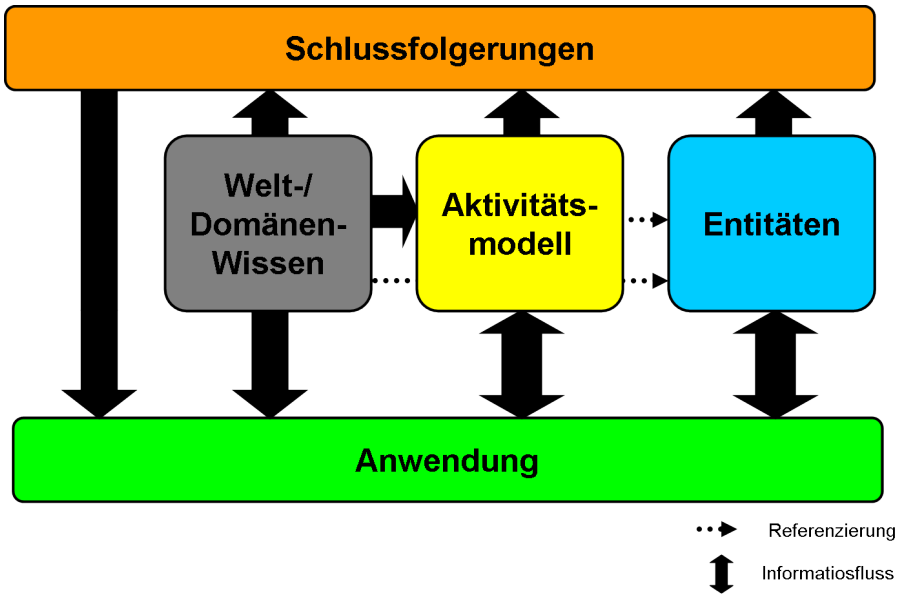

Abbildung 8.1: Teilbereiche und Informationsflüsse in der Middleware.

Kapitel 8.1 entwickelten Kontextdatenschemas für Aktivitäten verwaltet werden. Routineaufgaben beim Umgang mit den Daten des Kontexdatenschemas, wie z.B. die Bekanntmachung von Aktivitäten durch Software-Werkzeuge (Abschnitt 8.2), der Adressierung und das Abrufen von Entitäten (Abschnitt 8.5) und der Verwaltung von zeitlichen Verläufen (Abschnitt 8.6) sollen dabei von der Middleware übernommen bzw. vereinfacht und vereinheitlicht werden.

Der Kernbereich der Middleware liegt somit beim Umgang mit Aktivitäten und dem Verwalten von Entitäten. Wie bereits in Abschnitt 7.3.5 festgestellt, liegen die Erweiterungsmöglichkeiten im modellieren von Welt- bzw. Domänenwissen und der bereitstellung von Reasoning-Mechanismen. Abbildung 8.1 stellt die verschiedenen Bereiche und die Beziehungen zwischen diesen dar. Aufbauend auf den Daten der Entitäten, des Kontexdatenschemas für Aktivitätskontext und ggf. von domänenspezifischem Wissen können Mechanismen zum Reasoning und zur Ableitung von Wissen oder Annahmen eingesetzt werden.

Um es zu ermöglichen, den bereitgestellten minimaler Kern des Kontexdatenschemas zu erweitern, ist es erforderlich, dass die Middleware solche Erweiterungen durch Anwendungen ermöglicht und unterstützt. Erweiterungen des Schemas sollen ebenfalls von der Middleware verwaltet werden und von verschiedenen Anwendungen auf einfache Art bereitgestellt und gemeinsam genutzt werden können. Hierzu sind Mechanismen erforderlich, um weitere Module bereitstellen zu können (Abschnitt 8.3) und um Kommunikation zwischen Anwendungen und Modulen, bzw. der Module untereinander zu ermöglichen (Abschnitt 8.4).

8.2 Aktivitäten aus Anwendungssicht

Anwendungen, die den Verlauf von, durch sie realisierten Aktivitäten an die Middleware kommunizieren möchten, nutzen hierfür ein vereinfachtes Modell von Aktivitäten. In diesem Modell wird ein einfacher Lebenszyklus von Aktivitäten angenommen (vgl. Abschnitt 7.3.2). Einmal begonnene Aktivitäten können dem Modell zufolge unterbrochen und wieder aufgenommen werden. Sie können erfolgreich, ohne Erfolg oder ohne dass über Erfolg bzw. Misserfolg etwas bekannt ist, beendet werden. Unabhängig davon, an welcher Stelle sich eine Aktivität in ihrem Lebenszyklus befindet, können ihr jederzeit Entitäten hinzugefügt werden, die an ihr beteiligt sind.

Um den Status einer Aktivität an die Middleware zu kommunizieren, nutzt die Anwendung eine bereitgestellte API (vgl. Abschnitt 9.2.4). Die Anwendung kann über die API Objekte erzeugen, die Aktivitäten repräsentieren und über Methodenaufrufe auf diesem Objekt einer Aktivität Entitäten zufügen oder ihren aktuellen Zustand setzten. Die Middleware erzeugt daraufhin den Zustandsübergängen entsprechende Ereignisse und verarbeitet diese durch die Module des Frameworks weiter.

8.3 Modularisierung

Ähnlich der Struktur des ContextToolkits (vgl. 6.2.1.1) stellt die Middleware eine Infrastruktur bereit, in der Funktionalität jeweils in Komponenten für bestimmte Aufgaben modularisiert wird. Da grundsätzlich ereignis- und anfragebasierte Kommunikation ermöglicht werden soll, gliedern sich diese Komponenten in ereigniserzeugende und abfragbare Komponenten (vgl. Abbildung 8.2). Auch Mischformen, also Komponenten, die Ereignisse erzeugen, aber auch abgefragt werden können, sind möglich. Beispiel für eine ereigniserzeugende Komponente, wäre etwa ein Sensor, der das Telefonverhalten auf einem Mobiltelefon überwacht und sobald Telefonate beginnen oder beendet werden entsprechende Ereignisse erzeugt. Die Komponente, die das Kontextdatenschema für Aktivitätskontext (vgl. Abschnitt 7.3) implementiert, ist ein Beispiel für ein Modul, an das Anfragen gestellt werden können. Dieses Modul aktualisiert seine Daten auf Basis der beobachteten Ereignisse. Andere Komponenten oder Anwendungen, die Informationen über vergangene, zukünftige oder laufende Aktivitäten benötigen, können entsprechende Anfragen stellen, beispielsweise nach allen Aktivitäten, die in fünf Minuten beginnen sollen.

Abfragbare Komponenten können prinzipiell beliebige Schnittstellen für Abfragen bereitstellen. Um diese jedoch zu vereinheitlichen, werden standardisierte Schnittstellen für bestimmte Arten von Datenstrukturen definiert und angeboten. Diese Schnittstellen umfassen eine Definition für ID-basierte simple Anfragen, eine Definition für das pfadbasierte Abfragen von hierarchischen Strukturen und eine Definition für die SQL-basierte Abfrage von relationalen Daten. Diese Komponenten sind nicht bloß als passive Datenspeicher zu verstehen, sondern sie können Daten auch aktiv verarbeiten. Ein Beispiel für

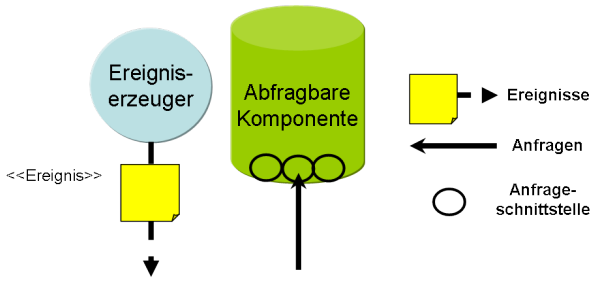

Abbildung 8.2: Ereigniserzeuger und abfragbare Module sind die grundlegenden Komponententypen der Middleware.

eine aktive Verarbeitung der Daten wäre etwa eine Komponente, die auf die Aktivitätsdaten zugreift und auf dieser Basis die augenblickliche Relevanz von Entitäten schätzt. Ein weiteres Beispiel ist eine Komponente die low-level-Sensordaten, wie z.B. Positionsdaten in verdichtetere, symbolische Daten etwa in eine Orts-Entität überführt und die höherwertigen Daten bereitstellt.

Ereigniserzeugende Komponenten brauchen lediglich die Ereignisse für deren Erzeugung sie zuständig sind erzeugen und an die Middleware geben. Diese übernimmt die weitere Verteilung des Ereignisses.

Beide Arten von Komponenten können die Daten, auf denen sie arbeiten sowohl über Anfragen, als auch durch den Empfang von Ereignissen oder über eine Mischform beziehen.

Auf welche Art und Weise diese Daten weiterverarbeitet werden, ist den Komponenten überlassen und wird nach außen hin gekapselt.

8.4 Kommunikationsmechanismen für lose gekoppelte Module

Um die lose Kopplung von Modulen zu erreichen, muss jedes Modul seine Eigenschaften deklarieren. Insbesondere muss das Modul angeben, an welcher Art von Ereignissen es interessiert ist und welche Arten von Abfrageschnittstellen es anbietet.

Die Ereignisse, an denen das Modul interessiert ist, können dabei durch unterschiedliche Kriterien spezifiziert sein. So kann das Modul deklarieren, an welcher Klasse von Ereignissen es interessiert ist, ob es über Ereignisse, an denen eine bestimmten Klasse von Entitäten beteiligt ist, informiert werden möchte oder lediglich Ereignisse zu einer bestimmten Entität erhalten möchte. Diese Kriterien können auch miteinander kombiniert werden, so dass es für ein Modul beispielsweise möglich ist, sich für alle Ereignisse zu registrieren, die den Beginn einer Aktivität signalisieren und an der eine bestimmte Person beteiligt ist. Auch zur Laufzeit können sich Module für den Erhalt bestimmter Ereignisse registrieren oder ihre Kriterien modifizieren. Durch ein Modul erzeugte Ereignisse werden durch die Middleware analysiert und an alle interessierten Module zugestellt (vgl. Abbildung 8.3, links).

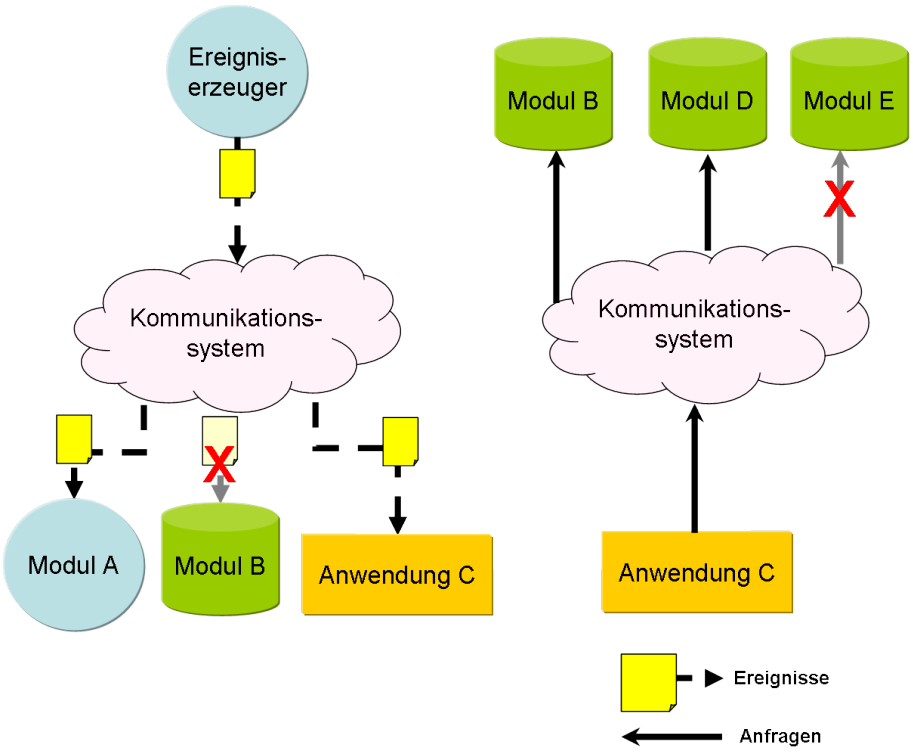

Abbildung 8.3: Many-to-many Kommunikation in der Middleware.

Das Kommunikationssystem stellt somit ein eventbasiertes Publish-Subscribe-System dar [Ait07]. Das Einschreiben für den Erhalt von Ereignissen basiert, wie oben beschrieben, auf inhaltsbasierten Filtern. Ein solches System hat den Vorteil, dass es die Komponenten örtlich, zeitlich und in ihrem Ablauf voneinander entkoppelt: Einzelne Komponenten benötigen keinerlei Kenntnis oder Referenzen voneinander. Das Zustellen von Ereignissen kann verzögert erfolgen, jederzeit können neue Komponenten oder Ereignisfilter dem System zugefügt oder entfernt werden. Komponenten, die Ereignisse erzeugen, blockieren nicht bis diese verarbeitetet wurden, sondern sind in ihrem Ablauf unabhängig von der Verarbeitung der Ereignisse an anderen Stellen [Ait07]. Aufgrund dieser Eigenschaften ist ein solches System hervorragend für verteilte Ubiquitous Computing Szenarien geeignet [Ait07].

Für den Fall, dass anfragebasierte Kommunikation genutzt werden soll, müssen Module, die Abfrageschnittstellen anbieten diese explizit deklarieren. Inhalt der Deklaration ist die Spezifikation der Abfragesprache und welche Ergebnismengen die Module liefern. Mögliche Ergebnistypen sind z.B. Entity-Listen, Entity-Listen eines bestimmten Typs oder tabellenförmige Strukturen.

Anfragen an das System werden an alle passenden Module gesendet. Die Ergebnisse durch die Middleware aggregiert und an das abfragende Modul zurück geliefert (vgl. Abbildung 8.3, rechts). Dieser Mechanismus ermöglicht es, Daten sowohl horizontal als auch vertikal zu fragmentieren [KE04, S. 447ff].

Da auf diese Weise sowohl Anfragen an mehrere Module gleichzeitig gestellt werden können als auch Nachrichten von mehren Quellen bezogen und an mehrere Empfänger gleichzeitig gesendet werden können, wird eine n:m Kommunikation realisiert. Dabei kommunizieren die Module nie direkt mit ihren Kommunikationspartnern, sondern stets vermittelt durch die Middleware. Diese Struktur ermöglicht eine hohe Flexibilität in der Kommunikationsstruktur und im Nachrichtenfluss, entkoppelt die einzelnen Module voneinander und ermöglicht die Verteilung und den transparenten Austausch der beteiligten Komponenten.

8.5 Entitäten und kontextabhängige, dynamische Namensschemata

Eine besondere Klassen von Anfragen sind die Anfragen nach Entitäten. Prinzipiell hat jede Entität einen eindeutigen und permanenten Namen, dieser entspricht einem Pfad-Schema. Solch ein Pfad-Schema besteht im einfachsten Fall aus dem Entitätstyp und einer eindeutigen ID, also beispielsweise `/Person/id/5/`. Zusätzlich zu ihrem permanenten und eindeutigen Namen kann eine Entität auch über beliebig viele weitere kontextabhängige Pfad-Schemata erreichbar sein. Diese Schemata sind nicht zwingended permanent, sondern vielmehr durch den aktuellen Kontext bestimmt. Zudem müsen sie nicht eindeutig sein. So liefert z.B. das Schema `/Person/id/5/related/Place` alle Orte, die aktuell in Verbindung mit Person 5 stehen. Im Gegensatz zum Context Filesystem [HC02] (vgl. Abschnitt 6.2.6) lassen sich alle Arten von Entitäten, nicht bloß Dateien, über diese Schemata adressieren.

Die Middleware stellt Mechanismen zur Verfügung, wie diese Pfad-Schemata transparent aufgelöst werden können. Bei der Auflösung der Schemata liegt ein weiterer Unterschied zum Context Filesystem vor. Die Auflösung erfolgt nicht durch eine zentrale Stelle, sondern dezentral in den einzelnen Komponenten. Komponenten können Teilsegmente des Schemas, die in ihren Zuständigkeitsbereich fallen auflösen und die von ihnen unauflösbaren Teile zur weiteren Verarbeitung durch andere Komponenten zunächst unverarbeitet lassen.

Wie diese dezentrale Auflösung im einzelnen funktioniert, wird im Abschnitt 9.2.2.5 erläutert.

8.6 Zeitliche Verläufe und zukünftige Ereignisse

Um zeitliche Abläufe und zukünftige Ereignisse zu unterstützen, stehen einige besondere Mechanismen zur Verfügung. Zunächst existiert neben einem grundlegenden Ereignistyp, der sofortige Ereignisse repräsentiert, dem *InstantEvent*, ein Ereignistyp zum Umgang mit zukünftigen Ereignissen. Ein solches *ScheduledEvent* kapselt ein weiteres Ereignis zusammen mit einem Zeitpunkt, an dem dieses stattfinden wird. Desweiteren

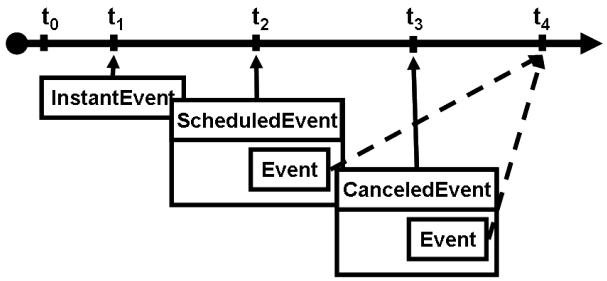

Abbildung 8.4: Verwaltung eines Zeitstrahls aufgrund von *InstantEvents*, *ScheduledEvents* und *CanceledEvents*.

enthält das Ereignis Metainformationen darüber, woher die Informationen über das zukünftige Ereignis stammen und ob dieses durch Komponenten des Systems vorhergesagt oder durch den Nutzer angegeben wurde (z.B. durch einen Kalendereintrag).

Bei zukünftigen Ereignissen bestehen ein paar grundsätzliche Problematiken: Zunächst ist nie wirklich sicher ob diese stattfinden werden, da es nach Kenntnis des Autors bis heute leider keine zuverlässige Methode gibt die Zukunft vorher zusagen. Auch kann bei Weitem nicht immer festgestellt werden, ob diese Ereignisse auch wirklich eingetroffen sind. Ein Kalendereintrag „hübsche Blumen kaufen“ etwa wird sich nur schwer durch eine Software überprüfen lassen. Falls die Ereignisse stattfinden und auch festgestellt werden können, werden sie in den seltensten Fällen exakt zu dem Zeitpunkt eintreten für den sie angekündigt wurden. Um mit dieser Art von Unvorhersehbarkeit umgehen zu können, ist es möglich, weitere Metainformationen für zukünftige Ereignisse anzugeben. So ist es möglich, einen frühesten und spätesten Zeitpunkt für den Eintritt des Ereignisses festzulegen und ob, falls nichts hierüber festgestellt werden kann, davon ausgegangen werden soll, dass dieses Ereignis auch wirklich stattfindet.

Wird bekannt, dass ein Ereignis doch nicht stattfinden wird, etwa weil der Nutzer einen Kalendereintrag wieder entfernt, kann die Middleware über ein *CanceledEvent* darüber informiert werden. Um auf bestimmte geplante Ereignisse zu verweisen, ist es an dieser Stelle nötig, dass Ereignisse eindeutig identifiziert werden können. Hierfür erhält jedes Ereignis einen eindeutigen Bezeichner in Form einer ID. Diese ID wird bei der Erzeugung des Ereignisses durch die Middleware generiert.

