Kontextadaptive Dienstnutzung in Ubiquitous Computing Umgebungen

Dissertation

an der

Fakultät für Mathematik, Informatik und Statistik
der
Ludwig-Maximilians-Universität München

vorgelegt von

Michael Samulowitz

Tag der Einreichung: 25. Juni 2002

Kontextadaptive Dienstnutzung in Ubiquitous Computing Umgebungen

Dissertation

an der

Fakultät für Mathematik, Informatik und Statistik
der
Ludwig-Maximilians-Universität München

vorgelegt von

Michael Samulowitz

Tag des Rigorosum: 15. Juli 2002

1. Berichterstatter: **Prof. Dr. Linnhoff-Popien**, Universität München
2. Berichterstatter: **Prof. Dr. Wirsing**, Universität München

Zusammenfassung

Die vorliegende Arbeit präsentiert einen Ansatz zur Spezifikation und Implementierung von kontextadaptiven Anwendungen in einer Ubiquitous Computing Umgebung. Grundlegend ist dabei das Konzept der *kontextadaptiven Dienstnutzung*, die sowohl die kontextadaptive Selektion als auch Ausführung von Diensten umfasst.

Die kontextadaptive *Selektion* erweitert grundlegende Techniken der Dienstvermittlung insofern, dass ein Matching nicht ausschließlich durch die Spezifikation von gewünschten Dienstattributen erfolgt, sondern auch Kontextinformationen Berücksichtigung finden. Die *Ausführung* eines Dienstes kann ebenfalls an kontextuelle Bedingungen geknüpft werden. Eine realisierte Kombination von kontextadaptiver Selektion und Ausführung ermöglicht eine sowohl personalisierte als auch situationsbezogene Bereitstellung von Diensten.

Kern der kontextadaptiven Dienstnutzung ist dabei ein *Datenzentrisches Protokoll*, welches die Weiterleitung (*Routing*) von Anwendungsdaten anhand kontextueller Einschränkungen erlaubt. Dieser Ansatz gestattet neben der kontextadaptiven Nutzung individueller Dienste auch die spontane Komposition von Diensten in einer Ubiquitous Computing Umgebung.

Ferner wird ein Konzept zur dynamischen Rollenverwaltung für Endgeräte in einer Ubiquitous Computing Umgebung entwickelt und ein Verfahren zur Konstruktion von Kontextinformationen innerhalb eines Ad-hoc-Sensornetzwerks vorgestellt.

Summary

One of the main issues dealt within this thesis is the design and implementation of an approach enabling context-aware service provision for ubiquitous computing environments.

Context-aware service provision implies the *selection* and *execution* of services based on contextual constraints. The notion of context-aware service selection augments classical approaches for service discovery by taking into account arbitrary contextual constraints, instead of solely relying on required service attributes. The execution of services may be controlled by contextual constraints, too. As shown in this thesis, the combined use of context-aware selection and execution allows providing a situation-aware and personalized provision of services.

Context-aware service use is implemented by a so-called *data-centric protocol*, which allows routing of application data based on contextual constraints. Besides, this approach facilitates the spontaneous composition of discovered services.

In addition a concept for managing roles among information appliances is proposed. Finally, this thesis presents a novel scheme for gaining context information using ad-hoc-sensor-networks.

Abbildungsverzeichnis

Kapitel 1

Einleitung

Ubiquitous Computing [Weis 93a, Weis 93b] steht für eine neue Qualität der Nutzung von Computer- und Informationstechnologien. Charakterisiert wird dieser Trend durch die allgegenwärtige Einbettung von Computern in alltägliche Umgebungen als eine Konsequenz der fortschreitenden Miniaturisierung von Komponenten, sowie durch die Realisierung durchgängiger Vernetzung im Zuge der Weiterentwicklung und Integration von diversen Netzwerktechnologien [SG 01].

Abseits von technologischen Ansprüchen steht Ubiquitous Computing für eine neue Vorstellung zur Nutzung von Informationstechnologien. Insbesondere soll die oft bemängelte Nutzungskomplexität von Computersystemen beseitigt werden. In diesem Kontext wurde die Vision formuliert, dass Computersysteme einen integralen Bestandteil der Umwelt bilden und ähnlich einfach zu handhaben sein sollen wie alltäglich gebrauchte Gegenstände. Dieser Aspekt wird als *Invisibilität* eines Computersystems [Nor 98] bezeichnet.

Im Rahmen der hier vorgestellten Arbeit werden Ubiquitous Computing Systeme primär als *Multidienst-Umgebungen* verstanden. Es wird davon ausgegangen, dass sich mobile Benutzer in Diensträumen bewegen, in denen die von lokalen Ressourcen und Eingebetteten Systemen [BW 00] angebotenen Dienste spontan genutzt werden können. Multidienst-Umgebungen sind mit einer Reihe von Forschungsaspekten verknüpft. Eine Herausforderung besteht in der Lokalisierung von Diensten in einer neu besuchten Umgebung. Diesem Anspruch werden so genannte Dienstvermittlungsprotokolle gerecht, wie etwa JINI [JINI] oder das *Service Location Protocol* (SLP) [Gutt 99]. Eine weitere Herausforderung betrifft die Beschreibung von Diensten, denn ein neu lokalisierter Dienst kann nur dann benutzt oder mit anderen Diensten kombiniert werden, wenn dessen Dienstbeschreibung auch interpretiert werden kann. Dieser Gesichtspunkt führt zum Konzept der abstrakten Schnittstellenbeschreibung [HK 99a, BBG$^+$ 00]. Eine dritte Herausforderung – und Gegenstand dieser Arbeit – betrifft die *kontextadaptive*[1] Dienstnutzung.

Das Prinzip *Kontextadaptivität* [DeyAb 00, Nel 98, SAW 94] bildet den bemerkenswerten Ansatz, dem oben erwähnten Anspruch der einfachen Benutz-

[1] engl. *context-aware*

barkeit zu entsprechen. Es ermöglicht die Entwicklung von Anwendungen, die sich implizit der jeweiligen Nutzungssituation anpassen. Die Arbeit präsentiert neben einer neuen Konzeption zur kontextadaptiven Dienstnutzung auch Ansätze zur spontanen Komposition von Diensten einer Ubiquitous Computing Umgebung.

Die folgenden Abschnitte skizzieren Struktur und Inhalt der vorliegenden Arbeit. Kapitel 2 vermittelt insbesondere Grundlagenwissen im Forschungsbereich Ubiquitous Computing. Die Einführung in dieses breite Themenfeld erfolgt über einen *top-down* Ansatz. Zunächst werden generelle Merkmale eines Ubiquitous Computing Systems herausgestellt. Dazu gehören: *einfache Bedienbarkeit* (2.1.1) von Geräten eines Ubiquitous Computing Systems, *Diversifikation* und die damit verbundene *Heterogenität* (2.1.2) und schließlich die Möglichkeit zur *spontanen Vernetzung* (2.1.3) von Endgeräten. Im Anschluss erfolgt eine Untersuchung des Ubiquitous Computing Paradigmas im Forschungskontext. Es werden zunächst Bezüge zu fundierenden Forschungsbereichen hergestellt. Da der Anspruch der Distribution von Systemkomponenten inhärent mit dem Verständnis von Ubiquitous Computing Umgebungen verbunden ist, wird zunächst die Relation zum Forschungsfeld der *Verteilten Systeme* [W 98] untersucht. Nutzer einer Ubiquitous Computing Umgebung sind insbesondere *mobile* Nutzer („*The world is not a desktop*“ [Weis 94]). Aus diesem Grund bildet der Arbeitsbereich Mobile Computing (2.2.2) einen essentiellen Aspekt im Rahmen der hier angestellten Untersuchung. Da sich die Idee des Ubiquitous Computing weder durch Konzepte des Bereichs Verteilten Systeme noch durch Konzepte des Bereichs Mobile Computing hinreichend erfassen lässt, erfolgt in Ergänzung eine Erörterung der Forschungsrichtung *Smart Spaces* [AS 00]. Oft wird dieser Begriff synonym mit dem Begriff Ubiquitous Computing verwendet. Hier soll er jedoch dazu dienen, die Gesichtspunkte hervorzuheben, welche nicht durch Arbeiten in den traditionellen Bereichen der Verteilten Systeme bzw. Mobile Computing abgedeckt werden. Es ist zu beachten, dass die Darstellung der oben genannten Forschungsfelder akzentuiert erfolgt, d.h. immer in Hinblick auf das Themengebiet dieser Arbeit. Daher erfolgt die Darstellung der Teilgebiete *Kontextadaptivität* (2.3), *Dienstvermittlung* (2.4) und *lokale Netzwerktechnologien* (2.5) in gesonderten Abschnitten, da sie für die in den folgenden Kapiteln beschriebenen Konzepte von tragender Bedeutung sind.

In Kapitel 3 wird ein grundlegend neuer Ansatz präsentiert, welcher die *kontextadaptive Dienstnutzung* in Ubiquitous Computing Umgebungen ermöglicht. Insbesondere wird ein Verfahren vorgestellt, welches die kontextadaptive *Selektion* und *Ausführung* von Diensten in einem Verteilten System erlaubt. Als Resultat einer Anforderungsanalyse wird ein Architekturvorschlag – *Context-Aware Packets Enabling Ubiquitous Services* (CAPEUS) – vorgestellt. Innerhalb dieses Ansatzes erfolgt die Vermittlung und Nutzung von Diensten basierend auf einem assoziativen Kommunikations-protokoll. Zum Datenaustausch dient ein uniformes Datenformat, das als *Context-*

Aware Packet bezeichnet wird. Es dient der Vermittlung von Dienstanforderungen und assoziierteren Applikationsdaten. Die Weiterleitung dieser Dateneinheiten erfolgt anhand kontextueller Einschränkungen (*Context Constaints*). Das Kapitel endet mit der Beschreibung einer prototypischen Implementierung und einer Bewertung des Ansatzes.

In Bezug auf die Ergebnisse aus Kapitel 3 wird in Kapitel 4 ein neues Konzept zur dynamischen Rollenverwaltung für Endgeräte in einer Ubiquitous Computing Umgebung entwickelt. Individuellen Geräten können so genannte *Interaktionsmuster* zugewiesen werden. Diese bestimmen das Verhalten von Geräten, indem sie die Interaktionen zwischen Geräten bez. kontextueller Einschränkungen vorschreiben. Zugewiesene Interaktionsmuster können zu jeder Zeit manipuliert oder auch gelöscht werden. Auf diese Weise ist es möglich, das Systemverhalten anzupassen, ohne unterliegende Softwarekomponenten modifizieren zu müssen. Schließlich wird ein Tool vorgestellt, das den Benutzer bei der Erstellung von Interaktionsmustern und deren Distribution auf Endgeräte unterstützt.

Während bei den in Kapitel 3 und 4 untersuchten Systemen insbesondere der Nutzen von Kontextinformationen im Vordergrund steht, soll in Kapitel 5 die Konstruktion und Komposition von Kontextinformationen untersucht werden. In Berücksichtigung der Tatsache, dass die Gewinnung von Kontextinformationen innerhalb einer Ubiquitous Computing Umgebung an mobile Entitäten gebunden ist, wurde dieser Prozess in Bezug zu Ad-hoc-Sensornetzwerken [ASSC 02] untersucht. Der Vorgang der Konstruktion von Kontextinformationen wird als eine Anwendung der Datenfusion [A 92] verstanden. Daher erfolgt zunächst eine Darstellung von prinzipiellen Techniken der Datenfusion. Daraufhin werden diese Erkenntnisse angewendet, um die Gewinnung von Kontextinformationen in Ad-hoc-Sensornetzwerken zu ermöglichen. Die Datenfusion in Sensornetzwerken ist mit einigen Besonderheiten verbunden. So ist insbesondere zu beachten, dass die Topologie eines Sensornetzwerkes möglicherweise ständigen Veränderungen unterliegt. Es muss insbesondere der Fall von ausfallenden bzw. nicht mehr erreichbaren Sensor-Knoten berücksichtigt werden. Ein daten-zentrisches Protokoll – genant *Smart Stack Routing* (SSR) – wird präsentiert, welches die Konstruktion von Kontextinformationen in einer dezentralen, sich selbst organisierenden Netzwerkstruktur realisiert.

Die Ausarbeitung endet mit einer Zusammenfassung der erzielten Ergebnisse und vermittelt einen Überblick über zukünftige Forschungsziele bezüglich der behandelten Thematik.

Kapitel 2

Kontextadaptivität in Ubiquitous Computing Systemen

2.1 Das Ubiquitous Computing Paradigma

Gemeinhin kann man die Evolution der Computertechnologie in drei Phasen gliedern [Weis 91]: Anfangs wurde das Bild der Computernutzung noch durch einen zentralen Ansatz geprägt. Leistungsfähige Mainframes stellten ihre Ressourcen einer Vielzahl von Personen mittels individueller Terminals zur Verfügung. Anfang der 80er Jahre trat der PC seinen Siegeszug an. Von nun an standen einem einzelnen Nutzer die gesamten Ressourcen eines abgeschlossenen Computersystems individuell zur Verfügung. Und schließlich, in einer dritten Phase der Evolution, können einem einzelnen Nutzer eine Vielzahl von Computersystemen zugeordnet werden. Diese kann der Nutzer dann zur Bewerkstelligung einer Aufgabe verwenden. Der Trend wird als *Ubiquitous* [Weis 93] oder *Pervasive Computing* [HMNS 01] bezeichnet; siehe auch Abbildung 2-1.

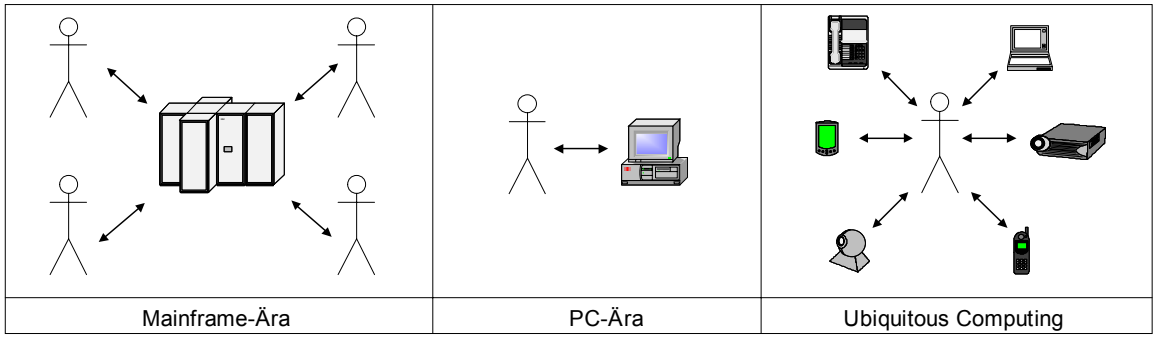

Abbildung 2-1: Evolution der Computertechnologie

Zu den Computersystemen oder Geräten, die einer Person ihre Dienste zur Verfügung stellen, gehören nicht nur mobile Geräte, wie etwa Laptops, PDA, Mobilfunktelefone und ähnliche. Auch Geräte, die sich in der Umgebung einer Person befinden, werden im Ubiquitous Computing Paradigma miteinbezogen. Dabei handelt es sich vor allem um so genannte *Eingebettete Systeme* [BW 00], wie man sie in Intelligenten Häusern oder Büroumgebungen findet. Ein Kon-

ferenzraum beispielsweise kann neben einem elektronischen Präsentationssystem auch ein Videokonferenzsystem zur Benutzung anbieten.

Ein Ubiquitous Computing System gestattet also die Nutzung von ggf. räumlich verteilten Ressourcen zur Erledigung einer bestimmten Aufgabe. Der Nutzungsprozess kann es erforderlich machen, dass eine Menge von Ressourcen gleichzeitig genutzt wird oder auch zeitlich versetzt. In beiden Fällen kann der Austausch von Informationen (z.B. Statusinformationen) zwischen Ressourcen erforderlich sein. Daher spielt aus technischer Sicht die *Kommunikation* bzw. *Koordination* zwischen diesen Ressourcen eine zentrale Rolle im Forschungsbereich Ubiquitous Computing.

Ubiquitous Computing Systeme erlauben die Realisierung neuer Anwendungsszenarien. Insbesondere kommen sie den Anforderungen mobiler Nutzer entgegen, indem sie die Nutzung von Ressourcen ortsunabhängig machen. So kann ein Nutzer in vielfältigen Lebensbereichen unterstützt werden, ohne an ein konkretes Computer-System gebunden zu sein (z.B. PC). Zudem soll die Nutzung von Ressourcen, wie etwa computerbasierten Diensten, intuitiv erfolgen. Im Sinne von [Weis 94] soll der Einsatz von computerbasierten Systemen so selbstverständlich wie die Nutzung von Alltagsgegenständen erfolgen.

Zusammenfassend kann festgestellt werden, dass Ubiquitous Computing für eine neue Qualität in der Anwendung von Computer- und Informationstechnologien steht. Gekennzeichnet wird diese Entwicklung durch die fortschreitende Miniaturisierung von Komponenten, sowie die Realisierung durchgängiger Vernetzung im Zuge der Weiterentwicklung und Integration verschiedener Netzwerktechnologien.

Die folgenden Teilabschnitte charakterisieren das Konzept Ubiquitous Computing bez. nachfolgender Gesichtspunkte:

- *Einfache Bedienbarkeit*
- *Diversifikation und Heterogenität*
- *Spontane Vernetzung*

2.1.1 Einfache Bedienbarkeit

Inhärent verbunden mit dem Konzept Ubiquitous Computing ist die Anforderung der *einfachen Bedienbarkeit*. Dieser Aspekt lässt sich bez. verschiedener Gesichtspunkte analysieren. Zum einen kann die Gestaltung der Mensch-Maschine Schnittstelle (*Human Computer Interaction*) [Sto 95] im Vordergrund stehen. Dann wird die Zielsetzung verfolgt, die Interaktionen mit einem Endgerät oder auch mit computerbasierten Diensten einer Umgebung möglichst intuitiv und einfach zu gestalten. Ein Ansatz, diesem Anspruch gerecht zu werden, ist, die Schaffung möglichst einheitlicher Schnittstellen für verschiedene Anwendungsszenarien bereitzustellen, damit der Wiedererkennungswert möglichst hoch ist und

der erforderliche Lernaufwand seitens des Nutzers gering gehalten werden kann. Denkt man beispielsweise an grafische Benutzeroberflächen, so wäre es wünschenswert, dass grafische Bedienelemente auch innerhalb einer heterogenen Geräte- und Dienstlandschaft über ein einheitliches Aussehen bzw. über dieselbe Funktion verfügen.

Ein weiterer Ansatz, um der Forderung der einfachen Bedienbarkeit zu entsprechen, bezieht sich auf die Berücksichtigung des Kontextes bzw. der Situation eines Benutzers. Dieser Gesichtspunkt bildet bei der Entwicklung von Ubiquitous Computing Systemen einen wesentlichen Beitrag und wird im Rahmen dieser Arbeit als *Kontextadaptivität* [AAHL+ 97, BroBo 97] bezeichnet. Orthogonal zum Aspekt der allgegenwärtigen[2] Verfügbarkeit von computerbasierten Diensten ermöglicht das Konzept der Kontextadaptivität, die konkrete Nutzung eines Dienstes an die lokale Situation automatisch anzupassen. Durch Anwendung dieses Prinzips ist es nicht nur möglich, die Nutzung von Diensten intuitiver zu gestalten, sondern insgesamt die Zahl der vom Benutzer zu leistenden Interaktionen/Eingaben zu verringern, da Informationen bez. der Situation des Nutzers implizit mit einfließen. Das Konzept der Kontextadaptivität lässt sich an einem einfachen Beispiel demonstrieren: Möchte ein Benutzer ein Dokument, das sich auf seinem PDA befindet, ausdrucken, so soll für diesen Auftrag ein Drucker ausgewählt werden, der sich möglichst in der direkten Umgebung des Nutzers befindet. Wie an diesem Beispiel zu erkennen ist, erfordert die Realisierung einer kontextadaptiven Anwendung, dass das System über die Fähigkeit verfügt, Umweltdaten zu erfassen. Im betrachteten Beispiel müsste der Ort des Benutzers feststellbar sein, um einen Drucker in dessen Nähe auswählen zu können. Die Erfassung von Umweltdaten erfolgt gewöhnlich über den Einsatz von Sensorelementen (siehe 2.3).

Da das Konzept der Kontextadaptivität wesentlich für den Inhalt dieser Arbeit ist, wird diese Thematik gesondert in Abschnitt 2.3 behandelt.

2.1.2 Diversifikation und Heterogenität

Wie bereits angedeutet, werden Ubiquitous Computing Applikationen von einer Menge unterschiedlicher Geräte bzw. Ressourcen ausgeführt. Diese Geräte bilden ein heterogenes System bez. diverser Aspekte. Zum einen unterscheiden sich die individuellen Geräte durch die von ihnen angebotenen Funktionen, die oft auf einen bestimmten Anwendungsbereich abgestimmt sind. Diese Diversifikation von Endgeräten im Gegensatz zu Multifunktionsgeräten, wie etwa dem PC, kann den Gebrauch von solchen Geräten merkbar vereinfachen [Nor 98]; siehe Abschnitt 2.1.1.

[2] ubiquitous = allgegenwärtig

Im Zuge der funktionellen Diversifikation differenzieren sich Geräte auch durch verwendete Betriebssysteme, vorhandene Hardware-Ressourcen (Speicher, CPU etc.), Kommunikationsmöglichkeiten (z.B. GSM, Bluetooth) und unterstützten Datenformaten.

Die geforderte Kommunikation bzw. Interoperabilität von multiplen Geräten kann durch diese plattformspezifischen Eigenschaften erschwert werden [HMNS 01]. So kann die netzwerkbasierte Kommunikation zwischen Geräten nur dann stattfinden, wenn diese kompatible Standards unterstützen. Dieses Problem wird durch neue, offene Standards wie WAP [WAP 02], Bluetooth [Blue] oder IrDA [IrDa] adressiert. Diese Standards definieren sowohl die erforderlichen Daten-Kommunikationsprotokolle, als auch die physikalischen Netzwerkschnittstellen.

Aber nicht nur die Vernetzung von Ressourcen einer Ubiquitous Computing Umgebung muss beachtet werden, sondern *Applikationen* und *Daten* müssen auch über verschiedene Gerätearchitekturen hinweg austauschbar sein. Ein Konzept zur Realisierung plattformunabhängiger Applikationen ist *Mobile Code* [BC 95]. Java [JAVA] ist ein Mobile-Code-Konzept, welches große Verbreitung gefunden hat. Java ist eine Programmiersprache, die einen plattformneutralen Code erzeugt, welcher von einem entsprechenden Interpreter (*Java Virtual Machine* [LY 99]) auf der Zielplattform ausgeführt werden kann. Java wird von einer Vielzahl von Gerätetypen unterstützt, so etwa von Smart Cards [SmartC], PDAs und Mobilfunktelefonen. Dieser Aspekt ist von großer Relevanz hinsichtlich der ausgeprägten Heterogenität von Ubiquitous Computing Systemen.

Eine Möglichkeit zur Vermittlung von Daten in einer heterogenen Systemumgebung ist die Formatsstandardisierung von Daten bzw. Metadaten. Diesbezüglich hat sich XML [Bos 97, GP 00] und RDF [RDF 99] als de facto Standard etabliert. Ein Beispiel dafür ist SyncML [Sync 02]. SyncML definiert basierend auf XML ein Protokoll zum Abgleich zwischen Datenbeständen von mobilen Endgeräten, so dass z.B. die Adressbestände zwischen einem Mobilfunktelefon und einem PDA automatisch abgeglichen werden können.

2.1.3 Spontane Vernetzung

Ein auszeichnendes Merkmal von Ubiquitous Computing Systemen ist die Möglichkeit der spontanen Kooperation mehrerer Geräte (oder Ressourcen) zur Erfüllung einer bestimmten Aufgabe. Dieser Vorgang wird hier als *Spontane Vernetzung* bezeichnet. Da Kooperationsbeziehungen möglicherweise sehr dynamisch sind, muss die Vernetzung der Geräte möglichst einfach und automatisch durchführbar sein.

Spontane Vernetzung ist ein Prozess, der sich über mehre Ebenen der Kommunikationshierarchie in einem Verteilten System [Tan 96] erstreckt [Kur 00]; siehe auch Abbildung 2-2:

- *Netzwerk-Ebene*
- *Infrastruktur-Ebene*
- *Dienst-Ebene*

Die **Netzwerk-Ebene** ist für die Kommunikation zwischen Dienstendpunkten zuständig. Die Spontane Vernetzung fordert hier die automatische Verhandlung von Kommunikationsparametern. Bei TCP/IP-Netzen [Stev 94] sind das beispielsweise IP-Adressen, welche den einzelnen Dienstendpunkten mittels DHCP [DL] zugewiesen werden müssen.

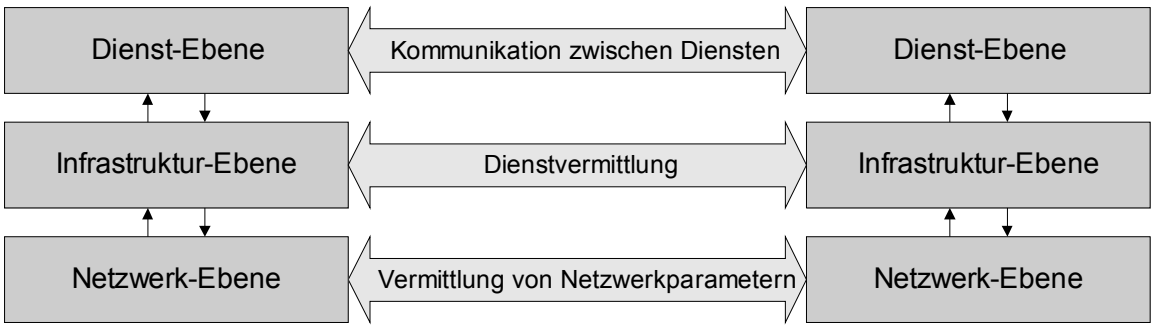

Abbildung 2-2: Kommunikationshierarchie zur Realisierung Spontaner Vernetzung

Im Allgemeinen wird die spontane Kommunikation zwischen Dienstendpunkten durch so genannte *Zero-Configuration-Netzwerke* [ZERO 02] realisiert. Diese Klasse von Netzwerken zeichnet sich dadurch aus, dass keine dedizierte Instanz zur Konfiguration bzw. Administration erforderlich ist. In diesem Zusammenhang spricht man auch von sich selbst konfigurierenden Netzen. Aus diesen Gründen bieten Zero-Configuration-Netzwerke einen großen Vorteil im Einsatz von lokalen Netzwerken im Home- und Office-Netzen, wie sie in Ubiquitous Computing Umgebungen verwendet werden.

Natürlich muss beachtet werden, dass der Prozess der Selbstkonfiguration nicht alle Parameter der Netznutzung in allgemeingültiger Weise bestimmen kann, z.B. im Bereich der Netzwerksicherheit. Es dominieren z.B. Aspekte der Netzwerksicherheit die Vorzüge der Selbstkonfiguration.

Einige der hier angeführten Punkte können direkt von der eingesetzten Netzwerktechnologie in einem Ubiquitous Computing System implementiert sein. So beispielsweise Bluetooth [Blue], wobei es sich um ein drahtloses LAN [Tan 96] mit kurzer Reichweite handelt (10m). Bluetooth erlaubt den spontanen Aufbau von Punkt-zu-Punkt bzw. Punkt-zu-Multipunkt-Verbindungen zwischen multiplen Geräten.

Eine Übersicht der im Bereich Ubiquitous Computing relevanten Netzwerktechnologien findet sich in Abschnitt 2.5.

Die **Infrastruktur-Ebene** realisiert den Austausch und die Vermittlung von Informationen, welche eine Kooperation von multiplen Ressourcen unterstützen. Dieser Vorgang wird auch als *Dienstvermittlung* (siehe 2.4) bezeichnet. Die Dienstvermittlung dient der Kommunikation von Diensteigenschaften zwischen den Ressourcen eines Systems. Analog zum Gebrauch einer Datenbank kann eine Ressource nach Diensten suchen, welche durch andere Ressourcen angeboten werden. Konnte ein Dienst mit den geforderten Attributen lokalisiert werden, kann dieser mittels den angebotenen Funktionen der Infrastruktur-Ebene benutzt werden.

In diesem Zusammenhang ist die Art der Beschreibung eines *Dienstes* von großer Relevanz. Bei der Beschreibung eines Dienstes kann man grundsätzlich zwei Methoden unterscheiden: Die Dienste sind im Voraus, d.h. zur Entwicklungszeit der dienstnutzenden Ressource, bekannt (*Closed World Assumption*) oder nicht (*Open World Assumption*). Im Falle der *Open World Assumption* muss eine semantische Beschreibung eines Dienstes gewählt werden, damit im Voraus unbekannte Kombinationen ermöglicht werden können. Die Dienstvermittlung basierend auf semantischen Informationen ist ein schwieriges Problem und wird in einigen Projekten mittels künstlicher Intelligenz behandelt, siehe AI-Trader [MGKR$^+$ 96] oder Ronin Agent Framework [Ronin]. Im Falle der *Closed World Assumption* reicht es aus, Dienste mittels syntaktischer Repräsentationen zu identifizieren. Dieses Verfahren wird bei etablierten Trader-Architekturen, wie etwa CORBA [CORBA] eingesetzt. Das Auswahlverfahren ist in diesem Fall einfacher, da die gewünschten Dienstkandidaten durch syntaktische Vergleichsoperationen bestimmt werden können.

Die **Dienst-Ebene** schließlich behandelt die eigentliche Nutzung eines durch die Infrastruktur-Ebene vermittelten Dienstes. Der Prozess der Dienstnutzung kann in manchen Fällen durch Vorgaben der Infrastruktur-Ebene mitbestimmt werden.

Vorwiegend determiniert die Dienst-Ebene die Protokolle, die zur Kommunikation zwischen einem Dienstnutzer und einem Diensterbringer eingesetzt werden. Das schließt z.B. auch den Methodenaufruf eines entfernten Dienstobjektes mit ein (*remote procedure call*).

Eine ausführliche Darstellung der Dienstvermittlung und -nutzung findet sich in Abschnitt 2.4.

2.2 Ubiquitous Computing Systeme im Forschungskontext

Der vorige Abschnitt hat das Thema Ubiquitous Computing vorgestellt. Dabei wurden Grundzüge der Dienstvermittlung, des Daten- und Code-Austausches erläutert. Hier soll nun der Bezug von Ubiquitous Computing bez. anderer

Forschungsrichtungen analysiert werden. Insbesondere sollen Relationen zu Konzepten aus den verwandten Themenbereichen *Verteilte Systeme* und *Mobile Computing* hergestellt werden. Es ist zu beachten, dass hier unter dem Oberbegriff Ubiquitous Computing einige Forschungsbereiche zusammengefasst wurden. Dazu gehören unter anderen: das eigentliche *Ubiquitous Computing* [Weis 92], *Invisible Computing* [Nor 98], und *Sentient Computing* [Sentient].

Abbildung 2-3 zeigt eine Darstellung des Forschungsbereichs Ubiquitous Computing mit korrelierenden Forschungszweigen, analog zu Satyanaraynan [Saty 01]. Liest man die Abbildung von links nach rechts, so kann man die Probleme ablesen, welche zur Verwirklichung eines Ubiquitous Computing Systems zu lösen sind. Offensichtlich handelt es sich bei Ubiquitous Computing Systemen um Verteilte Systeme, so dass viele Erkenntnisse aus diesem Bereich direkt angewandt werden können. Dasselbe gilt für Erfahrungen aus dem Bereich Mobile Computing. In der Abbildung werden Multiplikationssymbole verwendet, um anzudeuten, dass die Komplexitätszunahme nicht additiv sondern multiplikativ zunimmt. Wie bereits in vielen Arbeiten bestätigt, ist die Implementierung eines Ubiquitous Computing Systems um vieles komplexer als die eines Verteilten Systems mit vergleichbaren Anforderungen bez. Robustheit und Ausgereiftheit.

Es ist zu beachten, dass die Abbildung logische Relationen zwischen den individuellen Forschungsbereichen herstellt, nicht historische. Obgleich der angedeutete Entwicklungsverlauf in mancher Hinsicht den historischen Begebenheiten entspricht. Es gab jedoch Forschungsansätze, welche dem Bereich Ubiquitous Computing angehören, aber schon sehr früh unternommen worden sind. So hat beispielsweise die Forschung im Bereich Smart Spaces [AS 00] Anfang der 90er begonnen und entwickelte sich relativ unabhängig von Arbeiten des Mobile Computing.

Die folgenden Abschnitte vermitteln Aspekte der Bereiche Verteilte Systeme und Mobile Computing. Zudem folgt eine Darstellung zu dem Bereich Smart Spaces. Der in Abbildung 2-3 erwähnte Begriff der *Invisibility* wurde bereits in 2.1.1 unter dem Aspekt der einfachen Bedienbarkeit von Ubiquitous Computing Systemen erläutert. Das Konzept der *Kontextadaptivität* ist ein zentraler Gesichtspunkt dieser Arbeit und wird in 2.3 gesondert untersucht.

Da sich einige dieser Bereiche überschneiden, wird bei der Darstellung kein Wert auf Vollständigkeit gelegt. Ziel ist es, das vielfältige Thema Ubiquitous Computing aus verschiedenen Perspektiven zu erforschen.

2.2.1 Verteilte Systeme

Das Forschungsfeld Verteilter Systeme entstand durch das Aufkommen des PCs und die Möglichkeit, diese durch *Local Area Networks* (LANs) miteinander zu vernetzen. Die zwischen Mitte der 70er und Anfang der 90er Jahre geleistete Forschungsarbeit schaffte eine Konzeption und algorithmische Basis, welche sich

für die Kommunikation von verteilten Rechenknoten als grundlegend erwies [Saty 01]. Da es sich bei Ubiquitous Computing Systemen um immanent verteilte Anwendungen handelt, werden hier gemäß Abbildung 2-3 einige essentielle Aspekte erläutert. Die Darstellung erhebt keinen Anspruch auf Vollständigkeit, es werden Gesichtspunkte der hier vorgestellten Forschungsarbeit untersucht.

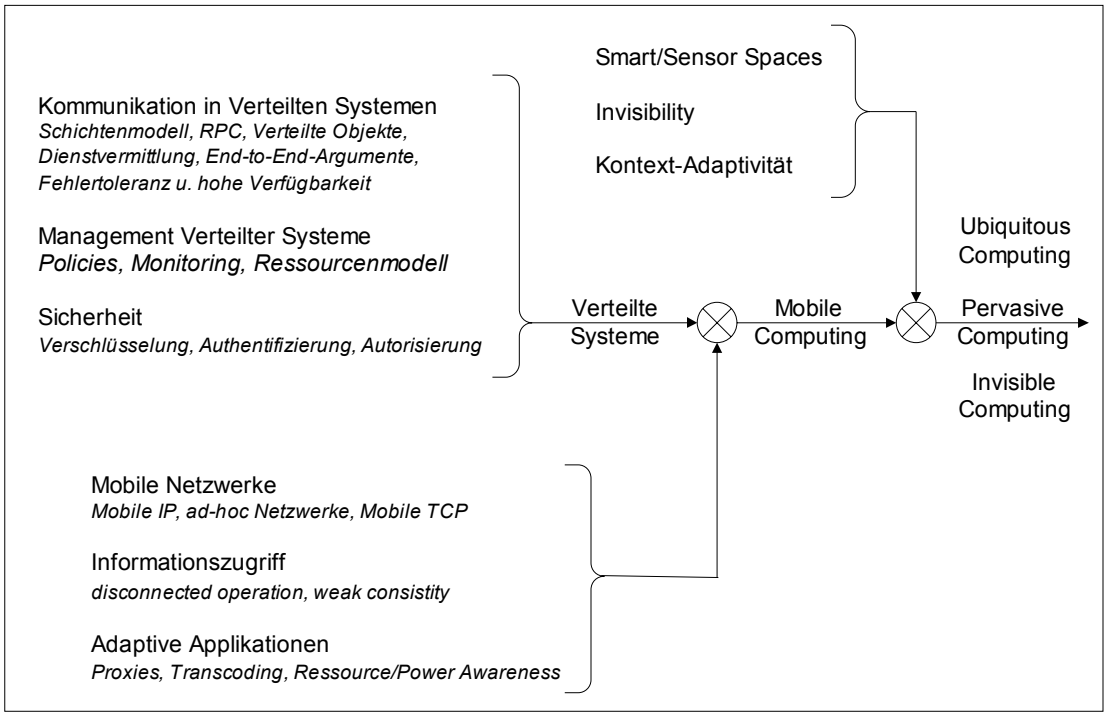

Abbildung 2-3: Ubiquitous Computing im Forschungskontext

Kommunikation in Verteilten Systemen

Ohne auf konkrete Ansätze zur Realisierung Verteilter Systeme wie etwa CORBA [L 98, OMG 99] einzugehen, soll hier die Basisproblematik dieses Forschungszweiges untersucht werden. Eine ausführliche Darstellung der Thematik Verteilter Systeme findet sich u.a. in folgenden Lehrbüchern: [W 98], [CDK 02].

Im folgenden sollen die Grundzüge und Anforderungen Verteilter Systeme aus der Perspektive der Softwaretechnik untersucht werden. Zu diesem Zweck wird ein Designmuster [GHJV 95] erläutert, welches für die Gestaltung eines Verteilten Systems wesentlich ist. Dabei handelt es sich um das Designmuster *Half Object Plus Protocol* [M 95], welches im nächsten Abschnitt erläutert wird.

Half Object Plus Protocol. Das *Half Object Plus Protocol* (HOPP)-Designmuster kann immer dann angewandt werden, wenn man im Rahmen einer Analyse, wie etwa der *Object Oriented Analysis* (OOA), die konstituierenden Objekte eines

Systems und deren Assoziationen bestimmt hat. Das HOPP-Designmuster ist hilfreich bei der Distribution von Objekten über verteilte Adressräume.

Eine der Herausforderungen bei der Distribution von Objekten über getrennte Adressräume bilden Fälle, in denen Objekte in verschiedenen Kontexten (*computing context*) zur Anwendung kommen – wenn z.B. auf ein Objekt von unterschiedlichen Adressräumen zugegriffen werden soll. Damit das Design verteilter Systemarchitekturen vereinfacht werden kann, soll von dem Unterschied der Anwendungen abstrahiert werden, die in singulären bzw. in multiplen Adressräumen ausgeführt werden.

Die Distribution von Objekten auf separierte Adressräume/Ressourcen wird u.a. durch folgende Faktoren beeinflusst: *Komplexität*, *Programmierumgebung*, *Verteilung der Ressourcen*, *Fehlertoleranz*, *Verfügbarkeit*, *Kosten* und *Performanz*.

In manchen Fällen kann die Dekomposition in einer Menge von Objekten resultieren, die jeweils in einem beliebigen Adressraum ausgeführt werden können. In anderen Fällen ist die Ausführung eines Objekts nur in bestimmten Adreßbereichen möglich, da die Ausführung an bestimmte Hardware-Ressourcen gebunden ist, z.B. an Sensoren oder Speicheraggregate.

Die Objektdekomposition wird erschwert, wenn eine gemeinsame Funktionalität in mehreren Adressräumen erforderlich ist oder wenn zur Ausführung eines Objekts Daten anderer Adressräume erforderlich sind. In diesem Fall müssen Objekte in der Lage sein, einkommende Anfragen anderer Objekte aus anderen Adressräumen zu verarbeiten.

In diesen Fällen kann eine Aufgabe nicht innerhalb eines einzigen Adressraumes erfüllt werden. Deshalb muss die Funktion eines Objekts auf mehrere Adressräume aufgeteilt werden. Die in Frage kommenden Objekte müssen in Teilobjekte zerlegt werden, die sich in unterschiedlichen Adressräumen befinden. Die Zerlegung von Objekten führt offensichtlich zu einer höheren Komplexität, da die Ausführung eines einzigen Objektes einfacher zu realisieren ist, als entsprechende Teilobjekte, die über ein Protokoll miteinander kommunizieren.

Kommt es zu hochfrequenten Interaktionen zwischen den Teilobjekten, so können die Kosten (Ausführungszeit, Verzögerung), die zur Generierung von Nachrichtobjekten bzw. deren Versendung erforderlich sind, ein inakzeptables Maß erreichen.

Die Splittung eines Objekts in zwei äquivalente Teilobjekte erlaubt es beiden Teilobjekten, auf viele lokale Anfragen direkt zu antworten, ohne dass dabei die Beanspruchung eines Dienstes des anderen Adressraums/Teilobjektes erforderlich wäre. Jedoch kann diese Vorgehensweise zu duplizierten Funktionalitäten führen. Ferner kann es erforderlich sein, dass die Datenbestände der beiden Teilobjekte synchronisiert werden müssen. Die Schnittstelle des Objektes ist unabhängig von dem eingesetzten Protokoll zwischen den beiden Teilobjekten.

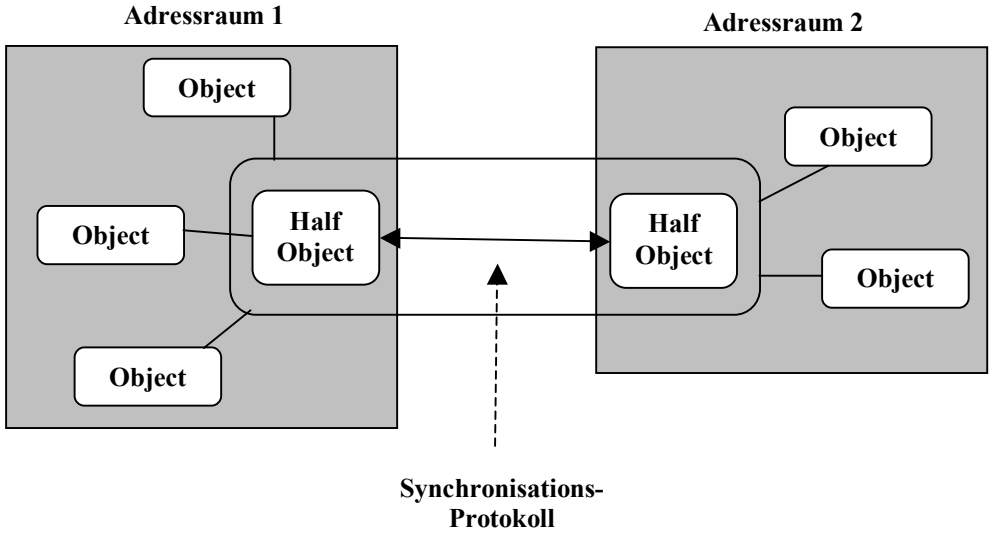

Abbildung 2-4: Half Object Plus Protocol (HOPP)

Als Konsequenz der oben formulierten Erwägungen besteht die Kernidee des HOPP-Desigmusters darin, ein einzelnes Objekt in zwei abhängige Teilobjekte (*Half Objects*) zu unterteilen, und zwar jedes in einem individuellen Adressraum, siehe auch Abbildung 2-4. Zur Kommunikation zwischen den Teilobjekten dient ein Kommunikationsprotokoll.