In der Middleware existieren spezielle Repositories, die *TimelineRepositories*, zur Verwaltung von zeitlichen Verläufen. Sie fungieren gleichermaßen als Historie wie als Speicher für zukünftige Ereignisse eines Types und stellen einen Zeitstrahl da (vgl. Abbildung 8.4). Dieser Zeitstrahl wird auf Basis von empfangener InstantEvents, ScheduledEvents und CanceledEvents aktualisiert und verwaltet. Er kann von anderen Modulen oder Anwendungen abgefragt werden. *TimelineRepositories* können beauftragt werden, zu bestimmten Zeitpunkten Ereignisse zu erzeugen, die auf bevorstehende Ereignisse hinweisen. So kann eine Anwendung den Auftrag geben, eine Stunde vor jedem Meeting eine Nachricht zu erhalten.

8.7 Erweiterungsmöglichkeiten

Im Rahmen dieser Arbeit kann nur eine minimale Version der Middleware realisiert werden. Es existieren jedoch eine Reihe von Erweiterungsmöglichkeiten, die beim Entwurf der Architektur bereits berücksichtigt wurden und sich gut zu einem späteren Zeitpunkt einfügen lassen.

8.7.1 Erweiterungen des Aktivitätsmodells

Das vorliegende Aktivitätsmodell ist absolut minimal gehalten. Erweiterungen sind an vielen Stellen denkbar. So könnte z.B. eine hierarchische Taxonomie von Aktivitäten als Weltwissen in die Middleware integriert und spezifische Eigenschaften von Aktivitäten modelliert werden. Die Beziehungen, die durch Aktivitäten hergestellt werden, ließen sich semantisch ausdifferenzieren. Module für das Erkennen von typischen Tätigkeitsmustern und darauf basierende Vorhersagen sind denkbar. Ein Modell zur Ermittlung von alternativen Handlungen bei der Blockierung von Aktivitäten könnte ebenfalls integriert werden.

8.7.2 Verteilung und Kooperation

Aufgrund der losen Kopplung der Komponenten und ihrer Kommunikationsstruktur folgt die Middleware einer Architektur, die sich prinzipiell auch als verteiltes System realisieren lässt. Dadurch wird eine Kooperation zwischen mehreren Nutzern denkbar, durch die Informationen über gemeinsame Aktivitäten untereinander geteilt und ergänzt werden. Ebenfalls ist eine Nutzung von Unternehmensinfrastruktur beispielsweise zur Abfrage von Domänenwissen oder der Integration von formal spezifizierten Workflows denkbar oder der Ausbau der Middleware für Ubiquitous-Computing-Szenarien. Insbesondere die verwendete, ereignisbasierte Kommunikation eignet sich gut für spontanes ein- und ausklinken einzelner Teilnehmer [Ait07].

8.7.3 Security und Privacy

Bevor die Middleware in einem realen Szenario eingesetzt werden kann, sind Mechanismen, welche die Privatsphäre der Nutzer gewährleisten, nötig. Dabei ist ein modulbasierter Zugriffsschutz sinnvoll. Module würden deklarieren, welche Art von Rechten zum Zugriff auf ihre Daten vorhanden sein müssen. Die Middleware wäre dann dafür zuständig nur solchen Modulen und Anwendungen den Zugriff auf diese Daten zu erlauben, denen von Seiten des Anwenders die entsprechenden Rechte eingeräumt wurden. Dadurch, dass sich innerhalb der Middleware auch Repositories implementieren lassen, die im Prinzip eingeschränkte Sichten[1] auf andere Repositories sind und zu diesen wie-

[1] Analog zu Sichten in relationalen Datenbanken

derum die nötigen Rechte spezifiziert werden können, ließe sich auch ein feinkörniger Zugriffsschutz für bestimmte Daten realisieren [KE04, S. 337f].

8.8 Fazit: Realisierung der Anforderungen

Kontextdatenschema und Middleware sind als eine Einheit zu betrachten. Die Middleware verwaltet eine Instanz des Kontexdatenschemas und ist mit diesem Teilweise eng verzahnt (vgl. Abschnitte 8.2, 8.5 u. 8.6), um einen möglichst einfachen Umgang mit den Kontextdaten gewährleisten zu können. Zusätzlich sind Mechanismen bereitgestellt, mit denen das Datenschema erweitert (Abschnitte 8.3f) und weitere Verzahnungen, etwa durch die dynamischen Namensschemata (vgl. Abschnitt 8.5), eingeführt werden können.

Durch die enge Verzahnung mit dem Kontextdatenschema werden die Anforderungen Abbildbarkeit des Kontextmodells (A1), Unterstützung für Aktivitäten (A2) und Einfachheit der Benutzung (A3) umgesetzt. Dabei wurden Mechanismen zum Umgang mit zeitlichen Verläufen (A5), die für den Umgang mit vergangenen oder zukünftigen Aktivitäten eine große Rolle spielen, aber auch für andere Kontextbereiche genutzt werden können, entwickelt (vgl. Abschnitt 8.6).

Es wurde darauf verzichtet, eine detaillierte Semantik einzelner Aktivitäten in die Middleware oder das Kontextdatenschema abzubilden. Das verwendete Modell ist abstrakt und minimal, wodurch die Kompaktheit (A9) des Systems begünstigt wird.

Die Middleware ist dabei kein geschlossener Block, sondern Erweiterungen für Anwendungsspezifische Komponenten können in der Middleware realisiert werden (A7). Hierfür wurden von vorneherein Möglichkeiten für den Umgang mit Daten unterschiedlichen Charakters vorgesehen und verschiedene Kommunikations- und Abfragemechanismen (A4) ermöglicht (vgl. Abschnitt 8.3f). Darüberhinaus existieren verschiedene Anknüpfungspunkte, um die Fähigkeiten der Middleware selbst zu erweitern (siehe 8.7).

Durch die Modularisierung in Komponenten kann eine Trennung von Schlussfolgerungen und Fakten (A8) realisiert werden, auch wenn die Middleware dieses nicht erzwingt. Es ist von vornherein vorgesehen, dass Komponenten und Anwendungen auch verteilt vorkommen können (A11), eine solche Verteilung wird durch die der Middleware zu Grunde liegenden Kommunikationsmechanismen unterstützt (vgl. Abschnitt 8.4).

Direkte Maßnahmen, um die Fehlertoleranz zu unterstützen, liegen in der Möglichkeit der Formulierung von Integritätsbedingungen z.B. für zeitliche Abläufe (vgl. Abschnitt 8.6). Außerdem sind die einzelnen Module von einander entkoppelt, so dass der Ausfall eines Moduls keine direkten Auswirkungen auf die anderen hat (A6) (vgl. Abschnitt 8.4).

Maßnahmen, die Privacy und Security (A10) gewährleisten, sind nicht direkt unterstützt, können aber an geeigneten Stellen in die Middleware integriert werden (vgl. 8.7.3).

8.9 Zusammenfassung

Um den Anforderungen an eine aktivitätsorientierte Kontextmiddleware gerecht zu werden, wurde eine modularisierte Architektur von lose gekoppelten Komponenten, die ereignis- und anfragebasierte Kommunikation ermöglicht, vorgeschlagen. Die vorgeschlagene Architektur unterscheidet sich von bisherigen Ansätzen vor allem darin, dass sie Abfrage und Kommunikationsmöglichkeiten weder zu sehr spezialisiert und überspezifiziert, wie es bei Optimierung für ein sehr eng gefasstes Kontextmodell der Fall wäre, noch zu sehr unterspezifiziert, wie dies bei einer dienstorientierten Struktur ohne Annahmen über das zugrundelegende Kontextmodell üblich ist. Desweiteren sind durch *TimelineRepositories* und die spezielle Ereignisklasse *ScheduledEvent* Mechanismen zum Umgang mit zeitlichen Abläufen und zukünftigen Ereignissen vorhanden.

Die, bisher in nur wenigen Ansätzen verfolgte, aber vielversprechende Idee von dynamischen kontextabhängigen Namensschemata wird im hier vorgestellten Ansatz aufgegriffen und eng in die Middleware integriert, um aus Anwendungssicht einen einfacheren Umgang mit kontextabhängigen Daten zu ermöglichen.

Die vorliegende Middleware weist zunächst nur eine möglichst minimale Funktionalität auf, ist aber in Hinblick auf ihre spätere Erweiterbarkeit entworfen worden.

9 Implementation

Das Kapitel zur Implementation gliedert sich in vier Abschnitte. Im ersten Abschnitt werden die technischen Plattformen, auf denen eine Implementation erfolgen kann kurz vorgestellt und die eigene Plattformwahl begründet (Abschnitt 9.1).

In den weiteren Abschnitten wird die Implementation selbst beschrieben. Abschnitt 9.2 gibt einen grundlegenden Überblick über Architektur der Middleware und die Realiserung ihrer Kernkonzepte.

Im darauf folgenden Abschnitt 9.3 wird die Middleware aus einer Perspektive beschrieben, die für Anwendungsentwickler, die mit der Middleware arbeiten möchten, bestimmt ist, in ihr finden sich eine Reihe von Code-Beispielen, die die Verwendung der Middleware praktisch darstellen. Der erste Teilabschnitt beschreibt, wie die vorhandenen Strukturen und Daten von Anwendungen genutzt und einfache Anfragen gestellt werden können und wie durch die Nutzung eines bereitgestellten kontextadaptiven GUI-Widgets die Entwicklung kontextadaptiver Anwendungen weiter vereinfacht wird (Abschnitt 9.3.1). Der nächste Teilabschnitt erläutert, wie das Implementationsschema erweitert werden kann, falls zusätzliche Kontextinformationen über die Middleware genutzt werden sollen (Abschnitt 9.3.2). Und der letzte Abschnitt beschreibt Möglichkeiten, die Fähigkeiten der Middleware selbst zu erweitern, etwa zur Nutzung neuer Kommunikationskanäle zwischen den Komponenten (Abschnitt 9.3.3).

Im letzten Abschnitt wird beschrieben, wie die Implementation der Demoanwendung erfolgte (Abschnitt 9.4), wobei die genutzen Datenquellen und auftretenden Datenflüße beschrieben werden (Abschnitt 9.4.1) und erläutert wird, wie die Heuristik, welche die aktuelle Relevanz von Entitäten feststellt, arbeitet (Abschnitt 9.4.2).

9.1 Wahl der Plattform

Die Implementation einer Anwendung, die das Beispielszenario prototypisch gemäß dem Szenario aus Kapitel 2 realisiert, soll auf einem Mobiltelefon erfolgen. Für die Entwicklung einer solchen Anwendung stehen unterschiedliche Plattformen verschiedener Gerätehersteller und Konsortien zur Verfügung.

9.1.1 Plattformanforderungen

Um die verschiedenen Plattformen hinsichtlich ihrer Eignung für die Realisierung von Middleware und Szenario bewerten zu können, wurden eine Reihe von Bewertungskriterien aufgestellt.

(P1) Abfangbarkeit von Anrufinformationen Informationen über ein- und ausgehende Anrufe sollen auf einfache Art und Weise abfangbar sein.

(P2) Abfangbarkeit weiterer Aktivitäten Informationen über weitere Aktivitäten des Benutzers, wie z.B. über das Versenden von Nachrichten, die Navigation zu bestimmten Orten oder dem Aufruf von Internetseiten, sollen ebenfalls auf möglichst einfache Weise abfangbar sein. Dies ist ein grundlegendes und nicht triviales Problem bei der Beobachtung von Nutzerverhalten [HBH+98].

(P3) Interprozesskommunikation Da Informationen zwischen der Middleware und den verschiedenen Anwendungen ausgetauscht werden müssen, sollten einfache und effiziente Möglichkeiten zur Kommunikation zwischen verschiedenen Prozessen bestehen. Die Daten, die ausgetauscht werden müssen, gehen dabei über einfache Signale hinaus und umfassen beispielsweise komplexere Anfragen und Antworten auf diese. Daher muss eine Kommmunikation auch komplexere Datenstrukturen unterstützen.

(P4) Verfügbarkeit eines Datenbanksystems Da ein relationales Modell genutzt werden soll, ist es wünschenswert, dass die Plattform ein für mobile Geräte optimiertes Datenbanksystem beinhaltet oder ein solches System als Erweiterung verfügbar ist.

(P5) Zugriff auf Positionsdaten Die Positionsdaten des mobilen Gerätes müssen abrufbar sein. Auf diese Daten soll auf einfache Art und Weise zugegriffen werden können.

(P6) Unterstützung für symbolische Positionsdaten Es ist wünschenswert, dass nicht nur mit geographischen (bzw. kartesischen) Positionsdaten umgegangen werden kann, sondern auch mit symbolischen (bzw. topologischen) Positionsdaten, wie Adressen, Räumen oder anderen bestimmten Orten [DRD+00] und diese ineinander überführt werden können.

(P7) Zugriff auf Kalenderdaten Um das Beispielszenario möglichst einfach zu realisieren, sollte nach Möglichkeit ein globaler und einheitlicher Zugriff auf Kalenderdaten möglich sein.

(P8) Gemeinsame globale Nutzung von Entitätsdaten Statt dass Anwendungen die Daten zu den Entitäten, mit denen sie operieren lokal verwalten, sollte auf Entitätsdaten, wie z.B. zu Personen oder Orten, global und einheitlich zugegriffen werden können.

(P9) Verfügbarkeit von Programmiersprachen Zur Vervollständigung des Überblicks werden die Programmiersprachen, mit denen für die jeweilige Plattform entwickelt werden kann, aufgelistet. Für die prototypische Implementation und für die spätere Anwendungsentwicklung sind moderne, objekt-orientierte high-level Programmiersprachen zu bevorzugen. Für den späteren leistungsoptimierten Produktionscode ist evtl. eine systemnähere Sprache von Vorteil.

Die existierenden gängigen Plattformen, die auf heutigen Mobiltelefonen im Einsatz sind, werden in den folgenden Abschnitten stichpunktartig erläutert und bewertet.

9.1.2 Symbian OS

Symbian OS ist ein propäritäres Betriebssystem für mobile Geräte, das sowohl den Anforderungen von Mobiltelefonen, als auch von PDAs entsprechen soll [GJNS08]. Es wurde ursprünglich von einem Konsortium der Firmen Ericsson, Nokia, Motorola und Psion entwickelt. Die Entwicklung von Symbian wurde unter Federführung des Nokia-Konzerns 2008 in eine Stiftung überführt, die einen Großteil des Systems unter Einbeziehung weiterer Firmen, als Open Source Software anbieten will [Sym09]. Symbian OS wird auf einem hohen Anteil der von Nokia produzierten Mobiltelefone eingesetzt, sowie auf einigen Geräten anderer Hersteller [Bla09].

Als Betriebssystem für mobile Geräte ist die Entwicklung auf dieser Plattform von einem relativ systemnahen Ansatz geprägt. Es existieren eine Reihe von Programmiersprachen, mit denen auf Symbian entwickelt werden kann [GJNS08]. Die native Sprache ist C++, zusätzlich kann Java ME, Python, Ruby und Flash Lite genutzt werden [Nok09]. Es existieren spezialisierte APIs, mit denen auf Positionsdaten [Sym08], Kalendereinträge [Sha06] und Kontakte (d.h. Personen) [EMC08] zugegriffen werden kann. Darüberhinaus gibt es verschiedene Methoden zur persistenten und global zugreifbaren Speicherung von Daten sowie zur Interprozess-Kommunikation [Bli07]. Es steht eine native Datenbank zur Verfügung sowie externe Implementationen von Datenbanken verschiedener Hersteller [Col06]. Ein einheitliches Konzept auf einer höheren Abstraktionebene, z.B. wie mit Entitätssarten umgegangen werden soll, fehlt jedoch. Auch ein Abfangen verschieder Aktivitäten ist nicht direkt möglich.

Um die Schwierigkeiten zu überkommen, die sich aus dem systemnahen Ansatz beim Zugriff auf Kontextdaten ergeben, existiert mit *ContextPhone* [ROPT05] eine Middleware für Kontext-Awareness, die auf Symbian aufbaut und den Zugriff auf Kontextinformationen vereinfacht (vgl. Abschnit 6.2.2). Auch wenn diese Middleware als nicht hinreichend für die Nutzung von Aktivitätskontext bewertet wurde (vgl. Abschnitt 6.3.2), könnte diese Middleware dennoch benutzt werden, um den Zugriff auf einige Kontextinformationen zu vereinfachen. So bietet *ContextPhone* beispielsweise direkte Schnittstellen, um die wichtigsten Kommunikationskanäle des Telefons, wie die Telefonie, den Versand von Kurznachrichten oder Kommunikation per Bluetooth zu überwachen und um zu prüfen welche Anwendungen aktiv sind.

9.1.3 Windows Mobile

Mit Windows Mobile versucht Microsoft ein Betriebssystem für mobile Geräte bereit zu stellen, das äußerlich an Microsoft Windows erinnert und den selben Programmierkonzepten folgt [Mic08b]. Mit dem *.NET Compact Framework* [Mic09b] steht in neueren Win-

dows Mobile Versionen eine für mobile Geräte optimierte Version des Softwareframeworks *.NET* zur Verfügung. Ähnlich wie Symbian OS bietet Windows Mobile Zugriff auf Kalendereinträge [Mic08a] und Kontakte [Mic08b] über spezialiserte APIs. Interprozesskommunikation lässt sich über das .Net Compact Framework realsieren [Mic09b] und es ist ein SQL-Datenbankserver für Windows Mobile verfügbar [Mic09a]. Ein einheitliches Konzept zum Umgang mit Entitäten ist auch in diese Plattform nicht integriert. Änderungen im Gerätezustand können zwar über eine API beobachtet werden [Sor], aber ein Abfangen von High-Level Aktivitäten des Nutzers ist nicht unmittelbar möglich.

9.1.4 iPhone OS

Anfang 2007 stellte Apple das *iPhone* der Öffentlichkeit vor [Job07] und trat somit in den Mobiltelefonmarkt ein. Das Gerät wurde vor allem für sein auf Multitouch-Gesten beruhendes Bedienkonzept in der Öffentlichkeit viel beachtet [Gro07]. Technische Grundlage für das iPhone war eine, an das Apple Betriebssystem *MAC OS X* [App09c] angelehnte, Entwicklungsplattform. Betriebssystem dieser Plattform ist mit Darwin [App09a] ein unix/posix-artiges Betriebssystem. Die oberste Abstraktionsebene bildet mit Cocoa touch [App09b] eine für Multitouch-Bedienung und mobile Geräte optimierte Version des Cocoa AppKits zur Entwicklung von grafischen Benutzungsschnittstellen. Entwicklungssprache ist das aus der Apple-Welt bekannte Objective C.

Zwar verfügt das IPhone OS auch über eine integrierte Datenbank [App09b] und eine API für den Zugriff auf Kontaktdaten [App09b]. Aber schon für die Kalenderdaten gibt es keine Zugriffsmöglichkeit. Auch in dieser Plattform gibt es keinen einheitlichen Ansatz zur Nutzung von Entitätsdaten.

Da Apple eine Politik verfolgt, die darauf abziehlt durch hohe Restrikionen in den Möglichkeiten einzelner Anwendung, diese möglichst weitgehend gegeneinander abzuschirmen, um eine hohe Stabilität und Zuverlässigkeit des Gesamtsystems zu gewährleisten, sind die Zugriffsmöglichkeiten für Anwendungen auf Daten, die außerhalb des lokalen Bereichs liegen, sehr eingeschränkt. Insbesondere werden keine Mechanismen für Interprozess-Kommunikation bereitgesetellt.

9.1.5 Freesmartphone.org Architecture

Die Freesmartphone.org Architecture ist ein Projekt, das Middleware zur Anwendungsentwicklung auf Mobiltelefonen ausschließlich auf Basis von Open Source Software bereitstellen will [fre09d]. Die Architektur setzt auf einen Linux-Kernel auf, wobei D-Bus [fre09b], ein leichtgewichtiges Framework für Interprozess-Kommunikation mit einem Object Request Broker im Kern zur Kommunikation zwischen den Schichten genutzt wird, wodurch die Anpassung des Systems auf neue Hardware oder Linux-Varianten vereinfacht wird. Das Projekt wird von Openmoko Inc finanziert. Die Firma hat mit dem gleichnamigen Openmoko [Ope09a] eine Linuxdistribution veröffentlicht,

die den Freesmartphone.org-Software-Stack implementiert. Mit dem OpenMoko Free-Runner [Ope09b] wurde bereits ein Referenzgerät auf den Markt gebracht.

Teil des Softwarestacks ist ein Eventsystem, das es ermöglicht, Nachrichten zwischen Anwendungen auszutauschen, um so beispielsweise eingehende oder ausgehende Anrufe zu registrieren, ein bisher undokumentiertes Subsystem für die gemeinsame Nutzung von PIM-Daten, wie Kalendereinträgen oder Kontakten und eine API für die Nutzung von Positionsdaten. Anwendungsentwicklung kann prinzipiell in allen Sprachen erfolgen, die über eine D-Bus Schnittstelle[1] verfügen [fre09c].

9.1.6 Java ME

Java ME ist eine Plattform für mobile Geräte zur Ausführung von Java-Programmen [Sun09b]. Java ME Anwendungen werden auf einer Java Virtual Machine in einer Sandbox ausgeführt, wobei unterschiedliche Konfigurationen für verschiedene Geräteklassen bereitstehen. Für optionale Fähigkeiten stehen zusätzlich eine Reihe standardisierter Bibliotheken zu Verfügung [Sun09c]. Dies macht eine einheitliche Anwendungsentwicklung auf einer hohen Sprachebene für eine sehr große Breite an unterschiedlichen Geräten möglich. Solche Erweiterungen existieren beispielsweise für die Nutzung von PIM-Daten [Sun09d] und Positionsdaten [Sun09e]. Java ME Anwendungen laufen mit sehr restriktiven Rechten in ihrer Sandbox, eine Kommunikation mit oder die Beobachtung von anderen Komponenten ist nur bedingt möglich. Für die Beobachtung von Anrufen existiert zwar eine Spezifikation [Sun09a], diese wurde bisher jedoch noch auf keinem Gerät implementiert. Eine Datenbank ist nicht Bestandteil der Java ME-Plattform, auch wenn diverse externe Implementationen verschiedener Hersteller existieren [Tri08].

Da die mobilen Geräte immer leistungsfähiger werden, wird die Entwicklung der Plattform mittelfristig von Sun zugunsten der Java Standard Edition aufgegeben [Sha07].

9.1.7 Android Application Framework

Das *Android Application Framework*, oft fälschlicherweise als Googles Betriebssystem für Handys bezeichnet, ist ein Software Stack, der auf einer speziellen Linuxdistribution für mobile Geräte läuft und auf oberster Ebene ein Framework zur Entwicklung von mobilen Anwendungen enthält [Goo09c]. Der Android Kern wurde als Open Source Software durch die Open Handset Alliance entwickelt. Diesem Konsortium gehören diverse Firmen, die die Bereiche Mobilfunknetzbetrieb, Mobiltelefonherstellung, Halbleiterherstellung und Softwaredienste umfassen, an [ope07]. Zugehörige Entwicklungswerkzeuge werden durch Google unter der Open Source Lizenz *Apache License v2* veröffentlicht.

[1] Das ist für viele gänge Sprachen wie z.B. Java, C++, Python, Perl und Ruby der Fall [fre09a].

Die Entwicklung von Android Applikationen erfolgt in einer java-gleichen Syntax[2]. Es stehen Bibliotheken zur Verfügung, die den Java Standard Edition Basisbibliotheken entsprechen und Erweiterungen, die die Fähigkeiten der Mobiltelefone nutzbar machen und zusammen mit Komponenten für GUI- und Anwendungsentwicklung ein eigenes Applikation-Framework bilden. Zusätzlich zur javabasierten Entwicklung wurden durch Google mitlerweile Entwicklungsumgebungen für C/C++ [Tur09] und für verschiedene Skriptsprachen wie Lua, Python, BeanShell und zukünftig auch Ruby und JavaScript [Goo09a] veröffentlicht. In diesen Umgebungen ist der Zugriff auf den Funktionsumfang des Android Applikation Frameworks und der Android Funktionen jedoch teilweise eingeschränkt.

Ein Teil des Frameworks sind `ContentProvider`, die ein einheitliches Konzept realiseren, wie alle möglichen Daten von Anwendungen global verfügbar gemacht werden können [Goo09b]. Darüberhinaus propagiert das Framework ein Konzept der Anwendungsentwicklung, das es erlaubt, unterschiedliche Anwendungen miteinander zu verzahnen und einzelne Komponenten der Anwendungen durch andere Anwendungen zu nutzen oder zu ersetzen.

9.1.7.1 Möglichkeiten zur Beobachtung von Aktivitäten in Android

Zentraler Teil des in der Android-Plattform propagierten Konzepts zur Anwendungsentwicklung sind `Intents` und `Activities` [Goo09b]. Eine Activity repräsentiert eine Tätigkeit, die auf einer Bildschirmseite ausgeführt wird. Eine Activity könnte zum Beispiel eine Seite sein, die das Telefonbuch des Nutzers anzeigt und es erlaubt einen Eintrag zu wählen. Die verschiedenen Activities, die zusammen eine Anwendung bilden, sind nur lose miteinander gekoppelt. Der Übergang von einer Activity zu einer anderen erfolgt, indem die erste Activity einen so genannten Intent an das Framework sendet, der die gewünschte nächste Activity beschreibt. Die Telefonbuch-Activity könnte beispielsweise einen Intent versenden, der anzeigt dass nun ein ausgewählter Telefonbucheintrag bearbeitet werden soll. Das Framework kann dann, unter Berücksichtigung der Präferenzen des Nutzers, eine Activity aus einer beliebigen Anwendung aufrufen, die deklariert hat in der Lage zu sein, Telefonbucheinträge zu bearbeiten (vgl. Abbildung 9.1.7.1). Nach Beendigung dieser Activity würde diese dann wieder zur ursprünglichen Activity zurückkehren.

Dieses Konzept soll die Verzahnung und Austauschbarkeit der einzelnen Komponenten gewährleisten. Es lässt sich sehr gut für die Beobachtung von Aktivitätskontext[3] nut-

[2] Die Android Plattform wurde nicht durch Sun lizensiert. Stattdessen wird mit Apache-Harmony eine alternative Implementation der Java Standard Librarys genutzt und mit Dalvik eine durch Google entwickelte eigene virtuelle Maschine eingesetzt, die auf einem eigenen Bytecodeformat (dex-files) aufbaut und für die Ausführung von mehreren gegeneinader abgeschirmeten Anwendungen auf mobilen Geräten optimiert ist [Bor08]. Durch den Einsatz dieser Technologien wird versucht, die Java-Lizensierung zu umgehen, weswegen Google nicht offiziel den Anspruch erhebt, dass Android eine Java-Plattform sei [Got07].

[3] Eine `Activity` ist nicht mit dem in dieser Arbeit entwickelten Aktivitätsbegriff zu verwechseln: Ein Activity ist ein eher technisches Konzept und liegt, wird sie mit dem hier verwedeten Begriff verglichen,

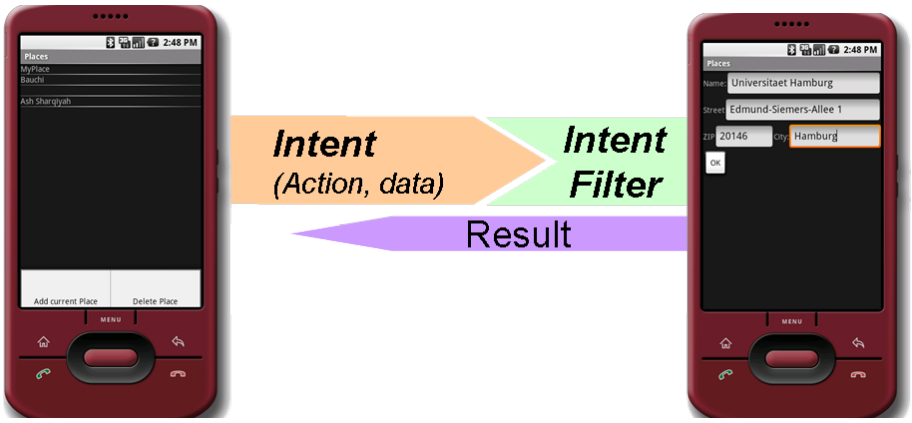

Abbildung 9.1: Lose Kopplung von `Activities` in Android: Eine Activity realisiert einen Screen. Eine andere Activity kann über einen *Intent* aufgerufen werden. Das Framework wählt eine Activity mit einem passenden `IntentFilter` zur Bearbeitung des Intents aus.

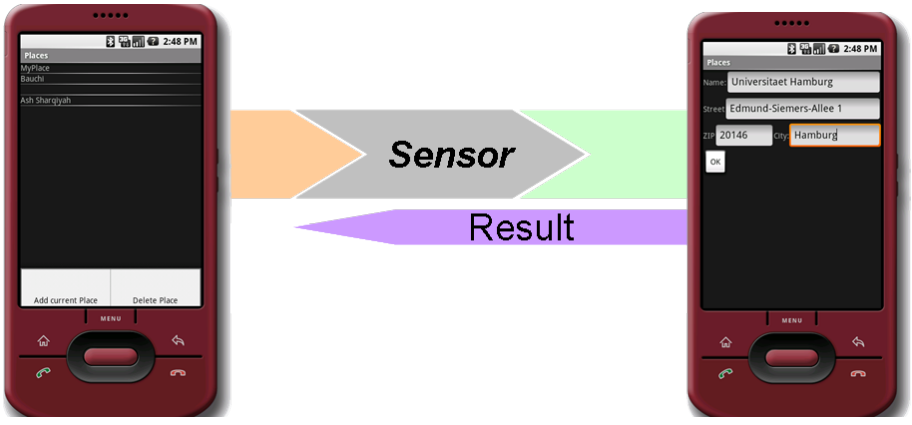

Abbildung 9.2: Das Intent-Konzept ermöglicht die Zwischenschaltung von Sensoren um Benutzeraktivitäten festzustellen.

		Symbian	Symbian +ContextPhone	Windows Mobile	iPhone OS	Freesmartphone.org	Java ME	Android
(P1)	Beobachtbarkeit von Telefonaten	+	++	+	+	++	−	++
(P2)	Beobachtbarkeit weiterer Aktivitäten	−	−	−	−−	−	−−	+
(P3)	Interprozess-kommunikation	++	++	++	−	++	−	++
(P4)	Datenbank	++	++	++	++	−	+	++
(P5)	geografische Positionsdaten	++	++	++	++	++	++	++
(P6)	topologische Positionsdaten	Alle: Lediglich Umgang mit Adressen						
(P7)	Kalenderdaten	++	++	++	−−	+	++	++
(P8)	weitere Entitäten	−	−	−	−−	−	−−	+
(P9)	Programmiersprache	C++[a]	C++[b]	Visual C++[a]	Object-ive C	C++[a]	Java	Java[c]

[a] div. weitere möglich
[b] Benutzung von ContextPhone schränkt auf C++ und Java ein
[c] Weitere mit eingeschränktem Funktionsumfang möglich.

++	Gute Unterstützung
+	Möglichkeit vorhanden
−	Möglichkeit bedingt vorhanden
−−	Keine Möglichkeit vorgesehen

Tabelle 9.1: Bewertung der Plattformen für mobile Anwendungen

zen. Da die Activities nur lose gekoppelt sind und über eine einheitliche Schnittstelle verbunden werden, ist es möglich Software-Sensoren an geeigneten Übergangsstellen zwischen Activities zu realisieren. Dies könnte entweder auf einer systemnahen Ebene als Teil des Frameworks geschehen oder indem ein Programm die gleichen Dienste einer Activity, die es eigentlich beobachten will, anbietet, dies jedoch mit einer höheren Priorität. Wird das Programm so aufgerufen, kann es die Anfrageparameter aufzeichnen und anschließend die eigentlich gewünschte Activity aufrufen. Abbildung 9.1.7.1 skizziert zwei Activities mit zwischengeschaltetem Sensor.

eher auf der Ebene von Aktionen und Operationen im Sinne der Aktivitätstheorie (vgl. 3.2). So verstanden kann eine Aktivität jedoch durchaus durch eine oder mehrere Activites realisert werden.

9.1.8 Fazit

Ein vergleichender Überblick über die verschiedenen Plattformen ist in Tabelle 5.2 dargestellt. Vor allem wegen der guten Möglichkeit dort Aktivitäten zu beobachten, wurde das Android Application Framework als Plattform für die Implementation von Middleware und Demoanwendung gewählt. Zusätzliche Pluspunkte bei der Wahl waren die Verfügbarkeit einer High-Level-API zum Zugriff auf Funktionen des Mobiltelefons und das von allen Ansätzen ausgereifteste Konzept für den Datenaustausch zwischen verschiedenen Anwendungen.

9.2 Architektur der Middleware

Die Architektur der Middleware muss es möglich machen, das in Abschnitt 7.3 erarbeitete Kontextdatenschema zu implementieren und soll darüberhinaus die in Kaptiel 8 entwickelten Kernkonzepte realisieren. Dabei wurde die Implementation so realisiert, dass die Middleware möglichst unabhängig von der Plattform auf der sie realisert wurde, ist (Abschnitt 9.2.1).

Als Grundlegende Funktionalität ist zunächst eine Kommunikationsinfrastruktur notwendig, welche die Modulariserung der Komponenten (vgl. Abschnitt 8.3) ermöglicht, dabei deren lose Kopplung und eine n:m Kommunikation realisert (vgl. Abschnitt 8.4) und die für spätere Erweiterungen offen ist (vgl. Abschnitt 8.7). Wie diese Kommunikationsinfrastruktur realisiert wurde, ist in Abschnitt 9.2.2 beschrieben.

Im darauf folgenden Abschnitt 9.2.3 werden Aufbau und Möglichkeiten der `EventStreams` (Ereigniserzeuger) und `Reposoitories` (abfragbare Komponenten) geschildert, welche die Bausteine für die in der Middleware realiserten Kontextspeicherungs- und Verarbeitungskomponenten bilden.

Der dritte grundlegende Teil der Middleware bildet die in Abschnitt 9.2.4 geschilderte Aktivitäts-API, die es Anwendungen ermöglicht, auf einfache Art den Verlauf von Aktivitäten an die Middleware zu kommunizieren.

9.2.1 Plattformunabhängigkeit

Die Middleware wurde in Hinblick auf eine möglichst große Plattformunabhängigkeit entwickelt, d.h. die Middleware setzt lediglich eine Java Standard Edition kompatible Laufzeitumgebung vorraus und ist unabhängig von Android. Das Trennungskonzept sieht vor, dass lediglich die Kommunikationsadapter (vgl. 9.2.2 u. 9.3.3.1) und die plattformspezifischen Sensorkomponenten auf das jeweilige System angepasst werden müssen. In der Praxis musste dieses Trennungskonzept jedoch bei der Benutzung der zu Android gehörigen SQLite Datenbank verletzt werden, da bisher nicht dokumentiert ist, wie diese Datenbank über die abstrakten, generischen Java-Datenbanktreiber [Sun] angesprochen werden kann und daher die Android-API zur Kommunikation mit der

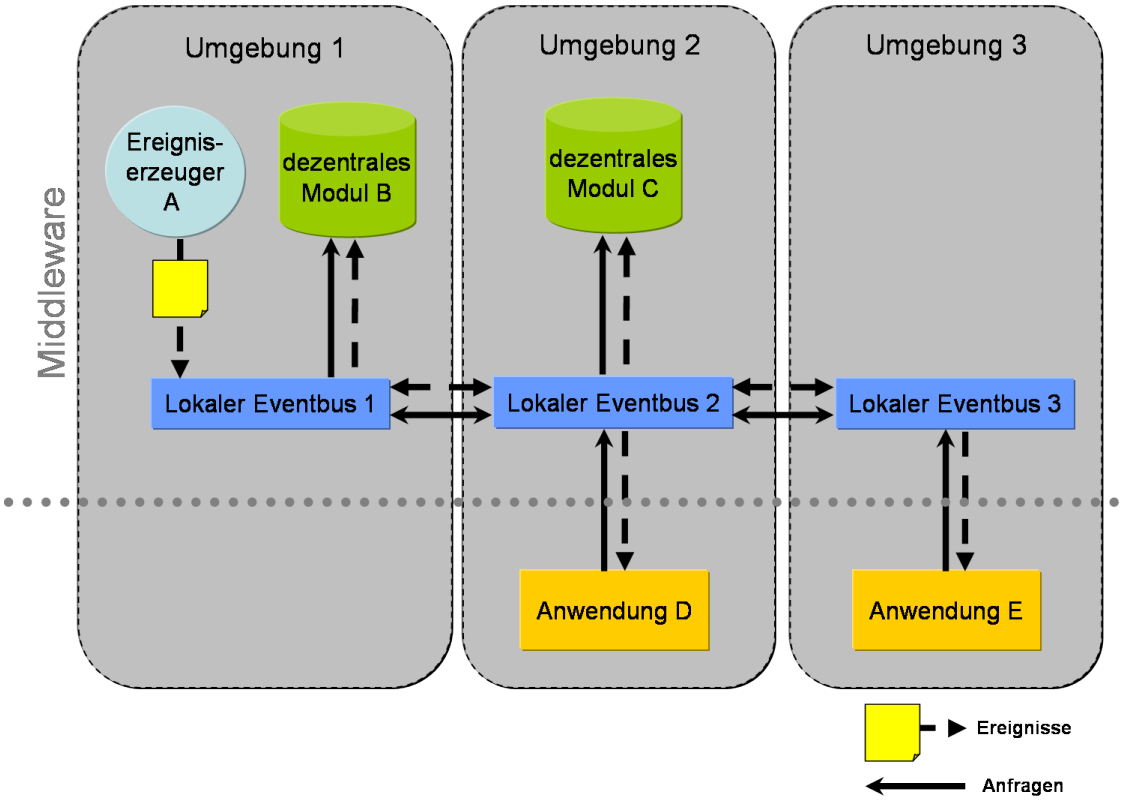

Abbildung 9.3: Komponenten und Anwendungen sind über einen Eventbus miteinander verbunden.

Datenbank genutzt wurde. Denoch sollte die weitgehend realiserte Trennung von plattformspezifischen Teilen eine Portierung auf andere Systeme deutlich erleichtern.

9.2.2 Kommunikation

Die Kommunikationsinfrastruktur besteht aus drei Schichten (vgl. Abbildung 9.4): Adapter für spezifische Kommunikationskanäle, einem Eventsystem und darauf aufbauend Mechanismen für Anfragen.

9.2.2.1 Kommunikationsschichten

Die zentrale Schicht ist das Eventsystem, das aus Sicht der Komponenten einen Eventbus bereitstellt (siehe Abbildung 9.3). Dieser Eventbus besteht aus mehreren verteilt vorliegenden Teilstücken, je ein Teilstück für jede Umgebung, welche die Middleware nutzt. Umgebungen können lokal vorliegende, gegeneinader abgegeschirmte Laufzeitumgebungen oder echte verteilte Systeme sein. Die Verteilung der einzelnen Bus-Teilstücke auf verschiedene Umgebungen wird dabei durch das Eventsystem verborgen. Die Kommunikation zwischen den einzelnen, verteilten Teilstücken des Eventbus erfolgt vermittelt durch Adapter für den jeweils genutzen Kommunikationskanal zur Herstellung der Verbindung zwischen den Stücken. Kommunikationskanäle können Mechanismen zur

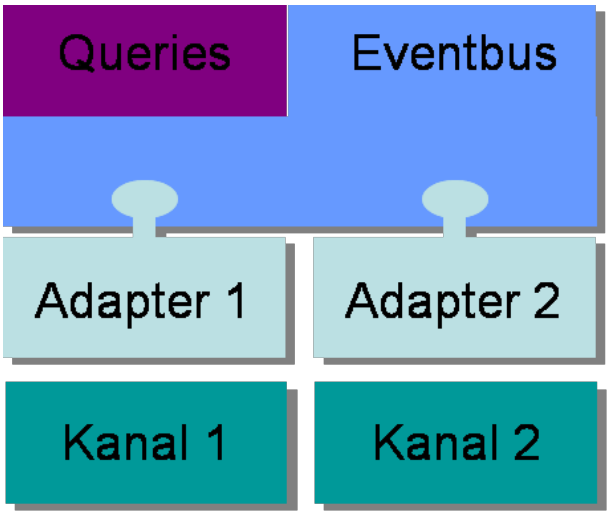

Abbildung 9.4: Ein verteilter Eventbus bildet die zentrale Kommunikationsschicht. Zur Abstraktion der verwendeten Kanäle werden Adapter genutzt. Die Behandlung von Anfragen und Antworten wird über das Eventsystem realisert.