Das Designmuster sieht nun vor, dass innerhalb beider Adressräume die benötigten Funktionalitäten implementiert werden, die für die effektive Kommunikation mit den Objekten im selben Adressraum erforderlich sind. Schließlich wird ein Protokoll definiert, welches die Aktionen der Teilobjekte koordiniert, indem relevante Datenfragmente zwischen den Adressräumen ausgetauscht werden.

Management Verteilter Systeme

Stellvertretend für den vielfältigen Forschungsbereich des Managements Verteilter Systeme, wird hier insbesondere das *Policy-basierte Management* [S 94] betrachtet, da einige Ansätze dieses Forschungszweiges in die hier vorgestellte Konzeption der kontextadaptiven Dienstnutzung (siehe Kapitel 3) miteingeflossen sind.

Das generelle Konzept der *Policy* dient zur Definition von Regeln, welche das Verhalten eines Systems bez. bestimmter Aspekte steuern. Eine der wesentlichen Motivationen für das Policy-basierte Management ist die Möglichkeit, Policies zu jedem Zeitpunkt ändern zu können, ohne dass die unterliegenden System-

komponenten modifiziert werden müssen oder die assoziierte Anforderungsanalyse neu strukturiert werden muss.

Policies werden in einem breiten Spektrum von Anwendungsgebieten eingesetzt. So beispielsweise auch im Bereich der Computersicherheit. In diesem Fall beschränken oder erlauben Policies die Nutzung oder den Zugriff auf eine Ressource. Innerhalb von Verteilten Systemen dienen Policies dazu, das Verhalten von Systemkomponenten zu beeinflussen. Insbesondere kann das mögliche Verhalten einer Komponente eingeschränkt (*authorisation*) oder ein gewünschtes Verhalten vorgeschrieben werden (*obligation*). Die Ausführung von Policies ist i.d.R. an Bedingungen geknüpft, so dass es möglich ist, das Systemverhalten bez. dynamischer Parameter zu kontrollieren (*triggering*).

Insbesondere beim Entwurf adaptiver Systeme, wie sie hier untersucht werden, konnte die Idee des Policy-basierten Managements gewinnbringend eingesetzt werden. Im Rahmen dieser Arbeit wurde ein Ansatz entwickelt, der die Auswahl bzw. Ausführung von Diensten in einer Ubiquitous Computing Umgebung an kontextuelle Einschränkungen bindet. Die Repräsentation und Interpretation von diesen kontextuellen Einschränkungen wurde maßgeblich durch die Arbeiten im Policy-basierten Management beeinflusst, so insbesondere auch die Formulierung von *Trigger-Bedingungen*.

Sicherheit

Die für Verteilte Systeme entwickelten Konzepte zur Sicherheit können zum großen Teil auch in Ubiquitous Computing Systemen angewandt werden. Grundlegende Anforderungen an ein sicheres System sollen im folgenden erläutert werden.

Generell wird unter dem Begriff der Sicherheit ein unfangreicher Katalog unterschiedlicher Anforderungen und Maßnahmen verstanden, die zum Schutz von Hard- und Software vor böswilligem Zugriff, Gebrauch, Veränderung, Zerstörung oder Enthüllung dienen. Sicherheit bezieht sich dabei auf die Bereiche Benutzer, Daten, Kommunikation, Medien und den physikalischen Schutz von Computerinstallationen [T 93].

Drei zentrale Anforderungen an ein sicheres System illustriert Abbildung 2-5 [O 92, FH 92, K 95]:

- **Vertraulichkeit** (*confidentiality*) dient dem Ziel, den unbefugten Informationsgewinn (Verletzung des Datenschutzes) zu unterbinden. Die Zusicherung von Vertraulichkeit stellt also sicher, dass Informationen nur legimitierten Personen zur Kenntnis gelangen.
- **Integrität** (*integrity*): Sie gewährleistet, dass nur autorisierte Modifikationen an Daten vorgenommen werden können, um die Konsistenz der Datenbasis zu erhalten. Teilziele zur Gewährleistung von Integrität sind

der Schutz vor Modifikationen durch nichtautorisierte Nutzer, der Schutz vor inkorrekten Modifikationen durch autorisierte Nutzer, die Einhaltung der Konsistenz voneinander abhängiger Daten (innere Konsistenz) sowie die Korrektheit der Abbildung des Ausschnitts der realen Welt, der durch die Daten repräsentiert wird (äußere Konsistenz). Es muss also zugesichert werden, dass alle beabsichtigten und unbeabsichtigten Veränderungen von Daten ausgeschlossen werden, welche diese Daten verfälschen oder Anwendungen falsch, unvollständig oder nur vorgetäuscht ablaufen lassen.

- **Verfügbarkeit** (*availability*): Sie sichert zu, dass geschützte Funktionen oder Daten weder unbefugt gelöscht noch beeinträchtigt werden.

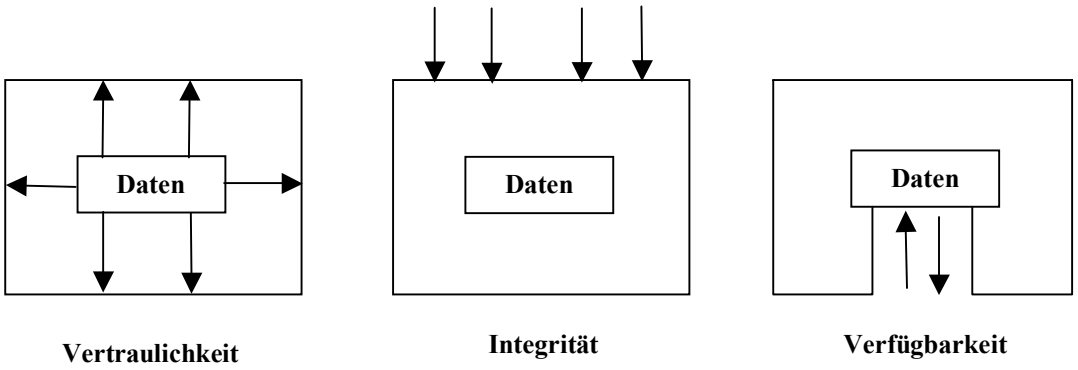

Abbildung 2-5: Elementare Sicherheitsanforderungen

Die Gewährleistung dieser Anforderungen umfasst eine Vielzahl bereits bestehender Methoden und Maßnahmen (Identifikation, Authentisierung, Protokollierung, etc.). Im Kontext von Ubiquitous Computing müssen diese Verfahren zunächst auf ihre Einsatzmöglichkeiten untersucht werden und ggf. angepasst werden. Eine einführende Analyse der Sicherheitsproblematik in Ubiquitous Computing Systemen kann in [RMW 99] gefunden werden.

2.2.2 Mobile Computing

Eine Vielzahl von Endgeräten in einer Ubiquitous Computing Umgebung sind mobil und können so an jedem Ort benutzt werden. Das Forschungsfeld Mobile Computing entstand in den frühen 90er Jahren parallel mit dem Aufkommen erster tragbarer Computer – wie etwa dem Laptop – und der Möglichkeit, diese jetzt vermittels drahtloser Netzwerktechnologien (z.B. WaveLan [AIR]) miteinander zu verbinden. Im Prinzip wurde der Forschungsbereich Verteilter Systeme um das Konzept des mobilen Klienten ausgedehnt. Auch wenn die Konzepte Verteilter Systeme hier angewandt werden konnten, wurden darüber hinaus neue Anforderungen identifiziert, welche die Entwicklung neuer Ansätze einforderten.

Zu den neu aufgedeckten Problemen in einer mobilen Umgebung gehören folgende Probleme: unvorhersagbare Änderungen in der Netzwerkqualität (z.B. verfügbare Bandbreite) und verminderte Robustheit von mobilen Elementen. Die lokale Ressourcen eines mobilen Endgerätes sind ferner beschränkt aufgrund von Einschränkungen bez. der Größe und Gewicht. Insbesondere spielt auch die Energieeffizienz (z.B. der Batterieverbrauch) eine zentrale Rolle [Saty 96].

Mobile Computing ist noch stets ein aktiver Forschungsbereich, und die vorläufigen Ergebnisse lassen sich in folgende Rubriken eingliedern:

- *Mobile Netzwerke*
- *Informationszugriff*
- *Adaptive Applikationen*

Das Thema Ortsabhängigkeit lässt sich auch dem Forschungskomplex Mobile Computing zuordnen. Es wurde jedoch im Rahmen von Ubiquitous Computing mit größerer Gewichtung behandelt. Zudem ist das Konzept der Ortsabhängigkeit ein zentraler Aspekt im Forschungsbereich der *Kontextadaptivität*, welcher in Abschnitt 2.3 dargelegt wird.

2.2.2.1 Mobile Netzwerke

Der Bereich Mobiler Netzwerke lässt sich in drei Unterbereiche teilen: 1. Systeme zur Mobilitätsunterstützung, 2. Ad-hoc-Protokolle und 3. Optimierung von Transportprotokollen bez. Anforderungen der Mobilität.

Befindet sich eine mobile Komponente in einem Verteilten System, so kann sich der Anschlusspunkt dieser Komponente ändern und damit u. U. auch die Netzwerkadresse. Um dennoch einen kontinuierlichen Datentransport zu gewährleisten, existieren Konzepte zur Mobilitätsunterstützung. Man unterscheidet in diesem Kontext zwischen zwei Varianten: Zum einem kann die Mobilitätsunterstützung auf Transportebene erfolgen, so etwa mittels Mobile IP [MIP], zum anderen kann sie auf der Anwendungsebene erfolgen. Beide Methoden haben je nach Anwendungsfall ihre Vor- und Nachteile.

Ad-hoc-Protokolle behandeln das Problem des Datentransports in Netzwerken, welche vorwiegend aus mobilen Netzwerknoten bestehen. Durch die Mobilität der einzelnen Knoten unterliegt die Netzwerktopologie stetigen Fluktuationen. Deshalb sind Routingprotokolle, die von einer statischen Netzwerktopologie ausgehen, hier nicht einsetzbar. Infolgedessen wurde eine neue Klasse von Routingprotokollen [RoTo 99] entwickelt, welche diesen Begebenheiten gerecht werden. Eine detaillierte Ausführung dieses Sachverhaltes findet sich in Kapitel 5.

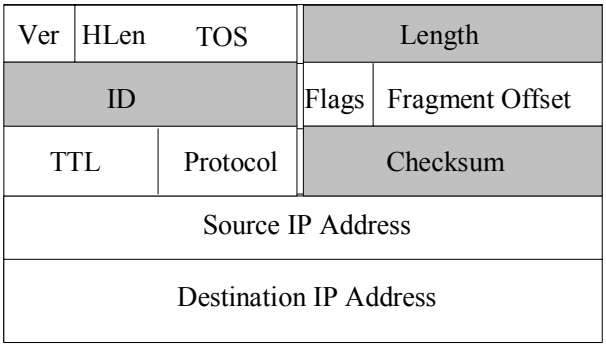

Abbildung 2-6: Felder im TCP/IP Header

Der dritte Themenbereich beschäftigt sich mit der Optimierung von Transportprotokollen in Bezug auf Ansprüche mobiler Systeme. Schwerpunkte in diesem Zusammenhang sind *Handoff-Strategien* [BB 95, BKAB+ 98] zwischen gleichartigen oder auch heterogenen Netzen und Verfahren, um redundanten *Protokoll-Overhead* zu minimieren.

Die Arbeit im Bereich der Handoff-Strategien zielt darauf ab, dem Benutzer kontinuierlichen Netzwerkzugang zu ermöglichen unabhängig von der aktuellen Abdeckung eines Netzwerktyps. Der Wechsel des Netzwerktyps soll für den Nutzer unter Berücksichtigung bestimmter QoS (Quality of Service) Parameter transparent sein. Auf diese Weise kann ein Benutzer in einer Büroumgebung einen WaveLAN-Zugang mit hoher Bandbreite nutzen, während er beim Verlassen dieser Umgebung das System automatisch auf GPRS wechselt, da der WaveLAN-Zugang nicht mehr erreichbar ist.

Die Verminderung von Protokoll-Overhead bezieht sich vor allem auf die Problematik der teilweise geringen Bandbreite in drahtlosen Netzwerken (im Vergleich zu Festnetzwerken). Ein bekannter Ansatz bezieht sich auf die Komprimierung von TCP/IP- Headern [HYBS 00]. Das Verfahren beruht darauf, nur die Abweichungen (*Deltas*) von aufeinander folgenden TCP-Paketen zu übermitteln und damit das zu übertragende Datenvolumen zu verringern.

Abbildung 2-6 verdeutlicht das Prinzip anhand eines TCP/IP-Headers. Die weißen Felder in einem TCP/IP-Header sind während einer gesamten Kommunikationssitzung konstant und müssen nur einmal zwischen Sender und Empfänger übertragen werden. Beim Feld *ID* müssen nur Deltas zwischen Sender und Empfänger übertragen werden; das Feld *Lenght* ist ableitbar. Das Feld *Checksum* lässt sich nicht komprimieren, da es pro Paket neu berechnet wird. Bei manchen Anwendungen lässt sich mittels Komprimierung der TCP/IP-Header das zu übertragende Datenvolumen um mehr als das Zehnfache verringern. Man beachte, dass bei diesem Verfahren nur die Header-Informationen komprimiert werden, nicht die eigentlichen Daten.

2.2.2.2 Informationszugriff

Trotz der bereits beschriebenen Handover-Strategien für mobile Klienten in einem Verteilten System kann es immer wieder zu unabsehbaren Verbindungsabbrüchen (*intermitted connectivity)* kommen. Aus diesem Grund lassen sich einige Techniken aus dem Forschungsbereich der Verteilten Systeme nicht anwenden, da impliziert wird, dass die einzelnen Komponenten eines Systems über ein statisches Netzwerk miteinander kommunizieren (ohne unabsehbare Verbindungsabbrüche).

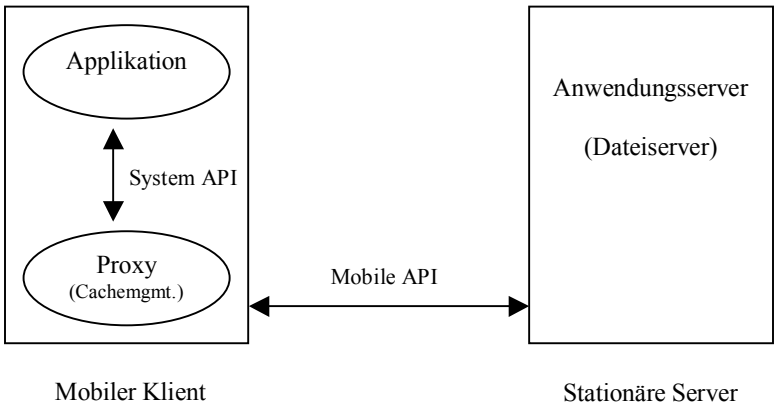

Abbildung 2-7: Cache-Management

Infolgedessen sind Systeme entwickelt worden, die sich den Variationen innerhalb mobiler Netzwerke anpassen können. Dabei werden nicht nur Verbindungsabbrüche beachtet, sondern auch drahtlose Verbindungen mit niedriger Bandbreite oder Netzwerkverbindungen, deren Nutzung mit hohen Kosten verbunden sind. Netzwerke dieser Charaktere erlauben also nur Verbindungen mit einer so genannten *weak connectivity* [MES 95].

Disconnected Operation [KS 92] ist ein Operationsmodus für mobile Klienten, welcher es erlaubt, auf kritische Daten zuzugreifen trotz vorübergehender Zugriffsprobleme auf den geteilten Datenbestand (*shared data repository*). Die Basis für Disconnected Operation ist ein Proxy, welcher die Funktion eines lokalen Caches auf der Seite des mobilen Klienten realisiert. Auf diesem kann eine Anwendung auch bei einer vorübergehenden Verbindungsunterbrechung zugreifen, siehe auch Abbildung 2-7.

Das Verfahren der Disconnected Operation kann auch dann hilfreich sein, wenn eine Netzwerkverbindung möglich wäre. Beispielsweise kann dank der Disconnected Operation die Lebensdauer von Batterien eines mobilen Endgerätes erhöht werden, da Sende- und Empfangsoperationen über das drahtlose Netzwerk vermieden werden können [MES 95]. Ebenso können Netzwerkkosten eingespart

werden, und eine Disconnected Operation kann auch im Falle eines Netzwerkzugangs mit unzureichenden QoS Parametern eine günstige Alternative sein.

Natürlich gibt es bei dem Ansatz der Disconnected Operation auch Beeinträchtigungen. Dazu gehören:

- Updates sind für andere Klienten nicht sichtbar
- Das Entstehen von Update Konflikten
- Die geforderten Daten befinden sich nicht im Cache (*cache misses*)
- Der Cache-Speicher ist ausgelastet

Die Relevanz dieser Aspekte nimmt mit abnehmender Netzqualität zu. Ein zentrales Problem bei der Realisierung von Disconnected Operation sind die Mechanismen zur Cacheverwaltung auf Seite des mobilen Klienten; diese sollen hier kurz beschrieben werden. Abbildung 2-8 zeigt das Zustandsdiagramm einer Cacheverwaltung nach [KS 92].

Hoarding

Befindet sich die Cacheverwaltung im Zustand *Hoarding*[3], werden Daten, welche nach einem möglichen Verbindungsabbruch am ehesten benötigt werden, im lokalen Cache des mobilen Klienten repliziert. Die vom Benutzer benötigten Daten (z.B. bestimmte Dateien eines Dateisystems) können von ihm explizit als kritisch markiert werden und werden wann immer möglich im Cache repliziert, so dass sie auch nach einem Verbindungsabbruch zur Verfügung stehen. Neben den explizit markierten Dateien werden im Cache auch die Dateien repliziert, welche gegenwärtig in Gebrauch sind. Dies ist erforderlich, um sich den aktuellen Benutzeranforderungen anzupassen und damit die Leistung des Systems zu erhöhen.

Viele Faktoren verkomplizieren den Hoarding Prozess, unter anderem folgende:

- Es kann nicht mit Sicherheit bestimmt werden, welche Dateien in Zukunft benötigt werden
- Verbindungsabbrüche bzw. das Wiedererlangen einer Verbindung sind zumeist nicht vorhersehbar
- Den Aktivitäten anderer Klienten muss Rechnung getragen werden, so dass im Falle eines Verbindungsabbruchs die aktuellste Version des Datenfragments im Cache ist
- Da der Cache-Speicher endlich groß ist, müssen weniger kritische Datenfragmente anderen weichen.

[3] deutsch: *horten*

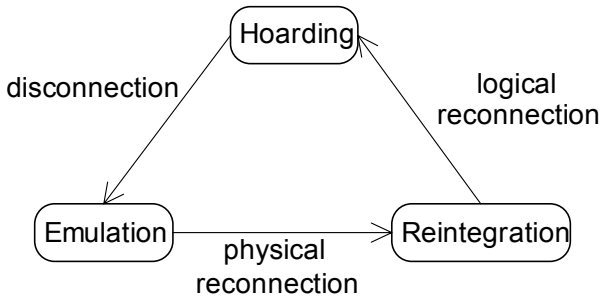

Abbildung 2-8: Zustandsdiagramm der Cache-Verwaltung

Um diesen Anforderungen gerecht zu werden, ist in [KS 92] ein prioritätbasiertes Cache-Verwaltungssystem vorgeschlagen worden. Dieses System evaluiert in periodischen Abständen den Inhalt des Caches und entscheidet aufgrund diverser Parameter (z.B. Zugriffsverhalten, markierte Elemente), welche Datenfragmente im Cache verbleiben bzw. welche gelöscht werden sollen.

Emulation

Im *Emulationsmodus* werden viele Operationen, die sonst auf dem entfernten Anwendungsserver ausgeführt werden, auf dem mobilen Klienten bewerkstelligt. Abgesehen von dem eigentlichen Datenzugriff wird auch die Zugriffs- und Modifikationskontrolle seitens des Klienten aufgelöst. Auch wenn das Cacheverwaltungsprogramm als *Pseudo-Server* agiert, müssen alle Modifikationen der lokalen Datenbasis bez. Integrität bei den wirklichen Anwendungsservern überprüft werden (Vertrauensverhältnis nur bez. Servern). Der Prozess der Cacheverwaltung wird nach den gleichen Prämissen bewerkstelligt wie im Zustand *Hoarding*.

Reintegration

In der Reintegrationsphase werden alle Datenmodifikationen, welche sich während der Emulationsphase ereignet haben, zu den korrespondierenden Anwendungsservern propagiert. Der Cache des mobilen Klienten wird mit den aktuellen Versionen der Datenfragmente auf der Serverseite abgeglichen.

Zusammenfassung

Die vorigen Abschnitte haben das Problem des Informationszugriffs von mobilen Klienten auf verteilte Server beschrieben. Die Ausführung hat sich insbesondere auf das Coda-Projekt [KS 92] bezogen. Es konnten nur Grundzüge dieser recht komplexen Thematik vermittelt werden. So wurden beispielsweise Strategien der Konfliktbehandlung nicht darlegt. Neben den Literaturhinweisen bez. Coda, [KS

92] und [MES 95], können beispielsweise auch die Arbeiten [TD 92], [Te 95] herangezogen werden, um das Verständnis der Thematik zu vertiefen.

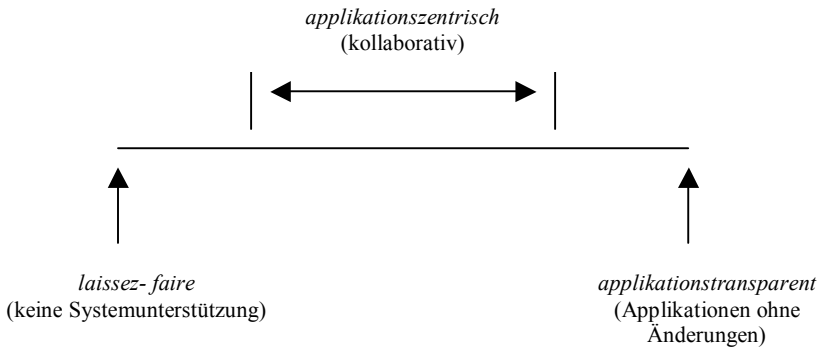

Abbildung 2-9: Adaptive Applikationen

2.2.2.3 Adaptive Applikationen

Das Angebot der einem mobilen Klienten zur Verfügung stehenden stationären Ressourcen (z.B. Applikationsserver) kann stetig variieren. Um dennoch Anwendungen zu unterstützen, welche auch bei solchen Konditionen operabel bleiben, werden Konzepte angewandt, die eine dynamische Anpassung an die Begebenheiten ermöglichen. Mit anderen Worten, das Ausführungsverhalten einer Anwendung seitens Klient und Server muss bez. der Variationen in einer mobilen Umgebung *adaptiv* sein [Ka 94, JHE 99].

In [Saty 96] wird eine Klassifikation von adaptiven Systemen dieser Art vorgestellt; siehe auch Abbildung 2-9. Dabei wird zwischen systemweiter und applikationsbezogener Adaption unterschieden. Der Bereich der Adaptionsstrategien wird durch zwei Extreme begrenzt. Zum einen besteht die Möglichkeit, dass die Verantwortlichkeit der Anpassung gänzlich von einer Applikation zu erfüllen ist, ohne jegliche Systemunterstützung. Dieser Ansatz wird in der Literatur auch als *laissez-faire* bezeichnet. Zum anderen besteht die Handhabung, dass die Anpassungsleistung ausschließlich vom System erbracht wird. Infolgedessen wird diese Vorgehensweise auch als *applikationstransparente* Adaption bezeichnet. Ein typischer Anwendungsfall für diese Vorgehensweise ist der Einsatz von Proxies.

Zwischen diesen beiden Extremen liegt ein ganzes Spektrum von Adaptionsstrategien, die als *applikationszentrisch*[4] klassifiziert werden. Diese Adaption bezieht sowohl das System als auch die jeweilige Applikation mit ein. Damit ist gemeint, dass Applikationen selbst das Anpassungsverhalten steuern, wobei das System Unterstützung bei der Überwachung von Ressourcen

[4] engl. *application-aware adaption*

(*Monitoring*) und der Umsetzung von Entscheidungen zur Reservierung von Ressourcen bietet. Die folgenden Abschnitte erläutern die einzelnen Adaptionsstrategien an Beispielen.

Applikationstransparente Adaption

Viele existierende Client/Server Applikationen implizieren die Voraussetzung, dass sich die Umgebung eines Klienten nicht verändert. Diese Anwendungen unterstützen nicht das Konzept der Mobilität und machen daher bestimmte Annahmen bez. der Disponibilität von Ressourcen.

Der Ansatz der *applikationstransparenten Adaption* versucht, diese Anwendungen ohne Modifikation auch in einer mobilen Systemumgebung benutzbar zu machen. Dies wird bewirkt, indem das System um eine Schicht erweitert wird, die von Unwegsamkeiten einer Umgebung mit Ressourcenvariationen abstrahiert. Beispiele für diese Vorgehensweise sind folgende Systeme: Coda [KS 92] (siehe 2.2.2.2), Little Work [HHRB 92] und WebExpress [HL 96]. Bei all diesen Modellen werden lokale Proxies auf dem mobilen Klienten eingesetzt, welcher eine Schnittstelle zu Serverdiensten implementiert. Aufgabe des Proxies ist es, störende Effekte, die durch die Mobilität des Klienten hervorgerufen werden, abzufangen.

Zur Demonstration des Konzeptes der applikationstransparenten Adaption soll hier das Modell des *Web Proxy* für mobile Klienten demonstriert werden. Web Proxies realisieren den Zugriff über drahtlose Netzwerkverbindungen auf das World Wide Web (WWW), ohne dass Browser oder Server modifiziert werden müssen. Web Proxies können den Zugang zu WWW-Daten mittels verschiedener Methoden verbessern:

- Zwischenspeichern von Daten in einem lokalen Cache
- Vorzeitige Zwischenspeicherung von Daten, welche zukünftig angefordert werden können (*prefetching*)
- Kompression und Transformation von Daten, z.B. um Bildmaterial über Netzwerke niedriger Bandbreite zu senden
- Unterstützung für *Disconnected Operations*; siehe auch 2.2.2.2.

Als Beispiel für die Anwendung des Proxy Modells für WWW-Zugriffe in mobilen Umgebungen soll hier WebExpress [HL 96] vorgestellt werden; siehe Abbildung 2-10. WebExpress benutzt diesen Ansatz, um den Kommunikationsverkehr zwischen einem mobilen Host und einem stationären Server zu optimieren. Um das zu erreichen, wurden zwei Komponenten in den Datenpfad zwischen Klient und Server eingefügt: der *Client Side Intercept* (CSI) seitens des Browsers und der *Server Side Intercept* (SSI) seitens des Web Servers.

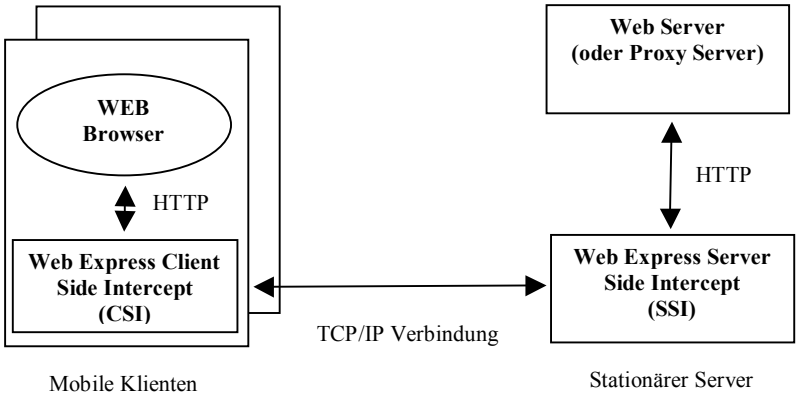

Abbildung 2-10: WebExpress

Die CSI Komponente überwacht (*intercepts)* die vom Browser initiierten HTTP Anfragen und modifiziert sie in Kooperation mit dem SSI so, dass die Bandbreitenausnutzung bzw. die Latenzzeit im drahtlosen Netz verringert werden kann. Aus Sicht des Browers agiert der CSI wie ein lokaler Web Proxy, so dass dieser über eine lokale TCP Verbindung via HTTP angesprochen werden kann (TCP/IP loopback). Das heißt, für die Kommunikation zwischen Browser und CSI ist kein Netzwerk erforderlich. Die beim CSI eingehenden HTTP-Anfragen werden mittels TCP/IP zum SSI versandt. Es ist zu beachten, dass zwischen CSI und SSI nicht HTTP zur Datenübermittlung verwendet wird. Hier wird ein reduziertes HTTP Protokoll via TCP/IP verwendet. Der SSI rekonstruiert den eingehenden Datenstrom, um diesen dann an den designierten Web Server weiterzuleiten. Analog erfolgt die Kommunikation von SSI zu CSI. Der CSI empfängt und rekonstruiert den einkommenden Datenstrom, um ihn dann über die lokale TCP Verbindung zum Browser zu senden. Seitens des Browsers erscheint die Kommunikation direkt mit dem korrespondierenden Web Server vonstatten zu gehen.

Der in WebExpress gewählte Ansatz erzielt den Vorteil der Transparenz sowohl bez. eines Web Servers (oder Proxies) als auch eines Browsers. Daher kann diese Lösung für jeden Browser – ohne Modifikationen - eingesetzt werden. Zudem implementiert das zwischen CSI und SSI angewandte Übertragungsprotokoll eine sehr effektive Datenreduktion, ohne die Interoperabilität mit einem Browser in irgendeiner Weise einzuschränken.

Im folgendem werden die Optimierungsverfahren von WebExpress zusammengefasst:

- *Caching*. Sowohl die CSI und SSI Komponente setzen Caching-Strategien ein, um die Performanz zu erhöhen. So agiert der CSI zum Teil wie ein konventioneller Proxy Cache.

- *Differencing*. Diese Methode wird eingesetzt, um möglicherweise nur Differenzen zwischen aufeinander folgenden HTTP-Anforderungen zwischen CSI und SSI zu übermitteln.
- *Protocol Reduction*. Alle HTTP Anforderungen und Antworten werden über eine einzelne TCP/IP Verbindung gemultiplext. Kostenaufwendige Operationen zum Verbindungsaufbau entfallen.
- *Header compression*. Dieser Ansatz ist analog zu dem Verfahren, welches bereits in 2.2.2 vorgestellt wurde. Allerdings handelt es sich hier um die Reduktion von HTTP-Headern, nicht um TCP/IP Header.

Zusammenfassend lässt sich feststellen, dass sich das Konzept der applikationstransparenten Adaption zur Unterstützung mobiler Web Klienten anbietet, da kaum Modifikationen an der vorhandenen Infrastruktur vorgenommen werden müssen, und dennoch ein beachtlicher Performanzgewinn erzielt werden kann [HL 96]. Eine vergleichbare Arbeit zu diesem Thema kann beispielsweise in [KRA 94] gefunden werden.

Applikationszentrische Adaption

Der Ansatz der applikationstransparenten Adaption macht es nicht erforderlich, existierende Anwendungen zu modifizieren, damit sie in einer Umgebung mit mobilen Klienten eingesetzt werden können. Jedoch kann diese Vorgehensweise die Performanz und Funktionalität einer Anwendung degradieren. Da Applikationen von Entscheidungen bez. der Adaption an Variationen der Umgebung keinen Einfluss nehmen können, ist es oft schwer, den Anforderungen diverser Applikationstypen gerecht zu werden. Als Konsequenz ist dann oft ein expliziter Eingriff des Benutzers erforderlich, um diesen Problemen gerecht zu werden (z.B. benutzerdefiniertes *prefetching* von Daten).

Im Kontrast dazu erlaubt die *applikationszentrische Adaption*, dass sich sowohl die Applikation selbst bei Änderungen anpassen kann, als auch assoziierte Systemkomponenten. Ein Ansatz, applikationszentrische Adaption zu bewerkstelligen, besteht in Kollaboration zwischen individuellen Applikationen und Systemkomponenten. Das System übernimmt das Monitoring von Ressourcen, benachrichtigt Applikationen über relevante Abweichungen und unterstützt beim Management von Ressourcen. Jede Applikation kann dann unabhängig entscheiden, welche Adaptionsstrategie beim Eingehen einer Meldung über Veränderungen angemessen ist.

Dieses Prinzip kann man gut am Beispiel einer Videoübertragung veranschaulichen. Die Applikation, welche das Video abspielt, kann die Bildqualität dynamisch verringern (und damit Ressourcen schonen), wenn die verfügbare

Bandbreite unzureichend ist. Andererseits kann sie immer versuchen, die Bildqualität zu erhöhen, indem sie zusätzliche Ressourcen lokalisiert.

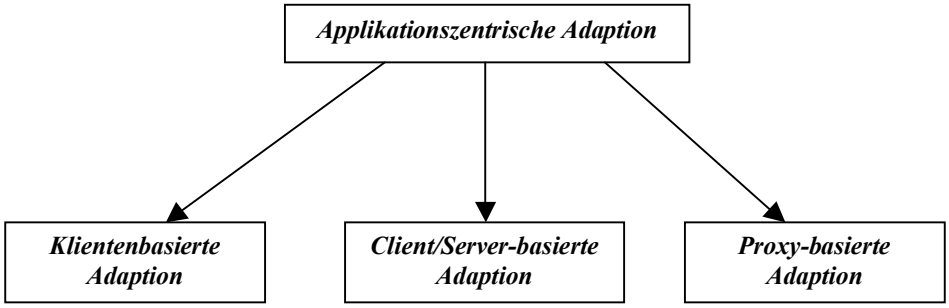

Abbildung 2-11: Unterklassen der Adaptions-Zentrischen Adaption

In Abhängigkeit davon, wo die Adaptionslogik umgesetzt wird, kann man den Ansatz der applikationszentrischen Adaption in verschiedene Unterklassen einordnen; siehe auch Abbildung 2-11.

Klientenbasierte Adaption erlaubt es Applikationen mobiler Klienten, auf Änderungen der Umgebung zu reagieren, während bei der *client/server-basierten Adaption* sowohl das System, als auch die Applikation auf Veränderungen reagieren kann. Beispiele für Systeme, welche diese Arten von Adaption anwenden, können in [NSN+ 97] und [JK 97] gefunden werden.

Die *proxy-basierte Adaption* unterstützt applikationsspezifische Berechnungen von einem Proxy im Festnetz. Die applikationsspezifischen Server werden als intermediäre Instanzen zwischen bereits vorhandenen Servern und heterogenen mobilen Klienten [BKAB+ 98] eingesetzt. Die Proxies können stellvertretend für mobile Klienten aufwendige Berechnungen vollziehen (z.B. Konvertierung von Datentypen, Bildmaterial) oder auch Speicherplatz zur Verfügung stellen.

Diese Ansätze können komplementär in Client/Server-Systemen angewandt werden. So kann beispielsweise eine klientenbasierte mit einer proxy-basierten Adaption in einem einzelnen System kombiniert werden, um Anforderungen mobiler Systeme zu erfüllen.

2.2.3 Smart Spaces

Die vorigen Teilabschnitte haben Aspekte Verteilter Systeme und Mobile Computing beschrieben. Viele Ansätze aus diesen Bereichen sind für das Forschungsfeld Ubiquitous Computing fundamental. Dennoch reflektieren sie nicht die Gesamtheit aller Anforderungen eines Ubiquitous Computing Systems. Diesbezüglich soll hier der Forschungsbereich *Smart Spaces* [AS 00, KLST 96] vorgestellt werden. Dieser Themenzweig zeichnet sich durch einen multi-

disziplinären Ansatz aus, der sich unter anderm mit folgenden Gesichtspunkten befasst:

- *Eingebettete Systeme: kooperative Entwicklung von Hard- und Software*
- *Ortsgebundene Ressourcen*
- *Dezentrale Systeme*
  - *Dynamische Rollenzuordnung*
  - *Spontane Vernetzung / Mobile Ad-hoc-Netzwerke*
  - *Sensor Netzwerke*
- *Human Computer Interaction* [Sto 95]

Ein Smart Space System kann anhand vier orthogonaler Perspektiven expliziert werden: *Smart Spaces*, *Smart Nodes*, *Netzwerk, Protokolle.* Die oben genannten Aspekte fließen ein und werden im Zusammenhang erläutert.

Smart Spaces

Ein Smart Space ist ein Raum, der von einer Menge von so genannten *Smart Nodes* aufgespannt wird. Smart Nodes können entweder mobil oder stationär sein. Dieser Raum ist entweder physikalisch begrenzt (z.B. durch ein Zimmer) oder durch die Netzwerkreichweite der einzelnen Smart Nodes.

Bemerkenswert bei dieser Konzeption ist, dass Smart Nodes immer auch bez. ihres Ortes identifiziert werden können. Bei konventionellen Ansätzen Verteilter Systeme verfolgt man in Kontrast dazu meist das Ziel, vom Ort eines Objekts zu abstrahieren. Man spricht in diesem Zusammenhang auch von Ortstransparenz, so z.B. bei Objekt-Trader-Architekturen. Im Kontext von Smart Spaces sind Ressourcen immer *ortsgebunden*.

Smart Nodes

Bei *Smart Nodes* kann es sich um herkömmliche Endgeräte handeln, wie etwa um Mobilfunktelefone oder PCs. Hier sollen vor allem Smart Nodes mit folgenden Eigenschaften betrachtet werden: es handelt sich dabei um Eingebettete Systeme [BW 00] mit einem drahtlosen Netzwerkzugang. Zudem verfügen diese Elemente oft auch über Sensoren (z.B. Temperaturmessung), um die Implementierung situationsadaptiver Applikationen [Dey 00, DeyAb 00] zu begünstigen. Des Weiteren zeichnen sich diese Elemente durch eine autonome Energieversorgung aus und bilden ein geschlossenes System innerhalb der Umgebung.

Diese Eigenschaften erlauben eine autonome Operation dieser Netzwerkelemente, welche die Kommunikation zwecks Erfüllung einer Aufgabe einschließt.

Charakteristisch für die Kooperation solcher Elemente ist ein rollenbasierter Ansatz. Das heißt, jedes Element hat im Rahmen eines bestimmten Kontextes eine bestimmte Rolle, welche einen bestimmten Aufgabenbereich festlegt. Die Zuordnung Rolle/Element kann sich dynamisch den Gegebenheiten anpassen. Natürlich muss beachtet werden, dass bestimmte Elemente nur spezifische Rollen annehmen können, da sie aufgrund ihrer eingebetteten Ressourcen eingeschränkt sind. Dabei kann es sich um statische Einschränkungen, wie etwa Rechenleistung oder Speicherkapazität, handeln, aber auch um dynamische, wie etwa die noch verfügbare Energie eines Elements.

Gerade die autonome Energieversorgung von Smart Nodes ist Bestand aktueller Forschung. Zum einen beziehen sich diese Arbeiten auf adaptive Algorithmen zur effektiven Nutzung von Energieressourcen, zum anderen wird der Einsatz neuer Energiesysteme erforscht. In diesem Zusammenhang ein interessantes Faktum: Die Batteriekapazität hat sich in den letzten 20 Jahren nur um 20% erhöht [Gel 00].

Außerdem stellt das Thema Smart Spaces neue Herausforderungen bez. des Managements von Systemen dar, da sich gegebenenfalls eine große Menge von Netzwerkelementen in einem Raum befinden. Als Beispiel: Befinden sich in einer Umgebung 1000 Netzwerkelemente, so gibt es selbst bei einer Verläßlichkeit von 99.9% einen technischen Defekt pro Tag.

Smart Nodes können in folgende Kategorien [AS 00] eingeordnet werden:

- *Personal Nodes*: Personenbezogenen Endgeräte (z.B. PDA ähnlich), die mit Smart Nodes in der Umgebung interagieren können
- *Wearable I/O nodes*: Head-up Displays, Mikrophone
- *Carryable I/O nodes:* Texteingabegeräte, Tastaturen, welche in Textilien integriert sind
- *Individual sensor nodes:* GPS, Temperatur, Biosensoren (z.B. Blutdruck)
- *I/O in smart rooms:* Drucker, Kameras, Displays
- *Server, Gateways, Storage nodes:* Anbindung ans Internet, Auslagerung von Berechnungen und Speicherdiensten.

Organisation von Smart Nodes

Nachdem die vorigen Abschnitte die Basiskonzepte von Smart Spaces erläutert haben, soll im weiteren die Organisation von Smart Nodes erläutert werden. Abbildung 2-12 vermittelt eine Übersicht. Smart Nodes werden innerhalb so genannter Föderationen organisiert. Föderationen kennzeichnen Gruppen von Smart Nodes, die bez. einer bestimmten Aufgabe miteinander in Interaktion stehen. Föderationen sind keine statischen Strukturen. Je nach Bedarf einer Appli-

kation bzw. nach Verfügbarkeit von Smart Nodes kann eine Reorganisation erforderlich sein.

Einzelne Föderationen können direkt über Smart Nodes miteinander vernetzt sein oder aber auch indirekt durch ein Backbone-Netzwerk, z.B. durch das Internet. So ist es möglich, Föderationen zu bilden, deren räumliche Ausdehnung nicht durch die Netzwerkreichweite der konstituierenden Smart Nodes beschränkt wird. Zudem können rechenintensive Aufgaben auf entsprechende Knoten des Backbone-Netzwerkes verlagert werden, so dass die oftmals knappen Ressourcen eines Smart Space Netzwerkes geschont werden können.

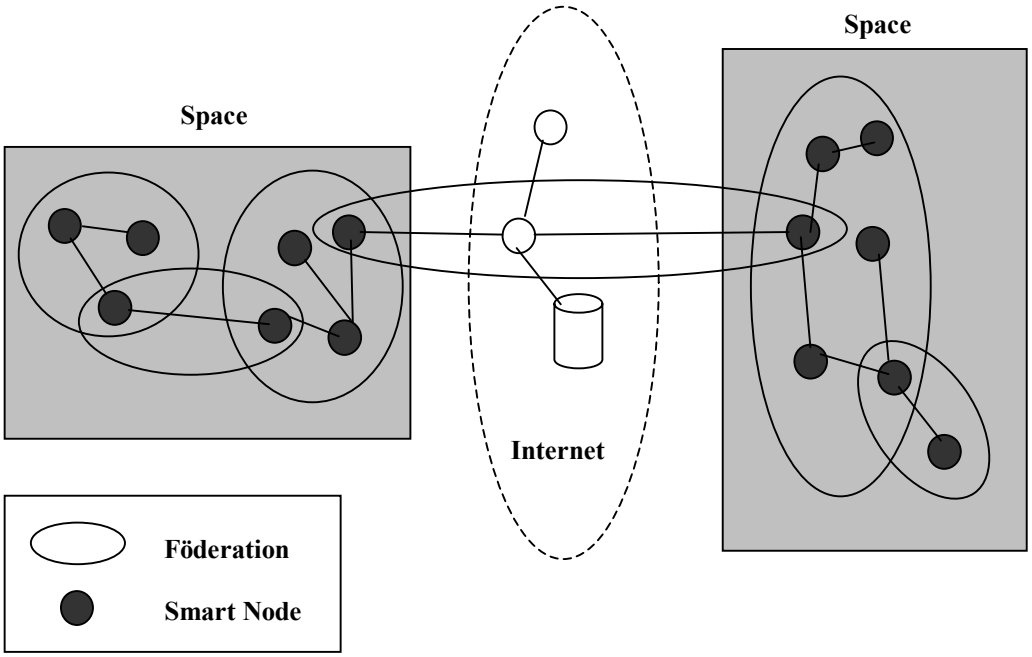

Abbildung 2-12: Organisation von Smart Nodes

Wie bereits erwähnt, wird die Organisation von Smart Spaces durch die jeweilige Anwendung bestimmt. Der Aspekt der Applikationsbezogenheit bez. der Vernetzung von Smart Nodes ist charakteristisch für Smart Spaces und stellt eine neue Herausforderung dar. Nicht nur die Vernetzung individueller Knoten ist abhängig von der jeweiligen Anwendung. Generell müssen je nach Anwendung Entscheidungen bez. der Zuordnung von Funktionalitäten auf die einzelnen Knoten getroffen werden. Applikationsspezifische Netzwerkstrukturen gewinnen zunehmend an Bedeutung. Dieser Gesichtspunkt wird in Kapitel 5 anhand des Problems der verteilten Datenfusion in einem drahtlosen Ad-hoc-Sensornetzwerk untersucht.

2.3 Kontextadaptivität

Das Konzept der Kontextadaptivität [SAW 94] erlaubt die Entwicklung von Anwendungen, die sich implizit der Situation eines Benutzers anpassen. Damit lassen sich nicht nur neue Anwendungstypen entwickeln, sondern auch bereits bestehende Anwendungen lassen sich um diesen Aspekt erweitern. Einer der Hauptbeweggründe für die Forschung im Bereich der Kontextadaptivität ist die Möglichkeit, die Benutzung von Anwendungssystemen wesentlich zu erleichtern. Bereits frühe Ansätze im Bereich Ubiquitous [Weis 93] und Wearable [ABOB 97] Computing haben die Relevanz des Kontextbegriffes hervorgehoben, indem sie insbesondere auch Aspekte der Mobilität berücksichtigt haben. Denn gerade in Anwendungsszenarien, wo die Mobilität von Benutzern (oder von Gegenständen) das Systemverhalten beeinflusst, lässt sich der Bedarf kontextadaptiver Systeme einfach begründen. Das ergibt sich daraus, dass sich die Umgebung eines mobilen Nutzers stetig ändert und infolgedessen auch sein assoziierter Kontext, dazu gehört etwa sein gegenwärtiger Aufenthaltsort, Personen und Gegenstände in direkter Nähe.

2.3.1 Kontextinformationen

Im Folgenden wird zunächst das Konzept der *Kontextinformation* erläutert. Zur Vermittlung des Begriffs werden verschiedene Ansätze zur Fassung des Konzeptes kurz beschrieben. Schließlich wird eine Definition nach Dey und Abowd [DeyAb 00] vorgestellt. Dieser Ansatz hat den Vorzug, dass mehrere Definitionsansätze mit eingeschlossen werden. Zudem ist sie nicht konkret auf bestimmte Anwendungsfälle bezogen und kann so auf eine Vielzahl von Szenarien angewandt werden. Die Ausführung beschränkt sich auf eine Auswahl von Ansätzen. Eine vollständige Abhandlung dieser Thematik findet sich in [Dey 00].

Der Begriff der *Kontextadaptivität* und *Kontextinformationen* wurde zum ersten Mal in einer Arbeit von Schilit und Theimer [ST 94] erwähnt. Dort wird Kontext als Aufenthaltsort und Identität von Personen und Objekten in der direkten Nähe eines Benutzers verstanden. Auch dynamische Aspekte werden in dieser Definition eingeschlossen, so etwa die Statusänderung eines Objekts oder das Fortbewegen einer Person. Ein ähnlicher Gebrauch des Kontextbegriffs wird in [BroBo 97] vorgeschlagen, er wird dort um Uhrzeit und einige Umgebungseigenschaften (z.B. Temperatur) erweitert. Interessant ist auch eine Definition nach Schilit und Adams et al. [SAW 94]. Hier wird Kontext in drei verschiedenen Dimensionen beschrieben:

- *Computing Environment*
- *User Environment*
- *Physical Environment*

Computing Environment bezeichnet die Menge der verfügbaren Ressourcen, welche dem Benutzer zu einem bestimmten Zeitpunkt zugänglich sind. Dazu gehören Geräte in der Umgebung des Benutzers, wie etwa Drucker oder Ein- und Ausgabekonsolen, aber auch netzwerkspezifische Informationen: die Kapazität oder die Kosten für die Nutzung eines Netzwerkdienstes. Zusammenfassend bezieht sich ein Computer Environment auf eine ortsbezogene Ausführungsumgebung für Benutzerdienste.

User Environment bezeichnet den Aufenthaltsort eines Benutzers und die Anwesenheit anderer Personen in der Nähe des Benutzers.

Physical Environment umfasst beispielsweise Informationen zur Umgebungslautstärke und Ausleuchtung. Im Rahmen dieser Arbeit wird die Definition nach Dey und Abowd [DeyAb 00] angewandt:

> *Als Kontextinformation (oder Kontext) wird jegliche Information bezeichnet, welche benutzt werden kann, um die Situation einer Entität zu charakterisieren. Eine Entität ist eine Person, ein Ort oder ein Objekt, welches bez. der Interaktion zwischen einem Benutzer und einer Anwendung als relevant einzuordnen ist. Dabei schließt man auch den Benutzer und die Anwendung mit ein.*

Die Nutzung von Kontextinformationen ermöglicht Anwendungen, die sich der jeweils gegenwärtigen Situation anpassen. In diesem Zusammenhang ist zu beachten, dass Kontextinformationen nur Hinweise auf eine bestimmte Situation geben, und die jeweilige Anwendung muss diese Information erst interpretieren. Die Interpretation von Kontextinformationen ist immer abhängig vom Typ der Applikation.

Die vorgestellte Definition für Kontextinformationen berücksichtigt neben impliziten Eingaben auch explizite Eingaben. So kann beispielsweise eine Identitätskontrolle via expliziter Eingabe eines Namens und des zugehörigen Passworts erfolgen, sie könnte aber auch implizit durch eine Kamera erfolgen, welche anhand von aufgenommenen Bildern die Identität des Benutzers feststellt. Aus Sicht der Anwendung handelt es sich in beiden Fällen um Informationen zur Identität des Nutzers und kann ihr Verhalten diesbezüglich angleichen. In dieser Hinsicht bildet das Konzept der Kontextadaptivität ein generelles Modell zur impliziten und expliziten Dateneingabe für Anwendungen. Natürlich ist im Rahmen kontextadaptiver Systeme der Fall der impliziten Dateneingabe von größerem Interesse, da sie dem ausgesprochenen Ziel entspricht, die benötigten expliziten Eingaben eines Benutzers zu verringern.

Tabelle 2-1 präsentiert einige Beispiele für Kontextinformationen, geordnet nach *Kontexttypen*.

Kontexttyp	Beispiele
Identität	Benutzername und Passwort
Ortsinformationen	Ort, Orientierung, Geschwindigkeit, Beschleunigung
Zeitinformationen	Uhrzeit, Datum, Jahreszeit
Umgebungsvariablen	Temperatur, Luftqualität, Geräuschspegel
Ressourcen in der Nähe des Benutzers	Erreichbare Server und Endgeräte
Aktivität	Sprechend, lesend, laufend, sitzend

Tabelle 2-1: Kontexttypen

Es existieren einige Kontexttypen, welche im praktischen Einsatz eine größere Relevanz haben als andere, dazu gehören *Identität*, *Ortsinformation* und *Aktivität*. Mittels dieser Kontexttypen lassen sich in einigen Szenarien bereits viele Aussagen bez. der Situation einer Entität erschließen. Zudem können von diesen Kontextinformationen andere abgeleitet werden. Aus der Identität eines Benutzers kann man z.B. dessen E-Mail Adresse oder Telefonnummer ableiten [Dey 00]. In diesem Zusammenhang spricht man auch von *primären* und *sekundären*, also abgeleiteten Kontexttypen.

2.3.2 Klassen der Kontextadaptivität

Kontextadaptivität zeichnet ein System aus, welches Kontextinformationen einsetzt, um eine bestimmte Funktionalität zu erfüllen. Ein kontextadaptives System interpretiert diese Informationen, um sich einer Situation anzupassen. Die zentralen Herausforderungen eines solchen Systems bestehen in der *Konstruktion*, *Repräsentation* und *Verarbeitung* von Kontextinformationen [Pas 98b]. Die Konstruktion von Kontextinformationen erfordert in der Regel zusätzliche Sensoren (siehe 2.3.4) und Softwarekomponenten. Damit Kontextinformationen zwischen verschiedenen Applikationskomponenten ausgetauscht werden können, ist ein einheitliches Repräsentationsformat erforderlich (siehe z.B. [Sch 00]). Schließlich behandelt der Prozess der Verarbeitung von Kontextinformationen, wie diese eingesetzt werden, um situationsadaptive Anwendungen zu gestalten.

Um das Konzept der Kontextadaptivität zu demonstrieren, sollen hier zunächst einige Anwendungsszenarien vorgestellt werden. Nach Schilit [Sch 95] unterscheidet man zwischen folgenden Anwendungsklassen:

- *Proximate selection*
- *Automatic reconfiguration*
- *Contextual Commands*

– *Context-triggered actions*

Diese Anwendungsfälle lassen sich bez. folgender, orthogonaler Dimensionen einordnen; siehe auch Abbildung 2-13. So unterscheidet man zum einen zwischen *aktions-* und *datenbezogenen* Operationen, und zum anderen zwischen *automatischen* und *manuell* ausgelösten Operationen.

Anwendungsfälle, welche Daten auf explizite Anforderung des Benutzers aber kontextbezogen zustellen, werden der Klasse *Proximate Selection* zugeordnet. Das Resultat einer solchen Operation könnte beispielsweise eine Liste von Objekten (z.B. Drucker) in der Umgebung des Benutzers sein.

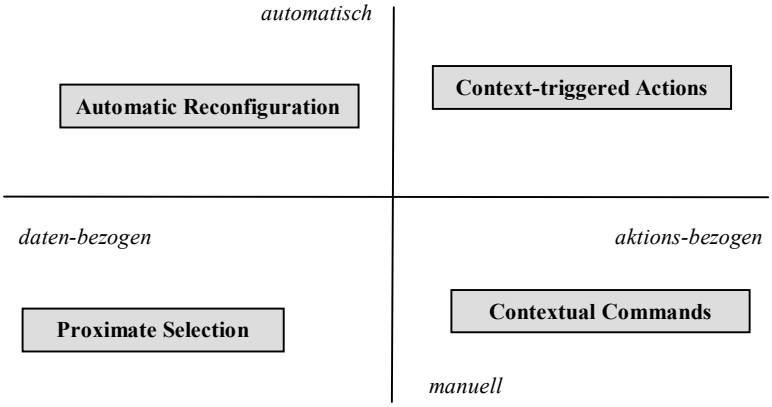

Abbildung 2-13: Taxonomie kontextadaptiver Anwendungen nach Schilit

Im Kontrast zu *Proximate Selection* werden Anwendungsfälle, bei welchen die Datenzustellung dem Kontext entsprechend automatisch erfolgt, der Klasse *Automatic Configuration* zugeordnet. Dabei handelt es sich um eine systemspezifische Operation, welche basierend auf den vorhandenen Kontextinformationen automatisch die Bindung zu einer Ressource erstellt.

Aktionen, die manuell vom Benutzer initiiert und in Bezug zum Kontext ausgeführt werden, bezeichnet man als *Contextual Commands*. Dieses Konzept realisiert zwei unterschiedliche Operationen. Zum einen besteht die Möglichkeit, Dienste kontextbezogen anzubieten, zum anderen kann die Ausführung von Diensten korrespondierend zum Kontext angepasst werden.

Schließlich werden Operationen, die der Klasse *Context-triggered Actions* zugezählt werden, vollständig autonom ausgeführt, d.h. sobald ein bestimmter Kontext festgestellt wird (der Benutzer ist nicht involviert). Die Bedingungen dafür werden bei Schilit als einfache *if-then* Regeln spezifiziert [Sch 95].

Neben der hier vorgestellten Klassifizierung von Kontextadaptivität gibt es eine Menge anderer Annäherungen an dieses Gebiet, siehe auch [Pas 98a, WJH 97,

HNBR 97]. Hier wurde die Taxonomie nach Schilit gewählt, da sie der in dieser Arbeit vorgestellten Thematik am nächsten kommt. Insbesondere die Konzeptionen *Context-triggered* und *Contextual Command* sind von Relevanz; siehe auch Kapitel 3.

Des Weiteren soll hier eine Definition nach Dey [Dey 00] vorgestellt werden, welche viele bekannte Definitionen für Kontextadaptivität zusammenfasst und große Verbreitung erlangt hat:

> *Man bezeichnet ein System als kontextadaptiv, wenn es Kontextinformationen benutzt, um dem Benutzer relevante Informationen bzw. Dienste anzubieten. Die Relevanz wird bestimmt durch die gegenwärtige Aufgabe des Benutzers.*

Diese allgemeine Definition wird durch eine Klassifikation von kontextadaptiven Anwendungen erweitert. Man unterscheidet drei Typen:

1. *Die Präsentation von Informationen*
2. *Die automatische Ausführung von Diensten*
3. *Die Speicherung (Tagging) von Kontextinformationen zur späteren Nutzung.*

Die erste Kategorie kombiniert und generalisiert die Konzepte Proximate Selection und Contextual Command nach Schilit [Sch 95]. Als Beispiel: Während sich ein Benutzer durch eine Büroumgebung bewegt, wird ihm fortwährend das Angebot verfügbarer Drucker angezeigt. Der zweite Anwendungsfall stimmt mit dem oben beschriebenen Konzept der Context-triggered Actions überein. Ein Beispiel für den dritten Fall ist die Speicherung aller Druckaufträge eines Benutzers. Sollte der Benutzer nach Dokumenten suchen, welche er vergessen hat abzuholen, kann er vom System dadurch unterstützt werden, dass ihm das System darüber informiert, welches Dokument wo ausgedruckt wurde.

2.3.3 Charakteristika kontextadaptiver Systeme

In Ergänzung zu den vorherigen Abschnitten werden hier noch einige kennzeichnende Eigenschaften kontextadaptiver Systeme erläutert, die für das Verständnis bzw. deren Modellierung unverzichtbar sind. Insbesondere werden folgende Konzepte aus [SG 01] miteinbezogen: das *Lokalitätsprinzip* und das *Prinzip der Zeitnähe*.

Lokalitätsprinzip

Kontextinformationen, wie sie in den vorigen Abschnitten eingeführt wurden, sind in ihrer Bedeutung örtlich eingeschränkt. Sie beziehen sich auf einen bestimmten Ort oder auch ein Raumsegment. Insbesondere im Bereich mobiler Systeme bzw.

eingebetteter Systeme ist der Ort der Nutzung und der Ort, an dem die Informationen entstehen, wesentlich. Der Ort ist immer ein kennzeichnender Parameter einer kontextadaptiven Anwendung. Diese Begebenheit ist offensichtlich bei einer ortsbezogenen Anwendung, wo der Ort einen direkten Einfluss auf das Verhalten eines Systems hat (z.B. Navigationssystem). Auch aus Sicht der kontextadaptiven Dienstnutzung, wie sie im Rahmen dieser Arbeit untersucht wird, kann das Prinzip der Lokalität motiviert werden. Möchte ein Nutzer beispielsweise einen Druckdienst in Anspruch nehmen, so ist es wünschenswert, dass das System zur Ausführung des Auftrags einen Drucker in der nächsten Nähe des Nutzers wählt, falls möglich im selben Raum.

Die Gewinnung von Kontextinformationen ist immer ortsgebunden. Am Ort der Gewinnung ist die Relevanz von Kontextinformationen am höchsten und mit zunehmendem Abstand vom Ursprungsort nimmt die Relevanz i. Allg. ab [SG 01].

Ein anderer Aspekt des Lokalitätsprinzips bezieht sich auf die Messdatenerfassung. Desto näher ein Sensor dem beobachteten Phänomen ist, desto sicherer bzw. störungsfreier sind die Beobachtungsdaten. Wird also - wie es oft der Fall ist - ein Phänomen vermittels multipler Sensoren gleicher Güte erfasst, so ist in der Regel der Sensor mit der kleinsten Entfernung für die Ermittlung von Kontextinformationen am relevantesten.

Um den Anforderungen der Lokalität gerecht zu werden, sollten diese in die Modellbildung eines kontextadaptiven Systems mit einfließen. Im Zusammenhang der kontextadaptiven Dienstnutzung wird diesen Anforderungen mittels eines ortsspezifischen Domänenmodells (siehe Kapitel 3) entsprochen.

Prinzip der Zeitnähe

Komplementär zum Lokalitätsprinzip ist das Prinzip der Zeitnähe zu verstehen. Die Erfassung einer Situation erfordert die Berücksichtigung zeitlicher Abläufe. Insbesondere die Gleichzeitigkeit ist bei der Bewertung und Auswertung von Messdaten von vorrangiger Relevanz. Das Prinzip der Zeitnähe ist ein essentieller Aspekt bei der Konstruktion von Kontextinformationen.

Analog zum Lokalitätsprinzip nimmt die Relevanz von erfassten Daten mit zunehmender zeitlicher Verzögerung ab. In Konsequenz wird das Signal eines Sensors, welches bez. eines Ereignisses mit geringster zeitlicher Verzögerung gemessen wird, vorrangig bei der Gewinnung von Kontextinformationen berücksichtigt. Die Gültigkeit einer Kontextinformation ist i.d.R. begrenzt auf einen beschränkten Zeitraum.

Aus Sicht der Softwaretechnik erfordert das Prinzip der Zeitnähe Echtzeiteigenschaften des kontextadaptiven Systems. So könnte beispielsweise der Einsatz echtzeitfähiger Betriebs- und Kommunikationssysteme erforderlich sein.

2.3.4 Konstruktion von Kontextinformationen

Wie bereits erwähnt, ist die Konstruktion von Kontextinformationen ein essentieller Bestandteil eines kontextadaptiven Systems. Um der Relevanz dieses Aspekts zu entsprechen, sollen im Folgenden zwei repräsentative Ansätze vorgestellt werden. Zunächst wird ein auf Prinzipen der Softwaretechnik basierender Ansatz vorgestellt, das *Context-Toolkit*. Dieses Projekt wurde innerhalb von Forschungsarbeiten im FCE entwickelt.

Der zweite Ansatz nach Schmidt und Laerhoven (siehe [SL 01]) unterbreitet einen generellen Architekturvorschlag zur Gewinnung und Nutzung von Kontextinformationen. Kennzeichnend für diesen Ansatz ist die Vorverarbeitung von Meßdaten mittels statistischer Funktionen.

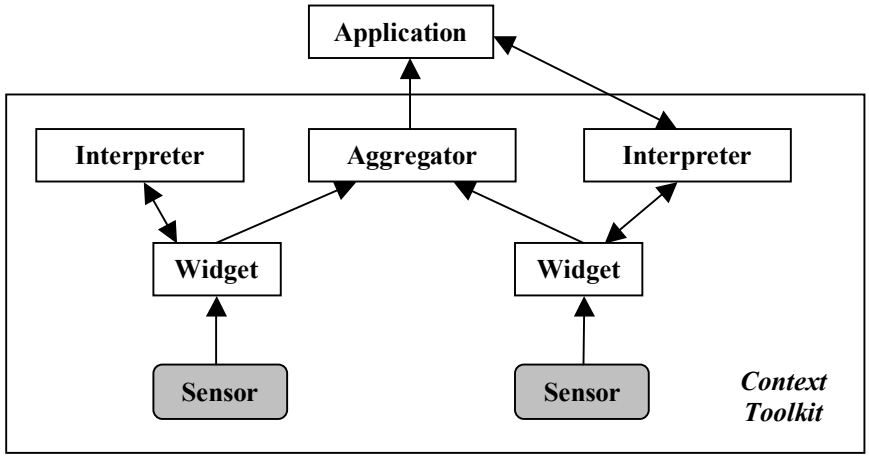

Abbildung 2-14: Architektur des Context-Toolkit

Context-Toolkit

Das Context-Toolkit [SDA 99] ist ein System zur Konstruktion von Kontextinformationen basierend auf Java [Java]. Es wurde bereits zur Realisierung zahlreicher kontextadaptiver Applikationsprototypen eingesetzt. Zu den Anwendungen basierend auf dem Context-Toolkit gehören u.a. folgende (vgl. [CT 01]): Das *In/Out Board*, welches Informationen darüber gibt, wo sich Mitarbeiter in einem Bürokomplex aufhalten. Die Applikation *Information Display*, welche den Aufenthaltsort von Bekannten des Nutzers darstellt. DUMMBO (*Digital Ubiquitous Mobile Meeting Board*), ein Digitalisierbrett [BTA 99], das nach Feststellen einer stattfindenden Diskussion („mehr als zwei Personen vor dem Digitalisierbrett“) automatisch den gesamten Diskussionsvorgang aufzeichnet und kontextadaptive Mailinglisten, mit denen E-Mails an Mitarbeiter gesendet werden, die derzeit im Gebäude sind.

Die Architektur des *Context-Toolkits* basiert auf drei Basiskomponenten [SDA 99]: *Widget*, *Aggregator* und *Interpreter*, siehe Abbildung 2-14. Ein Widget erfasst die von einem Sensor ermittelten Messdaten, z.B. Ortsinformationen bei Einsatz eines GPS-Sensors. *Widgets* bieten anderen Komponenten des Systems die eingelesenen Daten an einer normierten Schnittstelle an, wobei von technologischen Spezifika der eingesetzten Sensoren abstrahiert wird. *Aggregator* Komponenten vererben alle Fähigkeiten eines *Widgets*, können aber zudem die eingehenden Datenströme multipler *Widgets* kombinieren. Weiterhin fassen sie die Kontextinformationen, die mit einer Entität oder Person assoziiert sind, zusammen. *Interpreter* Komponenten wiederum kombinieren die eingehenden Kontextinformationen verschiedener Quellen (*Aggregator* und *Widget* Komponenten), um daraus anwendungsspezifische Informationen abzuleiten. Applikationen können über das HTTP-Protokoll an das Context-Toolkit angebunden werden.

Schmidt und Laerhoven

Abbildung 2-15 zeigt eine Architektur zur Konstruktion von Kontextinformationen nach Schmidt und Laerhoven [SL 01]. Sie wurde insbesondere zur Konstruktion kontextadaptiver Endgeräte entwickelt. Die Architektur unterscheidet folgende Abstraktionsbarrieren: *Sensors*, *Cues*, *Context* und *Applications*.

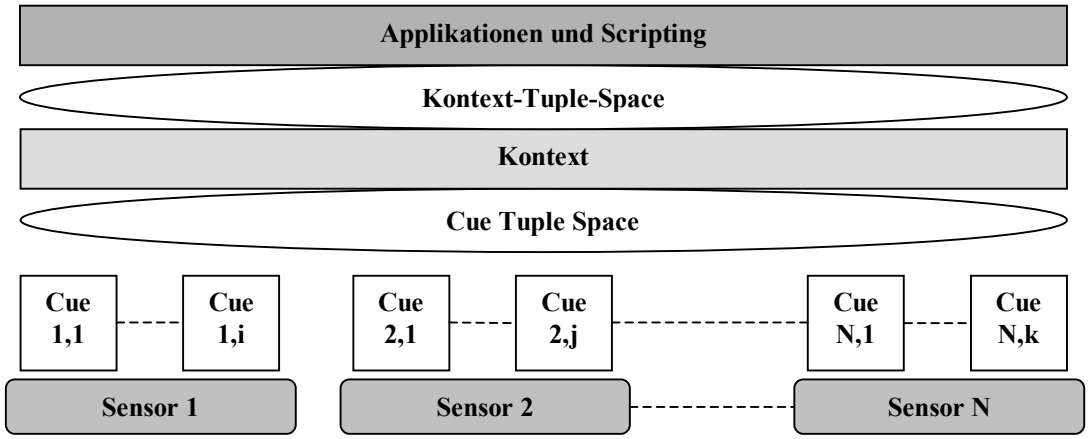

Abbildung 2-15: Geschichtete Architektur für die Konstruktion von Kontextinformationen

Sensors verkörpern hier sowohl so genannte *physische* als auch *logische* Sensoren. Physische Sensoren bezeichnen Hardwareeinheiten (wie etwa Temperatursensoren), die eine bestimmte physikalische Größe messen. Die Daten eines logischen Sensors beziehen sich auf Datenquellen des jeweiligen Endgerätes (z.B. Uhrzeit, GSM-Zelle).

In Abhängigkeit von der Anwendung und der Verteilung der einzelnen Systemkomponenten kann die Kommunikation zwischen den einzelnen Schichten gepuffert erfolgen. Zu diesem Zweck wird das Konzept des Tuple Spaces [RW 96] angewandt; siehe auch Abbildung 2-15.

Das Konzept der *Cues* abstrahiert von den unterliegenden Sensoren; jede *Cue* ist von genau einem Sensor abhängig. Einem einzelnen Sensor (logisch oder physisch) können mehrere Cues zugeordnet werden (1:n). Cues erfüllen maßgeblich folgende Aufgaben: Zum einen abstrahieren sie von Details der unterliegenden Sensor-Hardware (Datenformate, etc.). Dies ermöglicht, dass Sensor-Hardware ausgetauscht werden kann, ohne dass Änderungen in den oberen Schichten der Architektur erforderlich wären. Zum anderen kann das von den Sensoren gelieferte Datenvolumen vor der eigentlichen Verarbeitung reduziert werden.

Zur Datenreduktion werden in [SvL 01] statistische Funktionen vorgeschlagen. Diese können eingesetzt werden, um die anfallenden Daten zusammenzufassen. Zur Charakterisierung einer Messfolge kann beispielsweise kontinuierlich ein gleitender Mittelwert gebildet werden. Die Wahl der Funktion wird durch die geplante Anwendung bzw. durch den eingesetzten Sensor bestimmt. Alternativ zum Mittelwert kann beispielsweise die erste Ableitung (zur Ermittlung von Änderungen) oder die Standardabweichung kalkuliert werden (zur Bestimmung der Stabilität eines Signals).

Die Konstruktion von Kontextinformationen kann als ein Prozess der Datenfusion verstanden werden (vgl. Kapitel 5). Cues dienen in diesem Zusammenhang der Vorverarbeitung von einkommenden Beobachtungsdaten. Aus den resultierenden Daten können dann schrittweise Kontextinformationen zur Beschreibung einer Situation gewonnen werden. In einem System, das gekennzeichnet ist durch eine geringe Anzahl von Sensoren bzw. zu bestimmender Kontexttypen, kann die Ableitung von Kontextinformationen durch einfache Bedingungsregeln gefaßt werden. Beispiel:

```
IF (light is low and acceleration is nonzero):
    KONTEXT_INFORMATION = "MOVING IN THE DARK"
```

Als allgemeineren Ansatz schlagen die Autoren den Einsatz von *Selbstorganiserenden Sensorischen Karten* (SSK) [RMS 92] vor. SSK gehört der Klasse der Cluster-Algorithmen an. Bei der Anwendung dieses Verfahrens erfolgt die Zuordnung von bestimmten Messwertkonfigurationen zu Kontexten in einem überwachten Lernprozess. Zunächst werden die Sensordaten mittels SSK in Cluster unterteilt. Anschließend müssen diese Cluster explizit benannt werden. Der Einsatz von SSKs hat zum einen den Vorteil der Störungsunempfindlichkeit,

und zum anderen kann der Lernprozess zu jedem Zeitpunkt fortgesetzt werden, d.h. es können neue Kontexte „gelernt“ werden.

2.4 Dienstvermittlung

Bereits in Abschnitt 2.1.3 wurde der Gegenstand der Dienstvermittlung innerhalb von Ubiquitous Computing Systemen motiviert. Im Folgenden soll dieser Themenbereich näher erläutert werden.

Zur Klassifikation der Dienstvermittlungsprotokolle sollen hier zunächst einige Begriffe eingeführt werden:

- *Dienst* und *Dienstbeschreibung*
- *Dienstsignatur*
- *Lookup-* und *Discovery-Prozess*

Ein *Dienst* bezeichnet hier einen beliebigen Netzwerkdienst. Die *Dienstbeschreibung* spezifiziert die Funktionalität und die korrespondierende Schnittstelle eines Dienstes. Die Dienstsignatur kann beispielsweise Aussagen bez. QoS oder einer Netzwerkadresse machen. Weiterhin identifiziert eine Dienstsignatur eine Klasse konkreter Dienste und kann so auch zur Lokalisierung von Diensten angewandt werden. Das konkrete Format einer Dienstsignatur bezieht sich auf das jeweilige Dienstvermittlungsprotokoll, so etwa JINI [JINI].

Der *Dienstanbieter* bezeichnet die Ressource oder das Gerät, welches einen Dienst ausführt. Bei einem Diensterbringer kann es sich zum einen um einen konventionellen Computer handeln, wie etwa PCs oder PDAs, aber auch um spezialisierte Netzwerkgeräte, wie etwa Drucker oder Digital-Kamera (siehe auch 2.1.2). Es ist zu beachten, dass ein einzelner Diensterbringer mehrere Dienste anbieten kann. So kann beispielsweise ein Drucker neben Druck- auch Faxaufträge bearbeiten.

Der *Lookup-Prozess* bezeichnet den Vorgang des Auffindens eines beliebigen Objektes (z.B. einer Ressource oder eines Dienstes). Das Auffinden eines Objektes kann entweder anhand eines exakten Namens, einer Adresse, oder anhand eines bestimmten Matchingkriteriums erfolgen. Der Lookup-Prozess ist passiv in der Hinsicht, dass er immer von einer anfragenden Instanz initiiert werden muss. Zudem muss ein dedizierter Verzeichnisdienst existieren, welcher auf einkommende Anfragen reagiert. Ein Lookup-Prozess ist nur in einer statischen Umgebung möglich, die Datenbasis des Verzeichnisdienstes muss nicht modifizierbar sein. Zu den Verzeichnisdiensten, welche den Lookup-Prozess implementieren, gehören folgende Systeme: DNS [M 87], CORBA Naming Service [OMG 99], LDAP [WHK 97].

Im Kontrast zu einem Lookup-Prozess bezieht sich ein *Discovery-Prozess* auf einen spontanen Prozess, welcher das Auffinden von Objekten auch unabhängig

von einem dedizierten Verzeichnisdienst erlaubt. Objekte können ihre Verfügbarkeit anderen Objekten unmittelbar mitteilen und auch das Auffinden von Objekten kann durch direkte Kommunikation zwischen Objekten geschehen. In einem System, welches Discovery-Prozesse unterstützt, können auch immer Lookup-Prozesse abgebildet werden.

Ein Discovery-Prozess zeichnet sich durch folgende Merkmale aus [McG 00]:

- Spontane Lokalisierung und Konfiguration von Diensten und Netzwerkelementen
- Selektion von spezifischen Diensttypen anhand von Dienstsignaturen
- Keine explizite Administrationsinstanz erforderlich
- Automatische Anpassung an sporadische Verfügbarkeit von Objekten (z.B. aufgrund von Mobilität)
- Interoperabilität zwischen unterschiedlichen Systemarchitekturen

Die klassischen Verzeichnisdienste, wie etwa DNS [M 87], unterstützen die meisten der oben genannten Kriterien nicht. Diese zeichnen sich in der Regel durch folgende Eigenschaften aus:

- Dedizierter Verzeichnisdienst
- Explizite Administrationsinstanz erforderlich
- Die Datenbasis des Verzeichnisdienstes reflektiert nicht die gegenwärtige Verfügbarkeit eines Objektes
- Die Registrierung bzw. Deregistrierung von Objekten wird nicht bekannt gegeben (kein Event Mechanismus)

Verzeichnisdienste wie etwa LDAP [WHK 97] oder CORBA Naming Service [OMG 99] unterstützen nicht unmittelbar die spontane Vermittlung von Diensten mittels eines Discovery-Prozesses, dennoch können diese Systeme entsprechend erweitert werden.

Die folgenden Abschnitte erläutern die Funktionsweise von einigen Dienstvermittlungsprotokollen, welche die Dienstvermittlung anhand von Discovery-Prozessen realisieren. Diese Form der Dienstvermittlung entspricht den Anforderungen einer Ubiquitous Computing Umgebung; vergleiche 2.1.3.

2.4.1 Jini

Die folgenden Abschnitte beschreiben die Dienstvermittlung innerhalb von Jini [98]. Jini ist eine Entwicklung der Firma Sun Microsystems™ und basiert auf Java [Java]. Da das ganze System in Java implementiert wurde, ist es erforderlich, dass alle Geräte, die Jini einsetzen wollen, über eine *Java Virtual Machine* verfügen.

Neben der eigentlichen Dienstvermittlung bietet Jini noch eine Menge weiterer Funktionalitäten an. Dazu gehören beispielsweise das Transaktions- und Persistenzmanagement in Verteilten Systemen.

Basiskomponenten

Ein Jini-System besteht aus drei Komponententypen: *Klienten*, *Dienstinstanzen*, und einem *Lookup Service* (LUS). Der LUS implementiert die Dienstvermittlung auf Anfrage eines Klienten und verwaltet das aktuelle Dienstangebot. Die Jini-Spezifikation macht keine expliziten Einschränkungen bez. des eingesetzten Netzwerkprotokolls. Bisher existieren allerdings nur Implementierungen für das TCP/IP-Protokoll. Eine Voraussetzung für den Einsatz von Jini ist ein bereits vollständig konfiguriertes Netzwerk. Damit unterscheidet Jini sich z.B. von Ansätzen wie *Universal Plug and Play* (UPnP) [UPnP] (siehe auch 2.4.2), die auch Aufgaben der Netzwerkkonfiguration wie etwa die dynamische Zuordnung von Netzwerkadressen realisieren.

Die Plattformunabhängigkeit dieser Architektur erlaubt den Einsatz von mobilem Code zwischen den Basiskomponenten eines Jini Systems. Die Kommunikation zwischen verteilten Komponenten in Jini erfolgt durch RMI [RMI].

Dienstregistrierung

Die Dienstregistrierung erfordert zunächst die Lokalisierung eines zuständigen Lookup-Services. Dies erfolgt normalerweise mittels *Multicast* [Stev 94]. Der Klient sendet solange *Discovery-Nachrichten* an eine definierte Multicast-Adresse, bis sich ein geeigneter Lookup-Service meldet. Ist die Adresse des Lookup-Service bekannt, kann die Kommunikation unmittelbar durch einen *TCP Unicast* [Stev 94] erfolgen.

Nach der erfolgreichen Lokalisierung des Lookup Service sendet dieser eine Objektreferenz an den Klienten zurück. Dieses Objekt (*ServiceRegistrar)* implementiert die eine Managementschnittstelle, welche die Registrierung von Diensten ermöglicht. Der bisher beschriebene Vorgang wird im Rahmen von Jini als *Discovery* bezeichnet.

Nach erfolgtem Discovery-Prozess, wird der Dienst registriert. Dazu erzeugt die Dienstinstanz eine Datenstruktur aus einem *Dienst-Proxy-Objekt* und einer assoziierten Dienstbeschreibung, bestehend aus einer Menge von Dienstattributen, so genannten *Entries.* Der Prozess der Dienstregistrierung wird als *Join* bezeichnet.

Lokalisierung eines Dienstes

Die Lokalisierung eines Dienstes erfordert – wie bei der Dienstregistrierung – zunächst das Auffinden eines Lookup-Service. Dazu wird auch der oben beschriebene Discovery-Mechanismus angewandt. Der Klient konkretisiert seine

Anforderungen an den gesuchten Dienst mittels eines so genannten *Service Templates*. Ein Service-Template spezifiziert den Typ eines Dienstes und eine Menge von Attributen. Wird im Lookup-Service ein Diensteintrag gefunden, welcher dem eingehenden Service-Template genügt, so wird das korrespondierende *Dienst-Proxy-Objekt* zum Klienten gesendet. Mittels des Proxies kann der Klient mit dem Dienst kommunizieren (siehe Abbildung 2-16). Der Proxy implementiert eine lokale Dienstschnittstelle für den Klienten. Das Kommunikationsprotokoll zwischen Proxy und Dienst kann frei definiert werden.

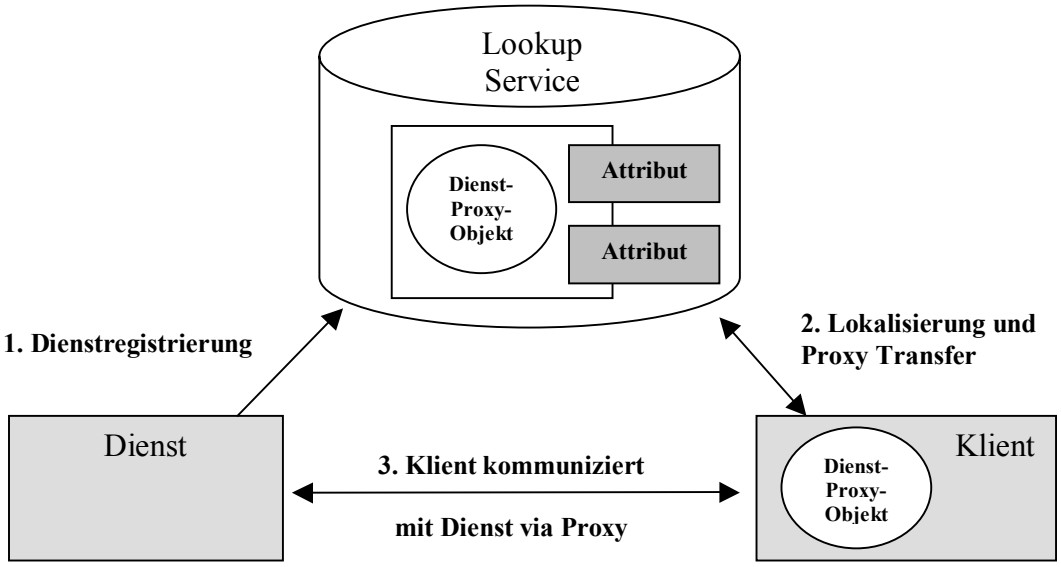

Abbildung 2-16: Dienstvermittlung in Jini

Leasing

Um die Aktualität des vom Lookup-Service verwalteten Dienstangebotes zu gewährleisten, wird das Konzept des *Leasings* angewandt. Leasing basiert auf der Idee, dass Referenzen auf Objekte nur eine vordefinierte Zeitspanne gültig und dann nicht mehr anwendbar sind. Diese Art von Objektreferenzen wird in der Konzeption von Jini als *Leases* bezeichnet. Die Zeitspanne eines Leases kann je nach Bedarf neu verlängert werden. Wird die assoziierte Zeitspanne nicht periodisch erneuert, ist die Referenz auf das Objekt ungültig. Die Referenz kann auch vor Ablauf der definierten Zeitspanne als ungültig markiert werden. Im allgemeinen realisieren Leases eine Methode zur Verwaltung von verteilten Ressourcen. Im Falle des Lookup-Service kann auch dann eine konsistente Datenbasis beibehalten werden, wenn die zukünftige Nichtverfügbarkeit eines Dienstes nicht angegeben wird, d.h. dass der zugehörige Lease nicht explizit gelöscht wird. In diesem Fall läuft die Gültigkeit des Leases ab, da keine

Erneuerung erfolgt. Entsprechend wird der Dienst aus der Datenbasis des Lookup-Services entfernt.

2.4.2 Universal Plug and Play

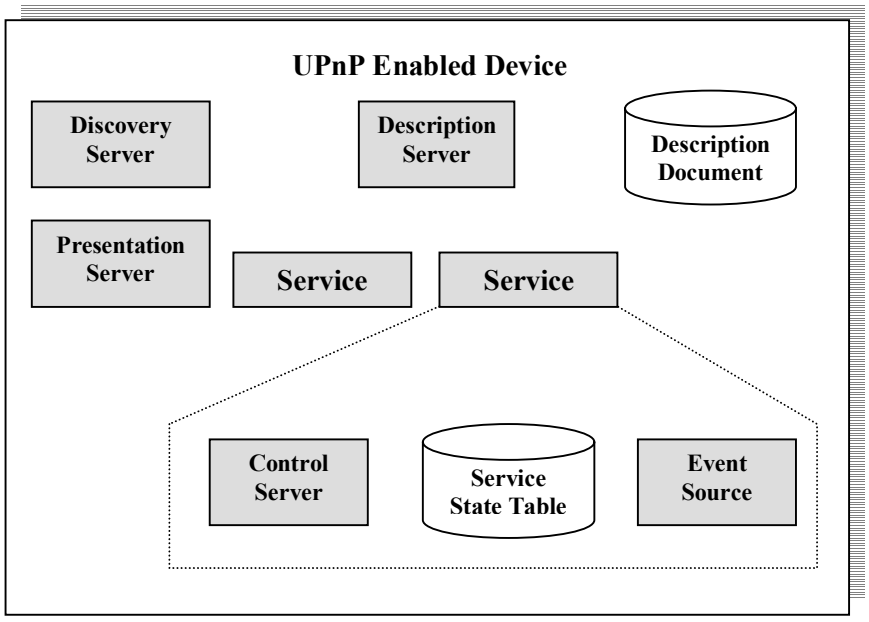

Abbildung 2-17: Modell eines UPnP Device

Basiskomponenten

Universal Plug and Play (UPnP) lässt sich in drei Schlüsselkonzepte gliedern. Es wird unterschieden zwischen UPnP-*Devices*, wie etwa einem Drucker oder eine Kamera, und den zugehörigen *Services*. Ein einzelnes Device kann einen oder mehrere Dienste anbieten. Diese Dienste werden gesteuert (*controlled)* oder angesprochen durch einen Web Browser oder eine beliebige Applikation, welche den *User Control Point* (UCP) Standard implementiert. Um auch Geräte zu integrieren, die nicht den UPnP Standard erfüllen, existiert ein *Bridge-Konzept*.