Interprozess-Kommunikation sein, wenn die Teilstücke auf einem lokalen System in unterschiedlichen Prozessen vorliegen. Genau dies ist in der vorliegenden Implementation der Fall. Es ist allerdings auch möglich Netwerkkanäle wie eine TCP/IP- oder Bluetooth-Verbindung zu nutzen, wenn die Teilstücke echt verteilt sind.

Durch die Entkopplung von Kommunikationskanal und Eventsystem durch die Adapter, lässt sich leicht der genutzte Kommunikationskanal austauschen oder auch eine hybride Struktur mit unterschiedlichen Kommunikationskanälen aufbauen.

9.2.2.2 Ereignisfilter

Komponenten der Middleware oder Anwendungen, die über bestimmte Ereignisse informiert werden wollen, registrieren sich bei ihrer jeweiligen lokalen Instanz des Eventbus'. Dabei melden sie Filter an, welche die Ereignisse, an denen sie interessiert sind, spezifizieren. Diese Filter können dazu eingesetzt werden, die Kommunikationseffizienz zwischen den lokalen Instanzen des Eventbus' zu optimieren, in dem die Filter an die anderen Instanzen weitergereicht werden und nur solche Ereignisse über die Kommunikationskanäle übertragen werden, die am anderen Ende auch verlangt werden.

9.2.2.3 Topologie des Eventbus

Im Rahmen dieser Arbeit wurde eine hierarchische Verwaltung des Eventsystems realisiert. Die lokalen Eventbusinstanzen kommunizieren mit einem zentralen `BusManager`-Service. Dieser sammelt alle angemeldeten Filter und erhält alle auftretenden Ereignisse. Die Ereignisse werden von dieser Stelle jedoch nur selektiv auf Basis der Filter an die

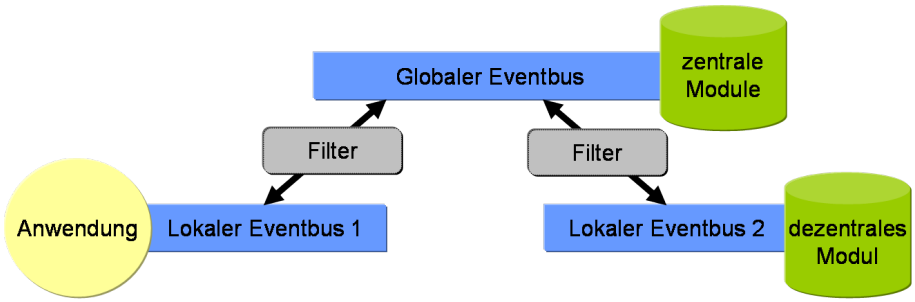

Abbildung 9.5: Der verteilte Eventbus wurde in dieser Implementation über eine hierarchische Topologie realisiert.

lokalen Eventbus-Instanzen weitergeleitet (vgl. Abbildung 9.5). Als Kommunikationskanal wurde der Android-Broadcastmechanismus [Goo09b] gewählt und entsprechende Adapter implementiert.

Für Ubiqitous Computing -Szenarien ließe sich statt der hierarchischen Topologie der Ereignisverteilung auch eine peer-to-peer basierte Struktur realisieren. Dies würde jedoch die Verwaltung der Filter und Routen erheblich komplexer werden lassen, da diese die dynamische Natur des Netzwerkes berücksihtigen müssten [Ait07]. Da in dieser Arbeit alle Komponenten auf der gleichen Hardware laufen und so die Erreichbarkeit eines zentralen Verwaltungsdienstes gewährleistet werden kann, ist die vorliegende Implementation für die Zwecke dieser Arbeit hinreichend.

9.2.2.4 Anfragemechanismen

Aufbauend auf dem Eventsystem wurden Mechanismen für pfad- und sql-basierte Anfragen prototypisch implementiert. diese bauen auf dem Eventsystem auf. Daher brauchen die Kommunikationsadapter lediglich einmal für das Eventsystem implementiert werden und nicht extra für die Anfragemechanismen angepasst werden. Da für die Weiterleitung der Anfragen das Eventsystem genutzt wird, lassen sich auch indirekte Adressierung und somit ein anonymer Request/Reply Mechanismus realiseren [Ait07].

9.2.2.5 Dynamische kontextabhängige Namensschemata

Die Mächtigkeit der indirekten Adressierung erlaubt es auch, dynamische kontextabhängige Namensschemata (vgl. Abschnitt 8.5) zu implementieren. Ein Repository kann durch reguläre Ausdrücke deklarieren, dass es bestimmte Fragmente eines pfadbasierten Namensschemas auflösen kann. Bleibt dabei ein Teil des Pfades unaufgelöst, so wird die Anfrage an Repositories weitergeleitet, die diese Segmente auflösen können, nachdem die auflösbaren Teile durch entsprechende Einträge substituiert wurden. Der Auflösemechanismus geht dabei im durch einen Pfad repräsentierten Namen von links nach rechts vor. Die Aggregation und Weiterleitung der Teilergebnisse muss dabei nicht durch das

Repository selbts implementiert werden, sondern erfolgt durch einen durch die Middleware bereitgestellten `RepositoryManager` (vgl. 9.2.3.1).

9.2.3 Eventstreams und Repositories

`EventStreams` und `Repositories` bilden die zentralen Bausteine für die Komponenten zur Kontextdatenverarbeitung und -speicherung (vgl. Abschnitt 8.3). In ihrer internen Realisierung sind diese Komponenten grundsätzlich frei. Eventstreams sind lediglich dadurch gekennzeichnet, dass sie Events an den lokalen Eventbus weitergeben, von wo aus diese ggf. weiter verteilt werden.

Repositories müssen mindestens eine Unterklasse des `IQueryable`-Interfaces implementieren. Derzeit existieren die Unterklassen `IQueryableByID` für id-basierte Abfragen, `IQueryableBySQL` für sql-basierte Abfragen und `IQueryableByPath` für pfadbasierte Abfragen.

Repositories, wie Eventstreams können selbst auch Empfänger von Ereignissen sein oder Anfragen an andere Repositories senden, um die Daten zu erhalten, die sie für ihre Funktionalität benötigen.

9.2.3.1 RepositoryManager

Die Middleware stellt pro Repository einen `RepositoryManager` bereit, der dieses Repository verwaltet. Dieser Manager kennt die Fähigkeiten des Repositories und verbindet das Repository mit dem Eventbus. Er leitet passende Anfragen, die auf dem Eventbus eingehen, an das Repository weiter, nimmt das Resultat der Abfrage vom Repository entgegen und kommuniziert dieses Resultat über den Eventbus an den entsprechenden Empfänger (vgl. Abbildung 9.6). Falls das Repository Segmente eines dynamischen Namenschemas auflösen kann (vgl. 9.2.2.5), erzeugt der Manager entsprechende Eventfilter, leitet ggf. die aufzulösenden Segmente als Anfrage an das Repository weiter und substituiert die Pfadsegmente durch das Auflösungsergebnis. Anschließend wird das Ergbenis zur weiteren Verarbeitung wieder auf den Eventbus gelegt.

Der RepositoryManager nimmt dabei nur Ereignisse entgegen, die in irgendeiner Weise Anfragen an das Repository darstellen oder verwaltungstechnische Bedeutung haben. Informationen, die von einem Repository zur Pflege und Aktualiserung seines Datenbestandes benötigt werden, werden direkt von dem Repository entgegengenommen und verarbeitet. Somit braucht das Repository nur den fachlichen Umgang mit seinen Daten zu implementieren. Die technischen Aspekte beim Umgang mit den Daten und der Middleware werden weitgehend durch den Manager übernommen.

9.2.3.2 Standardrepositories

Als Teil der Middleware wird ein `ActivityRepository` bereitgestellt, welches das in Abschnitt 7.3 entwickelte Kontextdatenschema für Aktivitäten implementiert. Zur Ver-

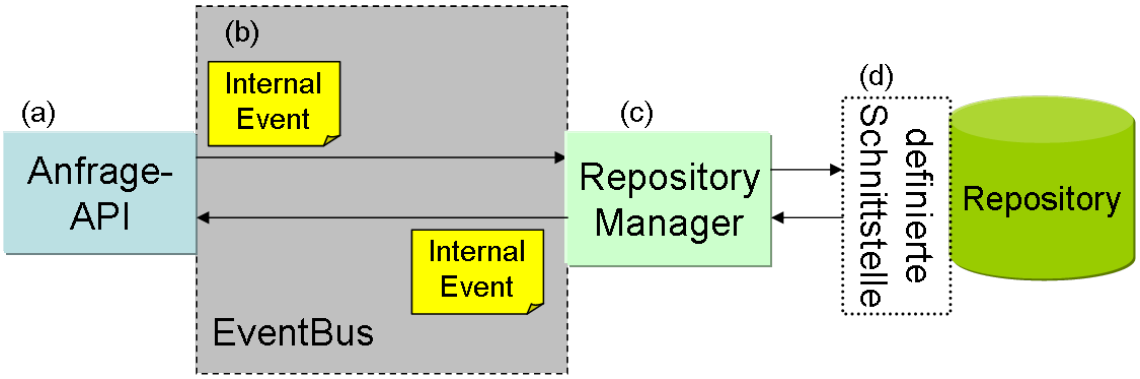

Abbildung 9.6: Anfragen werden intern über Ereignisse verschickt. Der RepositoryManager wertet diese aus und verwendet die Anfrageschnittstellen des Repositories.

waltung und persistenten Speicherung der Daten wird intern die von Android bereitgestellte SQLite-Datenbank genutzt. Dadurch können eingehende sql-basierte Anfragen direkt an die Datenbank weitergegeben werden. Das Repository wertet eigenständig alle Ereignisse, die durch die Middleware zur Information über laufende Aktivitäten erzeugt werden, die `ActivityStateChangedEvents` und `ActivityAddedEntity` sowie die mit diesen verbundenen Planungsevents (vgl. Abschnitt 8.6) aus und pflegt auf dieser Basis die Instanz des Kontextdatenschemas. Solche Ereignisse werden durch die Middleware automatisch erzeugt, sobald von einer Anwendung die API zur Bekanntmachung von laufenden Aktivitäten (vgl. Abschnitt 9.2.4) genutzt wird.

Als Teil der Demoanwendung ist zudem ein `RelevanceRepository` implementiert, das Auskunft über die zum aktuellen Zeitpunkt relevanten Entitäten gibt. Die Details hierzu sind in Abschnitt 9.4.2 geschildert.

9.2.4 Aktivitäts-API

Damit Anwendungen laufende Aktivitäten an die Middleware auf möglichst einfache Art und Weise übermitteln können, stellt die Middleware eine simple Aktivitäts-API zur Verfügung. Mit einer Factory-Methode kann ein neues Aktivitätsobjekt erzeugt werden. Zustandsübergänge im Sinne des Lebenszyklus einer Aktivität (vgl. Abschnitt 8.2) können durch Aufrufe entsprechender Methoden auf diesem Objekt signalisiert werden. Darüberhinaus können der Aktivität Entitäten zugeordnet werden. Es ist vorgesehen, dass auch Integritätsbedingungen, wie z.B. Timeouts, die angeben ab wann eine Aktivität spätestens als beendet angesehen werden soll, über diese API mitgeteilt werden können[4].

Beispielcode zur Nutzung der API findet sich in Listing 9.2.

[4]Diese Funktionalität ist derzeit noch nicht realisiert

9.3 Anwendungsentwicklung

In diesem Abschnitt wird geschildert, wie die Entwicklung (aktivitäts-)kontextadaptiver Anwendungen mit Hilfe der Middleware praktisch aussieht. Es wird die Kommunikation zwischen Anwendungen und Middleware geschildert (Abschnitt 9.3.1). Die hierfür genutzen Beispiele sind die Verwendung der Aktivitäts-API (Abschnitt 9.3.1.2) und das Stellen einer einfachen Anfrage nach aktuell relevanten Personen (Abschnitt 9.3.1.3). Anschließend wird erläutert, wie die Definition neuer Repositories erfolgt und so das Kontextdatenschema erweitert werden kann (Abschnitt 9.3.2) und schließlich welche Anknüpfungspunkte es zum Ausbau der Middleware selber (Abschnitt 9.3.3) gibt.

9.3.1 Kommunikation zwischen Anwendungen und Middleware

Die Kommunikation von Anwendungen mit der Middleware erfolgt in zwei Richtungen. Der Anwendung obliegt es, die Middleware über den Verlauf ihr bekannter Aktivitäten mittels der Aktivitäts-API zu informieren. Sie kann Informationen nutzen, indem sie Anfragen an die Middleware stellt oder sich für den Erhalt bestimmter Ereignisse registriert.

9.3.1.1 Instanzierung des Eventbus

Zur Kommmunikation mit der Middleware wird grundsätzlich eine lokale Instanz des Eventbus' benötigt. Diese muss bei Bedarf erzeugt werden (siehe Listing 9.1 Zeilen 2-5). Beim Erzeugen des Eventbus' wird diesem mitgeteilt, welchen Adapter er nutzen soll (Zeile 4). Der `AndroidBusAdapter` nutzt die Android-Interprozess-Kommunikationsmechanismen. Hierfür benötigt er eine Referenz auf die Umgebung[5], in der die Android-Applikation läuft.

Listing 9.1: Erzeugung einer lokalen Instanz des EventBus

```
1 //Einmaliges Erzeugen einer lokalen EventBus-Instanz
2 if (msgbus==null) {
3   AndroidBusAdapter adapter=new AndroidBusAdapter(ctx);
4   msgbus=new EventBus(adapter, adapter); // inputAdapter, outputAdapter
5 }
```

9.3.1.2 Nutzung der Aktivitäts-API

Zur Nutzung der Aktivitäts-API muss zunächst eine Instanz einer neuen Aktivität erzeugt werden. Dabei wird eine Bezeichnung für den Aktivitätstyp angegeben. Zustandsübergänge können durch entsprechende Methodenaufrufe erfolgen. Auch geplante Zustandsübergänge können übermittelt werden. Zu jedem Zeitpunkt können der Aktivität

[5] d.h. den Android-`Context`, der die Laufzeitumgebung der Anwendung repräsentiert und nicht mit dem eigenen Kontextbegriff zu verwechseln ist.

beteiligte Entitäten zugefügt werden. Entsprechende Aufrufe sind in Listing 9.2 aufgeführt.

Listing 9.2: Nutzung der Aktivitäts-API

```
1  //Erzeugen einer neuen Aktivität
2  Activity telActivity = Activity.create(msgbus,"telefonieren");
3
4  // Die Aktivität beginnt
5  telActivity.start():
6
7  // In 5 Minuten wird die Aktivität unterbrochen
8  long timeout=System.currentTimeMillis()+(5*60*1000);
9  telActivity.schedule(timeout, Activity.INTERRUPTED);
10
11 // Eine Entität wird zugefügt
12 telActivity.addEntity("people/11");
```

Durch Aufruf dieser Methoden werden von der Middleware entsprechende Events erzeugt, die im ActivityRepository registriert werden.

9.3.1.3 Senden von Anfragen

Anfragen können über Helfermethoden auf einfache Weise an Repositories gestellt werden. Das Beispiel in Listing 9.3 zeigt eine pfadbasierte Anfrage durch den kontextadaptiven Kalender der Demoanwendung nach aktuell relevanten Personen, in der das RelevanceRepository direkt addressiert wird. Sobald eine Antwort auf die Anfrage eintrifft wird die übergebene Callback-Methode aufgerufen und das Ergebniss übergeben.

Listing 9.3: Eine pfadbasierte Anfrage

```
1  RepositoryQuery.path(
2    msgbus,
3    "RelevanceRepository",
4    "relevant/people",
5    new IPathQueryResultListener() {
6      @Override
7      public void onQueryResult(RepositoryQuery q, PathQueryResult r) {
8        setPeopleSuggestions(r.getResult());
9      }
10   });
```

9.3.1.4 Kontextadaptive Android GUI-Widgets

Um die Entwicklung von kontextadaptiven Anwendungen weiter zu vereinfachen, wurde im Rahmen der Entwicklung der Demoanwendung (vgl 9.4) der `EntityPickerView`,

ein GUI-Widget für Android entwickelt, das es erlaubt, auf einfache Weise kontextabhängige Vorschläge für die Eingabe von Entitäten zu machen. Das Widget besteht aus einem freien Eingabefeld und einer Liste von Vorschlägen neben dem Eingabefeld (vgl. Abbildung 9.8). Sobald ein Vorschlag ausgewählt wird, wird dieser zum Eingabefeld hinzugefügt. Dem Widget kann bei Erstellung ein Entitätstyp zugeordnet werden und es ist in der Lage, die Middleware zu nutzen um relevante Vorschläge für den Entitätstyp abzurufen und anzuzeigen. Dieses GUI-Widget ist generisch und kann von allen Android-Anwendungen genutzt werden, die ein Widget zur Auswahl von Entitäten benötigen, in dem automatisch kontextabhängige Vorschläge gemacht werden sollen.

9.3.2 Erweitern des Implementationsschemas

Um neue Arten von Kontextinformationen zu verwalten, können in die Middleware neue Ereignisse und Repositories eingeführt werden. Das Erstellen neuer Ereignisklassen ist in Abschnitt 9.3.2.1 beschrieben. Für neue Repositories stehen zwei unterschiedliche Methoden zur Verfügung. Repositories, die eine Zeitlinie von Ereignissen repräsentieren, können als spezielle Instanzen eines `TimeLineRepositories` erstellt werden (Abschnitt 9.3.2.2). Die allgemeinere und flexiblere Möglichkeit ist das Erstellen einer neuen Repository-Unterklasse (Abschnitt 9.3.2.3).

9.3.2.1 Neue Events und Eventfilter

Um neue Ereignistypen zu kommunizieren, können neue Ereignissklassen definiert werden. Abbildung 9.7 zeigt die zur Verfügung stehenden grundlegenden Oberklassen. In der Regel wird ein neues Ereignis als eine direkte oder indirekte Unterklasse von `InstantEvent` realisiert. Diese Klasse repräsentiert fachliche, kontextbezogene Ereignisse, die zum Zeitpunkt des Versendens stattfanden. Die `InternalEvent`-Klasse repräsentiert interne Ereignisse der Middleware, die nicht direkt an Anwendungen oder Repositories weitergeben werden sollen, sondern durch die Middleware verarbeitet werden. So wird beispielsweise das Versenden von Anfragen und Antworten intern über solche Ereignisse realisiert. Die `ScheduledEvent`- und `CanceledEvent`-Klassen können beliebige InstantEvents kapseln und zeigen an, dass das gekapselte Ereigniss für die Zukunft geplant ist, bzw. abgesagt wurde. Sie brauchen nicht extra für neu entwickelte InstantEvents angepasst werden und finden stets unmodifiziert Verwendung.

Wenn neue Ereignisse eingeführt werden, kann es sinnvoll sein zusätzliche Filter zu definieren, die auf die fachlichen Besonderheiten der neuen Klasse zugeschnitten sind.

9.3.2.2 Neue Repositories für zeitliche Verläufe

Das `ActivityRepository`, welches das Kontextdatenschema für Aktivitäten aufzeichnet (vgl. Abschnitt 7.3), muss im Wesentlichen den zeitlichen Verlauf von Aktivitäten aufzeichnen.