UPnP Devices

Ein UPnP-Device ist eine Dienstinstanz, welche sich durch einen Satz von speziellen Software-Komponenten auszeichnet; siehe auch Abbildung 2-17. Der *Discovery Server* wird eingesetzt, um das Dienstangebot im Netzwerk bekannt zu machen (*Service Anouncement*). Auf das *Description Document*, welches die Funktionalität des Dienstes beschreibt, kann mittels des *Description Servers* zugegriffen werden. Ein so genannter *Presentation Server* implementiert die Schnittstelle zu einem Web Browser. Weiterhin bewerkstelligt der *Control Server*

die Steuerung des UPnP-Gerätes. Die Statusinformationen eines UPnP-Dienstes werden im *Service State Table* verwaltet. Die Statuseinträge können zum einen durch den *Control Server* und zum anderen durch die *Event Source* Komponente manipuliert werden. Die *Service State Table* enthält vielfältige Zustandsinformationen, neben dem logischen Zustand des Gerätes können auch Informationen zum physikalischen Zustand enthalten sein (z.B. Batteriespannung).

Die Datenvermittlung zwischen verteilten UPnP-Komponenten erfolgt mittels HTTP.

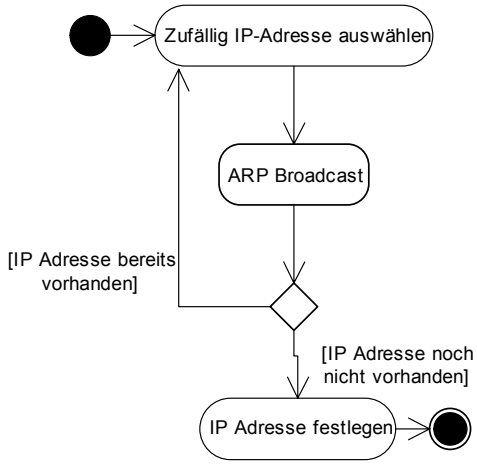

Abbildung 2-18: AutoIP

Spontane Vernetzung

Im Kontrast zu Jini (siehe 2.4.1) unterstützt UPnP die spontane Vernetzung von Geräten in einem Ad-hoc-Netzwerk. Zu diesem Zweck unterstützt UPnP folgende Standards: *DHCP* [DL 02], *AutoIP* [UPnP] und *Multicast DNS* [MDNS].

Die Integration eines neuen UPnP Gerätes erfolgt in drei Phasen. Zunächst sendet das Gerät einen *DHCP Broadcast* [DL 02], um festzustellen, ob ein DHCP Server im Netzwerk vorhanden ist. Ist dies nicht der Fall, wird dem neuen Gerät eine Adresse mittels AutoIP zugewiesen. Bei diesem Verfahren wird eine IP Adresse dynamisch erzeugt, ohne dass dafür ein spezieller Server erforderlich ist; siehe auch Abbildung 2-18.

Schließlich implementiert Multicast DNS einen dezentralen DNS Dienst. Das heißt, auch hier ist kein expliziter DNS-Server erforderlich, womit die Zuordnung von symbolischen Namen in einem abgeschlossenen Netzwerk ohne Zugang an das Internet erfolgen kann.

Lokalisierung eines Dienstes

Zur Lokalisierung eines Dienstes initiiert der *Discovery Client* – eine Komponente des *User Control Points* – eine Anfrage mittels des *Simple Service Discovery Protocols* (SSDP) [UPnP]. SSDP ist ein HTTP-basiertes Dienstvermittlungsprotokoll, welches das Auffinden von Diensten erlaubt. Ähnlich wie bei Multicast DNS und AutoIP (siehe voriger Abschnitt) ist auch hier kein dedizierter Server optional. Zudem erfordert das Protokoll keine explizite Administrierung. Da auch SSDP auf einer Multicast-Strategie basiert und daher nur begrenzt skalierbar ist, wird es vorzugsweise in kleinen Netzen (z.B. Home Networks) angewandt.

Eine Dienstanfrage kann sich entweder auf ein konkretes Gerät beziehen oder auf eine bestimmte Geräte- oder Dienstklasse. Geräte, die der Anfrage entsprechen, senden ihren assoziierten *Uniform Resource Locator* [Ber 96] zurück. Dieser verweist auf den *Description Server* des jeweiligen Gerätes. SSDP unterstützt ein Leasing Konzept, welches die Verwendbarkeit dieser Verweise temporal einschränkt (vgl. Jini).

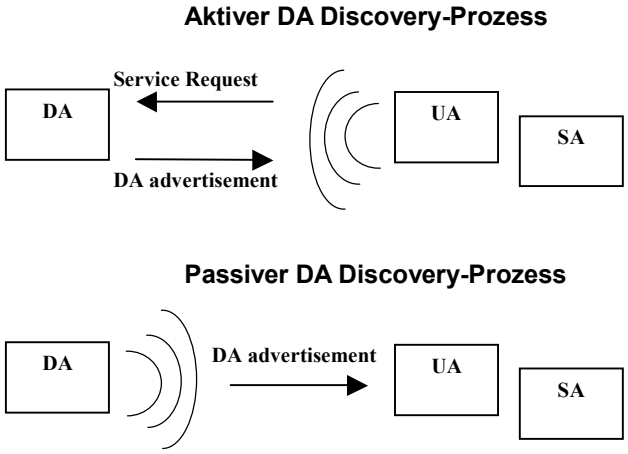

Abbildung 2-19: Aktiver/passiver DA Discovery-Prozess

2.4.3 Service Location Protocol

Das *Service Location Protocol* (SLP) [Gutt 99, RFC 2165, RFC 2608] ist ein Vorschlag der Internet Engineering Task Force (IETF), um die Dienstvermittlung in IP-basierten Netzwerken durchzuführen. SLP definiert sowohl die Protokolle zwischen den Systemkomponenten, als auch Spezifikationen zur Beschreibung von Diensten. In Kontrast zu Jini und in Analogie zu UPnP ist SLP nicht an einen bestimmten Sprachstandard gebunden.

Basiskomponenten

SLP definiert eine abstrakte Architektur aus drei unterschiedlichen Komponententypen: *User Agents*, *Service Agents*, *Directory Agents*. Der User Agent (UA) realisiert die Dienstsuche im Auftrag eines Klienten. Service Agenten (SA) verbreiten die Nachricht über das Vorhandensein eines Dienstes im Netzwerk (*service advertise)*. Zudem vermittelt der Service Agent auch eine Dienstbeschreibung. Der Directory Agent (DA) hat die Funktion eines zentralen Dienstverzeichnisses und aggregiert eingehende Dienstangebote. In SLP ist auch der Einsatz von multiplen Directory Agenten möglich. Zum einen besteht dann die Möglichkeit zur Replikation von Dienstdaten, um die Systemleistung zu optimieren (bez. der Fehlertoleranz). Zum anderen kann eine Hierarchie oder ein Graph von Dienstdomänen gebildet werden.

SLP kann in diversen Konfigurationen eingesetzt werden. So wird beispielsweise auch ein Modus ohne verfügbaren Directory Agent unterstützt. In diesem Fall werden Discovery Prozesse angewandt, die auf den bereits beschriebenen Multicast Strategien beruhen.

Es ist zu beachten, dass die Effizienz der Dienstvermittlung bei Vorhandensein von DAs besser ist, da die Kommunikation zwischen Komponenten dann vermittels Unicast erfolgt.

Lokalisierung eines Dienstes

Ein *User-Agent*, der einen bestimmten Dienst sucht, initiiert zunächst einen auf Multicast basierenden Discovery-Prozess zur Lokalisierung eines zuständigen Directory Agents. SLP unterstützt zu diesem Zwecke zwei unterschiedliche Strategien, man unterscheidet zwischen einem so genannten *aktiven* und *passiven* Discovery-Prozess; siehe auch Abbildung 2-19. Innerhalb des aktiven Discovery-Prozesses wird zunächst eine Multicast Nachricht von einem UA oder SA gesendet. Ein oder mehrere DAs können dann auf diese Frage antworten. Der passive Discovery-Prozess wird vom DA durch einen Multicast Aufruf initiiert. Dieser Aufruf wird periodisch wiederholt, für den Fall, dass er nicht von SAs/UAs empfangen werden konnte. SLP unterstützt auch die Lokalisierung mittels eines DHCP-Servers [DL 02].

Ein Dienst registriert sich über den Service Agent beim DA, indem eine Registrierungsmeldung gesendet wird. Individuelle Diensteinträge im DA haben eine maximale Gültigkeitsdauer von 18 Stunden, daher muss die Registrierung periodisch verlängert werden. Erfolgt keine Verlängerung vom Service Agent, wird der entsprechende Diensteintrag gelöscht. Mit diesem Verfahren wird erreicht, dass das vom DA verwaltete Dienstangebot nur für begrenzte Zeiträume inkonsistent ist (also nicht mit dem tatsächlichen Dienstangebot übereinstimmt).

Eine Lookup-Prozess wird vom User Agent zum bereits lokalisierten DA gesendet. Die Dienstanfrage enthält einen Suchfilter (vergleiche *Lightweight Directory Access Protocol Version 3* [How 97]), welcher das angeforderte Dienstprofil spezifiziert. Der DA antwortet mit einer Nachricht, welche eine Liste von Dienst Referenzen – Uniform Resource Locators (URLs) – enthält. Über die URLs können die Dienste direkt angesprochen werden.

Weiterhin besteht die Möglichkeit, die Dienstvermittlung ohne dedizierten DA auszuführen, diese erfolgt dann über eine Multicast-Strategie (vergleiche 2.4.2, UPnP).

	JINI	**UPnP**	**SLP**
Dienstbeschreibung	Java Attributes (object oriented matching)	XML	SLP Templates (String based Attributes)
zentral/dezentral	√ / -	√ / √	√ / √
Skalierbarkeit	+	+	++
Sprachspezifisch	√	-	-
Ad-hoc-Netzwerke	-	√	-

Tabelle 2-2: Vergleich von Dienstvermittlungsprotokollen

2.4.4 Vergleich

Die oben vorgestellten Dienstvermittlungsprotokolle sollen nun verglichen werden. Tabelle 2-2 macht Angaben bez. der verwendeten Dienstbeschreibung, ob das System mit einer (oder mehreren) zentralen Dienstvermittlungsinstanzen betrieben wird, oder ob das System auch einen dezentralen Modus unterstützt. Zudem wird eine qualitative Angabe zum Skalierbarkeitsaspekt des jeweiligen Protokolls gemacht.

Der Aspekt „*Sprachspezifisch"* ist von Relevanz, da dieser Parameter die Einsetzbarkeit eines Protokolls einschränken kann. So erfordert Jini den Einsatz einer *Java Virtual Machine* (JVM) auf jedem teilnehmenden Endgerät. Doch gerade bei Eingebetteten Systemen können die damit verbunden Systemanforderungen (bez. Speicher, CPU Leistung) oft nicht erfüllt werden.

Schließlich untersucht der Punkt *Ad-hoc-Netzwerke,* ob das verwendete Protokoll ein bereits konfiguriertes Netzwerk zur Operation voraussetzt.

Im Rahmen dieser Arbeit wird zur Dienstvermittlung SLP eingesetzt, obschon sich auch andere Dienstvermittlungsprotokolle einsetzen lassen. Somit wird auch der

zukünftigen Evolution von Dienstvermittlungsprotokollen Rechnung getragen. Probleme der Skalierbarkeit, wie sie beispielsweise in [CZHK 99] untersucht wurden, sind für das hier vorgestellte System nicht vorrangig relevant, da die Dienstvermittlung nur lokal begrenzt zum Einsatz kommt. Dieses Lokalitätsprinzip hat Einfluss auf die gesamte Systemarchitektur und wird in Kapitel 3 näher erläutert.

2.5 Lokale Netzwerktechnologien

Wie bereits erwähnt ist die Möglichkeit der spontanen Vernetzung von Ressourcen in einer Ubiquitous Computing Umgebung von zentraler Bedeutung. Wie in 2.1.3 beschrieben, erstreckt sich der Prozess der spontanen Vernetzung über mehrere Ebenen der Kommunikationshierarchie eines Verteilten Systems: die *Infrastruktur-Ebene* (siehe 2.4, Dienstvermittlung), die *Dienst-Ebene* und die *Netzwerk-Ebene*. Im Folgenden soll auf Aspekte der Netzwerk-Ebene eingegangen werden. Zu diesem Zweck werden zwei lokale Netzwerktechnologien vorgestellt: *Bluetooth* [Blue] und *IrDa* [IrDa].

Modus	Bandbreite
asynchron-symmetrischer Datenkanal	**433,9 kbit/s**
asynchron-asymmetrischer Datenkanal	**732,2 / 57 kbit/s**
synchroner Sprachkanal	**64 kbit/s**

Tabelle 2-3: Bluetooth Bandbreiten

2.5.1 Bluetooth

Die Entwicklung von Bluetooth [Blue] wurde 1998 von den Firmen Ericsson, Intel, IBM, Nokia und Toshiba initiiert. Es handelt sich dabei um eine lokale Netzwerktechnologie zur spontanen Vernetzung von Endgeräten. Aus technischer Sicht ist das Ziel, eine kostengünstige funkbasierte Netzwerktechnologie zur Verfügung zu stellen, welche auf einer Ein-Chip-Lösung basiert.

Bluetooth lässt sich insbesondere in folgenden Einsatzgebieten verwenden:

- Anbindung von Peripheriegeräten („Kabel ersetzen“)
- Unterstützung von lokalen Ad-hoc-Netzwerken
- Anbindung von mobilen Klienten an Backbone Netze

Bluetooth benutzt ein Frequenzband im 2,4 GHz-Bereich und erlaubt sowohl Punkt-zu-Punkt-, als auch Punkt-zu-Multipunkt-Verbindungen. Um die Kommunikation robuster und abhörsicherer zu machen, setzt Bluetooth ein Frequenz-

sprungverfahren (*Frequency Hopping*) ein. Die Reichweite bewegt sich zwischen 10 cm und 10 m bei einer Sendeleistung von 1mW. Alternativ kann Bluetooth auch mit einer Reichweite von 100 m eingesetzt werden, dann beträgt die Sendeleistung 100 mW. Es werden sowohl Sprach- als auch Datendienste angeboten.

Sprachverbindungen benötigen symmetrische, leitungsvermittelte Punkt-zu-Punkt-Verbindungen. Im Kontext von Bluetooth werden diese als *Synchronous Connection-Oriented Links* (SCO) bezeichnet. Die Kommunikation von Daten erfolgt über symmetrische oder asymmetrische Punkt-zu-Punkt- bzw. Punkt-zu-Mehrpunkt-Verbindungen. Die unter der Bezeichnung *Asynchron Connectionless Link* (ACL) zusammengefassten Kommunikationsmodi erfolgen paketvermittelt. Tabelle 2-3 zeigt die Bandbreiten der einzelnen Kommunikationsmodi.

Eine Gruppe kommunizierender Geräte bildet ein so genanntes *Pico-Netz*. Maximal acht Geräte können in einer solchen Struktur miteinander kommunizieren. Innerhalb eines *Pico-Netzes* existiert immer ein *Master-Knoten*, die übrigen werden als *Slaves* bezeichnet. Der Knoten, welcher die Kommunikation initiiert, hat die Rolle des Masters. Mittels des Masters wird das Frequenzsprungverfahren bzw. die internen Zeitgeber teilnehmender Knoten eines Pico-Netzes synchronisiert.

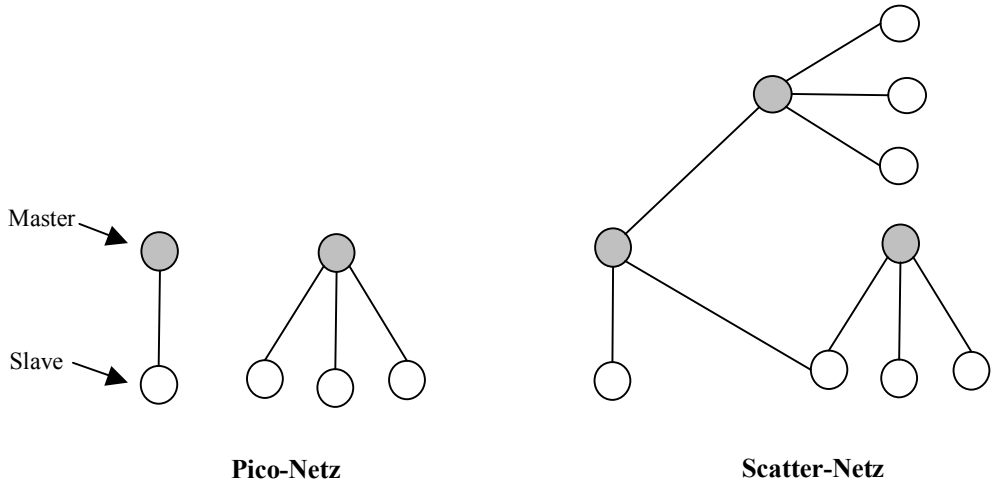

Abbildung 2-20: Pico- und Scatter-Netze

Mehrere Pico-Netze können innerhalb eines so genannten *Scatter-Netzes* zusammengeschlossen werden; vgl. Abbildung 2-20. Jedes Pico-Netz verfügt über einen dedizierten Master-Knoten. Die konstituierenden Pico-Netze eines Scatter-Netzes sind nicht synchronisiert, dennoch können Knoten aus beliebigen Pico-Netzen miteinander kommunizieren. Bez. eines anderen Pico-Netzes kann ein Master-Knoten die Rolle eines Slaves einnehmen.

Bluetooth Protocol Stack

Abbildung 2-21 gibt eine Übersicht über die Spezifikation des *Bluetooth Protocol Stacks* [HMNS 01, Blue]. Die Spezifikation berücksichtigt neben den Bluetooth-spezifischen Protokollen auch bereits existierende Protokollstandards, welche in Verbindung mit Bluetooth eingesetzt werden können. Dazu gehören beispielsweise OBEX [HMNS 01] und TCP/IP [Stev 94].

Die Pfeile in Abbildung 2-21 geben die Sichtbarkeit der einzelnen Protokollschichten aus Perspektive der Applikationsebene an. Dienste dieser Schichten können von Applikationen direkt angesprochen werden. Es ist zu beachten, dass Applikationen vorwiegend Protokolle der oberen Schichten verwenden, um so einen höheren Kompatibilitätsfaktor zu erreichen.

Die unteren Schichten – *Bluetooth Baseband* und *Radio* – definieren das physikalische Übertragungsschema (Bandbreite, Signaldefinitionen) und die Datenkodierung (Paketformate).

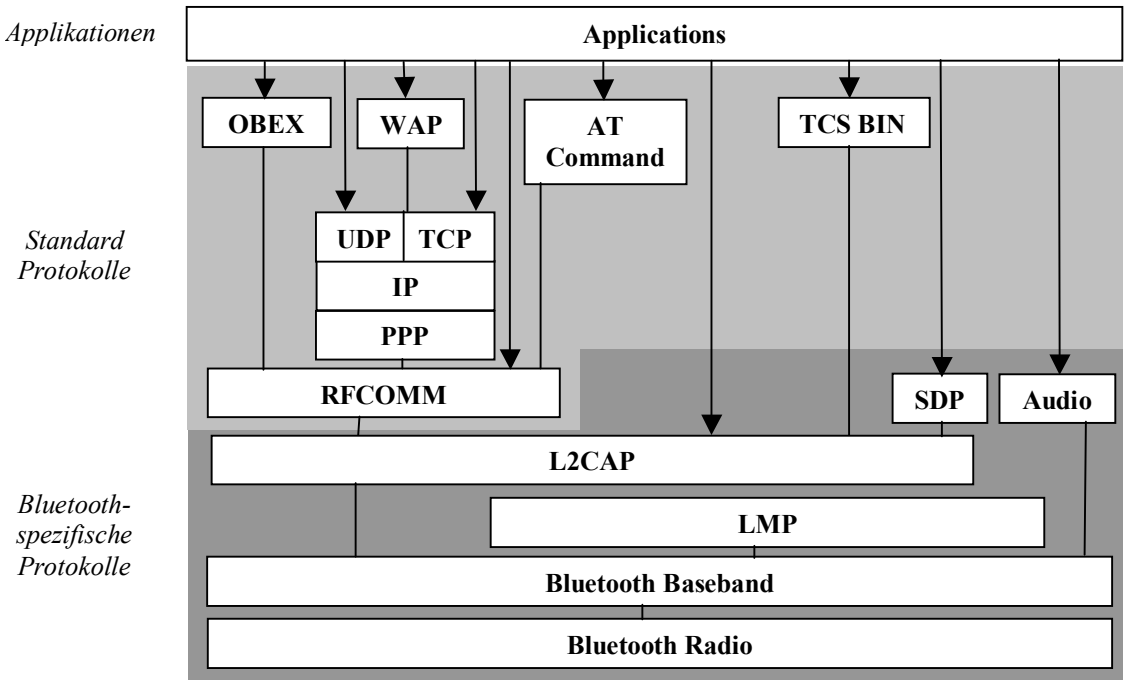

Abbildung 2-21: Bluetooth Protocol Stack

Das *Link Manager Protocol* (LMP) definiert Nachrichten, die zum Aufbau und zur Erhaltung von Verbindungen zwischen Knoten erforderlich sind. Auch Sicherheitsaspekte, wie etwa Verschlüsselung und Authentifizierung, werden innerhalb von LMP implementiert.

Das *Logic Link Control and Adaptation Protocol* (L2CAP) wird von den oberen Schichten zum Transport von Daten eingesetzt und kann als Vermittlungsinstanz zwischen den oberen Anwendungsprotokollen und den unteren Protokollschichten verstanden werden. L2CAP realisiert u.a. die Segmentation bzw. Desegmentation von Datenpaketen und transportiert *Quality of Service* (QoS) Parameter. L2CAP bietet den oberen Schichten sowohl verbindungsorientierte als auch verbindungslose Dienste an. L2CAP implementiert einen sicheren Datentransport für Datenblöcke mit einer Größe von maximal 64 Kilobyte. Zusammenfassend kann festgestellt werden, dass L2CAP einen Teil der *Data Link Layer* Funktionalität implementiert, wie auch das schon erwähnte LMP und die unterstützten Sprachdienstprotokolle (*Audio*).

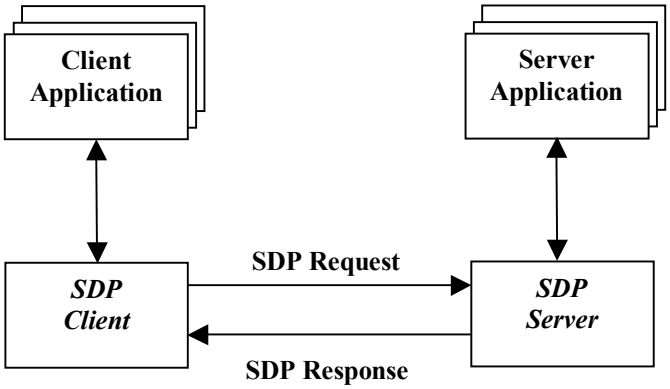

Abbildung 2-22: ***Service Discovery Protocol*** **(SDP)**

Eine zentrale Bedeutung hat das *Service Discovery Protocol* (SDP), welches Bluetooth bez. anderer WLAN-Standards auszeichnet. SDP implementiert ein Dienstvermittlungsprotokoll im Sinne von Abschnitt 2.4. Analog zu Jini [JINI] oder UPnP [UPnP] erlaubt SDP die Lokalisierung von Diensten, die durch andere Knoten/Endgeräte in Reichweite angeboten werden.

Pico-Netze können von nachbarschaftlich angeordneten Knoten spontan gebildet werden. Bei Aufnahme eines neuen Knotens, wird das Gesamtdienstangebot des Pico-Netzes um die Dienste dieses Knotens erweitert. Verlässt hingegen ein Knoten ein Pico-Netz, so vermindert sich das Dienstangebot um die von diesem Knoten dargebotenen Dienste. Es ist auch möglich, dass ein bereits registrierter Knoten sein Dienstangebot dynamisch modifiziert. Das Dienstangebot des Pico-Netzes steht jedem einzelnen Knoten zur Verfügung.

Zur Verwaltung eines Dienstangebotes realisiert Bluetooth SDP folgende Operationen:

- Suche von Diensten bez. einer vordefinierten Dienstklasse
- Suche von Diensten anhand von Dienstattributen

- Auflistung aller verfügbaren Dienste (*service browsing*)

Ein so genannter *SDP Server* verwaltet das Dienstangebot eines jeden Knotens und verarbeitet eingehende Suchanfragen; siehe Abbildung 2-22. Für jeden Dienst verwaltet der SDP Server einen individuellen Eintrag. Die Beschreibung eines Dienstes erfolgt über Dienstattribute. Die Bluetooth Spezifikation [Blue] definiert u.a. folgende Attributtypen: *Service Class*, *Provider Name*, *Service ID*, *Icon URL*.

Das *RF Communication Interface* (RFCOMM) implementiert die Emulation einer RS-232 konformen seriellen Schnittstelle. RFCOMM bietet zum einen einheitliche Schnittstelle für Geräte mit seriellem Anschluss, wie etwa Drucker oder Modems, zum anderen kann RFCOMM auch genutzt werden zur Kommunikation zwischen Bluetooth-fähigen Geräten. RFCOMM unterstützt die gleichzeitige Nutzung von bis zu 60 Verbindungsports [Blue].

Die *Telephony Control Protocol Specification – Binary* (TCS BIN) definiert Signalisierungsnachrichten zum Aufsetzen einer Sprachverbindung zwischen Bluetooth-Geräten. Die *Advanced Telephony Commands* (AT) definieren ebenfalls Direktiven zur Steuerung von Telefonverbindungen. Insbesondere werden AT-Kommandos zur Ansteuerung von Modems und Fax-Geräten benutzt.

Die übrigen Protokollkomponenten basieren alle auf der RFCOMM-Schnittstelle, siehe auch Abbildung 2-21. Dazu gehören TCP/IP [Stev 94] und das *Object Exchange Protocol* (OBEX)[HMNS 01]. OBEX (s.o.) wurde von der *Infrared Data Association* (IrDA) entwickelt. Es definiert einen Standard zum Austausch von Datenobjekten (z.B. Digitale Visitenkarte) zwischen Geräten über eine Infrarotschnittstelle.

Mittels IP kann auch das *Wireless Application Protocol* (WAP) [WAP 02] in Bluetooth eingebunden werden.

2.5.2 IrDA

Die *Infrared Data Association* (IrDA) [IrDA] hat einige Standards zur Datenübertragung mittels Infrarotlicht definiert. Die hohe Akzeptanz der IrDA-Standards hat zur Existenz eines breiten Spektrums unterstützter Hard- und Softwareplattformen geführt. Weltweit existieren derzeit ca. 150 Millionen Einheiten, die eine IrDA-konforme Datenkommunikation implementieren. Dazu gehören PCs, PDAs, Handys, Drucker und Kameras. Insbesondere wird IrDA auch in der Medizintechnik eingesetzt, da die infrarotlichtbasierte Datenkommunikation keine Störungen bei anderen elektronischen Geräten hervorruft.

Signale funkbasierter LANs breiten sich in alle Richtungen des umgebenden Raum aus, man spricht in diesem Zusammenhang auch von einer 360° Abdeckung. Bei der Verwendung von gerichteten Infrarotlichtquellen, wie etwa LEDs oder Laserdioden, breiten sich Signale in einem Konus von der Lichtquelle

aus, die Abdeckung liegt bei etwa 30°[5]. Diese Eigenschaft ermöglicht so genannte *Point-and-Shoot* Anwendungen. Möchten etwa zwei Personen ihre digitalen Visitenkarten austauschen, so ist dies möglich, indem die jeweiligen Geräte (z.B. PDA) aufeinander gerichtet werden. Umstehende Personen sind von diesem Kommunikationsvorgang ausgeschlossen, da sie sich nicht in der Abdeckung befinden.

Bluetooth und andere drahtlose LANs bilden ommnidirektionale Netzwerke (360° Abdeckung). Alle Geräte in Netzreichweite sind potentielle Kommunikationsendpunkte. Daher wäre zur Realisierung des oben angeführten Szenarios ein spezieller Selektionsprozess erforderlich [HMNS 01]. Ein Kommunikationsendpunkt könnte beispielsweise durch kennzeichnende Attribute bestimmt werden.

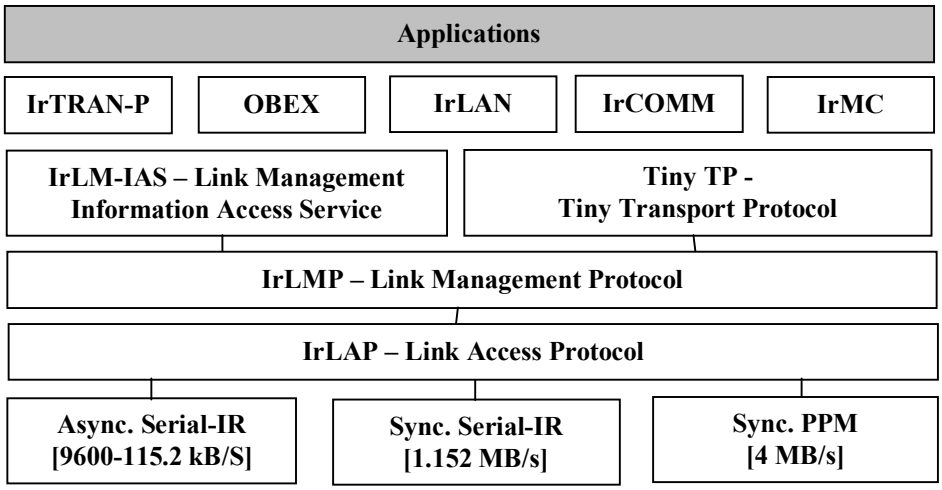

Abbildung 2-23: IrDA Protocol Stack

Eine vielversprechende Kombination bildet die Kombination von gerichteter (IrDA) und ungerichteter Kommunikation (Bluetooth). Vermittels gerichteter Kommunikation kann zunächst ein Kommunikationsendpunkt selektiert werden, die anschließende Kommunikation könnte dann per ungerichteter Kommunikation erfolgen. So kann ein einmal initiierter Kommunikationsvorgang auch dann fortgesetzt werden, wenn die Endpunkte nicht mehr im Abdeckungsbereich des gerichteten Netzwerkes liegen.

[5] Der Abdeckungswinkel wird durch die eingesetzte Lichtquelle determiniert. Es ist auch eine diffuse Signalausbreitung möglich.

IrDA Protocol Stack

Abbildung 2-23 vermittelt eine Übersicht über den *IrDA Protocol Stack*. Die von der IrDA definierte Physische Übertragungsschicht (*Physical Layer)* sieht drei verschiedene Kommunikationsmodi vor:

- Asynchrone Übertragung mit einer Bandbreite zwischen 9600 bps und 115.2 kbps.
- Synchrone Übertragung mit einer Bandbreite von 1.152 Mbps
- Synchrone Übertragung mit Pulsmodulation, *Pulse Position Modulation* (PPM), Bandbreite: 4 Mbps.

Die Reichweite von typischen Implementierungen variiert zwischen 0 und 2m. Bei energiesparenden Implementierungen liegt die maximale Reichweite zwischen 20 und 30 cm.

Das *Infrared Link Access Protocol* (IrLAP) bietet den oberen Schichten sowohl verbindungsorientierte als auch verbindungslose Datendienste an. Die Übertragung erfolgt gesichert.

Das *Infrared Link Management Protocol* (IrLMP) realisiert die Ad-hoc-Datenkommunikation zwischen IrDA-konformen Endgeräten. Zudem bietet IrLMP Multiplex-Funktionen, so dass mehrere Applikationen über einen einzigen Datenkanal miteinander kommunizieren können. Alternativ kann eine einzige Applikation auch exklusiven Zugriff auf den Datenkanal erwerben.

Neben den vorgestellten Basisprotokollen definiert IrDA auch einige Anwendungsprotokolle (siehe Abbildung 2-23):

- *Tiny TP* ist ein Transportprotokoll basierend auf IrLMP. Unter anderem verwaltet Tiny-TP die Segmentation bzw. Desegmentation von Datenpaketen. Dadurch kann die Komplexität von aufsetzenden Applikationen vermindert werden.
- *IrCOMM* implementiert eine Emulation für parallele und serielle Netzwerkschnittstellen (vgl. Bluetooth). Bereits existierende Anwendungen können dann ohne Änderungen übernommen werden.
- *IrLAN* bietet eine Schnittstelle zur Realisierung eines IrDA-basierenden LAN.
- *IrTRAN-P* ist ein Anwendungsprotokoll zum Transport von Bilddaten, beispielsweise von einer digitalen Kamera zu einem anderen Endgerät.
- *IrMC* ist ein Protokoll zum Austausch vom Telefon- und anderen Kommunikationsdaten.

IrOBEX

Das *Infrared Object EXchange Protocol* (IrOBEX) ist ein generelles Objekt zum drahtlosen Austausch von Datenobjeken (z.B. digitalen Visitenkarten). OBEX hat im Bereich der drahtlosen Ad-hoc-Kommunikation zunehmend an Relevanz gewonnen. Wie bereits erwähnt, wurde dieses Austauschformat auch in die Bluetooth-Spezifikation aufgenommen.

Wie in Abbildung 2-23 zu sehen, ist OBEX ein Anwendungsprotokoll, welches auf den unterliegenden Protokollschichten basiert. OBEX unterstützt insbesondere drei Anwendungsszenarien: die Synchronisation von Datenbeständen, die vom Sender initiierte Übertragung von Datenobjekten (*Object Push*) und den allgemeinen File-Transfer.

OBEX ist ein sitzungsbasiertes Protokoll mit einem binären Übertragungsmodus. Das OBEX-Protokoll hat Ähnlichkeiten mit dem textbasierten HTTP-Protokoll. Neben dem eigentlichen Datenobjekt enthält eine OBEX-Nachricht auch Metainformationen, die der Beschreibung bzw. Klassifikation des Datenobjektes dienen. Zu diesem Zweck definiert OBEX eine Reihe von Headern, die zur Klassifikation von Daten eingesetzt werden können. Die Menge der vordefinierten Header kann jederzeit um neue Header ergänzt werden. Zudem können auch HTTP-konforme Header eingesetzt werden.

Das Sitzungskonzept von OBEX realisiert sowohl verbindungslose als auch verbindungsorientierte Kommunikationsdienste. Aspekten der Sitzungssicherheit wird durch Protokollprimitive der Authentifikation bzw. Verschlüsselung Rechnung getragen.

2.6 Zusammenfassung

Dieses Kapitel hat Grundzüge vermittelt, welche für das Verständnis des Ubiquitous Computing Paradigma wesentlich sind. Zuerst wurde dieser Forschungsbereich bez. verwandter oder grundlegender Bereiche untersucht. Insbesondere wurde der Bezug zu den Forschungsrichtungen der Verteilten Systeme, der Mobilen Systeme und der Smart Spaces hergestellt. Ergebnisse aus diesen Gebieten bilden einen relevanten Beitrag bei der Entwicklung von Ubiquitous Computing Systemen, obschon die Berücksichtigung der veränderten Rahmenbedingungen teils eine neue Interpretation erfordert. Dazu zählen insbesondere generelle Aspekte der Verteilung, Sicherheitsaspekte und auch das Management (vgl. 2.2.1).

Ansätze für die Unterstützung mobiler Systeme wurden in 2.2.2 vorgestellt und bilden einen immanenten Bestandteil bei der Untersuchung bzw. Entwicklung von Ubiquitous Computing Systemen, da Aspekte der Mobilität einen primären Anspruch bilden. Infolgedessen können viele Konzepte übernommen werden.

Adaptionsstrategien (2.2.2.3) spielen diesbezüglich eine übergeordnete Rolle, da sie in der Konzeption der hier präsentierten Arbeit mit einfließen.

Schließlich erfolgte eine Darstellung von Arbeiten im Bereich der so genannten Smart Spaces (2.2.3). Hier wurde insbesondere auf Aspekte eingegangen, welche die Thematik Ubiquitous Computing von den Charakteristika Verteilter und Mobiler Systeme abgrenzt, wozu beispielsweise die Ad-hoc-Vernetzung von autonomen Netzwerkelementen gehört.

Aspekte der Kontextadaptivität, der Dienstvermittlung und lokalen Netztechnologien wurden in gesonderten Abschnitten behandelt, da sie für die in dieser Arbeit entwickelte Konzeption von vorrangiger Relevanz sind.

Das anschließende Kapitel beschreibt einen neuen Ansatz, der eine kontextadaptive Dienstvermittlung in Ubiquitous Computing Umgebungen erlaubt.

Kapitel 3

Ein System zur kontextadaptiven Dienstnutzung

Im Folgenden wird ein neuer Ansatz zur Beschreibung und Implementierung von kontextadaptiven Anwendungen vorgestellt. Grundlegend für diesen Ansatz ist das Konzept der *kontextadaptiven Dienstnutzung* in einer Ubiquitous Computing Umgebung. Die kontextadaptive Dienstnutzung wird hier als Obergriff für die kontextadaptive *Selektion* und *Ausführung* von Diensten verstanden.

Die kontextadaptive *Selektion* erweitert die grundlegenden Techniken der Dienstvermittlung, wie sie in 2.4 erläutert wurden, insofern, dass die Vermittlung nicht ausschließlich durch die Spezifikation von gewünschten Dienstattributen erfolgt, sondern auch kontextuelle Einschränkungen Berücksichtigung finden. Auch die *Ausführung* bzw. Nutzung eines Dienstes kann innerhalb des hier erläuterten Verfahrens an kontextuelle Bedingungen geknüpft werden. Die Kombination von kontextadaptiver Selektion und Ausführung ermöglicht eine personalisierte und situationsbezogene Bereitstellung von Diensten.

Die kontextadaptive Dienstnutzung wird im Rahmen dieser Arbeit durch ein *Datenzentrisches Protokoll* [EHAB 99] realisiert, welches die Weiterleitung (*Routing*) von Anwendungsdaten anhand von kontextuellen Einschränkungen erlaubt. Dieser Ansatz gestattet neben der kontextadaptiven Nutzung individueller Dienste auch die spontane Komposition von Diensten in einer Ubiquitous Umgebung.

Auszeichnend für den hier vorgestellten Ansatz ist, dass Kontextadaptivität als ein Aspekt der Inter-Dienst Kommunikation verstanden wird.

Zur Motivation des skizzierten Ansatzes soll zunächst ein Szenario erörtert werden.

Szenario

Abbildung 3-1 vermittelt einen Überblick über das betrachtete Szenario. Der Benutzer trägt drei Endgeräte bei sich: eine digitale Kamera, einen PDA und ein Mobiltelefon. Die einzelnen Geräte sind innerhalb eines *Personal Area Networks* (PAN), wie etwa Bluetooth (vgl. 2.5.1), organisiert. Im Kontext dieser Arbeit

bildet das PAN eine abgeschlossene Dienstdomäne[6], in denen die angebotenen Dienste der eingeschlossenen Geräte wechselseitig genutzt werden können.

Es existieren zwei weitere Dienstdomänen. Es gibt zum einen eine entfernte Dienstdomäne (nicht erreichbar durch das PAN), die ein so genanntes *Artdisplay* zur dauerhaften Anzeige von digitalem Bildmaterial bereitstellt, und zum anderen eine Dienstdomäne, die einen Drucker zur Verfügung stellt. Bei beiden Dienstentitäten handelt es sich um eingebettete Geräte, die womöglich Teil eines Gebäudes sind.

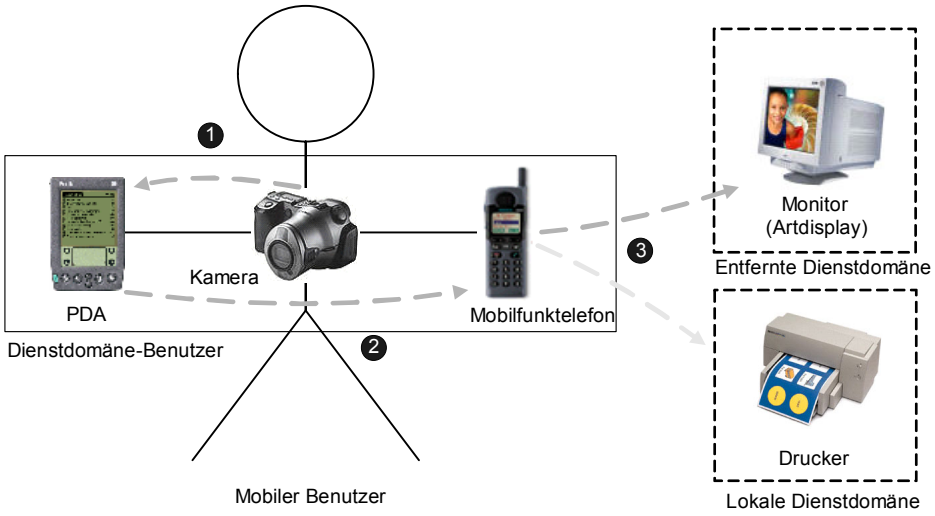

Abbildung 3-1: Szenario

Nachfolgend werden Wechselwirkungen zwischen den einzelnen Dienstentitäten anhand von Aktionen des Benutzers beschrieben.

- *Zunächst schießt der Benutzer Bilder mit seiner Digitalkamera. Dann veranlasst er, dass die gemachten Bilder auch auf seinem PDA anzuschauen sind.*
- *Neben dem Vorteil des größeren Displays bietet das PDA auch Möglichkeiten der Weiterverarbeitung von eingehenden Bildern. Zu diesem Zweck bietet das PDA dem Nutzer weiterführende Dienstoptionen an. Die Selektion der Dienstoptionen erfolgt kontextadaptiv (z.B. abhängig vom Dienstangebot der Umgebung, Datentyp, etc.).*
- *Im hier betrachteten Szenario entscheidet der Benutzer sich dafür, ein gemachtes Bild zu Hause auf einem Artdisplay anzubringen. Da sich das*

[6] Eine umfassende Definition des Domänenbegriffs erfolgt in 3.2.1

Artdisplay in einer entfernten Dienstdomäne befindet, wird es zunächst zum Mobiltelefon übertragen.

- *Das Mobiltelefon agiert hier als ein allgemeiner Kommunikationsserver. Es empfängt das Bild, welches für das Artdisplay bestimmt ist. Zu einem früheren Zeitpunkt hat der Benutzer verfügt, dass Daten, die ein bestimmtes Volumen überschreiten, aus Kostengründen nicht mittels Weitverkehrnetzen (z.B. GPRS [HMNS 01]) versendet werden dürfen. Da es sich nicht um zeitkritische Daten handelt, werden diese erst dann übertragen, wenn sich der Benutzer in Reichweite seines Heimnetzwerkes (z.B. WaveLan [Air]) befindet.*
- *Schließlich wird das Bild auf dem Artdisplay angezeigt.*

Alternativ wäre folgender Ablauf denkbar (vgl. Abbildung 3-1):

- *Statt der Anzeige des Bildes auf dem Artdisplay entscheidet sich der Benutzer für Dienstoption „Drucken auf lokal verfügbarem Drucker".*
- *Da sich gerade ein oder mehrere Drucker in direkter Nähe des Benutzers befinden, wählt das System entsprechend seines Kontextes (hier Ort, Benutzerprofil) einen Drucker aus.*
- *Der resultierende Druckauftrag wird ausgeführt.*

Das Szenario zeigt zum einen Anwendungen für das Konzept der kontextadaptiven *Selektion*, so etwa am Beispiel der Auswahl eines Druckers bez. kontextueller Einschränkungen. Zum anderen wird auch die Notwendigkeit einer kontextadaptiven Ausführung von Diensten demonstriert. So erfolgt die *Ausführung* des Anzeigedienstes (*Artdisplay*) erst dann, wenn die kontextuellen Einschränkungen bez. dieser Dienstnutzung erfüllt sind. In diesem Fall bezogen sich die Einschränkungen auf die Minimierung der anfallenden Kosten und die Erreichbarkeit eines lokalen Netzwerkes.

Zudem motiviert das angeführte Szenario eine kontextadaptive Weiterleitung von Datenfragmenten. In Konsequenz wurde im Rahmen dieser Arbeit ein *Datenzentrisches Protokoll* [EHAB 99] entwickelt, welches erlaubt, die Vermittlung von Daten inhaltsbezogen bzw. basierend kontextueller Einschränkungen zu gestalten.

3.1 Anforderungen und Designkriterien

Die Anforderungen und Designkriterien, die bei der Entwicklung der hier vorgestellten Konzeption berücksichtigt wurden, werden im Folgenden darlegt. Es wird zwischen benutzerspezifischen und daraus resultierenden technischen

Anforderungen unterschieden. Im nachfolgenden Abschnitt 3.1.3 werden in Übereinstimmung mit den formulierten Anforderungen Designkriterien erörtert, welche für die Konstruktion des hier beschriebenen Systems grundlegend waren.

3.1.1 Benutzerspezifische Anforderungen

In diesem Kapitel wird ein System für mobile Benutzer in Ubiquitous Computing Umgebungen vorgestellt. Diese Klasse von Systemen ist mit einer Vielzahl von speziellen Anforderungen verbunden, die das resultierende Design notwendig beeinflussen. Hier werden zwei Aspekte vorgestellt:

- *Kontextadaptivität*
- *Benutzermodellierung*

Kontextadaptivität

Eines der wesentlichen Ziele dieser Arbeit besteht darin, den Umgang mit Mobilen Systemen zu erleichtern. Insbesondere soll die Anzahl von erforderlichen Benutzer-System-Interaktionen gegenüber anderen Ansätzen verringert werden. Der Fokus richtet sich dabei auf Interaktionen in Multidienst-Umgebungen. Die Benutzung von Diensten soll aus Sicht des Nutzers *implizit* erfolgen. Das entwickelte System unterstützt den Nutzer bei der Auswahl und Ausführung von Diensten. Diese Unterstützung erfolgt in Anlehnung an 2.3 kontextadaptiv und kann damit aufgaben- oder tätigkeitsspezifisch geleistet werden [WG 00]. Aus dieser Anforderung heraus wurde eine Abstraktion gestaltet, die im Rahmen dieser Arbeit als *Kontextadaptive Interaktion* bezeichnet wird, siehe 3.2.

Benutzermodellierung

Neben dem Konzept der Kontextadaptivität kann auch das Verfahren der Benutzermodellierung [K 94] verwendet werden, um die Interaktionen in einer Ubiquitous Computing Umgebung zu rationalisieren. Hier wird das Konzept der Benutzermodellierung angewandt, um Interaktionen mit Diensten zu personalisieren. Ein bedeutender Aspekt in diesem Zusammenhang ist die Möglichkeit, persönliche Präferenzen auch in einer fremden Umgebung zu etablieren (so ist z.B. die explizite Konfiguration seitens des Nutzers nicht erforderlich). Neben der Verwaltung von benutzerbezogenen Einstellungen und Präferenzen ist auch das Anlegen von individuellen Interaktionsmustern vorgesehen. Dies geschieht toolgestützt und erfordert keine näheren Kenntnisse des Benutzers bez. der Systemarchitektur. Diese Thematik wird in Kapitel 4 behandelt.

3.1.2 Technische Anforderungen

Der Anspruch der Kontextadaptivität bzw. Benutzermodellierung für die Dienstnutzung in Ubiquitous Computing Umgebungen resultiert in einer Menge

von technischen Anforderungen an die Konzeption. Jene werden im folgenden erläutert:

- *Mobilitätsunterstützung*
- *Heterogene Netzwerkumgebungen*
- *Thin Clients*
- *Horizontale Integration*
- *Legacy Anwendungen*

Mobilitätsunterstützung

Der Aspekt der Mobilitätsunterstützung gewinnt im Zusammenhang der hier vorgestellten Thematik an Bedeutung. Insbesondere wird das Konzept der applikationsbasierten Mobilitätsunterstützung zum Zweck der kontextadaptiven Dienstnutzung erweitert. Ansätze der Mobilitätsunterstützung wie etwa Mobile IP [MIP] beziehen sich immer auf einen bestimmten Anwendungsfall. Dieser zeichnet sich im Wesentlichen dadurch aus, dass die Kommunikationen eines Klienten mit einer Infrastruktur (z.B. Applikationsserver) auch dann gewährleistet werden kann, wenn dieser mobil ist. Zusammenfassend kann man diese Ansätze als eine Erweiterung des klassischen Client/Server-Modells um mobile Klienten charakterisieren.

Das hier vorgestellte System verkörpert ein *Peer-to-Peer-Netzwerk* [P2P 02] und zeichnet sich somit nicht durch eine Netzwerktopologie mit dedizierter Client/Server Architektur aus. Jeder Knoten in einem Peer-to-Peer-Netzwerk kann sowohl Client- als auch Server-Funktionen übernehmen. Gemäß [S 00] lässt sich diese Art von Netzwerken durch folgende Eigenschaften charakterisieren:

1. Variable Netzwerkverbindungen mit temporären Netzwerkadressen
2. Netzwerkknoten mit signifikanten Autonomieeigenschaften

Diese Eigenschaften erfordern daher ein neues Modell zur Behandlung von Mobilitätsaspekten. Dabei soll vor allem von variierenden Netzwerkadressen und -verbindungen abstrahiert werden. Infolgedessen wird im Rahmen dieser Arbeit ein neues Adressierungsschema vorgestellt. Dabei handelt es sich um ein *assoziatives* Schema (siehe 3.1.3), welches eine Identifizierung von Netzwerkknoten unabhängig von konkreten Netzwerkadressen ermöglicht. Dies gilt auch für mobile Knoten, deren Netzwerkadressen sich dynamisch ändern können.

Heterogene Netzwerkumgebungen und Weak Connectivity

Ubiquitous Computing Umgebungen sind heterogene Netzwerkumgebungen in der Hinsicht, dass nicht nur verschiedene Netzwerktechnologien zum Einsatz kommen

(siehe 2), sondern auch diverse Netzwerkprotokolle wie etwa HTTP [HTTP], SMTP [P 82] oder TCP/IP [Stev 94]. Die Kommunikation zwischen Endknoten eines Ubiquitous Computing Systems kann über intermediäre Netzwerkknoten erfolgen, welche unterschiedliche Netzwerkprotokolle implementieren. Abbildung 3-2 deutet die Kommunikation zwischen zwei Endknoten über zwei intermediären Netzwerkknoten an.

Das im Rahmen dieser Arbeit vorgestellte Datenprotokoll zur kontextadaptiven Dienstnutzung unterstützt die Operation in einer solchen Multiprotokollumgebung, indem es von den diversen Netzwerkprotokollen abstrahiert (*multiple protocol bindings*). Dies erfordert ein einheitliches Kodierungsschema und eine Abstraktion von den Nachrichtenmodellen der unterliegenden Protokolle (vgl. [NT 01]).

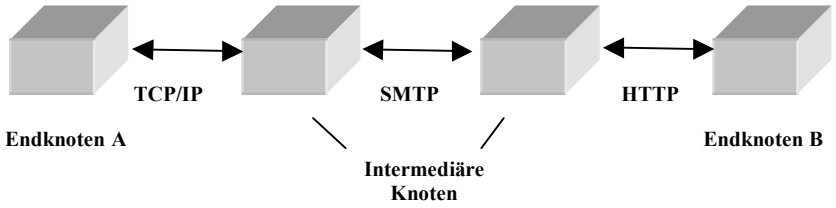

Abbildung 3-2: Multiprotokoll-Umgebung

Zudem soll das hier vorgestellte System auch den Anforderungen einer Umgebung mit drahtloser Netzwerktechnologie entsprechen. Diese ist u.a. durch Verbindungsabbrüche und niedrige Bandbreiten gekennzeichnet (vgl. 2.2.2, *weak connectivity*). Des weiteren finden auch die Spezifika von Ad-hoc-Netzwerken in der hier vorgestellten Konzeption Berücksichtigung.

Thin Clients

Eine weitere essentielle Forderung für die Realisierung des hier vorgestellten Systems ist die Unterstützung von so genannten *Thin Clients* (vgl. [JHE 99]). Diese Gattung von Klienten verfügt nur über knappe Systemressourcen (z.B. CPU- und Speicher-Kapazität). Dazu zählt man beispielsweise Smart Phones oder PDAs. Infolgedessen wird ein großer Teil der Applikationslogik auf stationären Server oder Proxies verlagert.

Horizontale Integration

Gegenwärtige Systemarchitekturen sind zumeist nach dem Prinzip der *Vertikalen Integration* [EHAB 99] entwickelt worden. Für diesen Ansatz ist charakteristisch, dass ein abgeschlossenes System die Gesamtlösung zu einer bestimmten Problematik implementiert. Diese Vorgehensweise ist durch hohe Kosten und Inflexibilität gekennzeichnet. So ist es beispielsweise meist schwierig, Infor-

mationen zwischen Systemen mit einem ähnlichen Dienstangebot auszutauschen. Ein Vorteil der Vertikalen Integration ist die Möglichkeit, dass die Administration eines Systems zentralisiert werden kann. Aus Benutzersicht ist die Vertikale Integration oft mit Nachteilen verbunden. Die Forderung eines Benutzers nur eine Untermenge der vom System angebotenen Dienstfunktionalität zu beanspruchen, kann oft gar nicht oder nur mit Schwierigkeiten erfüllt werden (*"take it or leave it dilemma")*. Zudem ist die schnelle Einführung von neuen Dienstalternativen oft nicht durchführbar.

Im Kontrast zur Vertikalen Integration erlaubt der Ansatz der *Horizontalen Integration* die Komposition von Basisdiensten zu komplexen Diensten. Bei diesem Ansatz lässt sich eine Anpassung an spezifische Benutzerwünsche einfacher verwirklichen. So ist es beispielsweise denkbar, in einer bereits existierenden Dienstkomposition einzelne Basisdienste auszutauschen, um neuen Anforderungen gerecht zu werden. Insbesondere wird so auch eine Dienstkomposition in Multi-Provider Szenarien, wie sie im Rahmen von Ubiquitous Computing typisch sind, begünstigt [SAHARA].

Im Rahmen dieser Arbeit wird der Ansatz der Horizontalen Integration verfolgt.

Legacy Anwendungen

In bestehenden Anwendungen ist der Aspekt der Kontextadaptivität zumeist ein inhärenter Bestandteil der Systemarchitektur. Daher müssen bereits bestehende Anwendungen i.d.R. neu entwickelt werden, um den Anspruch der Kontextadaptivität gerecht zu werden.

Um dieser Problematik zu entgehen, soll der Anspruch der Kontextadaptivität als ein Aspekt der Kommunikation aufgefasst werden. Dann ist es möglich, bereits bestehende Anwendungskomponenten zu integrieren, indem lediglich die Art der Kommunikation angepasst wird (*Wrapper*).

3.1.3 Designkriterien

In Korrespondenz zu den oben aufgeführten Anforderungen werden nunmehr die daraus resultierenden Designkriterien vorgestellt. Insbesondere sollen folgende Aspekte näher untersucht werden:

- *Datenzentrisches Protokoll*
- *Adaptivität*
- *Assoziative Adressierung*
- *Lokalitätsprinzip*

Datenzentrisches Protokoll

Zum Zwecke der kontextadaptiven Dienstvermittlung in Ubiquitous Computing Systemen wird hier ein *Datenzentrisches-Protokoll*[7] (vgl. [EHAB 99]) vorgeschlagen. Diese Art von Protokoll dient der Kommunikation von Applikationsdaten zwischen verteilten, autonomen Netzwerkknoten. Im Rahmen dieser Arbeit wird ein Datenzentrisches Protokoll durch folgende Eigenschaften charakterisiert:

- *Uniforme Datenformate für Applikationsdaten*
- *Applikationsspezifische Metadaten*
- *Daten-gesteuertes Routing (Content-based Routing)*
- *Mechanismen zur Daten- und Dienst-Komposition in Verteilten Systemen (in-network data composition)*

Ein uniformes Datenformat ermöglicht den Austausch von Applikationsdaten über verschiedene Systemarchitekturen hinweg. Somit kann der Anspruch auf Heterogenität in Ubiquitous Computing Systemen erfüllt werden, da das Datenformat unabhängig von Merkmalen einer bestimmten Systemarchitektur entwickelt werden kann. Ein einzelnes Datenpaket kann aus mehreren logischen Dateneinheiten bestehen.

Applikationsdaten, die zwischen Komponenten eines Systems kommuniziert werden, werden um *Metadaten* erweitert. Diese Metadaten beschreiben den Datenverarbeitungsprozess in einem Netzwerk autonomer Knoten. Dazu gehört ein Datenpfad [NT 01] (siehe 3.2.1), welcher den Datenfluss über eine Menge von autonomen Netzwerkknoten beschreibt. Auf jedem dieser Netzwerkknoten können Daten eines eingehenden Paketes verarbeitet werden, indem diese Daten modifiziert, gelöscht oder auch neue Daten hinzugefügt werden. Der spezifizierte Datenpfad ermöglicht auf diese Weise eine verteilte Komposition von Daten und Diensten im Sinne einer horizontalen Integration (siehe voriger Abschnitt). Des Weiteren können Metadaten u.a. auch Informationen zur Fehlerbehandlung oder Datenbeschreibung (z.B. Typ der versandten Daten) enthalten.

Im Rahmen dieser Arbeit werden Datenpfade durch *Context Constraints* [SMLP 01a] festgelegt. Diese Methode ermöglicht auch die Kommunikation zwischen lose gekoppelten Komponenten; siehe folgender Abschnitt (*Assoziative Adressierung*).

Die Spezifikation von Metadaten erlaubt vor allem auch die Unterstützung von *weak connectivity* und Ad-hoc-Netzwerken. So können beispielsweise Daten auch dann zu ihrem Empfänger geschickt werden, wenn die Netzwerkverbindung zum sendenden Applikationsprozess nicht mehr vorhanden ist, da der assoziierte

[7] engl.: data-centric protocol

Netzwerkpfad die Verarbeitung der versandten Daten vollständig beschreibt, ohne dass eine weitere Reaktion der versendenden Applikation erforderlich ist.

Der Einsatz eines Datenzentrischen Protokolls erlaubt auch den Einsatz von *Thin Clients*, da Operationen mit hohem Ressourcenaufwand auf leistungsstärkere Netzwerkknoten verlagert werden können [BKAB+ 98] (siehe auch 2.2.2.3).

Assoziative Adressierung

Ein relevanter Aspekt des hier vorgestellten Daten-zentrischen Protokolls ist der Aspekt der *Assoziativen Adressierung*, der im Folgenden vorgestellt wird.

Ein bezeichnender Trend im Bereich der Interprozesskommunikation ist durch eine lose Kopplung (*loosely coupled*) von Prozesskomponenten gekennzeichnet. Dies steht im Kontrast zu einer statischen Bindung (*tightly coupled*) von kooperierenden Prozesskomponenten.

Die statische Bindung von Prozesskomponenten erfordert, dass die Netzwerkadresse eines Empfängers bei der Initiierung eines Kommunikationsvorgangs bekannt ist. Diese Anforderung wird in den Fällen zum Problem, in denen die Adresse eines Empfängers nicht bekannt ist oder die Adresse sich aufgrund dynamischer Adresszuweisungen ändern kann. Auch eine variierende Zahl von Empfängern kann mit diesem Modell nicht zufriedenstellend abgebildet werden. Da jedoch eine Ubiquitous Computing Umgebung durch Anwendungsfälle dieser Art gekennzeichnet ist, eignet sich die statische Kopplung für den hier behandelten Themenkomplex nicht.

Besser angepasst auf diese Anwendungsszenarien ist die lose gekoppelte Kommunikation zwischen Prozesskomponenten. Bei diesem Ansatz entstehen keine statischen Abhängigkeiten, da Empfänger dynamisch innerhalb eines *Late Binding* [INS 99] Prozesses determiniert werden. Dieser Ansatz entspricht den Ansprüchen einer Ubiquitous Computing Umgebung besser. Der Vorgang der dynamischen Determinierung von empfangenden Netzwerkknoten wird im Rahmen dieser Arbeit als *Assoziative Adressierung* bezeichnet. Wie bereits im vorigen Abschnitt erwähnt, erfolgt die Spezifikation eines Empfängers über *Context Constraints* [SMLP 01a].

Adaptivität

Mit Bezug auf die in 2.2.2 vorgestellte Klassifikation von Adaptionsstrategien kann man den hier gewählten Ansatz als *applikationstransparent* bezeichnen, da der initiierende Klient Anforderungen der Mobilität – wie etwa Verbindungsabbrüche - nicht berücksichtigen muss. Aufgrund dessen ist die Entwicklung von Anwendungen oder Anwendungsframeworks erleichtert. Im Gegensatz zu *applikationszentrischen* Ansätzen, welche u.U. Mobilitätsstrategien auf Seiten des Klienten implementieren.

Neben der Mobilitätsadaption ist der Gesichtspunkt der Kontextadaptivität ein zentrales Kriterium für den hier vorgestellten Ansatz. Als Konsequenz wurde ein Verfahren zur kontextadaptiven Datenvermittlung bzw. Selektion und Ausführung von Diensten entwickelt; siehe 3.2.

Lokalitätsprinzip

Ein wesentlicher Charakterzug von kontextadaptiven Systemen ist der Ortsbezug, da Kontext und Situation Phänomene sind, die sich auf einen bestimmten Ort oder eine Region beziehen. Insbesondere im Bereich der mobilen Geräte und der Eingebetteten Systeme sind der Ort der Nutzung und der Ort, an dem die Information entsteht, wesentlich [SG 01]; vgl. 2.3.3.

In der hier vorgestellten Systemarchitektur ist das Konzept der Ortsbezogenheit immanent. Das bedeutet, Systemressourcen sind an einen bestimmten Ort gebunden. Deshalb ist die Angabe von Ortsinformationen zur Identifikation einer Ressource notwendig. Dieses Modell widerspricht den Architekturprinzipien von vielen anderen Verteilten Systemen, wo von dem Ort einer Ressource gewöhnlich abstrahiert wird.

Diesbezüglich wird in 3.2.1 ein Dienstmodell beschrieben, welches Ressourcen und Dienste mittels eines ortsbezogenen Domänenmodells modelliert.

3.2 Aspekte der kontextadaptiven Dienstnutzung

Im Folgenden wird das Konzept der Kontextadaptiven Dienstnutzung erläutert. Die Darstellung erfolgt aus zwei Perspektiven: aus *Benutzersicht* und *Systemsicht.* Beide Teile ergänzen sich und vermitteln einen integralen Überblick über die hier behandelte Thematik. Insbesondere die Darstellung aus Benutzersicht ist hier erforderlich, da das Konzept der kontextadaptiven Dienstnutzung die Handhabung eines Systems wesentlich beeinflusst.

Zunächst erfolgt jedoch eine Darstellung des im Rahmen dieser Arbeit verwendeten Dienstmodells.

3.2.1 Dienstmodell

Vor der Behandlung des hier zugrunde liegenden Dienstmodells sollen noch einige Begriffe erläutert werden:

- *Dienst*. Im Kontext dieser Arbeit bezeichnet ein Dienst ein Objekt (*compuational object*), welches mittels eines Dienstvermittlungsprotokolls (z.B. Jini, SLP) lokalisiert werden kann.
- *Diensterbringer.* Der Diensterbringer ist die Ressource (oder eine Menge von Ressourcen), auf der ein Dienst ausgeführt wird.

- *Dienstendpunkt.* Der Dienstendpunkt definiert den Kommunikationsendpunkt eines Dienstes. Dieser kann z.B. durch eine IP-Adresse oder eine Objektreferenz repräsentiert werden.
- *Dienstnutzung*. Die Dienstnutzung bezieht sich auf den Vorgang der Inanspruchnahme eines Dienstes. Es wird beispielsweise eine Methode aufgerufen.

Innerhalb des hier vorgestellten Verteilten Systems zur kontextadaptiven Dienstnutzung ist der Aspekt der Ortsbezogenheit mit in die Modellbildung eingeflossen (vgl. 3.1.2, *Lokalitätsprinzip*). Dies erleichtert die Formulierung kontextadaptiver Applikationen, indem es das Lokalitätsprinzip des hier verwendeten Kontextmodells (siehe 2.3) implizit mit einbezieht. Zudem wird auch der Einsatz lokaler Netzwerke, wie etwa Bluetooth (siehe 2.5.1), die sich durch einen begrenzten Wirkungsradius auszeichnen, begünstigt.

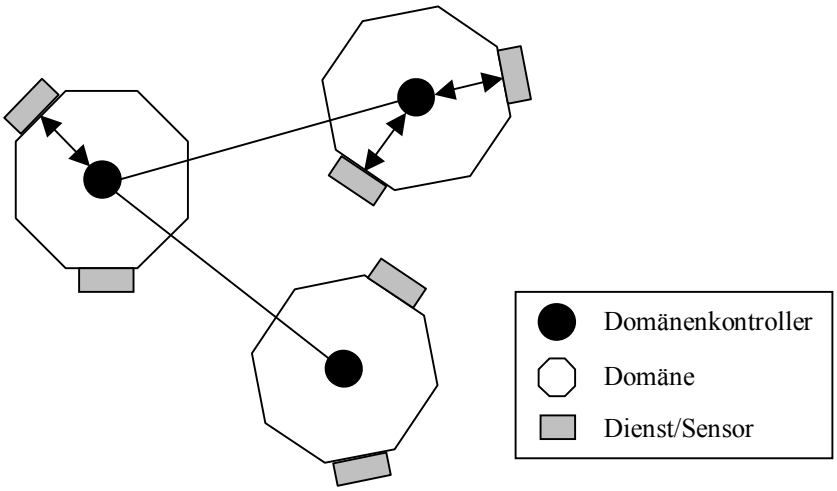

Abbildung 3-3: Dienstmodell

Architekturübersicht

Abbildung 3-3 veranschaulicht die Architektur des Dienstmodells. *Domänen* (oder auch *Dienstdomänen*) werden durch einen zugehörigen *Domänenkontroller* verwaltet. Dieser verwaltet die Dienste/Ressourcen und Sensoren in seiner Domäne. Die Domänenkontroller sind über ein Backbone Netzwerk miteinander verbunden, so dass es möglich ist, die Ressourcen mehrer Domänen in einen individuellen Verarbeitungsprozess mit einzubeziehen. Allerdings ist eine direkte Kommunikation zwischen Ressourcen unterschiedlicher Domänen nicht möglich. Die Kommunikation erfolgt immer indirekt über die entsprechenden Domänenkontroller. Diese Tatsache ergibt sich direkt als eine Konsequenz des

angewendeten Lokalitätsprinzips. In ähnlicher Weise wurde das Lokalitätsprinzip auch in dem vom MIT entwickelten Agentensystem *Hive* angewendet; vgl. [MGR+99].

Einer der Vorzüge dieses Prinzips ist die Möglichkeit, dass man die Kommunikation zwischen Domänenkontrollern standardisieren kann. Folglich lässt sich beispielsweise auch in einer heterogenen Dienst- bzw. Sensor-Umgebung ein einheitliches Datenformat zur Kommunikation einsetzen.

Das hier betrachtete Dienstmodell schließt auch *abgeschlossene* Dienstdomänen (siehe Abbildung 3-4) mit ein. In diesem Fall sind die Interaktionen zwischen Diensten auf eine individuelle Domäne beschränkt. Die Konzeption der abgeschlossenen Domäne wurde bereits anhand des einführenden Szenarios (siehe Abbildung 3-1) erläutert. In abgeschlossenen Domänen existiert kein expliziter Domänenkontroller, dennoch ist eine Anbindung an andere Domänen mittels eines Gateway-Dienstes möglich, im Szenario das Mobilfunktelefon.

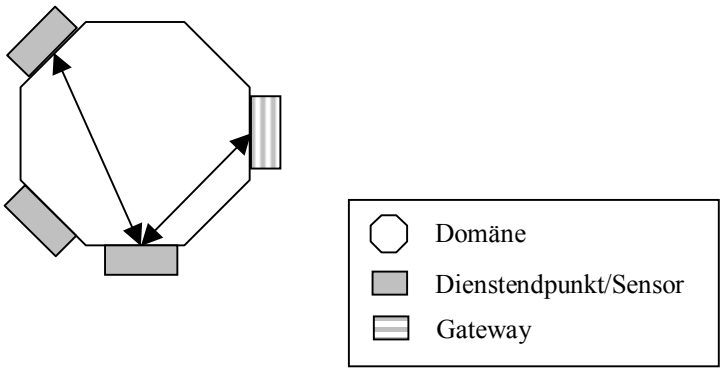

Abbildung 3-4: Abgeschlossene Domänen

Abgeschlossene Domänen werden insbesondere im Fall von Personal Area Networks (PANs) eingesetzt. Hier beschränken sich die Interaktionen i.d.R. auf Dienstendpunkte der jeweiligen Domäne. Zudem können unterliegende Dienste der eingesetzten Netzwerktechnologie direkt genutzt werden, so etwa das lokale Dienstvermittlungsprotokoll von Bluetooth (siehe 2.5.1).

Datenpfade

Im Kontext von SOAP [BEK+ 00] wurde das Konzept des Datenpfades (*Message Path*) [NT 01] vorgeschlagen. Ein Datenpfad beschreibt den Weg einer Nachricht von einem Sender – über intermediäre Knoten – bis zum Empfänger. Zudem wird festgelegt, welche Datenelemente der Nachricht in einem bestimmten Netzwerkknoten manipuliert oder verarbeitet werden.

Ein vergleichbarer Ansatz wird auch im Rahmen der hier vorgestellten Architektur zur kontextadaptiven Dienstnutzung angewandt. Jedoch erfolgt hier die Definition eines solchen Pfades assoziativ, d.h. es wird keine explizite Folge von zu besuchenden Knoten definiert. Die Festlegung des Datenpfades und die damit verbundene Datenverarbeitung erfolgt kontextadaptiv. Mittels dieses Ansatzes lassen sich Dienstinteraktionen über multiple Dienstdomänen bezüglich zu erfüllender Kontextbedingungen formulieren. Die abstrakte Entsprechung dieses Ansatzes wird im Folgenden als *Kontextadaptive Interaktion* bezeichnet (s.u.).

Zudem ist vorgesehen, dass ein einmal definierter Datenpfad auch noch während der Ausführung angepasst werden kann, um so beispielsweise auf Fehlersituationen reagieren zu können (vgl. [NCLL 01]).

Kontextadaptive Interaktion

Als Basis der Kontextadaptiven Dienstnutzung wird zunächst die Idee der *Kontext-adaptiven Interaktion* vermittelt. Diese Abstraktion wurde in Kongruenz mit den oben angeführten Anforderungen und Designkriterien entwickelt und wird im Folgenden schrittweise dargelegt. Zunächst folgt eine informale Definition:

> *Eine **Kontextadaptive Interaktion** verkörpert eine oder mehrere Dienstinteraktionen in einem Verteilten System mit autonomen Netzwerkknoten. Die Selektion und Ausführung der individuellen Dienste erfolgt kontextadaptiv.*

Das Konzept der Kontextadaptiven Interaktion soll in den nachfolgenden Abschnitten sowohl aus System- als auch aus Benutzersicht untersucht werden.

3.2.2 Benutzersicht

Aus Sicht eines Benutzers stellen Kontextadaptive Interaktionen eine Kommunikationsabstraktion dar, welche den Umgang mit lokalen Diensten in einer Ubiquitous Computing Umgebung erleichtern sollen. Als Repräsentant einer bestimmten Klasse von Diensten kann z.B. die Nutzung eines lokalen Druckers betrachtet werden. In einem konventionellen System lässt sich die Nutzung eines solchen Dienstes durch einen mobilen Nutzer in fünf Schritte untergliedern:

- *Sichtung des aktuellen Dienstangebots*
- *Selektion eines Dienstes entsprechend der aktuellen Aufgabe/Kontext (hier Druckerdienst)*
- *Verknüpfung mit Applikationsdaten*
- *Ausführung initiieren*
- *Ausführung kontrollieren (z.B. Quittierung)*

Einige dieser Arbeitsschritte können in komplexe Bedienungsszenarien münden, insbesondere dann, wenn der Benutzer ein mobiles Gerät benutzt, wie etwa ein Mobilfunktelefon, welches nur mit unzureichenden Bedienelementen bzw. Anzeigemöglichkeiten ausgerüstet ist.

Kontextadaptive Interaktionen vermögen nun einige dieser Schritte aus Sicht des Benutzers zu erleichtern bzw. hinfällig zu machen. Dies wird zum einen dadurch erreicht, dass die einzelnen Arbeitsschritte in einer logischen Einheit organisiert werden, wobei auch Abhängigkeiten zwischen den individuellen Schritten abgebildet werden können. Zum anderen berücksichtigen Kontextadaptive Interaktionen den aktuellen Kontext des Benutzers und können so die Interaktion mit einem Dienst der Benutzersituation anpassen. Beispielsweise werden der gegenwärtige Aufenthaltsort und Präferenzen des Benutzers bei der Selektion und Ausführung eines Dienstes berücksichtigt. In Verbindung mit verwandten Techniken aus Cyberdesk [DAPW 97] kann der Bedienungskomfort um ein weiteres erhöht werden, indem die Dienstselektion der gegenwärtigen Aufgabe des Benutzers angepasst wird.

Abgesehen von dem hier beschriebenen Anwendungsfall der lokalen Dienstnutzung für mobile Nutzer kann das Konzept der Kontextadaptiven Interaktionen in einem breiten Anwendungsspektrum eingesetzt werden; so kann sich beispielsweise eine einzelne Kontextadaptive Interaktion auf eine Menge von Dienstendpunkten beziehen und dadurch zur Dienstkomposition beitragen.

Da sich Kontextadaptive Interaktionen auf Kontexttypen beziehen, ist auch eine Personalisierung möglich, die Begriffe impliziert, die den Anforderungen eines Benutzers in natürlicher Weise entsprechen. So kann man die Verwendung eines Dienstes zur Erfüllung einer bestimmten Aufgabe anpassen an bestimmte Orte oder Zeiten. Hier folgt ein Beispiel, welches die Personalisierung bez. Ort und Zeit demonstriert: Immer, wenn ein Benutzer morgens sein Büro betritt, werden alle eingehenden Telefonanrufe automatisch von seinem Mobilfunktelefon auf sein Bürotelefon umgeleitet werden. Beim Verlassen des Büros könnten umgekehrt alle eingehende Büroanrufe automatisch auf das Mobilfunktelefon umgeleitet werden.

In Abschnitt 3.3.2 wird gezeigt, dass zur Implementierung von kontextadaptiven Dienstinteraktionen ein deklarativer Ansatz gewählt wurde, der es dem Benutzer erlaubt, das *Was* (deklarativ) einer Interaktion zu definieren, im Gegensatz zum *Wie* (operativ) einer Interaktion.

3.2.3 Systemsicht

Dieser Abschnitt behandelt die technischen Aspekte einer Kontextadaptiven Interaktion. Dies geschieht anhand eines konkreten Szenarios, bei dem ein Nutzer die Dienste seiner Umgebung in Anspruch nehmen möchte (z.B. eine Datei ausdrucken). Wie in [BBG$^+$ 00] beschrieben, soll das Endgerät des Benutzers (*Client Device*) als ein *Portal* in den umgebenden Dienstraum agieren. Das

Endgerät kann in diesem Zusammenhang als ein Zugriffspunkt in den umgebenden Dienst und Datenraum angesehen werden.

Die Idee des Portals soll hier noch verfeinert werden. Im Sinne der hier behandelten Thematik hält das Endgerät die Rolle eines *Mediators* inne. Dieser vermittelt die Dienstanforderungen eines Benutzers der umgebenden Infrastruktur. Dienstanforderungen resultieren im Wesentlichen aus dem gegenwärtigen Kontext (Tätigkeit/Aufgabe) eines Benutzers. Die Repräsentation und Interpretation von Benutzeranforderungen ist Kernpunkt des vorgestellten Systems.

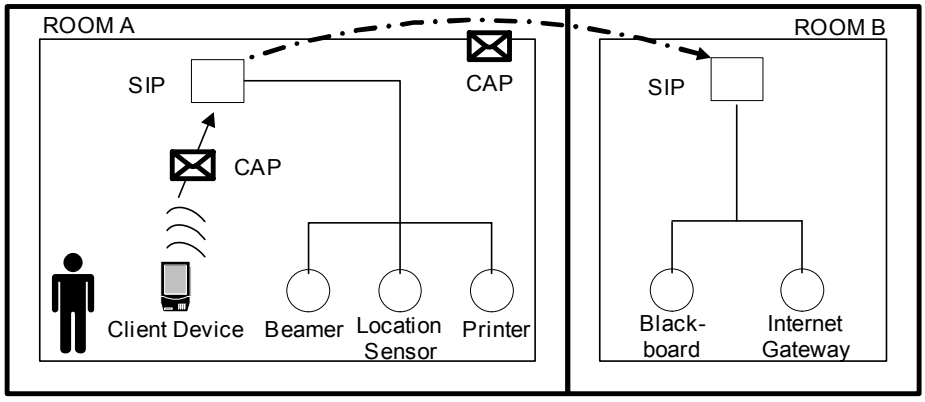

Abbildung 3-5: Kontextadaptive Interaktion (Systemsicht)

Als eine Lösung für die Vermittlung von Dienstanforderungen dient ein *Datenzentrisches Protokoll* mit *Assoziativer Adressierung* (siehe 3.1.3). Zentral für diesen Ansatz ist ein uniformes Datenformat, welches zur Repräsentation und Vermittlung von Dienstanforderungen zwischen den Komponenten eines Systems eingesetzt wird. Dieses Datenformat wird als *Context-Aware Packet* (CAP) bezeichnet. CAPs gestatten eine Darstellung von Dienstanforderung mit hohem Abstraktionsgrad, ohne dass Spezifika von konkreten Diensten der Umgebung bekannt sein müssen (z.B. Netzwerkadresse). CAPs werden vom Endgerät des Benutzers erzeugt und in den umgebenden Dienstraum "injiziert". Im Netzwerk des Dienstraumes werden diese CAPs evaluiert. Die Evaluation resultiert in der Selektion und Ausführung eines oder mehrerer Dienste in Konformität zu den spezifizierten Benutzeranforderungen. Benutzeranforderungen werden innerhalb von CAPs durch so genannte *Context Constraints* (siehe 3.3.2) repräsentiert, welche beschreiben, unter welchen Umständen bzw. Bedingungen Dienste ausgeführt werden. Die Konzeption *Context-Aware Packet* wird in Abschnitt 3.3 ausführlich dargestellt.

Abbildung 3-5 stellt die Evaluation eines CAPs aus Netzwerksicht dar: Zunächst sendet ein Benutzer - über ein drahtloses Netzwerk - ein CAP zu einem lokalen *Service Interaction Proxy* (SIP). Ein SIP ist ein spezieller Netzwerkknoten, der als

Proxy agiert, welcher die verfügbaren Dienste und Sensoren in einer Domäne verwaltet. In dem hier vorgestellten Szenario korrespondiert die vom SIP verwaltete Dienstdomäne mit den Ressourcen eines physikalischen Raums (z.B. Drucker oder Blackboard in einem Büroraum). Der SIP evaluiert das eingehende CAP; dieser Prozess lässt sich in zwei Phasen gliedern: *Selektions*- und *Ausführungsphase*.

In der Selektionsphase überprüft der SIP, ob sich das eingehende CAP auf einen Dienst seiner Domäne bezieht. Sollte dies nicht der Fall sein, wird das CAP im Rahmen eines Routingprozesses zu einem SIP gesendet, der den vom CAP vermittelten Dienstanforderungen entspricht. Der Routing-Prozess von CAPs wird von den im CAP enthaltenen Context Constraints gesteuert. In der zweiten Phase – der Ausführungsphase – wird der vorher selektierte Dienst ausgeführt. Hier kontrollieren die Context Constraints die Ausführung eines Dienstes. Somit wird z.B. geprüft, unter welchen Bedingungen (*triggering*) oder zu welchem Zeitpunkt ein Dienst ausgeführt werden soll.

Zur Steuerung von Selektions- und Ausführungsprozessen - basierend auf Context Constraints - ist es erforderlich, dass der aktuelle Kontext, wie etwa die Position eines Benutzers oder verfügbare Dienste, kontinuierlich bestimmt werden können. Daher hat der SIP Zugriff auf Sensordaten (z.B. Ortsbestimmung), welche zur Ableitung von Kontextinformationen herangezogen werden können. Zudem kann er über Dienstvermittlungsprotokolle, wie sie in 2.4 vorgestellt wurden, das aktuelle Dienstangebot bestimmen.

Zusammenfassend lässt sich feststellen, dass das hier vorgestellte System – unter Benutzung von CAPs – die Selektion und Ausführung von Diensten in einer Ubiquitous Computing Umgebung realisiert. Dieser Prozess wird durch so genannte Context Constraints determiniert; jene reflektieren die Dienstanforderungen eines Benutzers. Der Selektionsprozess bezieht sich auf die Auswahl und Lokalisierung eines Dienstes. Im Gegensatz dazu impliziert der beschriebene Ausführungsprozess temporale Einschränkungen bez. der Dienstnutzung.

3.3 Context-Aware Packets

Die bereits im vorigen Abschnitt erwähnte Konzeption der *Context-Aware Packets* (CAPs) soll im Folgenden erläutert werden. CAPs sind Datenstrukturen, welche zum Zwecke der Dienstvermittlung und Nutzung zwischen Netzwerkknoten eines Ubiquitous Computing Systems ausgetauscht werden.

3.3.1 Struktur

CAPs lassen sich in drei Teile untergliedern (siehe Abbildung 3-6): *Context Constraints*, *Meta Constraints* und *Data.*

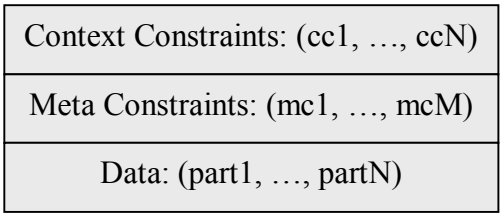

Abbildung 3-6: Context-Aware Packet (CAP)

Context Constraints

Context Constraints spielen eine zentrale Rolle, sie dienen der Vermittlung von Dienstanforderungen. Jedes Context Constraint definiert eine Interaktion (Selektion und Ausführung) mit einem bestimmten Dienst. Die Semantik und Repräsentation von Context Constraints wird im nachfolgenden Abschnitt erläutert. CAPs, die nur genau einen Context Constraint enthalten, werden als *einstufig* bezeichnet. CAPs mit mehreren Context Constraints als *mehrstufig*.

Meta Constraints

Meta Constraints definieren Regeln zur Evaluation von assoziierten Context Constraints. Meta Constraints können sich entweder auf alle Context Constraints eines CAPs beziehen oder auf individuelle Context Constraints. Typischerweise werden Meta Constraints eingesetzt, um die Modalitäten einer Dienstinteraktion zu bestimmen. Man unterscheidet zwischen folgenden Ausführungsdirektiven:

- *One shot*
- *Multiple shot*
- *Interactive/Non Interactive mode*

Mit den Direktiven *One shot* und *Multiple shot* kann bestimmt werden, ob Context Constraints einfach oder mehrfach angewandt werden. Wird die *Interactive mode* Option angewandt, wird ein selektierter Dienst nur nach expliziter Bestätigung ausgeführt. So könnte etwa die Bestätigung eines Benutzers erforderlich sein, bevor ein Dienst ausgeführt wird. Im *Non Interactive mode* erfolgt die Ausführung eines Dienstes autonom – ohne weitere Rückfragen.

Meta Constraints determinieren den von einem CAP induzierten Datenfluss.

Auch die Fehlerbehandlung kann mittels Meta Constraints beeinflusst werden, um z.B. das Standardverhalten zur Fehlerbehandlung zu modifizieren. Eine vertiefte Darstellung der Ausführungsdirektiven erfolgt in 3.4.

Schließlich können auch Zwischenergebnisse der Evaluation eines CAPs in Meta Constraints abgespeichert werden.

Data

Der Daten-Teil enthält die zur Ausführung einer Interaktion beim Empfänger benötigten Daten. Der Daten-Teil ist nicht auf ein bestimmtes Datenformat (z.B. MPEG) eingeschränkt. Die in Frage kommenden Daten können auch über einen *Uniform Resource Locator* (URL) [URL] spezifiziert werden. Die Anzahl der Dateneinträge eines CAPs (part1, … partN) kann im Verlauf einer Evaluation stetig variieren, da während eines Evaluationsprozesses Daten in einem CAP gelöscht, modifiziert oder auch neue Datenblöcke hinzugefügt werden können (siehe auch Abschnitt 3.4.).

CAPs werden mittels XML [GP 00] repräsentiert, siehe 3.5.1.

3.3.2 Context Constraints

Context Constraints steuern die Evaluation von Context-Aware Packets (CAP) und damit auch die Selektion und Ausführung von Diensten. Im Weiteren soll nun die Semantik und Repräsentation von *Context Constraints* erläutert werden.