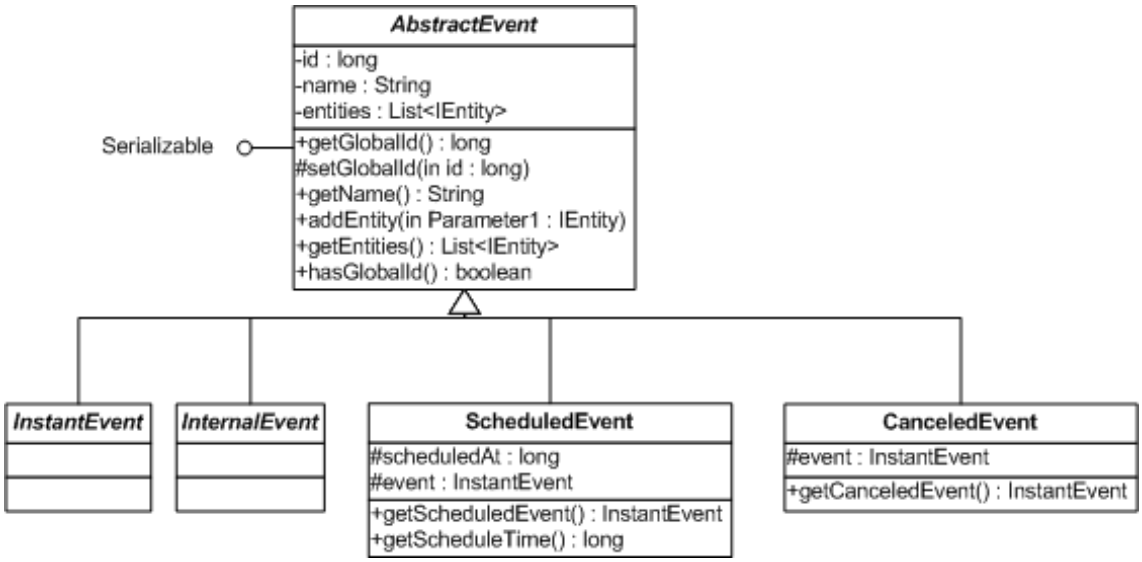

Abbildung 9.7: Hierachie der grundlegenden Ereignissklassen

Für Repositories, die eine Zeitlinie darstellen, steht eine generische `TimeLine-Repository`-Klasse zur Verfügung. Einem TimeLineRepository können Eventklassen übergeben werden, deren Verläufe es als Zeitlinie in einer relationalen Datenbank aufzeichnen soll. Dabei werden zugehörige Planungsereignisse (vgl. Abschnitt 8.6) ebenfalls automatisch berücksichtigt. Hierfür nutzt ein TimeLineRepository Java-Annotationen [Sun04] in den entsprechenden Eventklassen um festzustellen, welche Attribute der Events behandelt werden müssen. Listing 9.4 zeigt Annotationen in der ActivityAddedEntityEvent-Klasse. Diese Ereignisse werden durch die Aktivitäts-API erzeugt, wenn einer Aktivität neue Entitäten zugeordnet wurden. Die ID der Aktivität und die ID der Entität wurden als zu speichernde Informationen gekennzeichnet (siehe Listing 9.4, Zeile 1 u. 6).

Die Events, welche durch die Aktivitäts-API erzeugt werden, wurden so gewählt, dass das ActivityRepository als TimeLineRepository implementiert werden kann. Daher kann am Beispiel des ActivityRepositories gezeigt (Siehe Listing 9.5) werden, wie sich TimeLineRepositories definieren lassen. Zunächst wird ein Mapping erstellt, in dem angegeben wird, welche Ereignisklassen berücksichtigt werden sollen und in welche Tabellen diese abgebildet werden (Listing 9.5, Zeilen 1-4). Dieses Mapping wird bei der Instanzierung eines neuen TimelineRepositories übergeben (Zeile 9). Um nicht nur die Zeiten aufzuzeichenen an denen ein Event stattfand, sondern auch bis wann dieses Event Gültigkeit hat, kann definiert werden bis wann es gültig ist. In Zeile 12 wird eine solche Definition vorgenommen. Ein Aktivitätszustand wird als beendet angenommen, wenn ein neuer Zustand für dieselbe Aktivität (identifiziert über ihre ID) bekannt wird. Der Endzeitpunkt des alten Zustands wird in der Spalte `end_time` aufgezeichnet. Damit das Repository

in die Middleware integriert wird, benötigt es einen RepositoryManger. Dieser wird in Zeile 16 erzeugt.

Listing 9.4: Annotationen in der ActivityAddedEntityEvent-Klasse

```
1  @Persistence("entity␣as␣text")
2  public String getAddedEntity(){
3    return addedEntity;
4  }
5
6  @Persistence("activity_id␣as␣integer")
7  public long getActivityId(){
8    return activityId;
9  }
```

Listing 9.5: Das ActivityRepository ist eine Instanz eines generischen TimeLineRepositories

```
1  // Definition eines Mapping von Event-Klassen auf Tabellen
2  Map<Class,String> activityTables=new HashMap<Class, String>();
3  activityTables.put(ActivityStateChangedEvent.class,"activity_states");
4  activityTables.put(ActivityAddedEntityEvent.class,"activity_entities");
5
6  // Erzeugen des zugehörigen TimeLineRepositories
7  EventTimelineRepository repos=new EventTimelineRepository(
8    Activity.ACTIVITYREPOSITORYNAME,
9    activityTables,
10   ctx);
11 // Es soll aufgezeichnet werden, wann Zustände beendet wurden
12 repos.recordEndTimes("activity_states", "activity_id", "end_time");
13
14 // Ein Eventmanager zur Verwaltung des Repositories wird erzeugt
15 // und mit dem Eventbus verbunden
16 RepositoryManager manager=new RepositoryManager(repos,msgbus);
```

9.3.2.3 Neue Repositories mit eigener Logik

Sollte sich ein Repository nicht als TimeLineRepository realisieren lassen, kann es als Unterklasse der allgemeinen Repository-Klasse definiert werden. Hierzu muss es einige abstrakte Methoden implementieren. Die Methode `onRegisterEventFilters(msgbus)` wird bei Initialisierung des Repositories durch die Middleware aufgerufen und soll dazu genutzt werden, die Events zu deklarieren, über die das Repository informiert werden soll[6]. Darüberhinaus muss mindestens eine Anfrageschnittstelle implementiert

[6]Gemeint sind Ereignisse die das Repository zur Pflege der eigenen Daten benötigt und nicht Ereignisse die durch Anfragen an das Repository entstehen. Letztere werden bereits durch den RepositoryManager verwaltet.

werden. Das für die Demoanwendung entwickelte RelevanceRepository ist ein Beispiel für ein allgemeineres Repository. Es ist in der Lage, pfadbasierte Anfragen zur Relevanz zu beantworten. Listing 9.6 zeigt, welche Methoden hierfür implementiert werden müssen. In der `onRegisterEventFilters`-Methode deklariert das Repository, dass es über den Verlauf von Aktivitäten informiert werden möchte (Listing 9.6 Zeilen 10-18). Die `getByPath`-Methode wird vom RepositoryManager aufgerufen, sobald eine pfadbasierte Anfrage für das Repository eingegangen ist. Zum Parsen der Anfragen stehen Helferklassen zur Verfügung (Zeile 24). Wie das Ergebnis der Anfrage berechnet wird, bleibt dem Repository überlassen. Die `getSupportedSchemes()`-Methode muss einen regulären Ausdruck zurück liefern, der beschreibt, welche Pfadsegmente das Repository auswerten kann (Zeilen 36-39).

Listing 9.6: Implementation der Schnittstellen des RelevanceRepositories

```
1  public class RelevanceRepository
2    extends Repository
3    implements IEventListener,IQueryableByPath {
4
5    // [...]
6
7    @Override
8    public void onRegisterEventFilters(EventBus msgbus) {
9      // [...]
10     msgbus.registerEventListener(this,
11       new EventFilter().addCriterion(new IsInstanceOfFilterCriteria(
12         ActivityAddedEntityEvent.class)));
13     msgbus.registerEventListener(this,
14       new EventFilter().addCriterion(new IsPlaningEventForInstance(
15         ActivityStateChangedEvent.class)));
16     msgbus.registerEventListener(this,
17       new EventFilter().addCriterion(new IsInstanceOfFilterCriteria(
18         ActivityStateChangedEvent.class)));
19   }
20
21   @Override
22   public PathQueryResult getByPath(String path) {
23     MappingResult parsedQuery=queryMapper.map(path);
24     SortedSet<EntityRelevanceResult> sortedRelevances=
25       computeRelevanceTopList(parsedQuery.getSqlWhereClause(),
26         parsedQuery.arguments.toArray(new String[]{}));
27     Entity[] entities=new Entity[sortedRelevances.size()];
28     Iterator<EntityRelevanceResult> iter=sortedRelevances.iterator();
29     int i=0;
30     while (iter.hasNext() &&  i< entities.length) {
31       entities[i++]=iter.next().getEntity();
```

```
32      }
33     return new PathQueryResult(parsedQuery.getRemainingPath(),entities);
34    }
35
36    @Override
37    public String[] getSupportedSchemes() {
38      // Kann Pfadsegemente wie "related-to/argument" oder
39      // "between/argument1/argument2 auflösen
40      return new String[] {"related-to/[a-zA-Z]*",
41        "between/[a-zA-Z0-9]*/[a-zA-Z0-9]*"
42      };
43    }
44  }
```

9.3.3 Erweiterungsmöglichkeiten der Middleware

Die Middleware wurde so entworfen, dass ihre Funktionalität in bestimmten Bereichen leicht erweitert werden kann. Einer dieser Bereiche ist das Hinzufügen neuer Kommunikationskanäle, die durch die Middleware genutzt werden können (Abschnitt 9.3.3.1). Eine andere Möglichkeit ist die Definition neuer Anfrageschnittstellen (Abschnitt 9.3.3.2).

9.3.3.1 Adapter für andere Kommunikationsschichten

Wie bereits in Abschnitt 9.2.2 erläutert, nutzt die Middleware Kommunikationschichten nicht direkt, sondern vermittelt durch Adapter. Soll ein neues Transportmedium genutzt werden, muss ein entsprechender Adapter entwickelt werden. Dieser Adapter muss mit Ereignissen und Ereignisfiltern umgehen können. Ein Adapter erhält durch die Middleware alle lokal definierten Ereignisfilter. Diese können dann durch den Adapter über das Kommunikationsmedium weitergegeben werden. Über den Kanal können die Filter von anderen Stellen entgegengenommen werden und auf der lokalen Instanz des Eventbus' registriert werden. Der oder die entfernten Adapter werden so über alle zu den Filtern passenden Ereignisse informiert und können diese wiederum über den Kommunikationskanal an den interessierten Empfänger leiten.

Aus Sicht der Middleware wird nur erwartet, dass alle zu dem Filter passenden Ereignisse zurückgeliefert werden. Das dem Adapter zugrunde liegende Transportprotokoll oder die Topologie bleibt transparent. Daher könnten Adapter auch dynamisches Multi-Hop-Routing und peer-to-peer basierte Kommunikation [Ait07] intern realsieren und somit die Verteilung der Middleware in einem Ubiqitous-Computing-Umfeld ermöglichen.

9.3.3.2 Neue Anfrageschnittstellen

Die Middleware kann durch neue Anfrageschnittstellen erweitert werden. Sollte z.B. in einem Repository ein ontologiebasiertes Modell verwendet werden, wäre es möglich, dass hierfür eine eigene Anfrageschnittstelle benötig wird.

Anfragen werden intern als Ereignisse verschickt und vom RepositoryManager entgegen genommen und mittels der Schnittstellen des Repositories beantwortet (siehe Abbildung 9.6) Um so eine neue Schnittstelle zu implementieren, muss zunächst die Schnittstelle als Java-Interface definiert werden (Abbildung 9.6 (d)), welches ein entsprechendes Repository später implementieren muss. Dann müssen interne Ereignisse (b) für das Versenden von Anfragen und ggf. von Antworten definiert werden. Die Helferklassen für Anfragen, können dann um entsprechende Methoden erweitert werden, so dass diese Ereignisse über die API (a) erzeugt werden können. Schließlich muss noch die RepositoryManager-Klasse (c) so erweitert werden, dass die entsprechenden Ereignisse erkannt und an passende Repositories, unter Verwendung der definierten Schnittstelle, weitergeleitet werden.

9.4 Implementation der Demoanwendung

Als Demoanwendung wurde, wie in Kapitel 2 beschrieben, ein kontextadaptiver Kalender entwickelt, der in der Lage ist für neue Kalendereinträge zugehörige Personen und Orte vorzuschlagen. Hierfür wurde das bereits in Abschnitt 9.3.1.4 vorgestellte GUI-Widget entwickelt und genutzt. Abbildung 9.8 zeigt ein Screen der Demoanwendung zum Editieren eines Kalendereintrages, in dem Personen für diesen Termin vorgeschlagen wurden. Sobald ein neuer Kalendereintrag erstellt wird, leitet der Kalender mittels der Aktivitäts-API (vgl. 9.3.1.2) Informationen über zukünftig geplante Aktivitäten des Nutzers an die Middleware weiter.

Um Informationen über die Nutzeraktivitäten zu erhalten, die über die Kalendereinträge hinausgehen, wurde zusätzlich ein Sensor entwickelt, der die Telefonate des Nutzers beobachtet und als Aktivitäten an die Middleware weitergibt. Die in der Middleware realisierten Komponenten für die Funktionalität der Demoanwendung und die Datenflüsse zwischen diesen werden im nächsten Abschnitt 9.4.1 geschildert. Die wichtigste Erweiterung der Middleware ist eine Relevanz-Heuristik, die aufgrund bekannter Aktivitäten die aktuelle Relevanz von Entitäten schätzt. Wie diese Heuristik arbeitet, ist in Abschnitt 9.4.2 erläutert.

9.4.1 Datenquellen und Datenfluss

Abbildung 9.9 illustriert den Datenfluss in der Middleware und der Anwendung, welcher das Beispielszenario realisiert. Das ActivityRepository verwaltet einen Zeitstrahl von vergangenen, laufenden und zukünftigen Aktivitäten. Als Quelle für Aktivitätsdaten

Abbildung 9.8: Die kontextadaptive Kalenderanwendung schlägt Personen für einen Termin vor (rot hervorgehoben). Diese Art von kontextadaptiver Auswahl wird über ein bereitgestelltes, generisches Android-GUI Widget realisiert.

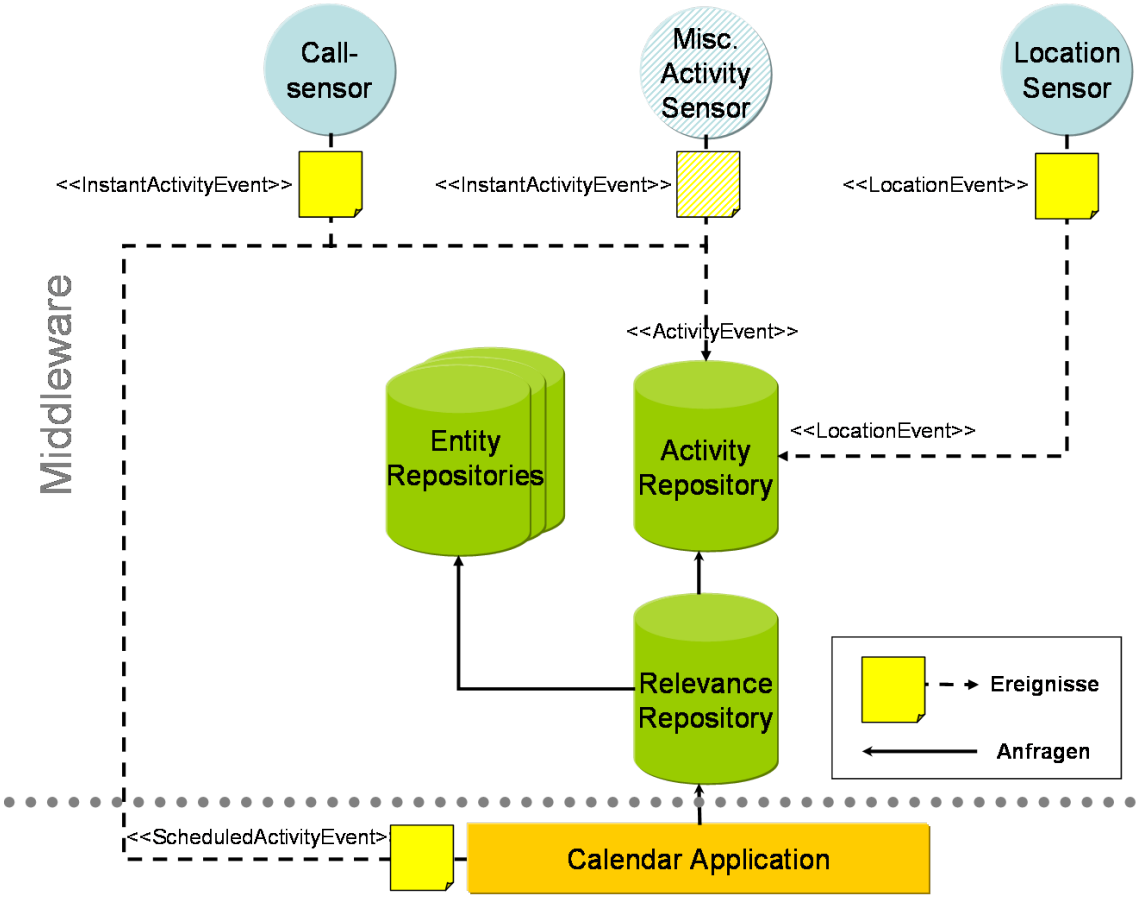

Abbildung 9.9: Datenfluss in der Demoanwendung.

wurde ein Sensor implementiert, welcher die Telefonate des Nutzers beobachtet. Dabei wurde die in Abschnitt 9.1.7.1 erläuterte Vorgensweise des Zwischenschaltens von Sensoren zwischen Android-Activities eingesetzt[7]. Die andere Quelle für Aktivitätsdaten ist die Kalenderanwendung, welche die Aktivitäts-API nutzt, um die Middleware über geplante Nutzeraktivitäten zu informieren. Das ActivityRepository nutzt zusätzlich einen Sensor für die aktuellen Positionsdaten, um laufende Aktivitäten mit Orten zu verknüpfen.

Die Relevanz-Heuristik (siehe 9.4.2) greift zur Schätzung der Relevanz von Entitäten auf die Daten des ActivityRepositories und ggf. einzelner Entitäts-Repositories (z.B. solche für Personen oder Orte) zu und ist in der Lage, auf dieser Grundlage die aktuelle Relevanz von Entitäten zu schätzen. Die Kalenderanwendung versendet Anfragen an das RelevanceRepository, in denen nach den aktuell relevanten Personen und Orten gefragt wird.

9.4.2 Relevanz-Heuristik

Die Entwicklung einer Heuristik, welche die aktuelle Relevanz von Entitäten aufgrund des Wissens über vergangene, laufende oder zukünftige Aktivitäten schätzt, ist nicht trivial. Es muss ein Verfahren gewählt werden, das sich auf einem mobilen Gerät einsetzen lässt, also nur eine geringe Zeit- und Speicherkomplexität besitzt. Nach eigener Erkenntnis liegen (noch) keine empirischen Daten zu Nutzerverhalten von mobilen Anwendungen vor, welche die Entwicklung einer Heuristik fundieren könnten. Allerdings ließe sich die entwickelte Middleware sehr gut dazu einsetzten, solche Daten zu gewinnen. Eine solche empirische Studie würde aber den Rahmen dieser Diplomarbeit sprengen. Zudem ist die Entwicklung einer solchen Heuristik nicht Themenschwerpunkt dieser Arbeit. Es wurde daher eine Heuristik entwickelt, die für mobile Geräte praktikabel ist und deren Angemessenheit plausibel erscheint.

9.4.2.1 Prämissen

Um eine plausible Relevanz-Heuristik zu entwickeln, wurden die im Folgenden geschilderten Annahmen gemacht, die sich in der Heuristik wiederspiegeln sollen.