Semantik

Mit Bezug zu Arbeiten in der Reglungstechnik [F 94] soll hier die Bedeutung von *Context Constraints* erläutert werden. Diese Vorgehensweise liegt nahe, da der Evaluationsprozess eines CAPs in einem dynamischen System erfolgt, welches sich durch Rückkopplungsprozesse auszeichnet. Zudem ermöglicht die Modellierung durch Regelkreise eine erste Dekomposition der Systemarchitektur, da funktionale Komponenten des Systems voneinander isoliert werden können. Des Weiteren können Aussagen über den Datenfluss zwischen Komponenten gemacht werden.

Abbildung 3-7 zeigt zwei Regelkreise: Regelkreis A verkörpert das generelle Modell eines Regelsystems mit offener Schleife, d.h. das Stellsignal ist von der Ausgangsgröße unabhängig. *Blöcke* in Abbildung 3-7 repräsentieren Operationen auf den eingehendem Signalen. *Mischelemente* - durch einen Kreis symbolisiert – zeigen die Kombination von mehren Signalen an (z.B. Addition, Subtraktion oder symbolischer Vergleich).

Regelkreis B modelliert die Evaluation eines CAPs in einem verteilten System. *Context Constraints* (CC) werden hier als *Sollwert* des Systems interpretiert, sie bestimmen eine Dienstinteraktion bez. kontextueller Einschränkungen. *Context Constraints* werden innerhalb einer Dienstdomäne mit aktuell bestimmten Kontextinformationen, dem *lokalen Kontext* (LK), verglichen. Der lokale Kontext bildet das Rückkopplungssignal oder auch den *Istwert* des Systems. Kommt es zu einer Übereinstimmung von lokalem Kontext und Context Constraints, so wird ein Dienst, der den Vorgaben der Context Constraints genügt, ausgeführt, d.h. die mitgeführten Daten des CAPs werden auf diesen Dienst angewendet.

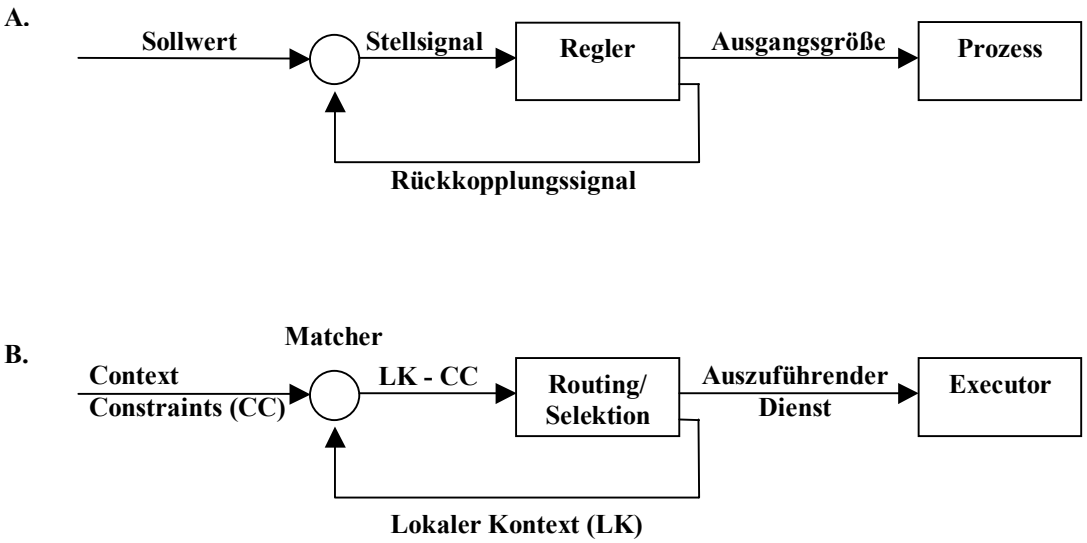

Abbildung 3-7: Darstellung der CAP Evaluation mittels Regelkreismodell

Im Fall der Nichtübereinstimmung kann entweder auf eine passende Situation gewartet werden, oder es wird festgestellt, dass die Bedingungen der Context Constraints in der gegenwärtigen Domäne überhaupt nicht erfüllt werden können. Sodann wird das CAP innerhalb eines Routing-Prozesses zu einer weiteren Domäne versendet. Dieser Prozess wiederholt sich so lange bis es zur erfolgreichen Dienstausführung kommt oder der Prozess explizit abgebrochen wird (beispielsweise durch den Benutzer).

Das Mischelement, hier als *Matcher* bezeichnet, ermittelt die Abweichung von Soll- und Istwert: *LK – CC*. Im Rahmen der Auswertung wird diese Abweichung an die Komponente *Routing/Selektion* weitergeleitet. Wird keine Abweichung mehr festgestellt, so wird eine Referenz (oder Beschreibung) des auszuführenden Dienstes an die *Executor* Komponente weitergeleitet; der Dienst wird ausgeführt.

Der nächste Abschnitt gibt einen Überblick über die logische Repräsentation von *Context Constraints*.

Abstrakte Repräsentation

Entitäten. *Entitäten* bezeichnen im Kontext dieser Arbeit alle Kommunikationsendpunkte, welche an einer Kontextadaptiven Interaktion teilhaben können. Der hier entwickelte Entitätsbergriff schließt neben den konstituierenden Systemkomponenten (Applikationskomponenten, Sensoren) einer Ubiquitous Computing Umgebung auch agierende Personen mit ein.

Der Entitätsbegriff wird informal gefasst:

Eine Entität ist eine Einheit, welche ein CAP sendet, empfängt, verarbeitet oder Effekt hat auf die Evaluierung eines CAPs.

Entitäten werde in zwei Unterklassen gegliedert: *Items* und *Service Endpoints*. Der Entitätstyp *Service Endpoint* wird noch weiter untergliedert; siehe Abbildung 3-8. *Service Endpoints* identifizieren Entitäten, welche Dienstaufrufe entgegennehmen, oder Dienste anderer Entitäten beanspruchen. Entitäten, welche Dienste in Anspruch nehmen, werden als *Actors* bezeichnet. Actors identifizieren Personen oder proaktive Systemkomponenten (z.B. Endgeräte). Als *Service Callee* werden Diensterbringer bezeichnet, deren Dienste im Rahmen einer Kontextadaptiven Interaktion in Anspruch genommen werden.

Die Rolle einer Entität kann sich in Abhängigkeit von einer Anwendung ändern. Dieselbe Entität kann in einem Anwendungsfall *Actor*, in einem anderen Fall *Service Callee* sein. In einigen Szenarien kann sich diese Zuordnung auch dynamisch ändern.

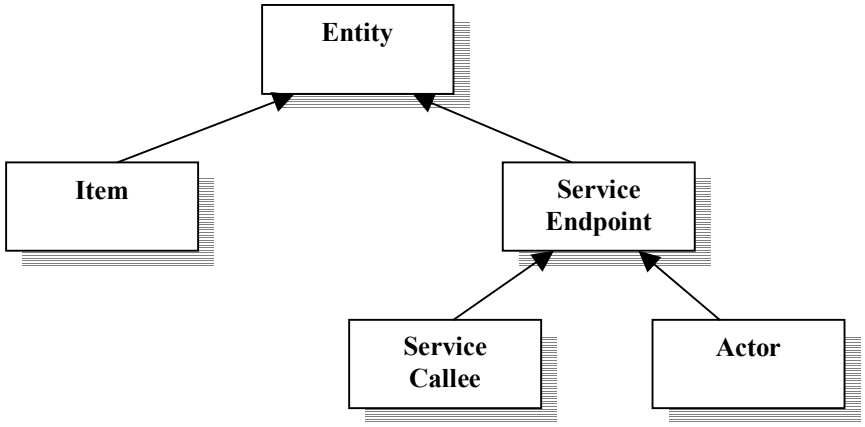

Abbildung 3-8: Unterklassen von *Entity*

Items bezeichnen Entitäten, die im Sinne des in 3.2.1 vorgestellten Dienstmodells weder Infrastrukturdienste noch Applikationsdienste anbieten. Zu dieser Klasse von Entitäten gehören insbesondere auch Sensoren. Ein Sensor kann eine sehr einfache Entität darstellen, welche lediglich periodisch oder auf Anfrage Zustandsdaten übermittelt. Ein Sensor gibt die Informationen über Umweltbedingungen in bestimmten Datenformaten wieder. Je nach Art des Sensors ist der Inhalt unterschiedlich, z.B. Druck, Raumtemperatur, Luftfeuchtigkeit, Lichthelligkeit, Entfernung zwischen zwei Objekten, ob eine Person sich in der Umgebung befindet oder eine Bürotür offen ist. Oft kann es auch erforderlich sein, die Informationen verschiedener Sensoren zu kombinieren (*data fusion*), um die gewünschten Daten zu erhalten.

Generell bezeichnen *Items* Entitäten - wie etwa Sensoren -, welche die Evaluation von CAPs beeinflussen können, aber an der direkten Verarbeitung eines CAPs keinen Anteil haben.

Die Adressierung von Entitäten erfolgt gemäß den in 3.1.3 aufgestellten Designkriterien assoziativ und erfordert im Regelfall keine konkreten Adressierungsschemata, wie etwa Netzwerkadressen. Die Adressierung erfolgt über abstrakte Beschreibungen der beteiligten Entitäten. Abbildung 3-9 zeigt eine vereinfachte Darstellung einer Entitätsbeschreibung für einen Drucker. Die Spezifikation von gewünschten Eigenschaften erfolgt über einfache Attributpaare. Im Rahmen der Evaluation von Context Constraints werden diese Informationen genutzt, um diese abstrakten Entitätsbeschreibungen konkreten Entitäten (z.B. Drucker) zuzuordnen.

Printer
Attributes
Colour = "yes"
Paper-size = "A4"
Duplex = "yes"
Technology = "laser"
Interface

Abbildung 3-9: Vereinfachte Darstellung einer Entitätsbeschreibung

Zusätzlich oder alternativ kann der Selektionsprozess eines Dienstendpunktes auch über eine Schnittstellenbeschreibung im Sinne der Dienstvermittlung eingeschränkt werden (vgl. 2.4).

Neben der Möglichkeit, Entitäten über abstrakte Beschreibungen zu identifizieren, besteht auch die Möglichkeit der konkreten, eindeutigen Adressierung (z.B. IP-Adresse). Der jeweilige Anwendungsfall bestimmt, welches der beiden Verfahren vorzuziehen ist.

Relationen. Die Repräsentation von individuellen Entitäten innerhalb von Context Constraints wurde im vorigen Abschnitt erläutert. Hier sollen nun Abhängigkeiten zwischen Entitäten modelliert werden. Eine Relation zwischen Entitäten kann in folgender Weise definiert werden:

> *Eine Relation deklariert räumliche oder temporale Abhängigkeiten zwischen individuellen Entitäten. Eine Relation wird aus einer Menge von Entitäten gebildet.*

Relationen werden eingesetzt, um den Dienstauswahlprozess, welcher durch ein Context Constraint determiniert wird, weiter einzuschränken. Das heißt, die Menge der Entitäten, die für eine Interaktion in Frage kommt, wird reduziert. Die

Relation *inRoom*, welche im Rahmen dieser Arbeit entwickelt wurde, definiert eine räumliche Einschränkung. Die Relation *inRoom* wird nur dann positiv ausgewertet, wenn sich alle enthaltenen Entitäten dieser Relationen im selben Raum befinden.

Die Konzepte der Entität und Relation bilden ein effektives Modellierungskonstrukt, um Aspekte der realen Welt abzubilden [Che 76]. Die Anwendung dieser Konzepte ermöglicht es im Rahmen der CAP, den Dienstauswahlprozess feingranular abzubilden. Ein Mechanismus, welcher es ermöglicht, den Ausführungsprozess von Diensten bez. anderer Aktivitäten zu synchronisieren, wird im nächsten Abschnitt beschrieben.

Events. Events beeinflussen den Ausführungsprozess von bereits ausgewählten Diensten:

> *Ein Event beschreibt einen Trigger, welcher die Ausführung eines Dienstes initiieren (positive trigger) oder abbrechen (negative trigger) kann.*

Generell bezeichnen Events Ereignisse oder Aktivitäten innerhalb von Computersystemen. Dazu gehört beispielsweise das Klicken einer Maustaste, das Drücken einer Taste oder auch ein Systemereignis, wie etwa die Meldung über ein unzureichendes Angebot an Speicherressourcen. Über *Call back* Mechanismen kann ein Prozess das Interesse für einen bestimmten Ereignistyp anmelden. Bei Auftreten dieses Ereignisses kann der Prozess entsprechend reagieren.

```
<AND>
  <condition subject = ”day” object = “monday”  type = ”=”/>
  <condition subject = “user\_location”
    object = “at_work” type = “ =”/>
</AND>
```

Abbildung 3-10: Event Beispiel: Erinnerung

Innerhalb von CAPs bezeichnen Events insbesondere Aktivitäten oder Ereignisse, welche von Sensorelementen (Items) gemessen werden. Events können zum einen die Ausführung eines Dienstes auslösen oder auch einen Dienst, der bereits ausgeführt wird, abbrechen. So könnte beispielsweise ein Druckauftrag unterbrochen werden, falls im selben Raum ein Meeting begonnen wird. In diesem Fall müsste das Ereignis *Meeting* durch einen Datenfusionsprozess [A 92] ermittelt werden.

Events werden durch logische Ausdrücke erster Ordnung spezifiziert, die Notation wurde durch Arbeiten im Bereich des Policy Management [SL 99, Slo 94]

inspiriert. Abbildung 3-10 demonstriert ein einfaches Beispiel: der Benutzer soll an etwas erinnert werden, wenn er am Montag zu Arbeit kommt.

3.3.3 Dienstinteraktionen

Wie bereits erwähnt, definieren CAPs eine Sequenz von Dienstinteraktionen in einem Verteilten System. Eine solche Dienstsequenz wird im Rahmen dieser Arbeit als eine *Kontextadaptive Interaktion* bezeichnet (vgl. 3.2). Im Weiteren sollen Kontextadaptive Interaktionen untersucht werden, welche typische Anwendungsfälle verkörpern. Die jeweiligen Interaktionstypen werden durch die *Context Constraints* eines CAPs festgelegt. In der bisherigen Konzeption werden folgende Interaktionstypen berücksichtigt: *Put*, *Get*, und *RPC* (vgl. 3.3.1). Es ist vorgesehen, dass das System zu jeder Zeit um neue Interaktionstypen erweitert werden kann.

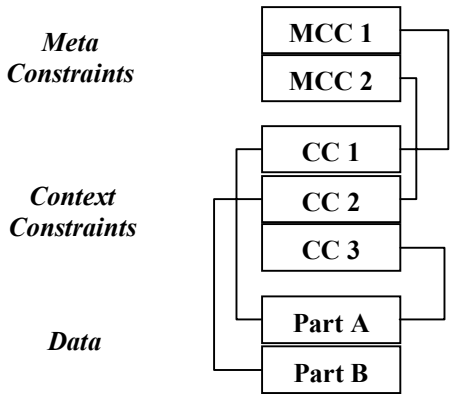

Abbildung 3-11: Beziehungsstruktur von Elementen eines CAP

Bevor die einzelnen Interaktionstypen untersucht werden, wird zunächst auf die Beziehungsstruktur von individuellen Elementen eines CAPs eingegangen, siehe Abbildung 3-11. Die Zuordnung von *Meta* zu *Context Constraints* ist entweder 1:1, oder 1:N für den Fall, dass sich ein *Meta Constraint* auf die Evaluierung aller *Context Constraints* bezieht. Die Beziehung zwischen Dateneinträgen (*Parts*) und *Context Constraints* ist N:1. Mehrere *Context Constraints*, die sich auf denselben Dateneintrag beziehen, werden als *korreliert* bezeichnet.

Bei jedem Evaluationschritt, also einer Dienstinteraktion, wird der entsprechende Eintrag für das *Context Constraint* gelöscht. In Abhängigkeit vom Interaktionstyp kann der korrespondierende Dateneintrag erhalten bleiben oder auch gelöscht werden. Der Dateneintrag bleibt bestehen, falls sich noch existierende *Context Constraints* auf ihn beziehen. Sie können dann Bestandteil späterer Operationen sein. Eine Referenz kann sich auch auf einen leeren Dateneintrag beziehen.

Put

Die durch *Put* bezeichnete Direktive appliziert die koresspondierenden Daten auf einen vorher selektierten Dienst; siehe Abbildung 3-12. Die *Context Constraints* des CAPs sind nicht korreliert, daher können die jeweiligen Dienstinteraktionen unabhängig voneinander bearbeitet werden. Etwaige Rückgabewerte des Dienstes werden bei einer Put-Interaktion ignoriert.

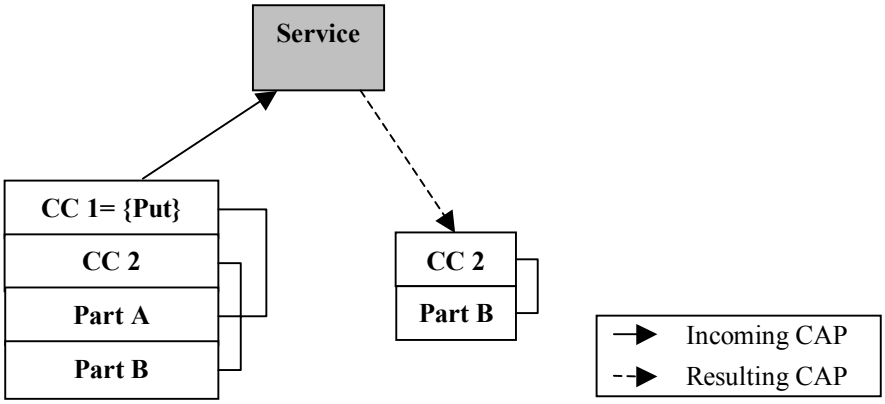

Abbildung 3-12: *Put* Interaktion

Get

Get implementiert das Einlesen von Daten, die von einem bestimmten Dienst zur Verfügung gestellt werden. Die Get-Interaktion basiert auf korrelierten *Context Constraints*.

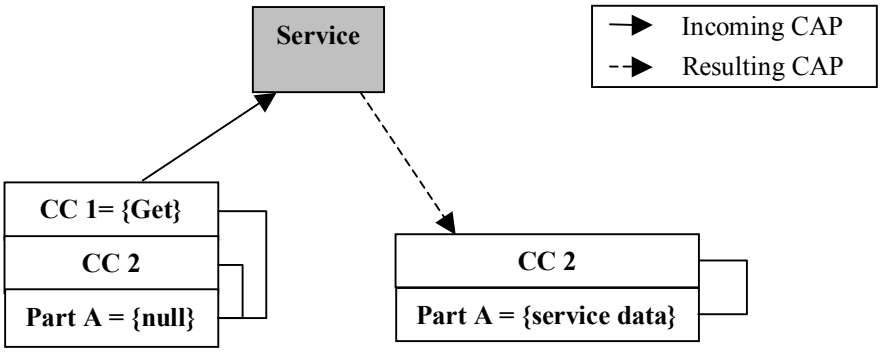

Abbildung 3-13: *Get* Interaktion

Zunächst werden am Dienst, der durch *Context Constraint 1* (CC 1) spezifiziert ist, Daten eingelesen; dann werden sie in das leere Datenfeld (*Part A*) kopiert. Diese Daten werden dann an den durch CC2 spezifizierten Dienst gesendet; siehe

Abbildung 3-13.

RPC

Der als *RPC* bezeichnete Interaktionstyp emuliert ein Verhalten, welches der Semantik eines *Remote Procedure Calls* [R 95] entspricht; siehe Abbildung 3-14. Zunächst werden die im CAP enthaltenen Daten auf den in Frage kommenden Dienst angewendet (Funktionsaufruf). Die rückgegebenen Daten werden in denselben Dateneintrag des CAPs zurückgeschrieben. Diese werden dann an den durch CC 2 spezifizierten Dienst gesendet.

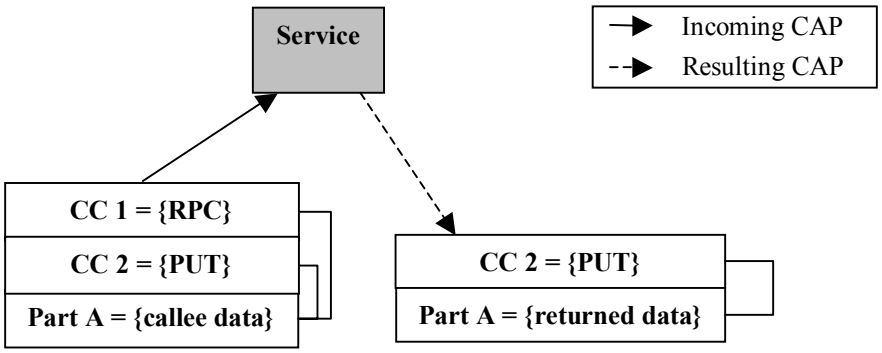

Abbildung 3-14: *RPC* Interaktion

Im Unterschied zu dem Konzept des *Remote Procedure Calls* können die Rückgabedaten an einen beliebigen Dienst gerichtet sein, müssen also nicht zwingend zum Sender des Aufrufs zurückgeschickt werden.

In Abschnitt 3.4.3 wird gezeigt, dass die durch ein CAP definierte Kontextadaptive Interaktion auch noch zur Laufzeit manipuliert werden kann. Dieses Verfahren kann insbesondere zur Behandlung von Fehlerbedingungen eingesetzt werden. Es wird im Rahmen dieser Arbeit als *Beugung eines Datenpfades* bezeichnet.

3.4 System Design

Die bisher beschriebenen Konzepte wurden innerhalb eines Architekturvorschlages zusammengefasst. Dieser wurde als *Context-Aware Packets Enabling Ubiquitous Services (CAPEUS)* [SMLP 01b] bezeichnet. Im Folgenden werden die Komponenten bzw. das Systemverhalten dieser Architektur erläutert.

3.4.1 Service Interaction Proxies

Wie bereits beschrieben, definiert ein mehrstufiges CAP eine Folge von Dienstinteraktionen über eine Menge von verteilten Dienstdomänen hinweg. In Abbildung 3-15 wird ein solcher Prozess angedeutet. Ein Benutzer tritt in Interaktion mit der Umgebung über einen so genannten *Interaction Organizer*, der als ein Benutzeragent fungiert. Benutzer- bzw. aufgabenbezogen erzeugt der *Interaction Organizer* CAPs, die an die Infrastruktur gesendet werden. Die Infrastruktur besteht aus einer Menge vernetzter Dienstdomänen, die durch jeweils einen SIP kontrolliert werden.

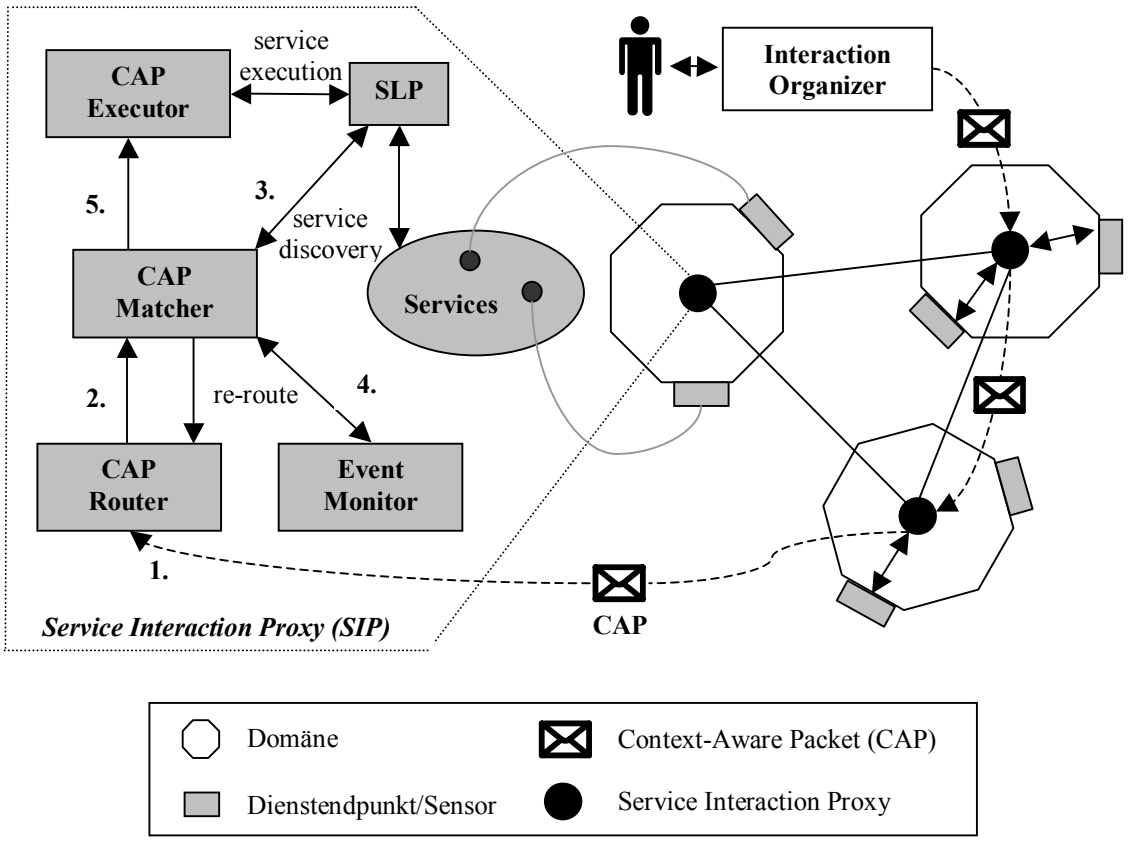

Abbildung 3-15: Komponenten des *Service Interaction Proxies* (SIP)

Die linke Seite der Abbildung stellt die konstituierenden Komponenten eines SIPs dar. Es werden insbesondere fünf zentrale Komponenten unterschieden: *CAP-Router*, *CAP-Matcher*, *CAP-Executor* und *Event-Monitor*. Zudem existiert noch ein Modul zu Implementierung des *Service Location Protocol* (SLP) [Gutt 99]. Die mit *Services* beschriftete Ellipse fasst die in einer Domäne verfügbaren Dienste zusammen.

Ein einkommendes CAP wird auf folgende Weise verarbeitet: (1) der *Router* empfängt das Paket und (2) leitet es an den *Matcher* weiter. Stellt der *Matcher* fest, dass das Paket für eine andere Domäne bestimmt ist (aufgrund nicht vereinbarer *Context Constraints*), wird es zurück zum *Router* geleitet (*re-route*). Dieser sendet das CAP im Rahmen eines Routing-Prozesses zu einer anderen Domäne. Können hingegen alle Relationen des CAPs erfolgreich ausgewertet werden (3) bzw. auch ein in Frage kommender Dienst lokalisiert werden, so verbleibt das CAP im SIP (4). Dann beobachtet der Event-Monitor (bzw. multiple Instanzen) die Umgebung, bis eine Situation entsteht, die einen der in den *Context Constraints* angegebenen Events entspricht (5). Schließlich wird der Dienst ausgeführt (*service execution*), indem der entsprechende Datenteil des CAPs und eine Dienstreferenz zum *Executor* gesendet werden.

Die folgenden Abschnitte erläutern die einzelnen Komponenten des SIPs. Die Komponente *Interaction Organizer*, die zur Generierung von CAPs dient, wird im nachfolgenden Kapitel behandelt.

CAP-Router

Sobald der SIP ein CAP empfängt, wird es zur Komponente *Router* weitergeleitet. Der *Router* realisiert im Wesentlichen folgende Aufgabenbereiche:

1. Verwaltung von logischen Netzwerkverbindungen
2. Routing von CAPs basierend auf Constraints
3. Weiterleiten des CAPs zum *CAP-Matcher*

Die Verwaltung von logischen Netzwerkverbindungen bezieht sich auf Interaktionen mit mobilen Engeräten. Es wird von der Annahme ausgegangen, dass Netzwerkverbindungen u.U. nur für einen kurzen Zeitraum existieren, diese aber zu einem späteren Zeitpunkt wieder aufgebaut werden können, auch dann wenn sich die Position des Endgerätes (bez. des Netzwerkes, Adresse) verändert hat. So könnte ein Benutzer temporär eine Verbindung zu einem lokalen SIP aufbauen, die nur so lange existiert, bis ein CAP mit den entsprechenden Dienstanforderungen übertragen wurde. Diese Art von Interaktionen wird insbesondere durch aufkommende Ad-hoc-Netzwerke wie etwa Bluetooth (vgl. 2.5.1) unterstützt.

Dennoch ist es manchmal erforderlich, den Benutzer auch nach erfolgreicher Übertragung eines CAPs zu kontaktieren. Dieser Fall kann beispielsweise dann eintreten, wenn bei der Auswertung des CAPs ein nicht behebbarer Fehler auftritt oder das CAP zur Ausführung eines Dienstes die explizite Bestätigung eines Benutzers erfordert (*Interactive CAP*).

Zum Wiederaufsetzen einer Verbindung unterstützt der Router je nach Systemumgebung verschiedene Optionen. Zum einen besteht die Möglichkeit, Mobile IP

[MIP] einzusetzen. Zum anderen kann die Rückmeldung selbst per CAP erfolgen, so dass die Routing-Dienste der Infrastruktur in Anspruch genommen werden können. Diese Methode kann beispielsweise dann erfolgreich angewandt werden, wenn die gegenwärtige Position eines Benutzers stetig mittels eines Tracking-Systems (z.B. *Active Badge* [Badge]) bestimmt werden kann.

Der zweite Aufgabenbereich bezieht sich auf die Weiterleitung von CAPs bez. symbolischer Attribute, z.B. „Raum D2 in Gebäude 10“, „In einem nahliegenden Raum“ oder „Wo der Benutzer ist“. Zur Implementierung dieser Funktionalität wurde eine Hierarchie von Routern gebildet, welche die Struktur eines Campus (Gebäude, Flure, Räume) reflektiert. Dieser Ansatz der Strukturierung ist analog zu [WH 99]. Generell wird die Wahl des eingesetzten Routingverfahrens immer durch die vom System unterstützten Lokalisierungsmethoden bzw. durch die Vernetzung der Domänen determiniert.

Schließlich kooperiert der Router auch mit der *Matcher* Komponente. Ein eingehendes CAP wird zum Matcher weitergeleitet, um zu prüfen, ob es sich auf die gegenwärtig erreichte Domäne bezieht bzw. eine Dienstinteraktion ausgeführt werden kann. Ist dies nicht der Fall, wird es vom Router gemäß den Routing-Vorschriften zu einem anderen SIP weitergeleitet.

Divergence		**Component/Action**
Location	***Vicinity***	**Router: try proximate domains**
	Other Location	**Router: route to specified address**
User Interaction		**Router: indicate state, user confirmation (interactive CAPs)**
Trigger		**Event Monitor: wait for matching situation**
Wrong Interface		**Matcher: convert data, nested CAP (siehe 3.4.3)**
Equal		**Executor: apply data on selected service**
General Error		**Router: Error message to user; Matcher: abort processing**

Tabelle 3-1: SIP Aktionstabelle

CAP-Matcher

Der *Matcher* ist für den Vergleich von Context Constraints eines CAPs mit aktuell gemessenen Sensordaten bzw. Kontextinformationen zuständig (vgl. 3.3.2). Wird eine Übereinstimmung festgestellt, so werden die assoziierten Daten des CAPs an den CAP Executor weitergeleitet. Der selektierte Dienst wird ausgeführt.

Ansonsten berechnet der *Matcher* die Abweichung (*Divergence*) zwischen Context Constraints und ermittelten Kontext. Im hier betrachteten Zusammenhang

bezeichnet die Abweichung, warum ein spezifizierter Dienst nicht selektiert oder ausgeführt werden konnte.

Wird beispielsweise im Rahmen eines Matching-Prozesses festgestellt, dass ein CAP für eine andere Dienstdomäne bestimmt ist, so bezieht sich die Abweichung auf den Ort (*Location*). Möglicherweise veranlaßt der Matcher diesbezüglich eine Weiterleitung des CAPs zu einer anderen Domäne. Zusammenfassend kann man feststellen, dass die Ermittlung der Abweichung dazu dient, dass das Systemverhalten in Übereinstimmung bez. der Context Constraints gesteuert werden kann.

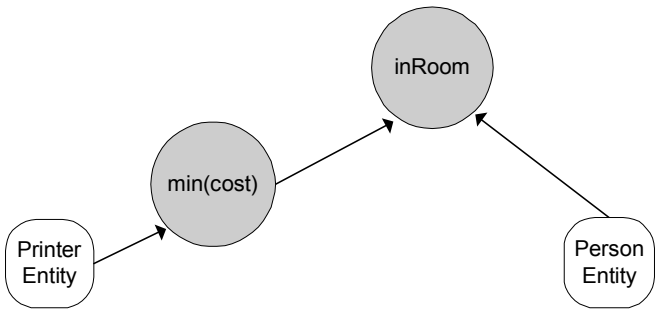

Abbildung 3-16: „Der kostengünstigste Drucker, der sich im selben Raum wie der Benutzer befindet“

Die resultierenden Abweichungen können nach folgenden Ereignistypen kategorisiert werden: *Location*, *User Interaction*, *Wrong Interface*, *Equal* und *General Error*. Tabelle 3-1 listet mögliche Abweichungstypen und deren assoziierte Aktionen auf. Der Abweichungstyp *Location* ist in zwei Untertypen kategorisiert worden: *Vicinity* und *Other Location*. *Vicinity* bezieht sich auf eine Menge benachbarter Dienstdomänen; *Other Location* bezieht sich auf genau eine spezifizierte Domäne.

Evaluation von Relationen. Innerhalb des Matchers werden geschachtelte Relationen eines CAPs als Bäume repräsentiert. Knoten repräsentieren Relationen und Blätter Entitäten. Abbildung 3-16 zeigt ein einfaches Beispiel, welches genutzt werden kann, um den kostengünstigsten Drucker auszuwählen, der sich im selben Raum befindet wie der Benutzer. Wie bereits erwähnt, erfordert die Relation *inRoom,* dass sich die enthaltenen Entitäten im selben Raum befinden. Die Relation *min(X)* selektiert aus einer Menge von Entitäten jene, welche bez. des Attributes X ein Minimum bildet.

Der Algorithmus zur Auswertung der Relationen bearbeitet den Baum nach einer *bottom-up* Strategie. In diesem Fall wird zunächst die *min(cost)* ausgewertet und anschließend die Relation *inRoom*. Bei Anwendung des Algorithmus wird die

Menge der in Frage kommenden Kandidaten (Entitäten) schrittweise verkleinert. Schließlich verbleibt eine Restmenge von Kandidaten, welche die Bedingungen aller Relationstypen erfüllt. Von diesen Kandidaten wird dann eine Entität zufällig ausgewählt.

Zum Zweck der Dienstvermittlung generiert das System Dienstanfragen basierend auf den Attributwerten der in Frage kommenden Entitäten. Das Dienstvermittlungssystem antwortet mit allen Diensten, die den spezifizierten Attributen genügen. In manchen Fällen ist es erforderlich, dass die Attribute der Entitätsbeschreibungen in ein Format konvertiert werden, welches dem eingesetzten Dienstvermittlungsprotokoll entspricht. Die Konvertierungsmaßnahmen wurden mittels des *Typed Object Models* (TOM) realisiert [WO 98]. Für das hier beschriebene System wurde SLP [Gutt 99] zur Dienstvermittlung eingesetzt. Es ist jedoch auch der Einsatz anderer Dienstvermittlungsprotokolle, wie etwa JINI [JINI] oder SLP [Gutt 99] möglich, ohne dass dadurch die Arbeitsweise anderer Systemkomponenten betroffen wäre. Die Anbindung der Dienstvermittlungsprotokolle erfolgt über das *Strategy Pattern* [GHJV 95].

Evaluation von Events. Die Auswertung der Relationen resultiert in der Bestimmung eines gewünschten Dienstes. Im Kontrast dazu steuern Events die Ausführung eines präselektierten Dienstes.

Für jedes in einem CAP definierte Event erzeugt der Matcher einen Event-Monitor Prozess. Ein Event-Monitor überwacht das Eintreffen eines in Frage kommenden Ereignisses. Wird ein bestimmtes Ereignis festgestellt, so wird der Matcher davon in Kenntnis gesetzt. Nachdem alle spezifizierten Ereignisse eines CAPs unter Berücksichtigung ihrer zeitlichen Abfolge festgestellt wurden, werden die entsprechenden Event-Monior Prozesse beendet. Anschließend leitet der Matcher die Daten des CAPs und eine Referenz des selektierten Dienstes zum *CAP-Executor*. Dieser führt den Dienst dann letztlich aus.

CAP-Executor

Der CAP-Executor bildet eine Schnittstelle zu den Diensten einer von einem SIP verwalteten Dienstdomäne. Der CAP Executor wendet den entsprechenden Datenteil des CAPs auf den selektierten Dienst an. Der Dienst wird durch eine Referenz identifiziert. Analog zum Matcher erlaubt auch der Executor die Anbindung verschiedener Dienstsysteme, so etwa JINI [JINI].

3.4.2 Lebenszyklus

Der Prozess einer Kontextadaptiven Interaktion bzw. der Evaluation eines CAPs wird durch zwei Faktoren bestimmt: zum einen durch Context Constraints (vgl. 3.3.2) und zum anderen von den aktuell bestimmten Kontextinformationen.

Der Lebenszyklus eines CAPs wird in Abbildung 3-17 durch einen Zustandsautomaten beschrieben. Es werden folgende Zustände unterschieden: *Find, Wait, Ready*, *Talk To,* und *Terminated.* Der Zustandsautomat beschreibt die Auswertung eines *one shot* CAPs mit einem enthaltenen Context Constraint (vgl. 3.3). *Multi Shot* CAPs oder CAPs mit multiplen Context Constraints lassen sich durch wiederholte Anwendung des Lebenszyklus abbilden.

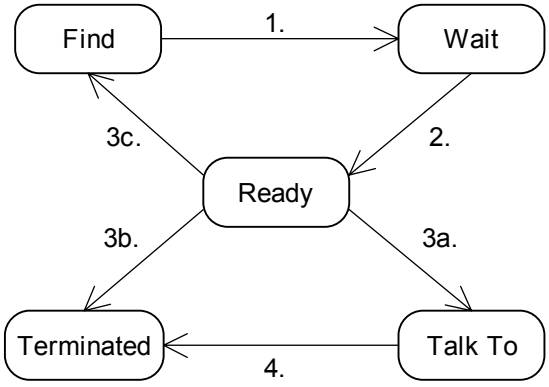

Abbildung 3-17: CAP Lebenszyklus

Find

Der *Find* Zustand benennt den initialen Zustand der Auswertung eines CAPs. In dieser Phase wird innerhalb eines Routingprozesses eine Dienstdomäne (vgl. 3.4) lokalisiert, welche für eine mögliche Dienstausführung in Frage kommt. Die Auswahl der Dienstdomäne basiert auf dem korrespondierenden Context Constraint.

Die einzig mögliche Transition vom Zustand *Find* führt zum Zustand *Wait*.

Wait

Hat die Auswertung den Zustand *Wait* erreicht, so ist eine geplante Dienstnutzung möglich. Alle räumlichen Bedingungen eines Context Constraints sind in der gewählten Domäne erfüllt. Und das System geht davon aus, dass auch die übrigen kontextuellen Einschränkungen erfüllt werden. Zu diesem Zweck werden die in Frage stehenden Kontexttypen kontinuierlich überprüft. Kann schließlich die in den Context Constraints geforderte Situation festgestellt werden, wechselt der Auswertungsprozess in den Status *Ready*.

Ready

In diesem Status entspricht die gegenwärtige Situation den Anforderungen, und der spezifizierte Dienst kann ausgeführt werden. Wird das CAP im *Non Interactive Mode* ausgewertet, so erfolgt eine unmittelbare Überführung in den Zustand *Talk To*. In diesem Fall erfolgt eine Dienstinteraktion. Im *Interactive Mode* hingegen ist zunächst eine explizite Bestätigung des Benutzers erforderlich, erst dann erfolgt eine Dienstinteraktion. Erfolgt eine negative Bestätigung, so wird die Evaluation terminiert (Transition zum Zustand *Terminated*).

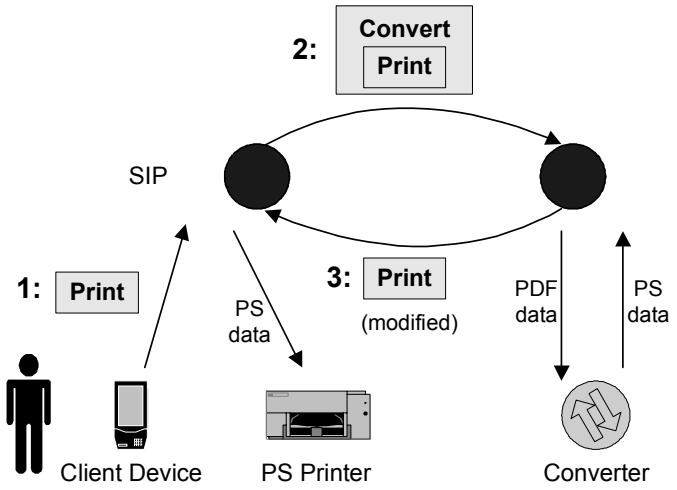

Abbildung 3-18: Ad hoc Dienstkomposition

Schließlich kann die Verarbeitung wieder in den *Find*-Zustand zurückfallen, da der gegenwärtig festgestellte Kontext nicht mehr dem der Context Constraints enthaltenen Anforderungen gleichkommt. Dies kann vor allem dann geschehen, wenn mobile Objekte Bestandteil der Context Constraints sind. Sollte sich die Position dieser Objekte verändern, sind assoziierte Einschränkungen u.U. nicht mehr erfüllt.

Talk To

Im *Talk To* findet die eigentliche Dienstinteraktion statt. Es werden also entweder Daten des CAPs auf einen bestimmten Dienst angewendet oder es werden Daten von einem Dienst angefordert, welche womöglich auf einen anderen Dienst angewendet werden. Nach einer erfolgreichen Dienstinteraktion ist die Evaluation eines CAPs beendet. Es folgt eine Transition in den Zustand *Terminated*.

Terminated

Ist schließlich der Zustand *Terminated* erreicht, so ist die Evaluation eines CAPs in Folge eines Abbruchs oder einer erfolgreich ausgeführten Dienstinteraktion abgeschlossen. Auf das entsprechende CAP kann nicht mehr zugegriffen werden; assoziierte Ressourcen werden freigegeben.

3.4.3 Beugung eines Dienstpfades

In den vorigen Abschnitten ist die Evaluation von CAPs in einem Verteilten System erläutert worden. Hier soll nun noch ein Sonderfall beschrieben werden: *die Beugung eines Dienstpfades*. Normalerweise werden die geplanten Dienstinteraktionen eines CAPs bei der Erstellung festgelegt. In manchen Fällen kann es jedoch hilfreich sein, wenn die durch das CAP definierte Kontextadaptive Interaktion zur Laufzeit verändert werden kann. Diese Funktionalität kann insbesondere in bei der Behandlung von Fehlern vorteilhaft eingesetzt werden.

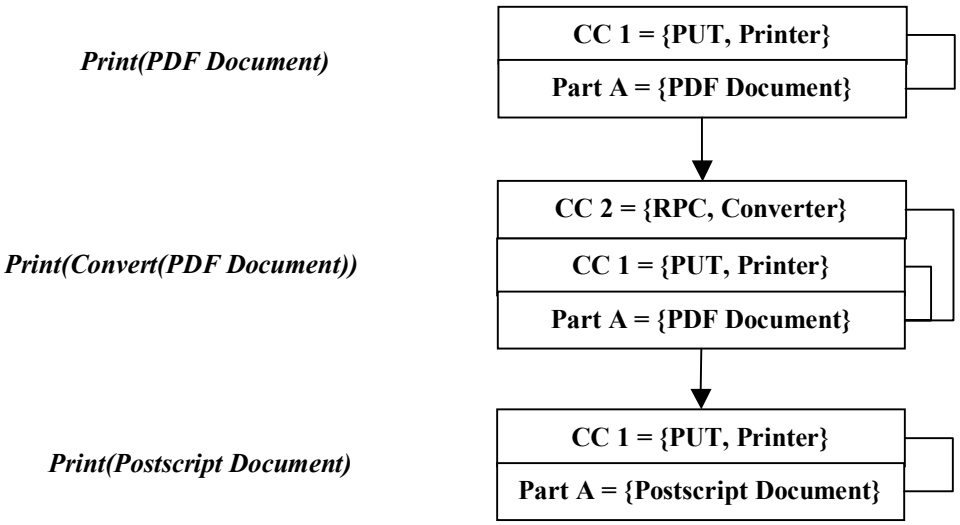

Abbildung 3-19: Phasendiagramm für Ad-hoc-Dienstkomposition

Abbildung 3-18 zeigt ein Beispiel für eine Ad-hoc-Dienstkomposition. In diesem Beispiel möchte der Benutzer von einem PDA aus den Ausdruck eines Dokuments im *Portable Document Format* (PDF) in Auftrag geben. Der in der Umgebung des Benutzers befindliche Drucker akzeptiert aber nur Dokumente im Postscript-Format. Anstatt dem Benutzer mitzuteilen, dass sein Druckauftrag nicht ausgeführt werden kann, veranlasst das System eine Konvertierung des Dokuments in das PDF-Format. Schließlich wird das Dokument, wie gewünscht, gedruckt.

Zu Erreichung dieser Funktionalität wird das ursprüngliche CAP *Print* gekapselt, d.h. der Druckauftrag wird zurückgestellt und das Dokument wird zunächst konvertiert. Nachdem die Konvertierung erfolgreich vonstatten gegangen ist, wird der ursprüngliche Druckauftrag wiederhergestellt, jetzt allerdings mit konvertierten Daten.

Die hier vorgestellte Dienstkomposition wurde mittels einer RPC-Interaktion (vgl. 3.3.3) realisiert. Abbildung 3-19 demonstriert dies anhand eines Phasendiagramms der einzelnen Evaluationschritte.

3.5 Referenzimplementierung

Der Architekturvorschlag *Context-Aware Packets Enabling Ubiquitous Services* (CAPEUS) [SMLP 01b] implementiert die oben beschriebenen Konzepte zur kontextadaptiven Dienstnutzung in Verteilten Systemen.

3.5.1 Datenrepräsentation von CAPs

Die Datenrepräsentation von *Context-Aware Packets* (CAPs) (siehe 3.3) erfolgt im Kontext von CAPEUS mittels XML[8] [Har 99, GP 00]. XML kann als Metasprache bezeichnet werden, da sie Sprachkonstrukte zur Definition von Markup-Sprachen (z.B. HTML [HTML]) festlegt. XML hat sich in den letzten Jahren als ein vorrangiges Konzept zum Datenaustausch in Verteilten Systemen etabliert, da es Interoperabilität auch in heterogenen Umgebungen durch ein offenes, flexibles, standardbasiertes Format unterstützt. Für den Einsatz in CAPEUS hat sich das hierarchische Strukturierungsschema von XML angeboten.

Für den Einsatz von XML stehen einige Hilfsmittel zur Verfügung, so existieren insbesondere Browser und Parser. Für das Parsing von CAPs bereits bestehende XML Parser eingesetzt werden, wodurch die Implementierung wesentlich erleichtert wurde. Mittlerweile existiert eine Reihe von XML Parsern mit unterschiedlichen Charakteristika. So z.B. folgende: XML4C [X4C], GNOME [GNM], XERCES-J [XRC], XJ-PARSER [XJP] und JAXP [JXP]. Eine vollständige Übersicht findet sich in [XSS]. Alle Parser implementieren eine Standardschnittstelle, welche das Lesen bzw. die Modifikation von XML Daten ermöglichen. Zudem werden Dienste zur syntaktischen Validierung angeboten.

Die Repräsentation von CAPs mittels XML ist Gegenstand des nachfolgenden Abschnitts.

XML Grammatik

XML benutzt so genannte *Tags* zur Abgrenzung von Datenelementen in einer Datenstruktur. XML-Tags spannen einen Datenbaum auf, wobei jedes Tag ein

[8] e**X**tensible **M**arkup **L**anguage

individuelles Datenelement identifiziert. Tags werden zusammengesetzt aus einem *Start*- und einem *End-Tag*. Der durch ein Start- und End-Tag abgegrenzte Datenbereich konstituiert ein Datenelement innerhalb eines XML Datenbaums. Die Schachtelung von Tags lässt eine Baumstruktur entstehen, welche sich insbesondere dazu eignet, die hierarchischen Strukturen eines CAPs abzubilden.

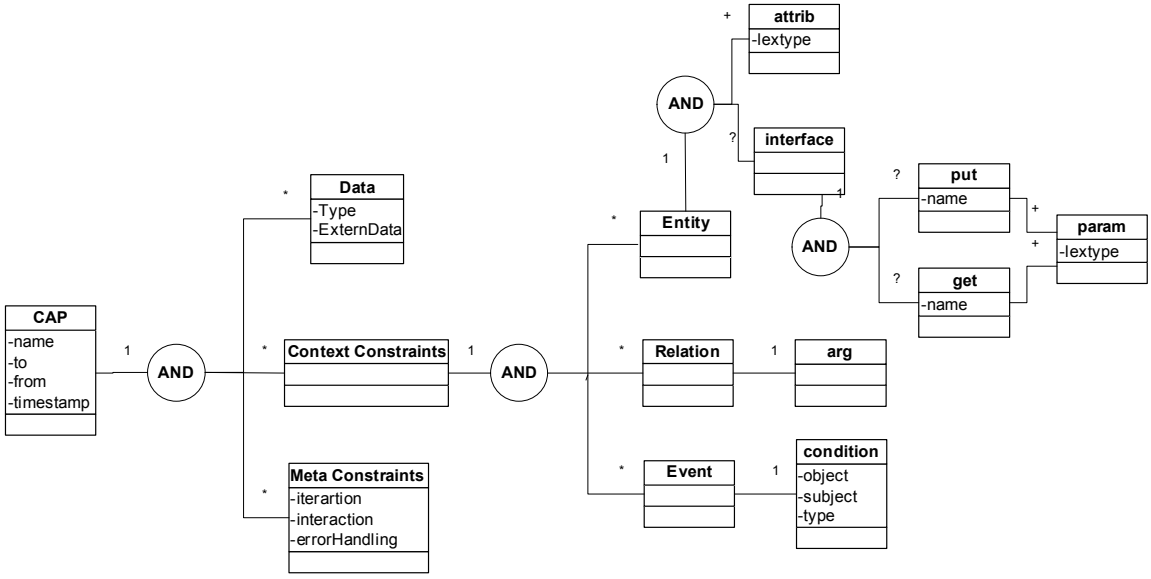

Abbildung 3-20: XML Strukturdiagramm für CAP

Tags können um *Attribute* erweitert werden, die eine zusätzliche Spezifikation der durch ein Tag eingeschlossen Daten ermöglichen. Die Interpretation eines XML Dokuments ist gänzlich anwendungsspezifisch. Die Document Type Definition (DTD) [GP 00] dient zur Deklaration von XML-Tags und anderen grammatikalischen Einschränkungen. Eine DTD beschreibt die Struktur eines XML Dokumenttyps und ist ein wesentlicher Bestandteil der XML Konzeption. Im Wesentlichen beschreibt sie, welche Tags in einem Dokument erlaubt sind, wie diese geschachtelt werden können und welche Attribute gültig sind. Eine ausführliche Erläuterung der DTD kann beispielsweise in [GP 00] gefunden werden. Im Folgenden soll die DTD für CAPs vorgestellt werden. Abbildung 3-20 illustriert die DTD eines CAPs anhand eines Strukturdiagramms gemäß der Richtlinien aus Abschnitt 3.3. Die Abbildung reflektiert den hierarchischen Aufbau eines CAP Dokumentes. Jedes UML Klassenelement repräsentiert im Kontext der Abbildung einen Knoten des XML Baums. Erweitert wurde jedes Element durch die korrespondierenden XML Attribute. Verbunden sind die einzelnen Elemente durch Assoziationen. In der Abbildung bedeutet das Symbol „?“: eins oder keins, „1“: genau eins, „*“: beliebige Anzahl und „+“: mindestens eins. Die Abbildung ist von links (dem Wurzelement *CAP*) nach rechts zu lesen.

Abbildung 3-21 zeigt ein Beispiel für ein CAP Dokument nach der oben beschriebenen DTD. Es handelt sich um eine einfache, einstufige Kontext-adaptive Interaktion, die einen Druckauftrag formuliert.

3.5.2 Statische Sicht

CAPEUS ist in Java [Java] implementiert worden. Dieser Abschnitt erläutert die konstituierenden Komponenten des Service Interaction Proxies (SIP), wie er in 3.4 beschrieben wurde. Die übrigen Komponenten (nicht dem SIP zugehörig) sind applikationsspezifisch und werden hier nicht erläutert.

```
...
<CAP name="printing_locally" from="Florian"
put_to="Printer">

<ContextConstraint>

   <Entity type="abstractDevice" name="Printer">
         <attrib lextype="colors">greyscale</attrib>
         <attrib lextype="papersize">a4</attrib>
         ...
         <interface>
          <put name="lpr">
           <param lextype="lpr_filter">no</param>
           <param lextype="MIMEtype">
                  stream/postscript
           </param>
           ...
          </put>
       </interface>

</Entity>
```

```
<Entity type="Person" name="Florian">
      <attrib lextype="date">02.02.2001 13:45</attrib>
      <attrib lextype="role">guest</attrib>
      ...
</Entity>

<Relation name="inRoom">
      <arg>Florian</arg>
      <arg>Printer</arg>
 </Relation>

 </ContextConstraint>

<Data type="application/postscript">
  <![CDATA[ %!PS-Adobe-2.0
    ...  ]]>
 </Data>

<MetaConstraint iteration="one-shot">

 </CAP>
```

Abbildung 3-21: Repräsentation eines CAPs basierend auf XML

Abbildung 3-22 gibt einen Überblick über die Pakete von CAPEUS. Das Hauptpaket *capeus*[9] besteht aus zwei Klassen – *Router* und *CAP* – und den Paketen (*packages*) *cap*, *process*, *corba*, *slp*, *exec, util* und *model*.

Router agiert als Hauptklasse des Systems und macht Gebrauch von den übrigen Paketen und Klassen. Sowohl *Router* als auch *CAP* kapseln die konzeptionellen Grundlagen aus den Abschnitten 3.3 und 3.4. Also insbesondere die Repräsentation von CAPs und deren Evaluation in einem Netzwerk von SIPs.

Im Folgenden werden einige Implementierungsaspekte der in Abbildung 3-22 dargestellten Java-Klassen und Pakete erläutert. Die Pakete *util*, *slp*, *corba* und *exec* werden in der folgenden Ausführung nicht berücksichtigt, da sie im Wesentlichen nur Unterstützungsdienste leisten, aber für das Verständnis der

[9] Nach Konvention werden Paketnamen klein geschrieben.

Kernfunktionen nicht unbedingt erforderlich sind. Eine vollständige und detaillierte Darstellung findet sich in [Mi 01].

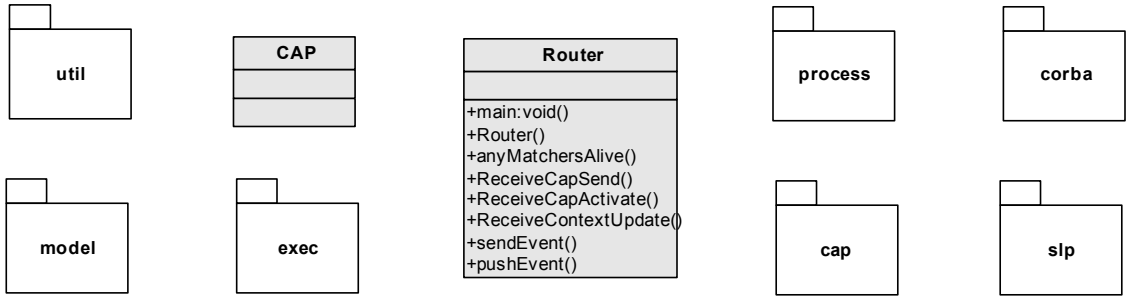

Abbildung 3-22: CAPEUS Pakete

Klasse *Router*

Die Router Klasse implementiert die zentrale Verarbeitung von CAPs innerhalb von CAPEUS. Insbesondere wird hier auch der in 3.4 vorgestellte Routing-Prozess realisiert sowie der Empfang und die Weiterleitung von eingehenden CAPs. Infolgedessen erzeugt Router auch neue *Matcher* (s.u.) Threads zum Abgleich von Context Constraints eines CAPs mit tatsächlich ermittelten Kontextinformationen. Eingehende Ereignisse werden an die entsprechenden Matcher Prozesse weitergeleitet, so z.B. Benutzerbestätigungen für CAPs im *Interactive mode*. Auch das Senden von Zustandsnachrichten über den Verarbeitungsprozess eines CAPs wird vom Router- Objekt veranlasst.

Klasse *CAP*

CAP bezeichnet die zweite Klasse im Hauptpaket *capeus*. Diese Klasse spezifiziert die Verarbeitung des CAP Datenformats (vgl. 3.5.1). Empfangene CAPs werden in Java-Klassen konvertiert, können dann manipuliert werden und anschließend wieder in ein CAP konformes XML Dokument umgewandelt werden. CAP erhält die Referenz auf ein XML Dokument vom Router-Objekt. Das eingehende CAP wird syntaktisch analysiert und später bez. der assoziierten DTD verifiziert. Schließlich werden die Daten des CAPs in Objekten des Pakets *cap* gespeichert, wo sie dann den Objekten *Router* und *Matcher* zur Evaluation zur Verfügung stehen.

Für die syntaktische Analyse wurde der Parser JAXP [JXP] der Firma Sun™ gewählt. Bei JAXP handelt es sich um einen so genannten SAX [SAX 00] Parser. Im Kontrast zu einem DOM [Woo98] Parser, der als Resultat der syntaktischen

Analyse das gesamte XML Dokument in einer Baumstruktur speichert, erlaubt SAX die sequentielle Analyse eine XML Dokuments. Als Konsequenz können mittels SAX auch Dokumente geparst werden, deren Größe die Speicherkapazität der verarbeitenden Ressource übersteigt. Dieser Gesichtspunkt ist vor allem im Bereich von Ubiquitous Computing Umgebungen von Bedeutung, da hier einige Ressourcen möglicherweise nur über beschränkte Systemressourcen verfügen.

Wie bereits erwähnt, erfolgt die syntaktische Analyse eines eingehenden CAPs in Entsprechung der zugehörigen CAP DTD. Sollte ein CAP nicht der geforderten Grammatik genügen, so wird der Evaluierungsprozess abgebrochen und die sendende Instanz benachrichtigt.

Parser innerhalb von CAPEUS werden über ein *Abstract Factory* [GHJV 95] Designmuster eingebunden. Somit ist es möglich, den unterliegenden Parser einfach auszutauschen, d.h. ohne dass die Modifikation anderer Klassen von CAPEUS erforderlich wäre.

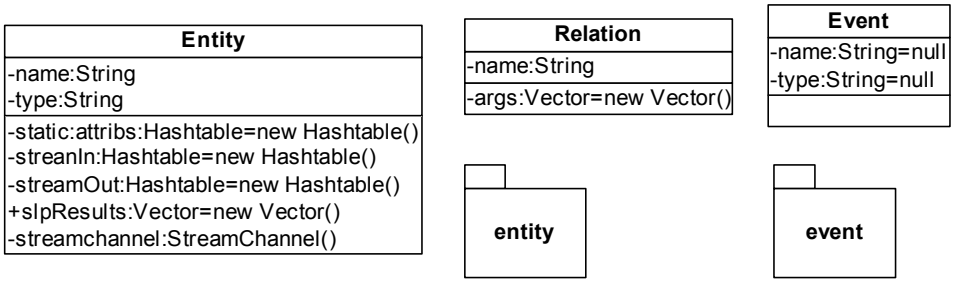

Abbildung 3-23: Paket cap: Übersicht

Paket *cap*

Innerhalb des *cap* Paketes wird die Repräsentation von *Context Constraints* implementiert (vgl. 3.3.2). Speziell wird die Abbildung von Entitäten, Relationen und Ereignissen implementiert. Das *cap* Paket besteht aus drei Klassen – *Entity*, *Relation* und *Event* – und zwei Unterpaketen – *entity* und *event*.

Entity ermöglicht den Zugriff und die Manipulation von Attributen einer CAP Entität. Einzelnen Attribute werden in einer Hashtabelle gespeichert. *StreamIn* und *StreamOut* korrespondieren zu den *Put* und *Get* Direktiven der CAP Schnittstellenbeschreibung.

Die Klasse *Relation* spezifiziert den Namen einer Relation und deren Mitgliedsentitäten.

Schließlich dient die Klasse *Event* und das zugehörige Paket *event* zur Repräsentation von CAP Events. Neben den Typ- und Namensattributen muss jedes Event mit einer Bedingung (*trigger condition*) verknüpft werden. Diese

Bedingungen werden mit zwei Klassen des Unterpaketes *event* modelliert: *JuncTerm* und *CondTerm*, siehe Abbildung 3-24.

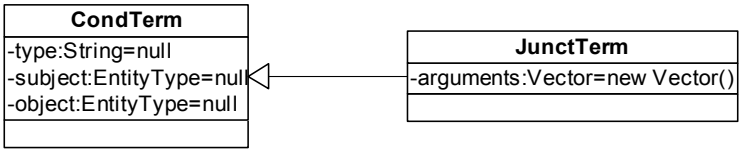

Abbildung 3-24: Paket *event*: Übersicht

CondTerm ermöglicht die Darstellung von logischen Grundtermen, die mittels der Klasse *JuncTerm* durch *AND* und *OR* verknüpft werden können. Zudem ist es möglich, Terme dieser Art in beliebiger Weise zu schachteln.

Zusammenfassend lässt sich feststellen, dass das *cap*-Paket die interne Repräsentation von *Context Constraints* innerhalb von CAPEUS implementiert.

Paket *process*

Das Paket *process* enthält aktive Komponenten, die den in 3.3.2 beschriebenen Matchingprozess für *Context Constraints* implementieren. Insbesondere gehören dazu die Klassen *Matcher* und *EventMonitor*. Die entsprechenden Objekte werden als Threads ausgeführt. Als Konsequenz können mehrere dieser Objektinstanzen parallel ausgeführt werden, was eine simultane Auswertung von multiplen CAPs auf einem *Service Interaction Proxy* (SIP) ermöglicht.

Jedes *Matcher* Objekt unterstützt die Evaluation eines individuellen CAPs während des gesamten Lebenszyklus. Dabei wird insbesondere der Matchingprozess (vgl. 3.4) ausgeführt. Es wird zwischen dynamischen und statischen Attributen unterschieden (*matchStaticAttributes, matchDynamic-Attributes)*. Die statischen Attribute werden mittels einer SLP [Gutt 99] Anfrage ermittelt, siehe auch *slp* Paket. Die dynamischen Attribute werden über das Objekt *EventMonitor* überwacht. Zudem ist der *Matcher* für das Aussenden von Status-informationen bez. des Evaluationsprozesses zuständig und die erforderliche Kommunikation (z.B. Einholen einer Benutzerbestätigung) für interaktive CAPs. Nach der vollständigen Abarbeitung eines CAPs wird der entsprechende *Matcher* Thread terminiert.

EventMonitor ist die zweite Klasse im Paket *process*. Wie in Abbildung 3-25 zu sehen ist, ist sie abhängig von der oben beschriebenen *Matcher* Klasse. Einem *Matcher* Objekt können mehrere *EventMonitor* Objekte zugewiesen sein. Jeder *EventMonitor* dient der Überwachung eines einzelnen CAP Events (siehe 3.3.2). Folglich können bei der Auswertung eines individuellen CAPs mehrere *Event-Monitor* Objekte einbezogen werden. Ein Beispiel für ein zu überwachendes Event

ist *closeby*. Das parametrisierte Event *closeby* ist erfüllt, falls sich zwei oder mehrere Objekte in einem eingeschränkten Raumsegment befinden.

Die Verwaltung von Relationen und Events wird von Klassen des Pakets *model* (s.u.) geleistet.

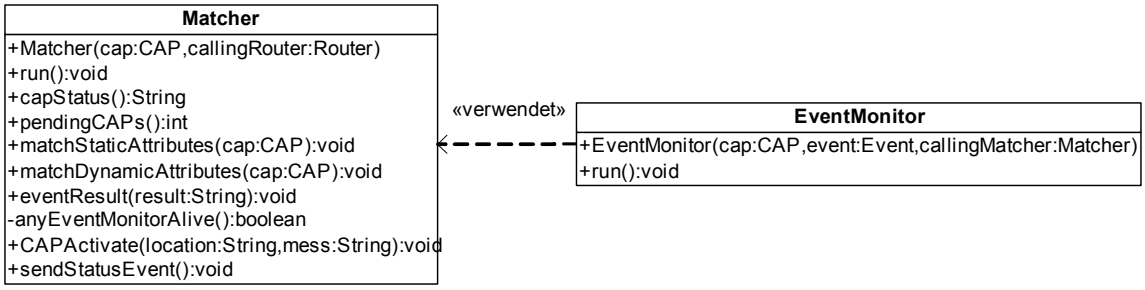

Abbildung 3-25: Paket *process*: Matcher und EventMonitor

Paket *model*

Das Paket *model* spezifiziert insbesondere Klassen zur Verwaltung von unterstützten Events und Relationstypen. Durch eine *plug-in* Architektur kann das System zur Laufzeit um Events und Relationstypen ergänzt werden. Standardmäßig implementiert *model* ein Grundpaket von Events und Relationen, dazu gehören beispielsweise das Event *closeby* und die Relation *inRoom*.

3.5.3 Dynamische Sicht

Um ein besseres Verständnis für die Funktion von CAPEUS zu vermitteln, sollen nun die Interaktionen zwischen einzelnen Komponenten des Systems dargelegt werden, um die in Abschnitt 3.5.2 beschriebene statische Sicht zu ergänzen.

Das Aktivitätsdiagramm in Abbildung 3-26 veranschaulicht die Evaluierung eines CAPs innerhalb von CAPEUS. Auch hier wird nur die Verarbeitung eines CAPs mit genau einem *Context Constraint* berücksichtigt, was genau einer Dienstinteraktion entspricht. Da Context Constraints immer unabhängig voneinander sind, kann der durch das Aktivitätsdiagramm beschriebene Prozess wiederholt angewendet werden, um ein mehrstufiges CAP zu behandeln.

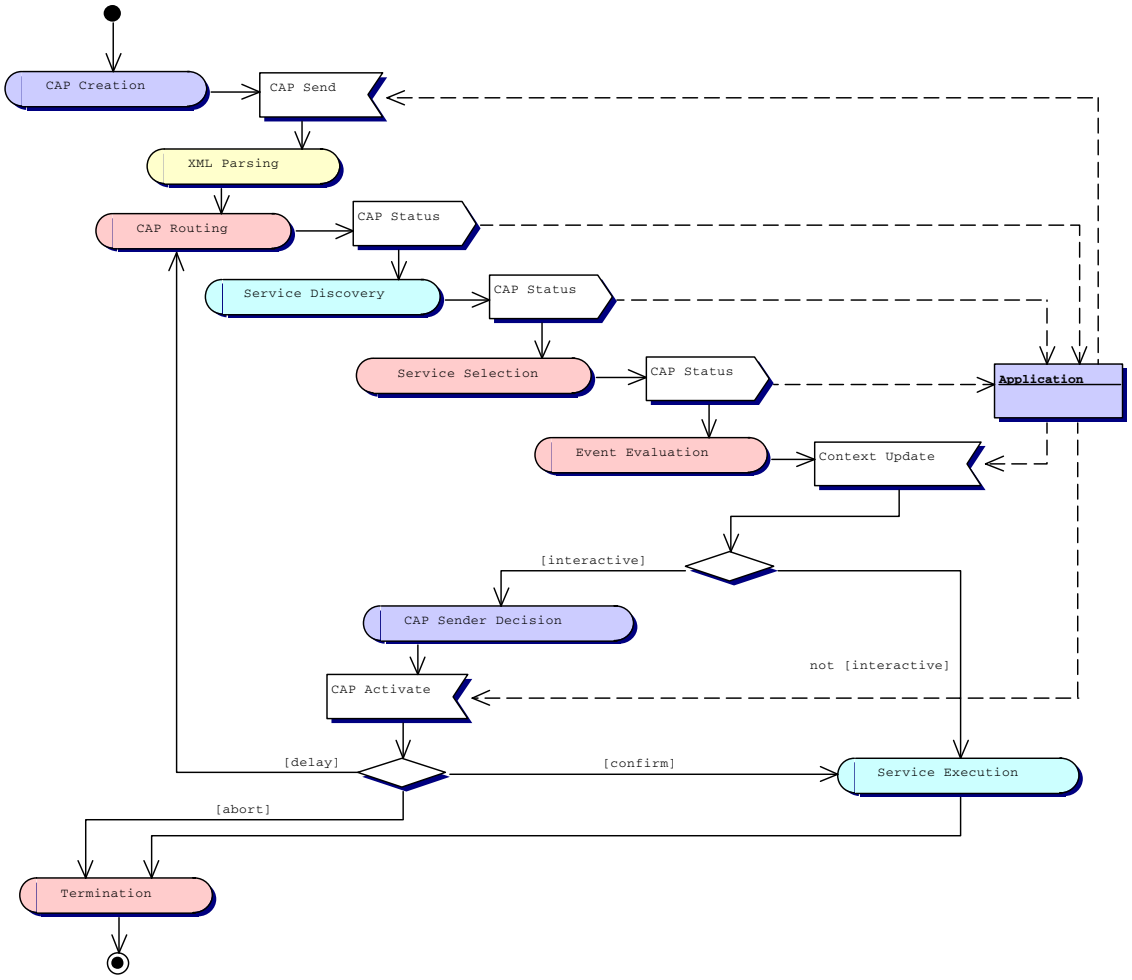

Abbildung 3-26: Aktivitätsdiagramm CAP Evaluation

In Anlehnung an Abschnitt 3.5.2, in dem die Hauptkomponenten von CAPEUS bez. ihrer Funktion vorgestellt wurden, liefert Abbildung 3-27 eine Übersicht über die Interaktion zwischen diesen Komponenten. Das Interaktionsdiagramm orientiert sich auch an der Auswertung eines einstufigen CAPs und beschreibt diesen Prozess von der initialen Sendung des CAPs, der daraus resultierenden Dienstinteraktion und dem Abschluss: der Auswertung und dem Verwerfen eines CAPs.

Zu Beginn empfängt das *Router*-Objekt das CAP und die XML Repräsentation wird an das *CAP* Objekt weitergeleitet, wo es geparst wird. Die resultierenden Java Objekte, welche nun das eingehende CAP verkörpern, werden an das *Router* Objekt zurückgegeben. In einem dritten Schritt kreiert der *Router* ein *Matcher* Objekt, welches die anschließende Evaluation des CAPs bewerkstelligt. Von nun an ist das *Router* Objekt wieder bereit, ein neues CAP zu empfangen.

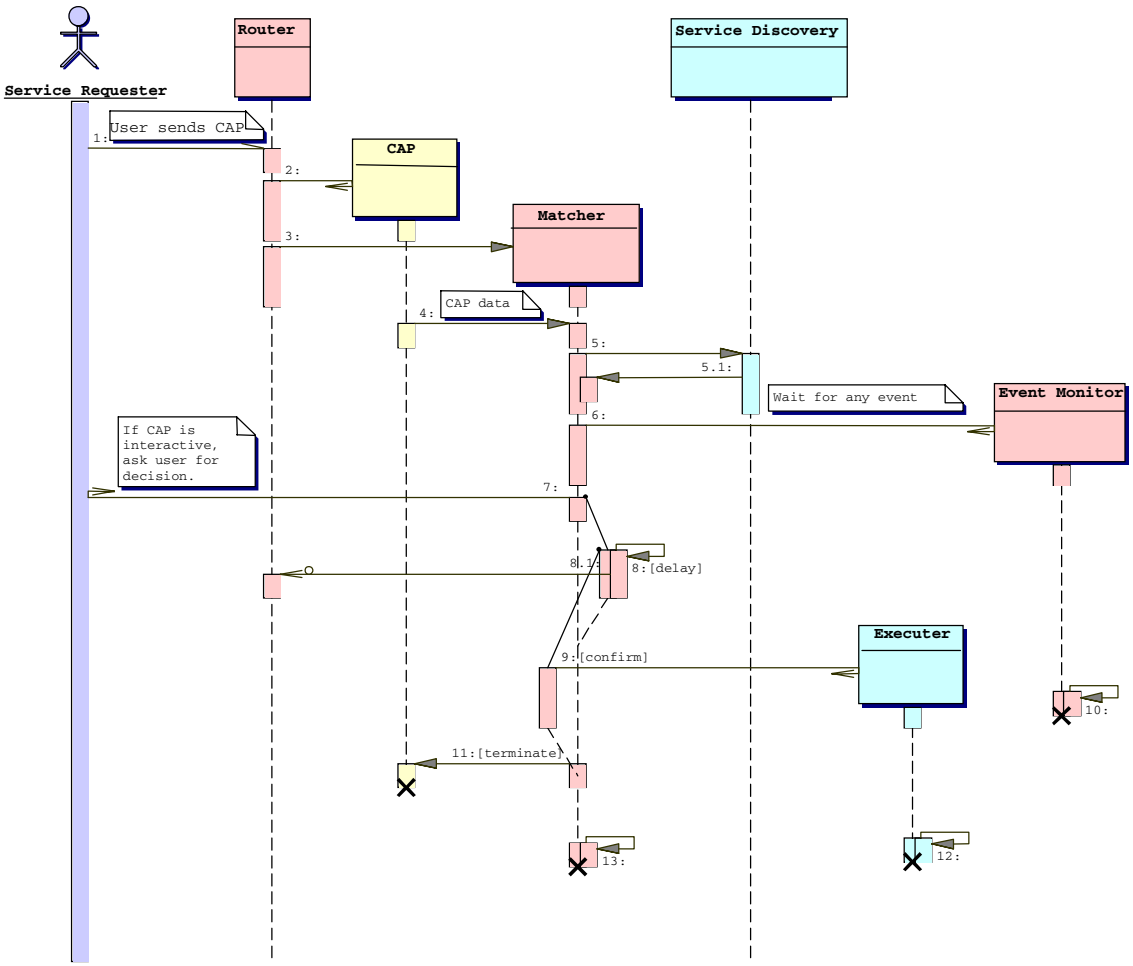

Abbildung 3-27: Interaktionsdiagramm CAPEUS

Der neu kreierte *Matcher* lokalisiert in Frage kommende Dienste mittels des *Service Discovery Objektes* (Paket slp). Im sechsten Schritt erzeugt der Matcher Instanzen der Klasse *EventMonitor*, diese werten die zeitlichen Bedingungen eines *Context Constraints* aus. Diese Auswertung führt entweder zum Abbruch der Evaluation eines CAP (*abort*) oder zu einer Dienstausführung.

Zudem können interaktive CAPs ausgeführt, verzögert oder abgebrochen werden. Dies ist abhängig von der expliziten Bestätigung des Benutzers. Zu diesem Zweck kommuniziert der *Matcher* mit dem *Service Requester*. Soll der Dienst ausgeführt werden, so wird die Kontrolle dem *Executer* übergeben. Im Falle einer gewünschten Verzögerung wird die Verarbeitung des CAPs an den *Router* zurückgegeben. Im Falle eines Abbruchs wird der *Matcher* Prozess abgebrochen und das CAP verworfen.

3.6 Bewertung

Ein zentraler Beweggrund für die Entwicklung von CAPEUS war, aufzuzeigen, dass sich typische kontextadaptive Applikationen implementieren lassen unter Berücksichtigung der Annahme, dass Kontextadaptivität als ein Aspekt der Kommunikation zwischen Komponenten aufgefasst werden kann. Offensichtlich lassen sich nicht alle Anwendungstypen auf diese Weise vollständig abbilden, doch auch in diesen Fällen kann das Konzept der Context-Aware Packets (CAPs) bzw. insbesondere das Konzept der Context Constraints unterstützend eingesetzt werden. Als Beispiel lässt sich die kontextadaptive Koordination von Systemkomponenten anführen.

Zur Demonstration des Konzeptes (*Proof of Concept)* wurden einige Anwendungen formuliert. Anschließend sollen zwei kontextadaptive Anwendungen, die mittels CAPEUS konzipiert wurden, repräsentativ vorgestellt werden. Weitere Anwendungsbeispiele für CAPEUS können in [Mi 01] gefunden werden, insbesondere auch die kontextadaptive Nutzung von lokalen Diensten in einer *Augmented Reality* [BMM+ 00a/b] Umgebung.

Context-Aware Notes

In Analogie zum *Stick-e Framework* [Pas 98b] realisiert die Anwendung *Context-Aware Notes* die Zuordnung von Notizen zu Kontexten. Das heißt, die vom Benutzer erstellten Notizen können an bestimmte kontextuelle Einschränkungen gebunden werden. So kann eine Notiz beispielsweise an einen bestimmten Ort gebunden sein; beim nächsten Besuch dieser Lokalität wird die Notiz dann wieder angezeigt. Derartige Notizen können auch als Erinnerungsstütze eingesetzt werden. Weiterhin soll es möglich sein, Notizen zu kreieren, die für jeden Nutzer sichtbar sind, welche einen entsprechenden Dienst benutzen, wie etwa zu Werbezwecken in einer Fußgängerpassage. Es ist zu beachten, dass eine Notiz nicht nur an Orte gebunden sein kann, sondern auch an beliebige Kontexte, so dass eine situationsbezogene Darstellung von Nachrichten möglich ist.

Diese Anwendung resultiert natürlich aus den von CAPEUS zur Verfügung gestellten Datenstrukturen und Systemprimitiven. Jede Notiz wird durch ein einzelnes CAP repräsentiert; siehe Abbildung 3-28. Je nach Anwendung kann die Notiz immer wieder angezeigt werden (Meta Constraints (MC) = *multi-shot*) oder bei Eintreten einer bestimmten Situation nur genau einmal (*one-shot*).

Der *Service Interaction Proxy* (SIP) überprüft kontinuierlich die in Frage kommenden Notizen. Bei Eintreten der durch die Context Constraints definierten Situation wird die jeweilige Notiz mittels eines Anzeigedienstes (*Note Display Service)* dargestellt.

Zur Ermöglichung persönlicher Notizen wird der SIP auf dem Endgerät des Benutzers ausgeführt. Ansonsten wird die Notiz zum zuständigen SIP in der Infrastruktur gesendet.

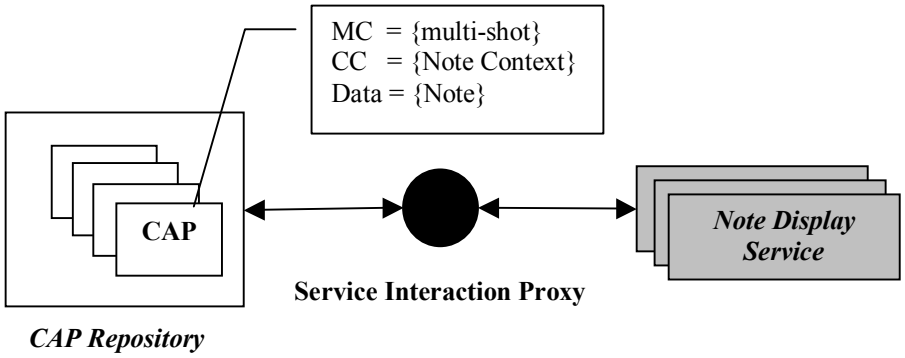

Abbildung 3-28: ***Context-Aware Notes***

Task-Aligned Service Use

Diese Anwendung realisiert die aufgabenspezifische Nutzung von lokalen Diensten. Insbesondere erlaubt die Anwendung die Manipulation und Anzeige von Dokumenten (Texte, Bilder, etc.). Zu jedem Dokument offeriert die Anwendung automatisch Dienstoptionen, die dem jeweiligen Typ des Dokuments entsprechen. So bietet das System bei Ansicht eines Textdokuments folgende Dienstoptionen an: *Print*, *Display* und *Note*. Jede Dienstoption wird durch ein entsprechendes CAP repräsentiert. *Print* veranlaßt den Ausdruck des Dokuments in Berücksichtigung der Benutzerpräferenzen. *Display* zeigt das Dokument auf einer digitalen Anzeigetafel (*Blackboard*) an. Die Interpretation des Dokuments als Notiz erfolgt durch *Note;* siehe voriger Abschnitt.

Wird vom Benutzer eine Dienstoption ausgewählt, so werden die Dokumentdaten in den Datenteil des CAPs kopiert. Die Context Constraints sind vordefiniert und reflektieren die Präferenzen des Benutzers. Zu jedem Zeitpunkt können dem System neue Dienstoptionen zugeordnet werden.

Angesichts der Tatsache, dass die Anwendung je nach Dokumenttyp angemessene Dienstoptionen vorselektiert (vgl. [DAPW 97]) und für den Benutzer alle Aspekte der verteilten Dienstnutzung transparent sind (kontextadaptive Selektion und Ausführung der Dienste), können die erforderlichen Benutzer-System Interaktionen signifikant reduziert werden. Zudem kann die oft knappe Anzeigefläche eines mobilen Endgerätes (z.B. Mobiltelefon) effizient genutzt werden.

3.6.1 Verwandte Arbeiten

Im Folgenden sollen Projekte und Ansätze umrissen werden, welche das Design von CAPEUS mitbeeinflusst haben. Sie vervollständigen die bereits erwähnten Ansätze, welche direkt in die Konzeption eingeflossen sind: so etwa Konzepte des Policy-basierten Managements (siehe 2.2.1) und Datenzentrischer Protokolle.

Stick-e Framework

Das *Stick-e Framework* [Pas 98b] präsentierte zuerst das Konzept virtueller Post-Its. In Korrespondenz zu ihren physischen Gegenstücken können auch die so genannten *Stick-e Notes* bestimmten Orten zugeordnet werden. Stick-es werden von PDAs verwaltet und bei Erreichen der zugeordneten Position eines Stick-es angezeigt. Zudem können Stick-e Notes generellen Kontexten zugeordnet werden. Stick-es sind in der Hinsicht persönlich, dass sie nur von den Personen gesehen werden können, auf deren Gerät sich das jeweilige Stick-e befindet.

Das *Stick-e Framework* besteht aus folgenden Komponenten, die auf einem mobilen Endgerät ausgeführt werden (z.B. PDA):

- *Triggering Component.* Diese Komponente vergleicht kontinuierlich den Kontext des Benutzers mit den in den Stick-e Notes vordefinierten Kontext-Bedingungen. Bei einer erfüllten Kontext-Bedingung wird das jeweilige Stick-e Note zur Ausführung gebracht.
- *Execution Component.* Diese Komponente implementiert die Ausführung eines Stick-e Notes, d.h. die Notiz wird auf dem PDA angezeigt.
- *Sensor Component* abstrahiert von der unterliegenden Sensortechnologie und liefert Kontextinformationen an die *Triggering Component*.

Insbesondere das kontextuelle Triggering-Konzept hatte Einfluss auf die hier vorgestellte Konzeption. Jedoch realisiert CAPEUS insgesamt einen allgemeineren Ansatz, der Anwendungsfall des Post-Its bildet einen Sonderfall (siehe voriger Abschnitt, *Context-Aware Notes*).

Cyberdesk

Cyberdesk [DAPW 97] implementiert ein System zur kontextadaptiven Integration von Anwendungskomponenten einer Desktop-Umgebung. Das System integriert Komponenten automatisch zur Laufzeit in Abhängigkeit von den Tätigkeiten des Benutzers. Die Funktionsweise von Cyberdesk lässt sich gut an einem einfachen Beispiel demonstrieren: Ein Benutzer empfängt eine E-Mail mit einer darin

enthaltenen Kontaktadresse. Diese möchte er nun seiner Adressdatenbank hinzufügen. Dazu markiert der Benutzer die Adresse innerhalb der empfangenen E-Mail. Das System analysiert automatisch die markierten Daten und stellt fest, dass es sich wahrscheinlich um Adressdaten handelt. Demzufolge bietet Cyberdesk dem Benutzer die Möglichkeit, diese Daten automatisch in die Adressdatenbank einzutragen. Da die markierte Kontaktinformation auch eine URL enthält, bietet Cyberdesk zudem die Option an, die URL mittels eines Web Browsers anzuzeigen. Nachdem der Benutzer die gewünschte Dienstoption gewählt hat, wird der entsprechende Dienst ausgeführt.

Selbst an diesem einfachen Beispiel ist zu erkennen, dass Cyberdesk die Anzahl der notwendigen Benutzer-System-Interaktionen wesentlich reduzieren kann, indem es die Funktionen von Systemkomponenten kontextadaptiv/datenzentrisch integriert.