- Je zeitlich näher eine Aktivität mit einer Entität stattgefunden hat, desto relevanter ist die Entität aktuell.
- Je häufiger eine Aktivität mit einer Entität stattgefunden hat, desto relevanter ist diese Entität.

[7] Da der Umgang mit Anrufen in Android aus sicherheitstechnischen Gründen nicht vollständig über den Intent-Mechanismus erfolgt und eingeschränkt ist, könnte an dieser Stelle das Konzept nicht in Reinform umgesetzt werden. Zur Feststellung von eingehenden Anrufen musste ein anderer, indirekter, Weg gewählt werden. Bei anderen Aktivitäten sollte sich das Vorgehen aber durchaus vollständig wie vorgeschlagen realisieren lassen.

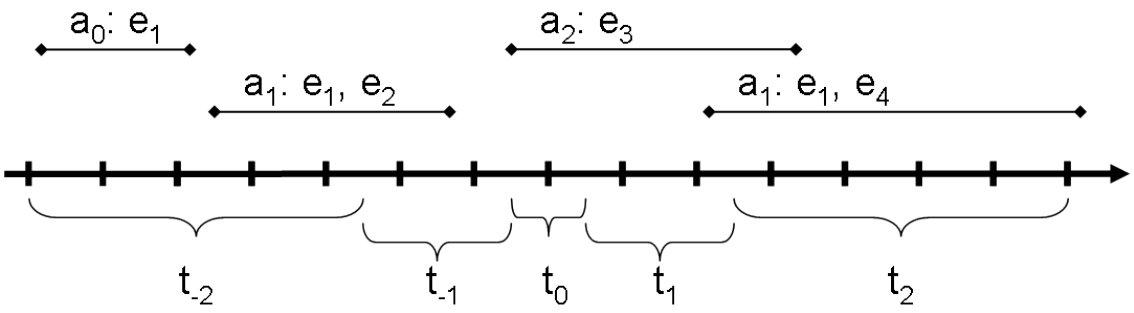

Abbildung 9.10: Für die Relevanzheursitik wird die Zeit in Intervalle aufgeteilt und die Häufigkeit von an Aktivitäten beteiligten Entitäten innerhalb der Intervalle festgestellt.

- Stehen zwei Entitäten e_1 und e_2 über Aktivitäten miteinander in Beziehung gilt:
  - Je relevanter e_1 aktuell ist, desto relevanter ist e_2.
  - Je häufiger die Entitäten mit einander in Beziehung stehen, desto relevanter ist e_2.

9.4.2.2 Methode

Einen einführenden Überblick über Inferenztechniken, die genutzt werden können um Bedürfnisse des Nutzers zu prognostizieren, findet sich bei Jöst [JÖ7]. Meiners stellt in seiner Arbeit vergleichend Methoden zur (allgemeinen) Kontextdatenprognose auf mobilen Geräten vor [Mei09, Kap. 3.3] und nimmt eine detailirte Bewertung vor. Meiners wählt Bay'sche Netze [RN03, Kap. 14f], die dahingehend eingeschränkt werden, dass Berechnungen von Wahrscheinlichkeiten nur in eine Richtung erfolgen, als allgemeine Methode für Kontextdatenprognose, die sich auf mobilen Geräten realisieren lässt. Greene und Finnegan verwenden eine Heurisik für einen ähnlichen Problembereich, allerdings in einem logischen Modell. Sie stellen Beziehungen, die durch Interaktion entstehen, in einem kantengewichteten Graph zwischen Entitäten dar, wobei die Gewichtung der Kanten der Häufigkeit einer Beziehung entspricht. Auf die Zuordnung von Zeit und Ort wird jedoch verzichtet [GF03].

Für die eigene Arbeit wurde ein ähnlicher Ansatz gewählt, der sich auch als vorwärtsgerchtes Bay'sches Netzt darstellen lässt.

Konkret wird für die Heuristik die Zeit um den aktuellen Zeitpunkt in Intervalle eingeteilt. Die Intervalle werden mit zunehmendem Abstand zum aktuellen Zeitpunkt größer (siehe Abbildung 9.10). Für jedes Intervall werden die Aktivitäten und die zugehörigen Entitäten festgestellt, die in dem Intervall stattfinden. So kann die relative Häufigkeit einer Entität in einem Intervall festgestellt werden. Je höher die relative Häufigkeit desto releventer ist die Entität in diesem Intervall. Die Werte für die unterschiedlichen Intervalle werden nun zu einem gemeinsamen Wert zusammengeführt, wobei zeitlich nähere

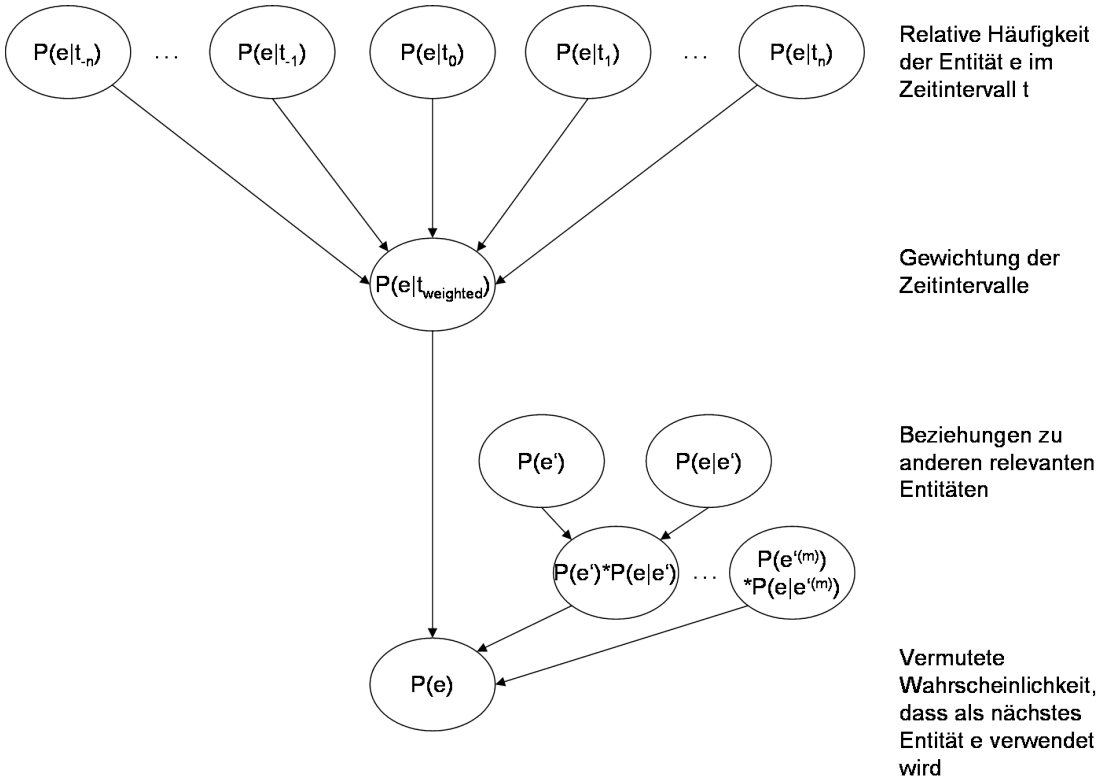

Abbildung 9.11: Die Relevanzheuristik dargestellt als Bay'sches Netz.

Intervalle stärker gewichtet werden. Dieses Zwischenmaß erfüllt die ersten beiden im vorherigen Abschnitt genannten Prämissen. Ein zweites Maß gewichtet die Relevanz aller Entitäten, die mit einer Entität in Beziehung stehen mit der Häufigkeit, mit der die Beziehung auftritt und erfüllt somit die verbleibenden Prämissen. Über ein unscharfes Oder[8] [RN03, S. 500] werden die beiden Maße zu einem gemeinsamen kombiniert. Diese Relevanzheuristik ist in Abbildung 9.11 als Bay'sches Netz dargestellt.

Da die Relevanz anderer Entitäten verwendet wird, ist dieses Vorgehen rekursiv. Die Rekursionstiefe wird auf zwei Schritte beschränkt. Dies entspricht den im Szenario (vgl. Abschnitt 2.2.2) ermittelten Beziehungen 2. Grades. So können Entitäten, die indirekt in Verbindung stehen, z.B. dadurch, dass sie an einem gemeinsamen Ort verwendet werden, noch festgestellt werden, ohne dass der Berechnungsaufwand zu hoch ist. Ohnehin fällt die Relevanz mit jedem Rekursionsschritt stark ab und dürfte schnell gegen Null streben, so dass weitere Schritte zu vernachlässigen sind.

9.4.2.3 Komplexität

Um die Zeiteffizienz des Verfahrens zu optimieren, werden die Häufigkeiten der Entitäten in den Zeitscheiben nicht für jede Anfrage neu ausgezählt, sondern gespeichert vorgehalten. Die Zeitintervalle werden nicht kontinuierlich verschoben, sondern in kurzen

[8] engl.: noisy-Or

diskreten Schritten. Dabei muss ledigleich der Grenzbereich zwischen alten und neuen Intervallen untersucht werden um die Häufigkeiten zu aktualisieren. Die zeitliche Aktualisierungs-Komplexität ist damit linear Abhängig von den Aktivitäten, die in diesen Grenzen beginnen oder aufhören: $f_{updatetime} \in o(|A|)$, wobei A die Menge der Aktivitäten bezeichnet. An Speicherplatz müssen die Häufigkeiten der Entitäten in den Zeitslots und die Häufigkeiten der Beziehungen zwischen zwei Entitäten gespeichert werden. Daraus ergibt sich eine polynomielle Platzkomplexität von $f_{space} \in o(|E|^2)$ mit E der Menge der Entitäten. Dabei kann davon ausgegegangen werden, dass die Beziehungsmatrix zwischen den Entitäten ($|E|^2$) in der Praxis dünn besetzt ist. Die Zeitkomplexität bei Anfragen ist ebenfalls polynomiel. Für das Zwischenmaß der gewichteten Häufigkeiten ist die Komplexität lediglich abhängig von der Anzahl der Zeitslots und somit konstant. Für jeden Rekursionsschritt im zweiten Maß erhöht sich die Komplexität abhängig von der Anzahl der in Beziehung befindlichen Entitäten. Auch hier ist die voraussichtlich dünne Besetzung der Beziehungsmatrix vorteilhaft. Es ergibt sich $f_{querytime} \in o(|E|^2)$.

9.5 Zusammenfassung

Als Plattform für die Entwicklung der Middleware wurde das Android Application Framework gewählt, da es hervorragende Möglichkeiten für das Erfassen von Nutzeraktivitäten bietet. Intern nutzt die entwickelte Middleware filterbasierter Kommunikation in einem verteilten Eventbus. Für die Anwendungsentwicklung wurden simple Schnittstellen für Anfragen und zur Bekanntmachung von Aktivitäten bereitgestellt. Dabei ließ sich sogar die Bereitstellung eines GUI-Widgets für Android umetzen, dass eigenständig kontextadaptives Verhalten realisiert.

Das Beispielsszenario wurde in einem kontextadaptiven Kalender implementiert. Dieser nutzt eine Relevanz-Heuristik, die auf den festgestellten Aktivitäten beruht und eine gute Zeit- und Platzeffizienz aufweist und somit für den Einsatz auf mobilen Geräten geeignet ist.

10 Zusammenfassung und Ausblick

Im Rahmen dieser Arbeit wurde untersucht, wie sich Aktivitätskontext zur Verbesserung der Interaktion mit mobilen Anwendungen einsetzen lässt. Dabei wurde der Fokus auf die Beziehungen zwischen Entitäten gelegt, die durch Aktivitäten entstehen. Diese Beziehungen konnten genutzt werden, um dem Nutzer Vorschläge bei der Eingabe von Entitäten in mobilen Anwendungen zu machen. Hierdurch wird der Eingabeaufwand bei passenden Vorschlägen deutlich reduziert. Eine entsprechende Demoanwendung konnte implementiert werden und zeigte die prinzipielle Umsetzbarkeit dieses Ansatzes auf heutigen mobilen Geräten.

Um sowohl die Entwicklung der Demoanwendung als auch der zukünftigen mobilen Anwendungen, die Aktivitätskontext nutzen, zu vereinfachen sowie um Aktivitätsdaten zwischen Anwendungen austauschbar und gemeinsam nutzbar zu machen, wurde eine Middleware für die Verwaltung der Kontextdaten entworfen und in Teilen implementiert. Diese Middleware nutzt ein generisches Kontextdatenschema für Aktivitätskontext, welches eine einheitliche Sicht auf Aktivitäten erlaubt. Das gewählte Kontextdatenschema für Aktivitäten wurde möglichst abstrakt gehalten und verzichtet darauf, eine detaillierte Semantik für einzelne Aktivitäten zu spezifizieren, wodurch sich ein sehr allgemeines und gleichzeitig kompaktes Modell realisieren ließ.

10.1 Ergebnisse

Es zeigten sich gegenüber anderen Bereichen von Kontext vor allem drei praktische Probleme beim Umgang mit Aktivitätskontext. Das erste Problem besteht darin, dass Anwendungen nicht nur Aktivitätskontext abfragen, sondern die Verwendung der Anwendung auch ein Teil des Kontextes ist. Also müssen Anwendungen in der Lage sein, nicht nur Kontextinformationen abzurufen, sondern auch befähigt sein diese bereit zu stellen.

Ein weiteres Problem besteht in der Verwaltung von Aktivitätskontext, da hier der Umgang mit zeitlichen Verläufen, also aktuellen, vergangenen und zukünftigen, geplanten oder prognostizierten Tätigkeiten notwendig ist. Hierbei ist besonders der Umgang mit zukünftigen Tätigkeiten, wegen der prinzipiell unzuverlässigen Planungen oder Prognosen eine besondere Schwierigkeit.

Als drittes Problem müssen, im Gegensatz zu über Sensoren erfassten Kontextbereichen, die oft nur eine Menge von Messwerten verwalten und auswerten müssen, beim Umgang mit Aktivitäten die Beziehungen zwischen Entitäten effizient verwaltet werden.

Es konnten Mechanismen in die Middleware integriert werden, die, eng mit dem Kontextmodell verzahnt, diese praktischen Schwierigkeiten reduzieren. Ein spezielle API wurde entworfen, um den Verlauf von Aktivitäten durch Anwendungen auf einfache

Art und Weise rückmeldbar zu machen. Das der Middleware zugrundeliegende Event-System erlaubt den Umgang mit in der Zukunft liegenden Ereignissen und die kontextabhängigen dynamischen Namensschemata erlauben es, kontextabhängige Beziehungen aus Sicht einer Anwendung transparent aufzulösen.

Wenn man das in dieser Arbeit entwickelte Verständnis von Aktivitäten auf den Forschungsstand zu Kontext und Kontextadaption rückbezieht, werden einige Zusammenhänge und Ergebnisse dieser Arbeit deutlich, die eher auf einer theoretischen Ebene liegen. Zunächst zeigt sich, dass sich der Zimmermannsche Kontextbegriff (vgl. Abschnitt 4.1.2) [ZLO07], der Kontextinformationen zu einer Entität in die Bereiche *Individualität*, *Aktivität*, *Ort*, *Zeit* und *Beziehungen* aufteilt, im Kontextdatenschema für Aktivitätskontext wiederfinden lässt. Im eigenen Modell stellen *Aktivitäten* die funktionalen *Beziehungen* zwischen Entitäten her. Den Aktivitäten wiederum sind *Ort* und *Zeit* zugeordnet, wodurch sich weitere Beziehungen, etwa durch räumliche Nähe oder zeitliche Muster, herstellen lassen. Entitäten selbst sind im Kern des Kontextdatenschemas zwar nur über einen eindeutigen Bezeichner erfasst, jedoch ist vorgesehen, dass sich Informationen zu ihrer *Individualität* über diesen Bezeichner an anderen Stellen abrufen lassen.

An einer weiteren Stelle lässt sich das eigene Aktivitätsverständnis gut auf die Diskussion zum Umgang mit Kontext beziehen. Der von Dey und Abowd definierte Bereich des Primärkontextes besteht aus *Ort*, *Identität*, *Zeit* und *Aktivität* (vgl. Abschnitt 4.1.3.3) [DA99]. Sekundärkontext sind alle Informationen, die sich aus diesen Bereichen ableiten lassen. Hier zeigt sich, dass die in dieser Arbeit getroffene Entscheidung, auf die Modellierung detaillierter Semantik einzelner Aktivitäten zu verzichten, dazu führt, dass das Modellierte komplett in den Bereich von Primärkontext fällt. Sollten nun bestimmte ableitbare Informationen, also Sekundärkontext, genutzt werden, kann das im Kern der Middleware stehende Kontextdatenmodell um entsprechende Domänen- bzw. Weltwissen repräsentierende Bereiche erweitert werden.

Der dritte Punkt, in dem das in dieser Arbeit entwickelte Verständnis von Aktivitäten und Kontext von theoretisch-philosophischem Belang ist, zeigte sich bereits am Anfang der Arbeit in der Kritik von einem auf Sensordatenverarbeitung fixierten Verständnis von Kontext (vgl. Abschnitt 3.3.4). Als Alternative zu diesem Ansatz wurde vorgeschlagen, dass Entitäten ihre Bedeutung und somit auch ihre Bedeutung für den Kontext in einer digitalen Form selbst repräsentieren. Diesem Gedanken konnte im Rahmen dieser Arbeit nicht in radikaler Form nachgegangen werden. Der Gedanke findet sich jedoch zum Teil in dem Ansatz wieder, dass Softwarewerkzeuge mit Hilfe der Middleware, selbständig über die durch sie realisierten Aktivitäten Auskunft geben, anstatt dass eine Beobachtung von außen versucht wird.

Diese Arbeit konnte also Wege zeigen, Nutzeraktivitäten zu erfassen und Aktivitätskontext als eigenständigen Kontextbereich zu nutzen, um die Bedienung mobiler Geräte

zu vereinfachen. Eine auf Android einsetzbare Middleware und ein intelligenter Kalender als Demoanwendung, welcher aufgrund des Aktivitätskontextes Eingabevorschläge macht, zeigten die praktische Umsetzbarkeit diesen Konzeptes. Somit hat diese Arbeit eine Basis dafür geschaffen, Aktivitätskontext stärker als bisher im Context Aware Computing zu nutzen. Diese Arbeit hat damit einen Beitrag dazu geleistet, kontextadaptiven Anwendungen ein breiteres und vollständigeres Spektrum an Kontextinformationen zur Verfügung zu stellen.

10.2 Weiterer Forschungsbedarf

Aufbauend auf dieser Arbeit lassen sich eine Reihe weiterer Forschungsfragen definieren. Zunächst konnte die entwickelte Heuristik zum Vorschlag von Entitäten (vgl. Abschnitt 9.4.2) nicht evaluiert werden. Es wurde zwar gezeigt, dass diese auf mobilen Geräten performant ist, ob sie jedoch in ihren Ergebnissen auch nützlich ist und ob Nutzer die Entitätsvorschläge als hilfreich empfinden, müsste in einer empirischen Untersuchung überprüft werden. Dabei könnte untersucht werden, inwiefern Nutzer überhaupt sinnvolle Vorschläge wünschen und nutzen. Auch ob die Heuristik im Vergleich zu primitiveren Methoden, Entitäten vorzuschlagen, wie etwa dem Vorschlagen der am häufigsten benutzten Möglichkeiten, überhaupt einen relevanten Vorteil bietet, sollte dabei überprüft werden.

Die entwickelte Middleware lässt sich aber auch als (gutes) Werkzeug für allgemeine empirische Studien zum Nutzerverhalten im Zusammenhang mit mobilen Geräten einsetzen, da sie eine Grundlage zur Erfassung und Analyse von Nutzungsmustern aller Art bietet. Dieses Potential sollte genutzt werden, um die bisher rar gesäten empirischen Studien [CK06] auf diesem Gebiet zu erweitern.

Desweiteren könnte untersucht werden, wie sich die in dieser Arbeit nicht näher betrachteten User-Modelling-Techniken (vgl. Abschnitt 3.1.3) auf Basis der Middleware umsetzen lassen. Ein mögliches Vorgehen könnte dabei sein, die in der Middleware erzeugten Ereignisse zu Aktivitäten durch *Complex Event Processing Techniken* [Luc02] auszuwerten und an ein User Model zu binden.

Das in dieser Arbeit erläuterte Konzept der *Activities* und *Intents* in Android und die mit diesen verbundene gute Beobachtbarkeit von Aktivitäten (vgl. Abschnitt 9.1.7.1) geben Anlass zur Frage, wie Kontextadaption stärker direkt in Frameworks zur Anwendunsentwicklung integriert werden kann, statt diese ausschließlich in einer über den Anwendungen liegenden Middlewareschicht zu behandeln, wie dies auch schon in anderen Ansätzen angedacht wurde [RHC+02][HIM05].

Ein weiterer Bereich, in dem sich Forschungsanstrengungen lohnen würden, ist der Ausbau der Middleware im Hinblick auf Ubiqitous-Computing-Szenarien. Die bisher lokal auf einem Gerät laufende Middleware ist für eine Erweiterung, zum Einsatz auf echte verteilte Systeme geeignet. Auch die Integration anderer Kontextbereiche ist denkbar

und wünschenswert. Eine Nutzung der Middleware für Ubiqitous Computing ist somit gut vorstellbar. Dabei sollte insbesondere der Ansatz verfolgt werden, dass verteilte Softwarewerkzeuge ihre Bedeutungen und die durch Werkzeuge realisierten Aktivitäten selbst transportieren (vgl. Abschnitt 3.3.4).

Literaturverzeichnis

[Ait07] AITENBICHLER, Erwin: Event-Based and Publish/Subscribe Communication. In: MÜHLHAUSER, Max (Hrsg.) ; GUREVYCH, Iryna (Hrsg.): *Handbook of Research on Ubiquitous Computing Technology for Real Time Enterprises*. Idea Group Publishing, Dezember 2007. – ISBN 1599048329, S. 152–171

[App09a] APPLE INC.: *Darwin*. http://developer.apple.com/Darwin/. Version: 2009. – (Abruf: 22.6.2009)

[App09b] APPLE INC.: *iPhone Application Programming Guide: Introduction*. http://developer.apple.com/iphone/library/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/Introduction/chapter_1_section_1.html. Version: Juni 2009. – (Abruf: 24.7.2009)

[App09c] APPLE INC.: *Mac OS X*. http://developer.apple.com/macosx/. Version: 2009. – (Abruf: 22.6.2009)

[AS97] AKMAN, Varol ; SURAV, Mehmet: The use of situation theory in context modeling. In: *Computational Intelligence* 13 (1997), 427—438. http://dx.doi.org/10.1.1.55.4523. – DOI 10.1.1.55.4523

[Bar05] BARDRAM, Jakob E.: The Java Context Awareness Framework (JCAF) - A Service Infrastructure and Programming Framework for Context-Aware Applications. In: *in Pervasive Computing. 2005: Munchen* (2005), 98—115. http://dx.doi.org/10.1.1.94.9175. – DOI 10.1.1.94.9175

[BBH97] BACON, J. ; BATES, J. ; HALLS, D.: Location-oriented multimedia. In: *Personal Communications, IEEE* 4 (1997), Nr. 5, S. 48–57. http://dx.doi.org/10.1109/98.626983. – DOI 10.1109/98.626983. – ISSN 1070–9916

[BD04] BALDAUF, Matthias ; DUSTDAR, Schahram: A survey on context-aware systems. In: *International Journal of Ad Hoc and Ubiquitous Computing* (2004), 2004. http://dx.doi.org/10.1.1.58.9253. – DOI 10.1.1.58.9253

[BG04] BRICKLEY, Dan ; GUHA, R.V.: *RDF Vocabulary Description Language 1.0: RDF Schema*. http://www.w3.org/TR/rdf-schema/. Version: Februar 2004. – (Abruf: 5.6.2009)

[BHH+04] BECHHOFER, Sean ; HARMELEN, Frank van ; HENDLER, Jim ; HORROCKS, Ian ; MCGUINNESS, Deborah L. ; PATEL-SCHNEIDER, Peter F. ; STEIN, Lynn A.: *OWL Web Ontology Language Reference*. http://www.w3.org/TR/owl-ref/. Version: Februar 2004. – (Abruf: 5.2.2009)

[BHLT06] BRAY, Tim ; HOLLANDER, Dave ; LAYMAN, Andrew ; TOBIN, Richard: *Namespaces in XML 1.0 (Second Edition)*. http://www.w3.org/TR/REC-xml-names/. Version: August 2006

[BK04] BAUER, Christian ; KING, Gavin: *Hibernate in Action*. Manning Publications, 2004 (In Action Series). – ISBN 193239415X

[Bla09] BLANDFORD, Rafe: *All About Symbian Devices*. http://www.allaboutsymbian.com/devices/. Version: 2009. – (Abruf: 22.6.2009)

[BLHL01] BERNERS-LEE, Tim ; HENDLER, James ; LASSILIA, Ora: The Semantic Web. In: *Scientific American Magazine* (2001), Mai

[Bli07] BLISZCZAK, Krzysztof: *Data Sharing Tips*. 1. Symbian Press, 2007 (Using Symbian OS). http://developer.symbian.com/china/documentation/books/books_files/pdf/data_sharing_tips.pdf. – (Abruf: 22.6.2009)

[Bor08] BORNSTEIN, Dan: *Dalvik Virtual Machine Internals*. Version: Juni 2008. http://sites.google.com/site/io/dalvik-vm-internals (Vortragsaufzeichnung, Abruf: 17.6.2009)

[BP83] BARWISE, J. ; PERRY, J.: *Situations and Attitudes*. MIT Press, 1983

[BP99] BRÉZILLON, Patrick ; POMEROL, J. ch: Contextual Knowledge Sharing And Cooperation In Intelligent Assistant Systems. In: *Le Travail Humain* 62 (1999), S. 223—246

[BPSM+08] BRAY, Tim ; PAOLI, Jean ; SPERBERG-MCQUEEN, C. M. ; MALER, Eve ; YERGEAU, François: *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. http://www.w3.org/TR/2008/REC-xml-20081126/. Version: November 2008

[Bra07] BRAUBACH, Lars: *Architekturen und Methoden zur Entwicklung verteilter agentenorientierter Softwaresysteme*, Universität Hamburg, Fachbereich Informatik, Verteilte Systeme und Informationssysteme, Diss., 2007

[CDME01] CHEVERST, Keith ; DAVIES, Nigel ; MITCHELL, Keith ; EFSTRATIOU, Christos: Using Context as a Crystal Ball: Rewards and Pitfalls. In: *Personal and Ubiquitous Computing* 5 (2001), 8—11. http://dx.doi.org/10.1.1.31.7607. – DOI 10.1.1.31.7607

[CFJ03] CHEN, Harry ; FININ, Tim ; JOSHI, A.: Using OWL in a Pervasive Computing Broker. In: *Proceedings of the Workshop on Ontologies in Agent Systems (OAS)*, 2003

[Che04] CHEN, Harry: *An Intelligent Broker Architecture for Pervasive Context-Aware Systems*, University of Maryland, Baltimore County, Diss., Dezember 2004

[CK00] CHEN, Guanling ; KOTZ, David: A Survey of Context-Aware Mobile Computing Research. 2000. – Forschungsbericht

[CK02] CHEN, Guanling ; KOTZ, David: Solar: An open platform for context-aware mobile applications. In: *In Proceedings of the First International Conference on Pervasive Computing (Pervasive 2002* (2002), 41—47. http://dx.doi.org/10.1.1.12.3991. – DOI 10.1.1.12.3991

[CK06] COURSARIS, Constantinos K. ; KIM, Dan J.: A Qualitative Review of Empirical Mobile Usability Studies. In: *Proceedings of the Twelfth American Conference on Information Systems (AMCIS)*, 2006

[CMD99] CHEVERST, Keith ; MITCHELL, Keith ; DAVIES, Nigel: Design of an object model for a context sensitive tourist GUIDE. In: *Computers and Graphics* 23 (1999), 24—25. http://dx.doi.org/10.1.1.100.7976. – DOI 10.1.1.100.7976

[Col06] COLEMAN, Regan: *Symbian Database Components*. Version: Juni 2006. http://www.ddj.com/mobile/189601913 (Abruf: 22.6.2009)

[DA99] DEY, Anind ; ABOWD, Gregory: Towards a Better Understanding of Context and Context-Awareness. Version: 1999. http://gvu.cc.gatech.edu/reports/1999/abstracts/99-22. 1999. – Forschungsbericht

[DAW98] DEY, Anind K. ; ABOWD, Gregory D. ; WOOD, Andrew: CyberDesk: a framework for providing self-integrating context-aware services. In: *Knowledge-Based Systems* 11 (1998), September, Nr. 1, 3–13. http://dx.doi.org/10.1016/S0950-7051(98)00053-7. – DOI 10.1016/S0950–7051(98)00053–7. – ISSN 0950–7051

[Dou04] DOURISH, Paul: What we talk about when we talk about context. In: *Personal and Ubiquitous Computing* 8 (2004). http://dx.doi.org/10.1.1.18.8587. – DOI 10.1.1.18.8587

[DRD+00] DIX, Alan ; RODDEN, Tom ; DAVIES, Nigel ; TREVOR, Jonathan ; FRIDAY, Adrian ; PALFREYMAN, Kevin: Exploiting space and location as a design framework for interactive mobile systems. In: *ACM Trans. Comput.-Hum. Interact.* 7 (2000), Nr. 3, S. 285—321. http://dx.doi.org/http://doi.acm.org/10.1145/355324.355325. – DOI http://doi.acm.org/10.1145/355324.355325. – ISSN 1073–0516

[EM99] ENGESTRÖM, Yrjö ; MIETTINEN, Reijo: Introduction. In: *Perspectives on Activity Theory*. Cambridge University Press, 1999 (Lerning in doing: Social, kognitive and cumputational perspectives). – ISBN 0–521–43127–1, S. 1–18

[EMC08] EMCC SOFTWARE LTD.: *Using the Symbian OS Contacts API*. http://developer.symbian.com/main/downloads/papers/Using_Symbian_Contacts_Model.pdf. Version: Februar 2008. – (Abruf: 22.6.2009)

[Eng87] ENGSTRÖM, Yrjö: *Learning by expanding*. Helsinki : Orienta Consultit, 1987

[Eng99] ENGESTRÖM, Yrjö: Activity Theory and individual and social transformation. In: *Perspectives on Activity Theory*. Cambridge University Press, 1999 (Lerning in doing: Social, kognitive and cumputational perspectives). – ISBN 0–521–43127–1, S. 19–38

[FHH+00] FENSEL, Dieter ; HORROCKS, Ian ; HARMELEN, Frank van ; DECKER, Stefan ; ERDMANN, Michael ; KLEIN, Michel C. A.: OIL in a Nutshell. In: *Knowledge Acquisition, Modeling and Management*, 2000, 1–16

[fre09a] FREEDESKTOP.ORG: *D-Bus Bindings*. Version: Juni 2009. http://freedesktop.org/wiki/Software/DBusBindings (Abruf: 24.7.2009)

[fre09b] FREEDESKTOP.ORG: *What is D-Bus?* Version: Juli 2009. http://freedesktop.org/wiki/Software/dbus (Abruf: 24.7.2009)

[fre09c] FREESMARTPHONE.ORG: *Architecture - freesmartphone.org*. Version: April 2009. http://www.freesmartphone.org/index.php/Architecture (Abruf: 22.6.2009)

[fre09d] FREESMARTPHONE.ORG: *Manifesto - freesmartphone.org*. Version: 2009. http://www.freesmartphone.org/index.php/Manifesto

[Gar09a] GARTNER INC.: *Gartner Says Worldwide Mobile Phone Sales Declined 8.6 Per Cent and Smartphones Grew 12.7 Per Cent in First Quarter of 2009*. Version: Mai 2009. http://www.gartner.com/it/page.jsp?id=985912

[Gar09b] GARTNER INC.: *Gartner Says Worldwide Mobile Phone Sales Grew 6 Per Cent in 2008, But Sales Declined 5 Per Cent in the Fourth Quarter*. Version: März 2009. http://www.gartner.com/it/page.jsp?id=904729

[GF03] GREENE, Stephen ; FINNEGAN, Jason: Usability of mobile devices and intelligently adapting to a user's needs. In: *Proceedings of the 1st international symposium on Information and communication technologies*. Dublin, Ireland : Trinity College Dublin, 2003, 175–180

[GG01] GHIDINI, C. ; GIUNCHIGLIA, Fausto: Local Models Semantics, or Contextual Reasoning = Locality Compatibility. In: *Artifical Intelligence* 127 (2001), Nr. 2, 221–259. http://dx.doi.org/10.1.1.23.7598. – DOI 10.1.1.23.7598

[GJNS08] GJERTSEN, Freddie ; JODE, Freddie ; NORTHAM, Phil ; SHACKMAN, Mark: *Getting Started on Symbian OS*. 4. Symbian Press, 2008 (Using Symbian OS). http://developer.symbian.com/main/documentation/books/books_files/pdf/Getting_Started_final.pdf. – (Abruf: 17.12.2008)

[Goo09a] GOOGLE INC.: *android-scripting - Project Hosting on Google Code*. http://code.google.com/p/android-scripting/. Version: 2009. – (Abruf: 24.7.2009)

[Goo09b] GOOGLE INC.: *Application Fundamentals*. http://developer.android.com/guide/topics/fundamentals.html. Version: 2009

[Goo09c] GOOGLE INC.: *What is Android?* http://developer.android.com/guide/basics/what-is-android.html. Version: 2009. – (Abruf: 6.6.2009)

[Got07] GOTTIPATI, Hari K.: *Dalvik - Google's tweaked, non-standard JVM for Android!!!!* http://www.oreillynet.com/onjava/blog/2007/11/dalvik_googles_tweaked_nonstan.html. Version: November 2007. – (Abruf: 23.6.2009)

[Gro07] GROSSMAN, Lev: Invention Of the Year: The iPhone - The Best Inventions Of The Year. In: *Time Magazine* 170 (2007), Dezember, Nr. 20. http://www.time.com/time/specials/2007/article/0,28804,1677329_1678542,00.html. – ISSN 0040–718X

[GS01a] GRAY, Philip D. ; SALBER, Daniel: Modelling and Using Sensed Context Information in the Design of Interactive Applications. In: *Proceedings of the 8th IFIP International Conference on Engineering for Human-Computer Interaction*, Springer-Verlag, 2001. – ISBN 3–540–43044–X, 317–336

[GS01b] GROSS, Tom ; SPECHT, Marcus: Awareness in Context-Aware Information Systems. In: OBERQUELLE, Horst (Hrsg.) ; OPPERMANN, Reinhard (Hrsg.) ; KRAUSE, Jürgen (Hrsg.): *Mensch und Computer 2001*. Bad Honnef, Deutschland : Teubner-Verlag, 2001, S. 173–182

[HA07] HARTMANN, Melanie ; AUSTALLER, Gerhard: Context Models and Context Awareness. In: MÜHLHAUSER, Max (Hrsg.) ; GUREVYCH, Iryna (Hrsg.): *Handbook of Research on Ubiquitous Computing Technology for Real Time Enterprises*. Idea Group Publishing, Dezember 2007. – ISBN 1599048329, S. 235–256. – (Kapitel 9)

[Hal01] HALPIN, Terry: *Information modeling and relational databases: from conceptual analysis to logical design*. Morgan Kaufmann Publishers Inc., 2001. – 760 S. – ISBN 1–55860–672–6

[Har06] HARBECK, Mathias: *Java-Agenten unterwegs. BDI-Agentensysteme auf mobilen Geräten*. 1. Vdm Verlag Dr. Müller, 2006. – ISBN 3865508529

[HBH+98] HORVITZ, Eric ; BREESE, Jack ; HECKERMAN, David ; HOVEL, David ; ROMMELSE, Koos: The Lumiere project: Bayesian user modeling for inferring the goals and needs of software users. In: *In Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence* (1998), 256—265. http://dx.doi.org/10.1.1.39.8472. – DOI 10.1.1.39.8472

[HBS02] HELD, Albert ; BUCHHOLZ, Sven ; SCHILL, Alexander: Modeling of Context Information for Pervasive Computing Applications. (2002). http://dx.doi.org/10.1.1.20.3976. – DOI 10.1.1.20.3976

[HC02] HESS, Christopher K. ; CAMPBELL, Roy H.: A Context File System for Ubiquitous Computing Environments / Department of Computer Science, University of illinois. Urbana, Illinois, Juli 2002 (UIUCDCS-R-2002-2285 UILU-ENG-2002-1729). – Technical Report

[Hen03] HENRICKSEN, Karen: *A framework for context-aware pervasive computing applications*, University of Queensland, Diss., September 2003. http://henricksen.id.au/publications/phd-thesis.pdf. – 201 S.

[HI05] HENRICKSEN, Karen ; INDULSKA, Jadwiga: Developing context-aware pervasive computing applications: models and approach. In: *Pervasive and Mobile Computing, In* 2 (2005). http://dx.doi.org/10.1.1.59.8031. – DOI 10.1.1.59.8031

[HIM05] HENRICKSEN, Karen ; INDULSKA, Jadwiga ; MCFADDEN, Ted: Middleware for Distributed Context-Aware Systems. In: *International Symposium on Distributed Objects and Applications (DOA* 3760 (2005), 846—863. http://dx.doi.org/10.1.1.59.8932. – DOI 10.1.1.59.8932

[HIR02] HENRICKSEN, Karen ; INDULSKA, Jadwiga ; RAKOTONIRAINY, Andry: Modeling context information in pervasive computing systems. 2414 (2002), 167—180. http://dx.doi.org/10.1.1.19.8899. – DOI 10.1.1.19.8899

[HIR03] HENRICKSEN, Karen ; INDULSKA, Jadwiga ; RAKOTONIRAINY, Andry: Generating Context Management Infrastructure from High-Level Context Models. In: *In 4th International Conference on Mobile Data Management (MDM) - Industrial Track* (2003), 1—6. http://dx.doi.org/10.1.1.12.6109. – DOI 10.1.1.12.6109

[HSP+03] HOFER, Thomas ; SCHWINGER, Wieland ; PICHLER, Mario ; LEONHARTSBERGER, Gerhard ; ALTMANN, Josef ; RETSCHITZEGGER, Werner: Context-Awareness on Mobile Devices - the Hydrogen Approach. In: *Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03)*, IEEE Computer Society, 2003. – ISBN 0–7695–1874–5, 292.1

[HWMI05] HENRICKSEN, Karen ; WISHART, Ryan ; MCFADDEN, Ted ; INDULSKA, Jadwiga: Extending Context Models for Privacy in Pervasive Computing Environments. In: *Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications Workshops*, IEEE Computer Society, 2005. – ISBN 0–7695–2300–5, 20–24

[Int86] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION (ISO): *ISO 8879: Standard Generalized Markup Language (SGML)*. `http://www.iso.org/iso/catalogue_detail.htm?csnumber=16387`. Version: 1986

[JÖ7] JÖST, Matthias: Adapting to the User. In: MÜHLHAUSER, Max (Hrsg.) ; GUREVYCH, Iryna (Hrsg.): *Handbook of Research on Ubiquitous Computing Technology for Real Time Enterprises*. Idea Group Publishing, Dezember 2007. – ISBN 1599048329, S. 282–295

[Job07] JOBS, Steve: *Macworld 2007 Keynote*. `http://www.engadget.com/2007/01/09/live-from-macworld-2007-steve-jobs-keynote/`. Version: 2007. – (Vortragsaufzeichnung, Abruf: 22.6.2009)

[KA04a] KAENAMPORNPAN, Manasawee ; AY, Bath B.: An Intergrated Context Model: Bringing Activity to Context. In: *In Workshop on Advanced Context Modelling, Reasoning and Management - UbiComp* (2004). `http://dx.doi.org/10.1.1.58.6428`. – DOI 10.1.1.58.6428

[KA04b] KAENAMPORNPAN, Manasawee ; AY, Bath B.: Modelling context: an Activity Theory approach. In: *Ambient Intelligence: Second European Symposium, EUSAI 2004*, Springer, 2004, S. 367—374

[Kap99] KAPTELININ: The activity checklist: a tool for representing the Space of context. In: *interactions* 6 (1999), Nr. 4, S. 39, 27

[KE04] KEMPER, Alfons ; EICKLER, Andre: *Datenbanksysteme*. 5. Oldenbourg, 2004

[Kjæ07] KJÆR, Kristian E.: A survey of context-aware middleware. In: *Proceedings of the 25th conference on IASTED International Multi-Conference: Software Engineering*. Innsbruck, Austria : ACTA Press, 2007, 148–155

[KLW95] KIFER, Michael ; LAUSEN, Georg ; WU, James: Logical foundations of object-oriented and frame-based languages. In: *Journal of the ACM* 42 (1995), S. 741—843

[KN96] KAPTELININ, Victor ; NARDI, Bonnie: Activity theory: implications for human-computer interaction. Version: 1996. http://www.ics.uci.edu/~corps/phaseii/nardi-ch5.pdf. In: *Context and Consciousness: Activity Theory and Human-Computer Interaction*. MIT Press, 1996, 116, 103. – (Kap. 5)

[KP88] KRASNER, Glenn E. ; POPE, Stephen T.: A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System. In: *Journal of Object Oriented Programming* 1 (1988), Nr. 3, S. 26–49

[KPC06] KOFOD-PETERSEN, Anders ; CASSENS, Jörg: Using activity theory to model context awareness. In: *Modeling and Retrieval of Context: Second International Workshop, MRC 2005, Revised Selected Papers. Volume 3946 of LNCS (LNAI)* (2006), 1—17. http://dx.doi.org/10.1.1.80.5088. – DOI 10.1.1.80.5088

[Kun05] KUNZE, Christian P.: DEMAC: A Distributed Environment for Mobility Aware Computing. In: *Adjunct Proceedings of the Third International Conference on Pervasive Computing* Bd. 191, 2005 (Advances in Pervasive Computing)

[Kuu96] KUUTTI, Kari: Activity Theory as a potential framework for human-computer interaction research. In: *Context and Consciousness: Activity Theory and Human-Computer Interaction*. MIT Press, 1996, S. 17–44

[LK06] LEINER, Ulrich ; KRZONKALLA, Sonja: Mobile Endgeräte: Einfache Alleskönner - ein Widerspruch? In: *i-com - Zeitschrift für interaktive und kooperative Medien* 1 (2006), Nr. 1, S. 41—47

[LSD+02] LEI, Hui ; SOW, Daby M. ; DAVIS, I. I. John S. ; BANAVAR, Guruduth ; EBLING, Maria R.: The design and applications of a context service. In: *SIGMOBILE Mob. Comput. Commun. Rev.* 6 (2002), Nr. 4, 45–55. http://dx.doi.org/10.1145/643550.643554. – DOI 10.1145/643550.643554

[Luc02] LUCKHAM, David: *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison Wesley Professional, 2002

[Mal07] MALAKA, Rainer: Intelligent User Interfaces for Ubiquitous Computing. In: MÜHLHAUSER, Max (Hrsg.) ; GUREVYCH, Iryna (Hrsg.): *Handbook of Research on Ubiquitous Computing Technology for Real Time Enterprises*. Idea Group Publishing, Dezember 2007. – ISBN 1599048329, S. 470–486

[Mcc93] MCCARTHY, John: Notes on Formalizing Context. In: *null* (1993), 555—560. http://dx.doi.org/10.1.1.35.8117. – DOI 10.1.1.35.8117

[Mei09] MEINERS, Matthias: *Kontextdatenprognose auf mobilen Geräten*, Universität Hamburg, Fakultät für Informatik und Naturwissenschaften, Verteilte Systeme und Informationssysteme, Diplomarbeit, 2009. – (laufend)

[Mic08a] MICROSOFT DEVELOPER NETWORK: *AppointmentCollection Class (Microsoft.WindowsMobile.PocketOutlook)*. http://msdn.microsoft.com/en-us/library/microsoft.windowsmobile.pocketoutlook.appointmentcollection.aspx. Version: März 2008. – (Abruf: 22.6.2009)

[Mic08b] MICROSOFT DEVELOPER NETWORK: *Windows Mobile 6 SDK Documentation*. http://msdn.microsoft.com/de-de/library/bb158486(en-us).aspx. Version: Oktober 2008. – (Abruf: 22.6.2009)

[Mic09a] MICROSOFT: *SQL Server Compact 3.5 for Windows Mobile*. http://www.microsoft.com/downloads/details.aspx?FamilyID=38ed2670-a70a-43b3-87f3-7ab67b56cbf2&displaylang=en. Version: 2009. – (Abruf: 22.6.2009)

[Mic09b] MICROSOFT DEVELOPER NETWORK: *.NET Compact Framework*. http://msdn.microsoft.com/en-us/library/f44bbwa1.aspx. Version: 2009. – (Abruf: 22.6.2009)

[Nok09] NOKIA GROUP: *Documentation - Runtime environments*. Version: 2009. http://developer.symbian.com/main/documentation/runtime_environments/ (Abruf: 22.6.2009)

[ope07] OPEN HANDET ALLIANCE: *Open Handset Alliance FAQ*. http://www.openhandsetalliance.com/oha_faq.html. Version: November 2007. – (Abruf: 6.6.2009)

[Ope09a] OPENMOKO INC.: *Introduction - Openmoko*. http://wiki.openmoko.org/wiki/Introduction#Software. Version: Juni 2009. – (Abruf: 24.7.2009)

[Ope09b] OPENMOKO INC.: *Neo FreeRunner - Openmoko*. http://wiki.openmoko.org/wiki/Neo_FreeRunner. Version: Juni 2009. – (Abruf: 24.7.2009)

[ORT05] OULASVIRTA, Antti ; RAENTO, Mika ; TIITTA, Sauli: ContextContacts: redesigning Smartphone's contact book to support mobile awareness and collaboration. In: *IEEE Photon. Technol. Lett*, Portal, 2005, S. 167—174

[Oul05] OULASVIRTA, Antti: The fragmentation of attention in mobile interaction, and what to do with it. In: *ACM interactions* 12 (2005), Nr. 6, 16–18. http://dx.doi.org/10.1145/1096554.1096555. – DOI 10.1145/1096554.1096555

[Pea02] PEASE, Adam: *Why Use DAML?* Version: April 2002. http://www.daml.org/2002/04/why.html (Abruf: 5.6.2009)

[PS97] PEDERSEN, Elin R. ; SOKOLER, Tomas: AROMA: abstract representation of presence supporting mutual awareness. In: *Proceedings of the SIGCHI conference on Human factors in computing systems*. Atlanta, Georgia, United States : ACM, 1997. – ISBN 0–89791–802–9, 51–58

[PSGP01] PHAM, Thai-Lai ; SCHNEIDER, Georg ; GOOSE, Stuart ; PIZANO, Arturo: Composite Device Computing Environment: A Framework for Situated Interaction Using Small Screen Devices. In: *Personal Ubiquitous Comput.* 5 (2001), Nr. 1, S. 25—28. http://dx.doi.org/http://dx.doi.org/10.1007/s007790170024. – DOI http://dx.doi.org/10.1007/s007790170024. – ISSN 1617–4909

[PV05] PEDELL, Sonja ; VETERE, Frank: Visualizing use context with picture scenarios in the design process. In: *Proceedings of the 7th international conference on Human computer interaction with mobile devices and services*. Salzburg, Austria : ACM, 2005. – ISBN 1–59593–089–2, 271–274

[RDF+05] RIEKKI, Jukka ; DAVIDYUK, OLEG ; FORSTADIUS, JARI ; SUN, JUNZHAO ; SAUVOLA, JAAKKO: Enabling Context-Aware Services for Mobile Users. In: *Proceedings of IADIS Virtual Multi Conference on Computer Science and Information Systems*, 2005, S. 360–369

[RG05] RYAN, Caspar ; GONSALVES, Atish: The effect of context and application type on mobile usability: an empirical study. In: *Proceedings of the Twenty-eighth Australasian conference on Computer Science - Volume 38*. Newcastle, Australia : Australian Computer Society, Inc., 2005. – ISBN 1–920–68220–1, 115–124

[RHC+02] ROMÁN, Manuel ; HESS, Christopher ; CERQUEIRA, Renato ; CAMPBELL, Roy H. ; NAHRSTEDT, Klara: Gaia: A Middleware Infrastructure to Enable Active Spaces. In: *IEEE Pervasive Computing* 1 (2002), 74—83. http://dx.doi.org/10.1.1.115.6723. – DOI 10.1.1.115.6723

[RHI07] ROBINSON, Ricky ; HENRICKSEN, Karen ; INDULSKA, Jadwiga: XCML: A Runtime Representation for the Context Modelling Language. In: *Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications Workshops*, IEEE Computer Society, 2007. – ISBN 0–7695–2788–4, 20–26

[RN03] RUSSELL, Stuart ; NORVIG, Peter: *Artificial Intelligence: A Modern Approach*. 2. Prentice Hall, 2003. – ISBN 0137903952

[Rom03] ROMÁN, Manuel: *An Application Framework for Active Space Applications*, Graduate College of the University of Illinois at Urbana-Champaign, Diss., 2003

[ROPT05] RAENTO, Mika ; OULASVIRTA, Antti ; PETIT, Renaud ; TOIVONEN, Hannu: ContextPhone: A Prototyping Platform for Context-Aware Mobile Applications. In: *IEEE Pervasive Computing* 4 (2005), Nr. 2, S. 51—59. http://dx.doi.org/http://dx.doi.org/10.1109/MPRV.2005.29. – DOI http://dx.doi.org/10.1109/MPRV.2005.29. – ISSN 1536–1268

[Sat96] SATYANARAYANAN, M.: Fundamental challenges in mobile computing. In: *PODC '96: Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing*. New York, NY, USA : ACM, 1996. – ISBN 0–89791–800–2, S. 1—7

[SAT+99] SCHMIDT, Albrecht ; AIDOO, Kofi A. ; TAKALUOMA, Antti ; TUOMELA, Urpo ; LAERHOVEN, Kristof V. ; VELDE, Walter Van D.: Advanced interaction in context. In: *In Proceedings of First International Symposium on Handheld and Ubiquitous Computing*, Springer Verlag, 1999, S. 89—101

[SAW94] SCHILIT, Bill N. ; ADAMS, Norman ; WANT, Roy: Context-aware computing applications. In: *In Proceedings of the Workshop on Mobile Computing Systems and Applications* (1994), 85—90. http://dx.doi.org/10.1.1.29.5833. – DOI 10.1.1.29.5833

[SBG99] SCHMIDT, Albrecht ; BEIGL, Michael ; GELLERSEN, Hans w: There is more to context than location. In: *Computers and Graphics* 23 (1999), 893—901. http://dx.doi.org/10.1.1.37.2933. – DOI 10.1.1.37.2933

[Sch95] SCHILIT, William N.: *A system architecture for context-aware mobile computing*, Columbia University, Diss., 1995. http://portal.acm.org/citation.cfm?id=220826

[Sch00] SCHMIDT, Albrecht: Implicit human computer interaction through context. In: *Personal and Ubiquitous Computing* 4 (2000), Juni, Nr. 2, 191–199. http://dx.doi.org/10.1007/BF01324126. – DOI 10.1007/BF01324126

[SDA99] SALBER, Daniel ; DEY, Anind K. ; ABOWD, Gregory D.: The context toolkit: aiding the development of context-enabled applications. In: *Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit*. Pittsburgh, Pennsylvania, United States : ACM, 1999. – ISBN 0–201–48559–1, 434–441

[Sha06] SHACKMAN, Mark: *Using the new Symbian OS v9 Calendar API*. http://developer.symbian.com/main/downloads/papers/Using%

20the%20new%20v9%20Calendar%20API.pdf. Version: September 2006. – (Abruf: 22.6.2009)

[Sha07] SHANKLAND, Stephen: *Sun starts bidding adieu to mobile-specific Java*. Version: Oktober 2007. http://news.cnet.com/8301-13580_3-9800679-39.html?part=rss&subj=news&tag=2547-1_3-0-5 (Abruf: 22.6.2009)

[SLP04] STRANG, Thomas ; LINNHOFF-POPIEN, Claudia: A Context Modeling Survey. In: *In: Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp 2004 - The Sixth International Conference on Ubiquitous Computing, Nottingham/England* (2004). http://dx.doi.org/10.1.1.2.2060. – DOI 10.1.1.2.2060

[SLPF03a] STRANG, Thomas ; LINNHOFF-POPIEN, Claudia ; FRANK, Korbinian: Applications of a Context Ontology Language. In: *UNIVERSITY OF SPLIT, CROATIA* (2003), S. 14—18

[SLPF03b] STRANG, Thomas ; LINNHOFF-POPIEN, Claudia ; FRANK, Korbinian: CoOL: A Context Ontology Language to enable Contextual Interoperability. In: *Proceedings of 4th IFIP WG 6.1 International Conference On Distributed Applications And Interoperable Systems* Bd. 2893. Paris. France, 2003 (LNCS), S. 236—247

[SNE] STEINIGER, Stefan ; NEUN, Moritz ; EDWARDES, Alistair: *Foundations of Location Based Services - Lecture Notes on LBS, Lesson 1.* www.geo.unizh.ch/publications/cartouche/lbs_lecturenotes_steinigeretal2006.pdf. – (Abruf: 1.9.2008)

[Sor] SORCINELLI, Don: *Your Windows Mobile 5.0 Applications Can Monitor Clients and Respond to Change*. http://www.developer.com/ws/pc/article.php/3547381. – (Abruf: 22.6.2009)

[SS04] STAAB, Steffen ; STUDER, Rudi: *Handbook on Ontologies*. Birkhäuser, 2004. – 660 S. – ISBN 3540408347, 9783540408345

[SS07] SCHILL, Alexander ; SPRINGER, Thomas: *Verteilte Systeme*. Springer, 2007. – 368 S. – ISBN 3540205683, 9783540205685

[ST93] SPREITZER, Mike ; THEIMER, Marvin: Providing location information in a ubiquitous computing environment (panel session). In: *SIGOPS Oper. Syst. Rev.* 27 (1993), Nr. 5, 270–283. http://dx.doi.org/10.1145/173668.168641. – DOI 10.1145/173668.168641

[Sun] SUN MICROSYSTEMS. INC.: *JDBC Overview*. http://java.sun.com/products/jdbc/overview.html. – (Abruf: 24.6.2009)

[Sun04] SUN MICROSYSTEMS. INC.: *Java Annotations*. http://java.sun.com/j2se/1.5.0/docs/guide/language/annotations.html. Version: 2004. – (Abruf: 24.7.2009)

[Sun09a] SUN MICROSYSTEMS. INC.: *The Java Community Process(SM) Program - JSRs: Java Specification Requests - detail JSR# 235*. http://jcp.org/en/jsr/detail?id=235. Version: 2009. – (Abruf: 22.6.2009)

[Sun09b] SUN MICROSYSTEMS. INC.: *Java ME - the Most Ubiquitous Application Platform for Mobile Devices*. http://java.sun.com/javame/index.jsp. Version: 2009. – (Abruf: 17.6.2009)

[Sun09c] SUN MICROSYSTEMS. INC.: *Java ME Technology - Java Community Process*. http://java.sun.com/javame/technology/jcp.jsp. Version: 2009. – (Abruf: 22.6.2009)

[Sun09d] SUN MICROSYSTEMS. INC.: *Java ME Technology - Mobile Service Architecture - JSR 179*. http://java.sun.com/javame/technology/msa/jsr179.jsp. Version: 2009. – (Abruf 22.6.2009)

[Sun09e] SUN MICROSYSTEMS. INC.: *Java ME Technology - Mobile Service Architecture - JSR 75*. http://java.sun.com/javame/technology/msa/jsr75.jsp. Version: 2009. – (Abruf: 22.6.2009)

[Sun09f] SUN MICROSYSTEMS. INC.: *The Squawk Project*. http://research.sun.com/projects/squawk/#Pubs. Version: 2009. – (Abruf: 17.6.2009)

[SWM04] SMITH, Micael K. ; WELTY, Chris ; MCGUINNESS, Deborah L.: *OWL Web Ontology Language Guide*. http://www.w3.org/TR/owl-guide/. Version: Februar 2004

[Sym08] SYMBIAN DEVELOPER NETWORK: *Introduction to Location-Based Services on Symbian OS*. http://developer.symbian.com/main/downloads/papers/LBS/Introduction_to_Location-Based_Services_on_Symbian_OS.pdf. Version: Februar 2008. – (Abruf: 22.6.2009)

[Sym09] SYMBIAN FOUNDATION LTD: *About the Symbian Foundation*. http://www.symbian.org/about/index.php. Version: 2009. – (Abruf: 22.6.2009)

[Tec09] TECHBLOGY: *Stop! iPhone Rotation | TechBlogy*. http://techblogy.com/stop-iphone-rotation/. Version: Juni 2009. – (Abruf: 6.6.2009)

[Tri08] TRIVEDI, Kamal: *List of Database in J2ME*. http://www.coderanch.com/t/230853/Java-Micro-Edition/Mobile/List-Database-J-ME. Version: September 2008. – (Abruf: 22.6.2009)

[Tur06] TURJALEI, Mirwais: *Integration von Context-Awareness in eine Middleware für mobile Systeme*, Universität Hamburg, Fakultät für Mathematik, Informatik und Naturwissenschaften, Verteielte Systeme und Informationssysteme, Diplomarbeit, Juni 2006

[Tur09] TURNER, Davis: *Introducing Android 1.5 NDK, Release 1.* http://android-developers.blogspot.com/2009/06/introducing-android-15-ndk-release-1.html. Version: Juni 2009. – (Abruf: 24.7.2009)

[WB97] WEISER, Mark ; BROWN, John S.: The coming age of calm technolgy. Version: 1997. http://portal.acm.org/citation.cfm?id=504934. In: *Beyond calculation: the next fifty years*. Copernicus, 1997. – ISBN 0–38794932–1, 75–85

[Wei91] WEISER, Mark: The Computer for the 21st Century. In: *Scientific American* (1991), Nr. 265, S. 94—104. – (Abruf: 6.11.2008)

[Wei93] WEISER, Mark: *Ubiquitous Computing*. http://www.ubiq.com/hypertext/weiser/UbiCompHotTopics.html. Version: August 1993

[Win01a] WINOGRAD, Terry: Architectures for context. In: *Hum.-Comput. Interact.* 16 (2001), Nr. 2, 401–419. http://portal.acm.org/citation.cfm?id=1463126

[Win01b] WINOGRAD, Terry: Interaction spaces for 1st century computing. In: *Human-Computer Interaction in the New Millennium*. Addison-Wesley Longman, Amsterdam, Mai 2001. – ISBN 0201704471, S. 259–276

[WSA+96] WANT, Roy ; SCHILIT, WN ; ADAMS, NI ; GOLD, Rich ; PETERSEN, K ; GOLDBERG, D ; ELLIS, JR ; WEISER, M ; IMIELINSKI, T ; KORTH, HF: The ParcTab Ubiquitous Computing Experiment. Version: 1996. http://sandbox.xerox.com/parctab/csl9501/paper.html. In: *Mobile Computing*. 1996, 45–102. – (Kapitel 2)

[ZLO07] ZIMMERMANN, Andreas ; LORENZ, Andreas ; OPPERMANN, Reinhard: An Operational Definition of Context. Version: 2007. http://dx.doi.org/10.1007/978-3-540-74255-5_42. In: *Modeling and Using Context*. 2007, 558–571

Abbildungsverzeichnis

Tabellenverzeichnis

Eidesstattliche 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.

Ich bin mit einer Einstellung in den Bestand der Bibliothek des Fachbereiches einverstanden.

Hamburg, den 27.7.2009 Jan D.S. Wischweh