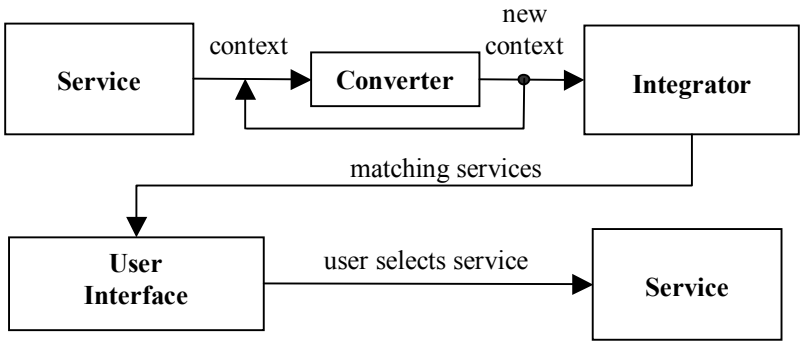

Abbildung 3-29: Die Laufzeitumgebung von Cyberdesk

Abbildung 3-29 zeigt die Laufzeitumgebung von Cyberdesk. Pfeile deuten den Datenfluss zwischen den Komponenten an. Daten werden innerhalb von Cyberdesk mittels eines einheitlichen Datenformats ausgetauscht. Die Architektur von Cyberdesk unterscheidet fünf Komponententypen. Die Komponente *Registry* verwaltet das Dienstangebot von Cyberdesk. Hier sind alle Komponenten samt ihrer Schnittstellenbeschreibungen gespeichert. Bereits existierende Anwendungen können durch *Wrapper* ins System integriert werden. *Converter* dienen der Transformation von Datentypen. So war bez. des oben angeführten Beispiels u.a. die Transformation von einem String in einen Datentyp Adresse erforderlich. Die Komponente *Integrator* bestimmt die Menge der matchenden Dienste (*Services)* bez. eines bestimmten Datentyps. Dazu greift sie auf den Datenbestand der *Registry* Komponente zurück.

Schließlich visualisiert *User Interface* die matchenden Dienstoptionen. Jetzt kann der Benutzer eine entsprechende Dienstoption selektieren.

Die Idee der kontextadaptiven Dienstselektion ist in das Design von CAPEUS mit eingeflossen. Der *Integrator* von *Cyberdesk* hat eine ähnliche Funktion wie der *Matcher* von CAPEUS. Es ist jedoch zu beachten das Cyberdesk für Desktop-Systeme entwickelt wurde, CAPEUS für verteilte Anwendungsumgebungen.

Portolano

Schließlich motivierte das Projekt *Portolano* [EHAB 99] die Konzeption Datenzentrischer Protokolle in Ubiquitous Computing Umgebungen. Portolano fokussiert insbesondere auf folgende Belange: benutzerorientierte Endgeräte (einfache Handhabung) und geringer Wartungsbedarf (*low maintenance*). Um diesen Anforderungen gerecht zu werden zu können, werden Datenfragmente inhaltsbezogen zu den vom Benutzer registrierten Diensten geleitet. Der Prozess der Datenzustellung soll für den Benutzer transparent erfolgen, andererseits sollen etwaige Fehler vom System autonom behoben werden. Allerdings beschreibt Portolano bisher nur eine Vision. Konzepte oder Systemarchitekturen, die ein solches System realisieren könnten, sind bisher ausgeblieben. Dennoch hatte die Vision der datenzentrischen Vermittlung einen zentralen Einfluss auf die Konzeption von CAPEUS.

3.7 Zusammenfassung

Dieses Kapitel hat ein neues Konzept zur kontextadaptiven Dienstnutzung vorgestellt. Ein kennzeichnendes Merkmal dieser Konzeption ist die Annahme, dass Kontextadaptivität als ein Aspekt der Kommunikation verstanden wird. Zur Realisation dieser Idee wurde zunächst ein Dienstmodell vorgestellt, in dessen Entwurf die Anforderungen des Lokalitätsprinzip (3.1.3) mit eingeflossen sind. Basierend auf diesem Modell wurde die Abstraktion der *Kontextadaptiven Interaktion* formuliert; sie beschreibt die kontextadaptive Nutzung einer Sequenz von verteilten Diensten.

In der vorgestellten Architektur *Context-Aware Packets Enabling Ubiquitous Services* (CAPEUS) wurde das Konzept der Kontextadaptiven Interaktion durch eine uniforme Datenstruktur repräsentiert, die als Context-Aware Packets bezeichnet wurden. CAPs beschreiben den Vorgang der kontextadaptiven Dienstnutzug anhand von kontextuellen Einschränkungen, so genannten *Context Constraints*. Context Constraints werden in Bezug auf die zu ermittelnden Kontextinformationen evaluiert und steuern die Selektion und Ausführung von Diensten in einer Ubiquitous Computing Umgebung.

Im Anschluss daran erfolgte eine Beschreibung der prototypischen Implementierung basierend auf Java. Das Kapitel endete mit einer Bewertung des vorgestellten Konzeptes.

Ein Ausblick auf weiterführende Forschungsansätze erfolgt in 6.1.1.

Kapitel 4

Kontextadaptive Integration von Diensten

Dieses Kapitel stellt ein neues Konzept vor, welches die Rollenverteilung von diversen Dienstendpunkten dynamisch verwalten kann. Dies ermöglicht die spontane und kontextadaptive Integration von Diensten in einer Ubiquitous Computing Umgebung. Zudem wird eine Tool-Architektur vorgestellt, die dem Benutzer erlaubt, so genannte *Interaktionsmuster* (*Interaction Templates*) zu erzeugen, welche das Rollenverhalten eines Endgerätes bez. kontextueller Einschränkungen definiert. Das spezifizierte Rollenverhalten bezieht sich insbesondere auf Dienstinteraktionen, die das vom Benutzer gewünschte Verhalten bez. einer bestimmten Situation realisieren.

Das durch Interaktionsmuster definierte Rollenverhalten wird durch Kontextadaptive Interaktionen bzw. Context-Aware Packets (CAPs) (vgl. Kapitel 3) beschrieben. Diesbezüglich bilden Interaktionsmuster Vorlagen zur systemgestützten Generierung von CAPs. Das heißt, das hier vorgestellte System erweitert die Konzeption von CAPEUS [SMLP 01b] insbesondere um einen Ansatz zur Konstruktion von CAPs. Dieser Aspekt wurde bisher nicht berücksichtigt.

4.1 Interaktionsmusterbasierte Dienstintegration

Der Prozess der *kontextadaptiven Dienstintegration* kann als eine Verallgemeinerung des Begriffs der *Kontextadaptiven Interaktion* (siehe 3.2) verstanden werden. Innerhalb von CAPEUS wurde vor allem die Evaluation von CAPs untersucht. Im Rahmen der kontextadaptiven Dienstintegration, wie sie hier erläutert wird, soll CAPEUS um folgende Aspekte ergänzt werden:

- Kontextadaptive Generierung von CAPs
- Unterstützung von Sitzungen *(Sessions*)
- Lokales Routing

Kontextadaptive Generierung von CAPs

Die *kontextadaptive Generierung von CAPs* erlaubt in Anhängigkeit von kontextuellen Bedingungen die Initiierung von kontextadaptiven Interaktionen. Während also CAPEUS die Evaluation von CAPs bewerkstelligt, können mit dem

in diesem Kapitel vorgestellten System CAPs erzeugt werden. Zu diesem Zweck wurde die bereits in Kapitel 3 angedeutete Komponente *Interaction Organizer* konzipiert. Diese Komponente erzeugt CAPs in Abhängigkeit vom Status einer Anwendung und einem so genannten *Interaktionsmuster* (siehe folgender Abschnitt). Interaktionsmuster definieren das Prozessverhalten eines Endgerätes in Bezug auf Kontextänderungen.

Lokales Routing

Innerhalb von CAPEUS erfolgt das Routing von CAPs immer über eine Indirektion - den *Service Interaction Proxy* (SIP) -, welche die Ressourcen einer Domäne verwaltet. Innerhalb der hier vorgestellten Konzeption ist auch eine direkte Zustellung von CAPs möglich. Dieser Kommunikationsmodus kann insbesondere innerhalb *abgeschlossener Dienstdomänen* (siehe 3.2.1) gewinnbringend eingesetzt werden.

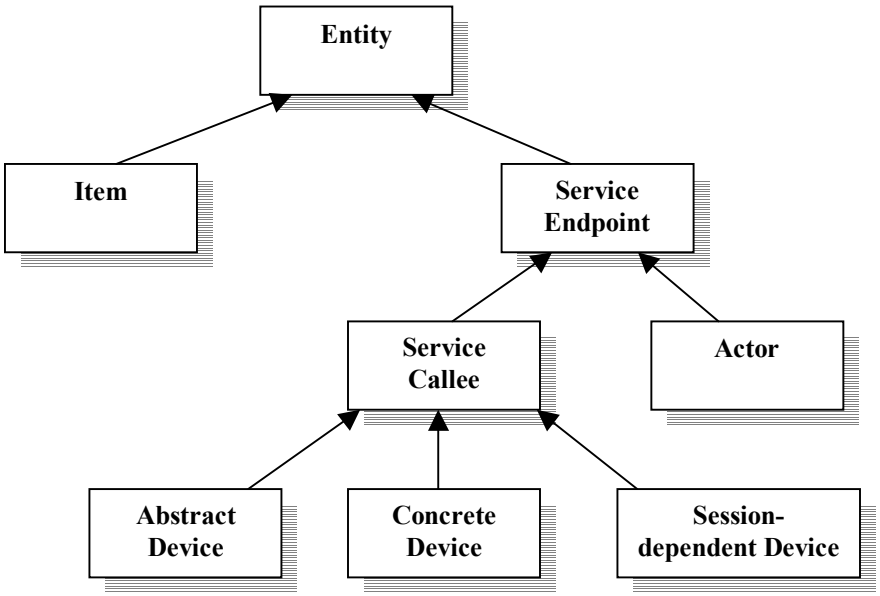

Abbildung 4-1: Erweiterte Klassifikation von Entitäten

Unterstützung von Sessions

Mittels CAPEUS erfolgt die Zustellung von CAPs entweder bez. konkreter Adressinformationen (z.B. IP-Adresse) oder durch ein assoziatives Zustellungsverfahren basierend auf kennzeichnenden Attributen einer Entität. Die assoziative Zustellung von CAPs kann in manchen Anwendungsszenarien zu Zuordnungsproblemen führen. Zur Verdeutlichung sei folgendes Szenario angenommen: Ein

CAP bezieht sich auf eine Entität des Typs A. In der betrachteten Dienstdomäne befinden sich jedoch zwei Entitäten dieses Typs A. Da die Entitäten aufgrund ihres identischen Typs innerhalb einer assoziativen Zustellung nicht unterschieden werden können, erfolgt die Zustellung nicht-deterministisch zu einer der beiden Entitäten. Das heißt, es kann nicht garantiert werden, dass eine Folge von individuellen CAPs mit Bezug auf eine Entität des Typ A immer derselben Entität zugestellt werden. Dieses Zuordnungsproblem kann offensichtlich zu Inkonsistenzen führen.

Motiviert durch dieses Zuordnungsproblem wurde ein Sitzungskonzept entwickelt. Eine Sitzung (*Session*) wird initiiert durch ein CAP mit einer abstrakten Entitätsbeschreibung. Sitzungen beziehen sich nun auf genau die Entität, die beim Routing-Prozess des ersten CAPs ermittelt wurde, d.h. alle nachfolgenden CAPs werden dieser Entität zugestellt. Dementsprechend muss die Klassifikation von Entitäten (siehe Abbildung 3-8, Seite 76) erweitert werden. Neben den bereits bestehenden Entitätstypen *Abstract Device* und *Concrete Device* wird der Entitätstyp *Session-dependent Device* hinzugefügt.

4.2 Das Konzept des Interaktionsmusters

Die kontextadaptive Dienstintegration basiert auf dem Konzept des *Interaktionsmusters* (*Interaction Template*) [SL 01], welches im Folgenden erläutert werden soll. Interaktionsmuster ordnen Endgeräten bzw. Dienstendpunkten ein bestimmtes Prozessverhalten zu. Das gewünschte Verhalten wird – analog zu *Context Constraints* – kontextadaptiv determiniert. Die erfolgreiche Evaluierung eines Interaktionsmusters mündet (1) in der kontextadaptiven Generierung eines oder mehrerer CAPs, welches die geforderten Aktionen mittels kontextadaptiver Interaktionen abbildet, (2) einem lokalen Methodenaufruf oder (3) der rekursiven Erzeugung eines weiteren Interaktionsmusters (vgl. 4.2.2).

Grundlegend für das Konzept des Interaktionsmusters ist der Begriff der *Rolle*. Der Begriff Rolle stammt ursprünglich aus der Soziologie. In diesem Zusammenhang versteht man unter einer Rolle ein erwartetes Verhaltensmuster von Personen, die bestimmte Funktionen innerhalb einer Gruppe zu erfüllen haben [Sch 86]. Der gleiche Begriff wird gleichfalls in der Informatik eingesetzt, so z.B. bei der Modellierung von Software-Engineering-Prozessen. Projiziert man den Begriff der Rolle auf Entitäten einer Ubiquitous Computing Umgebung, so lassen sich neue Konzepte entwickeln, die der menschlichen Denkweise nahe kommen. Lupu und Sloman beschrieben bereits in [LS 97] die auf Objektmodellierung basierende *Role Policy* für verteilte Anwendungen.

Wie bereits im einführenden Szenario von Kapitel 2 beschrieben, können mobile Dienstendpunkte in einer Ubiquitous Computing Umgebung spontan vernetzt werden. Das dadurch entstehende drahtlose Ad-hoc-Netzwerk oder auch *Personal*

Area Network (PAN) bildet eine Gruppe von interagierenden Anwendungsgeräten. Gruppenmitglieder können sowohl Informationen austauschen als auch angebotene Dienste gegenseitig aufrufen. Um die Interaktionen zwischen den Gruppenmitgliedern zu beschreiben, wird nun der Begriff der Rolle eingeführt. Die informale Begriffsdefinition der Rolle ist wie folgt gegeben:

> *Rollen werden im Kontext von Interaktionsmustern als ein erwartetes Prozessverhalten von Entitäten verstanden, die eine bestimmte Funktion innerhalb einer Gruppe erfüllen sollen. Als Ausprägung einer Rolle versteht man die Regeln oder Policies, die eine Rolle determinieren.*

Die Zuteilung von Rollen erfolgt dynamisch. Zur Laufzeit können Rollen modifiziert, gelöscht und hinzugefügt werden. Dabei ist eine Anpassung der unterliegenden Softwarekomponenten und Dienste nicht erforderlich. Als Konsequenz kann man die rollenbasierte Dienstintegration mit dem Ansatz des Scriptings [Ous 98, Rom 98] vergleichen. Bei dieser Vorgehensweise können Softwarekomponenten über einen so genannten *Scipting Layer* mittels einer Skriptsprache (z.B. Python [Py]) integriert werden; diesen Prozess bezeichnet man auch als *gluing*. Allerdings implementieren Scripting-Konzepte nativ kein Rollenkonzept, können jedoch herangezogen werden, um dieses zu implementieren.

Die Zuordnung von Rollen zu Entitäten kann in Abhängigkeit vom Kontext variieren. Die Ausprägung einer Rolle ist jedoch konstant und ändert sich nicht kontextadaptiv. Zur Rollenverwaltung benötigt man ein Mittel, das die erwarteten Interaktionen einer Rolle zusammenfasst, und bei einem korrespondierenden Kontext aktiviert. Diese Aufgabe kann der in dieser Arbeit beschriebene *Interaction Organizer* (siehe 4.4.1) mit Hilfe des Interaktionsmusters erfüllen. Die Grundidee vom Interaktionsmuster ist, das erwartete Verhalten einer Entität in einem uniformen Format zusammenzufassen und kontextadaptiv auszuwerten. Deswegen ermöglichen Interaktionsmuster eine kontextadaptive Anpassung des Verhaltens einer Entität.

> *Ein Interaktionsmuster (Interaction Template) ist eine Dateneinheit, welche die zu einer bestimmten Aufgabe gehörenden Interaktionen und ihre Ausführungsbedingungen abstrakt beschreibt, um den Entitäten ihre Rollen zuzuordnen.*

Interaktionsmuster werden lokal auf der entsprechenden Entität bzw. dem Dienstendpunkt ausgeführt. Wenn die Bedingungen für die Aktivierung eines Interaktionsmusters erfolgreich validiert wurden, so werden die assoziierten Interaktionen der Reihenfolge nach ausgeführt.

Ein Interaktionsmuster einer Entität fasst also die Interaktionen zusammen, die der Rolle zugeordnet sind. Interaktionsmuster können Entitäten zur Laufzeit dynamisch zugewiesen werden. Die von einem Interaktionsmuster resultierenden

Kontextadaptiven Interaktionen werden innerhalb einer individuellen Session ausgeführt. Eine detaillierte Darstellung der Struktur eines Interaktionsmusters erfolgt im anschließenden Abschnitt 4.2.1. Die durch ein Interaktionsmuster induzierten Kommunikationsmuster (*Communication Pattern*) [SL 01] werden in Abschnitt 4.3 zusammengefasst.

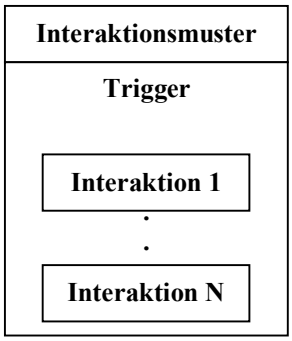

Abbildung 4-2: Abstrakte Struktur eines Interaktionsmusters

4.2.1 Struktur eines Interaktionsmusters

Zur Spezifikation eines Interaktionsmusters benötigt man insbesondere folgende Attribute: *Name, Location, Iteration, SessionID* und *Trigger*. Analog zur Spezifikation von CAPs (vgl. 3.3.2) können für *Iteration* folgende Optionen gewählt werden: *one-shot* oder *multi-shot*. Im Falle von *one-shot* wird das Interaktionsmuster nach einmaliger Aktivierung gelöscht; *multi-shot* indiziert die wiederholte Ausführung eines Interaktionsmusters. Die *SessionID* identifiziert ein Interaktionsmuster eindeutig. *Trigger* definiert, unter welchen Bedingungen das System das Interaktionsmuster aktiviert, er bezieht sich auf den internen Status einer Entität, zu dem etwa die Eingabe eines Benutzers oder der Aufruf eines Programms führt. Einem *Trigger* liegt ein logischer Ausdruck zugrunde, den man mit Variablen und logischen Operatoren (`AND`,`OR`,`NOT`) und Vergleichsoperatoren (`<`, `>`, `<=`, `>=`, `==`, `!=`) konstruieren kann. Das heißt, falls der logische Ausdruck erfüllt ist, so werden die korrespondierenden Interaktionen ausgeführt.

Ein Interaktionsmuster besteht explizit aus der abstrakten Beschreibung der Interaktionen, die sowohl durch Parameter und die zu bearbeitenden Daten definiert werden.

4.2.2 Spezifikation einer Interaktion

Wie bereits erwähnt, kann es sich bei den in einem Interaktionsmuster inkludierten Interaktionen um lokale Methodenaufrufe, Kontextadaptive Interaktionen oder auch um ein weiteres Interaktionsmuster handeln. Die lokale Methode wird lokal auf dem Dienstendpunkt ausgeführt, während eine Kontextadaptive Interaktion durch ein CAP realisiert wird. Außerdem kann ein Interaktionsmuster ein Kind-Interaktionsmuster für eine Teilaufgabe erzeugen, dass wiederum Interaktionen enthält und zur gleichen Sitzung des Mutter-Interaktionsmusters gehört. Zur Darstellung der Kompositionshierarchie von Interaktionen wurde hier das *Composite* Designmuster angewandt; siehe [GHJV 95].

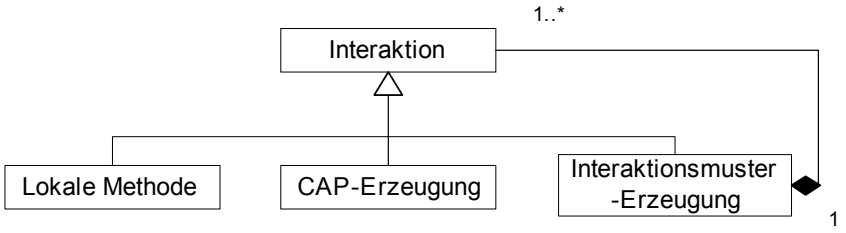

Abbildung 4-3: Rekursive Komposition von Interaktionen

Abbildung 4-3 zeigt die Kompositionshierarchie im Überblick. Lokale Methoden und die CAP-Erzeugung (*Create-CAP)* werden als primitive Interaktionen angesehen, während die Interaktionsmuster-Erzeugung (*Create-Template*) als Rahmen der Primitive gelten kann.

Zur Ausführung einer Interaktion können Parameter benötigt werden, die von vielfältiger Natur sein können, so z.B. vom Typ String, Integer oder einer Objektreferenz. Jene Parameter lassen sich in Aus- und Eingabeparameter klassifizieren. Manche Interaktionen benötigen überhaupt keine Eingabe-parameter, wie z.B. das Ausschalten eines Lichtes, während andere Interaktionen mehrere Parameter erfordern. Eine Interaktion mit Ausgabe kann beispielsweise den Download von Daten eines entfernten Rechners bezeichnen. Alle diese Ausprägungen müssen bei der Modellierung der Interaktion berücksichtigt werden.

Im folgenden werden die Anforderungen bez. der Parametrisierung einzelner Interaktionstypen untersucht.

Lokale Methode

Lokale Methoden sind ein Oberbegriff für gerätespezifische Dienste eines Dienstendpunktes. Ein Beispiel für eine lokale Methode einer digitalen Kamera ist der Bildauslösemechanismus. Die Parameter einer lokalen Methode sind in der Regel anwendungsspezifisch und können nicht weitergehend klassifiziert werden.

CAP-Erzeugung

Die CAP-Erzeugung wird mittels lokaler Methodenaufrufe auf der Entität realisiert. Eine Entität, welche die lokale Erzeugung von CAPs unterstützt, muss eine entsprechende API zu Verfügung stellen; siehe 4.4.2. Das entstandene CAP kann dann entweder lokal an den entsprechenden Adressaten gesendet werden oder zu einem Service Interaction Proxy (SIP) in der Infrastruktur, der das CAP an den Empfänger weiterleitet.

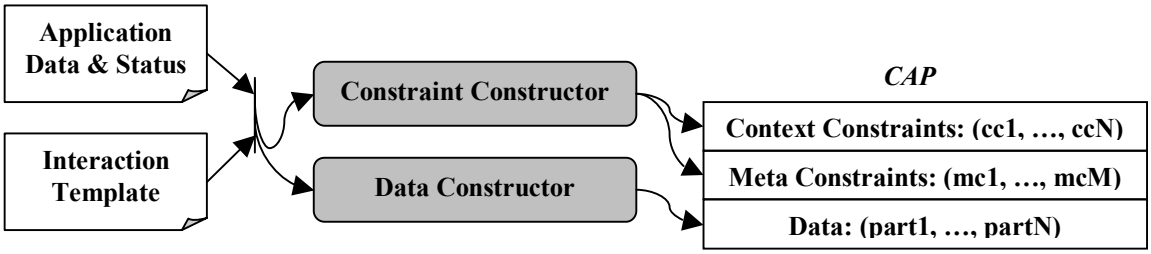

Abbildung 4-4: Konstruktion eines CAPs

Wie bereits in 3.3 beschrieben, besteht ein CAP aus drei Einheiten: *Meta Constraints*, *Context Constraints* und *Data*. Zur Generierung der *Meta* und *Context Constraints* wurde ein *Constraint Constructor* definiert. In Übereinstimmung mit dem Interaktionsmuster, den Applikationsdaten und Statusinformationen konstruiert der *Constraint Constructor* z.B. Entitätsbeschreibungen und Relationen, die im resultierenden CAP angelegt werden. Zur Konstruktion der *Data* Einheit existiert ein entsprechender *Data Constructor*. Dieser verweist zunächst auf die relevanten Daten einer Applikation. Während der Erzeugungsphase werden diese Daten im neu angelegten CAP repliziert.

Interaktionsmuster-Erzeugung

Schließlich kann die Auswertung eines Interaktionsmusters auch in der Erzeugung eines neuen Interaktionsmusters resultieren. Zur Konstruktion eines Interaktionsmusters werden die Angaben der am Anfang dieses Abschnitts beschriebenen Attribute und die Informationen der enthaltenen Interaktionen benötigt, die lokale Methoden aufrufen, eine Kontextadaptive Interaktion/CAP erzeugen oder rekursiv ein Interaktionsmuster erzeugen können. Die Angaben der Parameter für diese Interaktionen wurden bereits erläutert.

Trigger und Events

Eine Interaktion enthält neben den zur Ausführung notwendigen Parametern u.U. auch eine Event-Bedingung, womit eine Interaktion bez. Ereignissen der Umgebung synchronisiert wird oder auch abgebrochen werden kann. Events können sich auf externe Ereignisse beziehen, etwa das Annähern einer Person. Im Kontrast dazu können sich die in Interaktionsmustern enthaltenen Trigger nur auf interne Statusänderungen eines Dienstendpunktes beziehen.

Events beziehen sich in der Regel meist auf gemessene Sensorsignale der Umgebung.

4.2.3 Einschränkungen der Interaktionskomposition

Der vorherige Abschnitt hat die gültigen Interaktionstypen eines Interaktionsmusters klassifiziert. Um die Komplexität eines Interaktionsmusters zu begrenzen, werden die verknüpften Interaktionen eingeschränkt.

- **Lokale Methoden:** Ein Interaktionsmuster kann eine beliebige Anzahl von lokalen Methodenaufrufen definieren.
- **CAP-Erzeugung:** Ein Interaktionsmuster kann eine Sequenz von CAPs in beliebiger Anzahl erzeugen, unter der Bedingung, dass sie keine sitzungsabhängige (*session dependent)* Entität im ersten CAP und nur sitzungsabhängige Entitäten in den nachfolgenden CAPs enthalten dürfen.
- **Interaktionsmuster-Erzeugung**: Ein Interaktionsmuster kann beliebige Kind-Interaktionsmuster erzeugen, die die oben genannte Einschränkung erfüllen. Alle Kind-Interaktionen haben die gleiche *SessionID* wie das Mutter-Interaktionsmuster.

Die hier formulierten Einschränkungen definieren die Syntax für Interaktionen eines Interaktionsmusters. Weil ein Interaktionsmuster nur genau einer Sitzung zugeordnet werden darf, kann man aus den oben aufgeführten Einschränkungen folgende Aussage extrapolieren:

> *Jedes Interaktionsmuster kann nur eine einzige Ziel-Entität per CAP direkt erreichen.*

Alle Interaktionen, die den oben genannten Einschränkungen genügen, können innerhalb eines Interaktionsmusters definiert werden. Es müssen jedoch auch semantische Bedingungen berücksichtigt werden: Interaktionen eines Interaktionsmusters müssen einen logischen Zusammenhang bilden, um eine bestimmte Aufgabe abzubilden.

4.2.4 Dekomposition eines Interaktionsmusters

Falls in einem Interaktionsmuster Interaktionen existieren, die zwar semantisch korrekt sind, aber die oben erläuterten Interaktionseinschränkungen (siehe 4.2.3) verletzten, müssen diese soweit zerlegt werden, bis sie den Bedingungen genügen.

Die zerlegten Interaktionsmuster können trotzdem durch Trigger miteinander verbunden werden (siehe Abbildung 4-5), damit die enthaltenen Interaktionen hintereinander ausgeführt werden können. Die zerlegten Interaktionsmuster liefern dann dieselben Effekte wie zuvor.

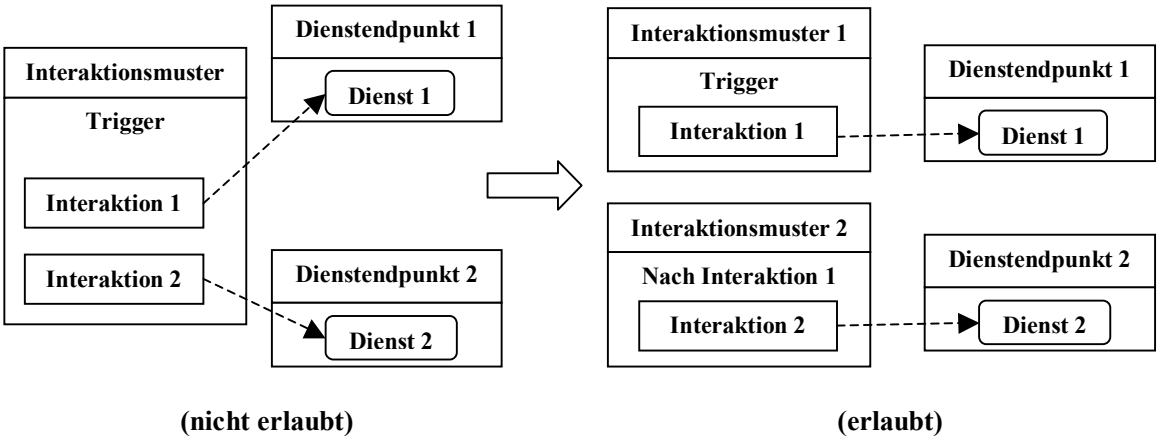

Abbildung 4-5: Dekomposition eines Interaktionsmusters

Die linke Hälfte der Abbildung zeigt ein nicht erlaubtes Interaktionsmuster, welches sich auf zwei individuelle Dienstendpunkte bezieht. Dieser Konflikt kann aufgelöst werden (rechte Bildhälfte), indem das ursprüngliche Interaktionsmuster in zwei eigenständige unterteilt wird. Die sequentielle Verkettung dieser Interaktionsmuster erfolgt über Trigger.

4.3 Kommunikationsmuster

Interaktionen können innerhalb eines Interaktionsmusters zusammengefasst werden, wenn die in 4.2.3 beschriebenen Einschränkungen erfüllt werden. Ein Interaktionsmuster kann rekursiv weitere Interaktionsmuster erzeugen.

Es besteht zudem die Möglichkeit, Interaktionsmuster zu kombinieren oder zusammenzufassen, um eine komplexe Aufgabe abzubilden. Im Folgenden werden die Kombinationsmöglichkeiten und die dadurch induzierten Kommunikationsmuster (*Communication Patterns)* erläutert. Es werden folgende Kommunikationsmuster unterschieden: *Sequenz, Verzweigung,* und *Zyklus.*

4.3.1 Sequenz

Interaktionsmuster können so verknüpft werden, dass eine sequenzielle Dienstverkettung gebildet wird. Die Kombination basiert entweder auf Daten oder auf Dienstendpunkten. Das daraus entstehende Interaktionsmuster besteht aus Interaktionen, die entweder lokale Methoden sind oder CAP-Erzeugungen verkörpern. Eine Sequenz enthält also keine Interaktionen, die in der Erzeugung eines neuen Interaktionsmusters resultieren. Interaktionsmuster implementieren eine „if-then" Semantik (Trigger), welche die Abbildung von Verzweigungen zulässt (s.u.).

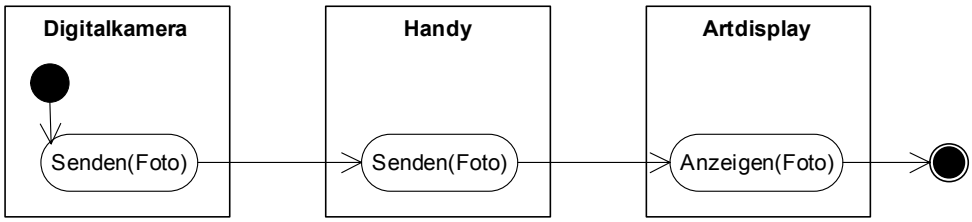

Abbildung 4-6: Datenbasierte Sequenz

Datenbasierte Sequenz

Dies ist die intuitivste Möglichkeit, Interaktionsmuster zu kombinieren. Die datenbasierte Sequenz lässt sich immer dann anwenden, wenn in einer Folge von Interaktionen die Ausgabe einer vorangehenden Interaktion zugleich Eingabe einer nachfolgenden Interaktion ist. In diesem Fall lassen sich diese Interaktionen in einem Interaktionsmuster zusammenfassen.

Anhand des einführenden Szenarios aus Kapitel 3 (siehe Abbildung 3-1) kann man das Prinzip der Datenbasierten Sequenz gut demonstrieren; siehe Abbildung 4-6.

Das Übertragen des Fotos zum Handy und das Speichern im Albumdienst im Internet bezieht sich auf dieselben Daten – in diesem Falle das erstellte Foto. Man definiert ein Interaktionsmuster, das die Bedingung „Foto erstellt" als den Trigger spezifiziert und eine CAP-Erzeugung als Interaktion enthält. Ist die Trigger-Bedingung erfüllt, so konstruiert die Digitalkamera das CAP, welches zunächst das Mobilfunktelefon beauftragt, das entstandene Foto zum Anzeigedienst zu senden, und schließlich der Ziel-Entität den Befehl „Anzeige des Fotos" erteilt. Die Interaktionssequenz wird automatisch ausgeführt.

Komplementär zum Konzept der datenbasierten Sequenz können sich Sequenzen auch auf einen einzigen Dienstendpunkt beziehen, z.B. wenn in Reaktion auf ein Ereignis mehrere Dienste eines individuellen Endpunkts beansprucht werden sollen. Dieser Fall wird dann als *dienstendpunkt-basierte Sequenz* bezeichnet.

4.3.2 Verzweigung und Zyklus

Ein Interaktionsmuster besteht im Wesentlichen aus den Interaktionen und einem Trigger. Die Interaktionen werden nur dann ausgeführt, wenn die formulierte Bedingung erfüllt ist. Deshalb haben Interaktionsmuster die Semantik von Verzweigungen. Ein Interaktionsmuster, welches das Kommunikationsmuster der Verzweigung realisiert, muss mindestens ein Kind-Interaktionsmuster konstruieren. Mit anderen Worten: Wenn man bei der Repräsentation eines Prozesses eine Verzweigung benötigt, so muss an dieser Stelle ein Interaktionsmuster generiert werden.

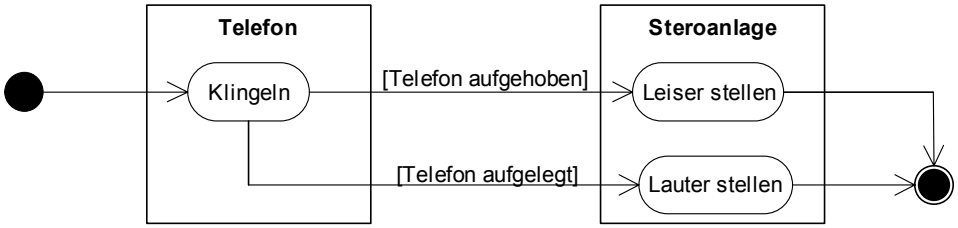

Abbildung 4-7: Verzweigung

Wie bereits erwähnt, kann man in einem Interaktionsmuster einen Auslöser definieren, der sich auf die Bedingung der Verzweigung bezieht. Die Verzweigung wird hier mittels des folgenden Szenarios erläutert: Beim Telefonieren soll die Lautstärke einer Stereoanlage reduziert werden; nach dem Beenden des Telefongespräches soll die Lautstärke der Stereoanlage wieder entsprechend erhöht werden.

Man definiert am Telefon ein mehrmaliges (*multi-shot*) Interaktionsmuster, das das Abheben des Telefonhörers als Auslöser hat und aus zwei Interaktionen besteht. Die erste Interaktion konstruiert ein CAP, um die Lautstärke der Musik zu verringern. Im Anschluss generiert das Interaktionsmuster ein einmaliges (*one-shot*) Interaktionsmuster, das beim Auflegen des Telefons ein weiteres CAP erzeugt, um die Musik an derselben Stereoanlage lauter zu machen. Das einmalige Interaktionsmuster wird nach der Ausführung der Interaktion gelöscht. Es wird deutlich, dass durch die Konstruktion des zweiten Interaktionsmusters ein Prozess mit einer Verzweigung entsteht.

Es ist offensichtlich, dass sich mit dem Konstrukt der Verzweigung auch ein *Zyklus* darstellen lässt, also ein Prozess der solange wiederholt wird, bis eine bestimmte Bedingung erfüllt ist.

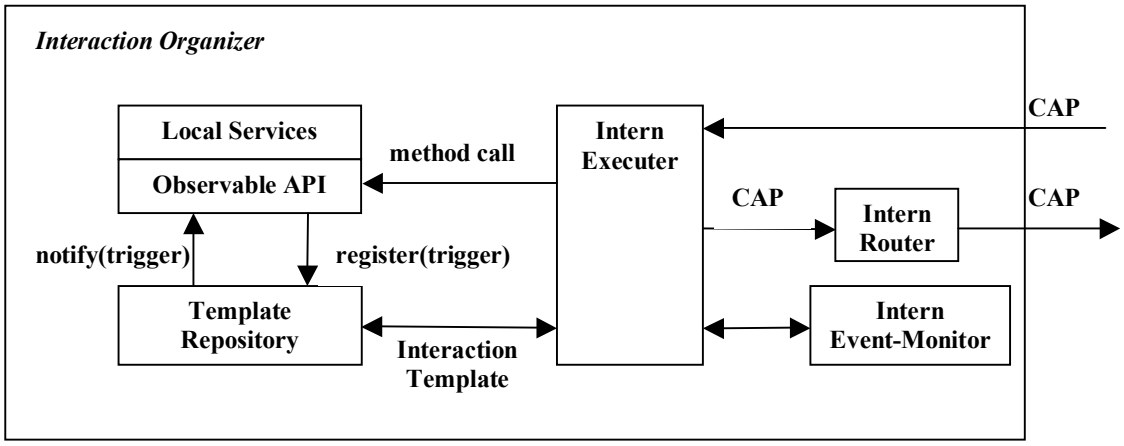

Abbildung 4-8: Architektur des Interaction Organizers

4.4 Eine Architektur zur Evaluation von Interaktionsmustern

Die Architektur von CAPEUS [SMLP 01b] wurde in Kapitel 3 beschrieben. Diese behandelt insbesondere die Dienstauswahl und Ausführung in einem Ubiquitous Computing System mittels CAPs, die durch spezielle Routerelemente - Service Interaction Proxies (SIPs) -, an die entsprechenden Dienstendpunkte geleitet werden. Allerdings wurde die Erzeugung von CAPs im Rahmen von CAPEUS nicht behandelt. In diesem Abschnitt wird die Komponente *Interaction Organizer* vorgestellt, die basierend auf der Evaluation von Interaktionsmustern (*Interaction Templates*) CAPs konstruiert. Zudem wird die CAPEUS-Architektur um einen wichtigen Aspekt ergänzt: Neben der bereits unterstützten konkreten und abstrakten Auswahl von Entitäten wird nun auch eine sitzungsabhängige Adressierung für CAPs unterstützt (vgl. 4.1). Im Anschluss an diesen Abschnitt soll die Architektur des Interaction Organizers erläutert werden.

4.4.1 Interaction Organizer

Der Interaction Organizer besteht aus den nachfolgenden Basiskomponenten: *Observable API, Templates Repository, Intern Executer, Intern Event Monitor* und *Intern Router* (siehe Abbildung 4-8). Die Bezeichnung der Komponenten orientiert sich an Komponenten der CAPEUS-Architektur mit vergleichbarer Funktion.

Observable API (vgl. [Woo 96]) implementiert eine Dienstschnittstelle mit Reflektionsmechanismen. Das *Template Repository* ist eine Komponente zur Verwaltung von Interaktionsmustern. Wenn die Entität mit Sensoren ausgestattet ist – z.B. einem GPS-Sensor –, so kann der *Intern Event Monitor* das vordefinierte Interaktionevent überwachen und gegebenenfalls die Ausführung starten oder abbrechen. Der *Intern Executer* dient der Ausführung von Interaktionen und der

Koordination der anderen Komponenten. Im Falle des lokalen Routings ermittelt der *Intern Router* die Zieladresse eines CAPs und leitet es an die Ziel-Entität weiter. Im Folgenden werden die einzelnen Komponenten und ihre Funktion bei der Evaluation von Interaktionsmustern erläutert.

4.4.2 Observable API

In den meisten Systemen sind die Operationen auf Schnittstellen von Komponenten [Krieg 98] nicht sichtbar für eine dritte Komponente. Erfolgt beispielsweise ein Methodenaufruf der Komponente A auf einer Komponente B, so existiert keine implizite Unterstützung, um eine dritte Komponente davon zu informieren. Analog ist es oft von Nutzen, interessierte Komponenten über einen etwaigen Statuswechsel einer anderen Komponente zu benachrichtigen. Diese Funktionalitäten sind in einem herkömmlichen Komponentenmodell nur dann möglich, wenn die entsprechenden Komponenten diese Art von Interaktionen über ihre Schnittstelle anbieten. Jedoch müssen bei dieser Vorgehensweise die gewünschten Interaktionstypen (z.B. Statuswechsel oder Methodenaufruf) a priori festgelegt werden, also während des Designs einer Komponente. Da sich aber die gewünschten Interaktionstypen nicht immer bereits zur Designphase bestimmen lassen, wird hier ein spezieller Schnittstellentyp verwendet: *Observable API* (analog zu [Woo 96]). Dieses Konzept erlaubt es, u.a. Interaktionstypen zur Laufzeit eines Systems zu parametrisieren. Insbesondere bei der spontanen Vernetzung von Komponenten, wie sie für Ubiquitous Computing Systeme charakteristisch ist, kann das Konzept der Observable APIs gewinnbringend eingesetzt werden.

Observable APIs realisieren folgende Basisoperationen:

- *Examination*
- *Manipulation*
- *Observation*

Examination erlaubt die Inspektion einer Schnittstelle (vgl. *Java Reflection* [Ref]) und das Abfragen von Statusinformationen einer Komponente. Komplementär erlaubt die Operation *Manipulation* die Beeinflussung des Status einer Komponente. Schließlich erlaubt die Operation *Observation* die Überwachung von Statusinformationen einer Komponente. Die Registrierung von interessierten Komponenten erfolgt nicht bez. eines a priori definierten Eventtyps, sondern über einen logischen Ausdruck, der sich auf Zustandsvariablen der Komponente bezieht und festlegt, bei welcher Zustandskonfiguration oder -änderung eine interessierte Komponente zu benachrichtigen ist.

Innerhalb des *Interaction Organizers* registriert das *Template Repository* Trigger des Interaktionsmusters bei der korrespondierenden *Observable API* (siehe

Abbildung 4-8). Wird eine Trigger-Bedingung erfüllt, so wird das *Template Repository* benachrichtigt. Zur Realisierung der *Observable API* wurde das *Observer* Designmuster angewendet; siehe [GHJV 95]. Jedoch wurde es um den Aspekt erweitert, dass der Zustand eines Objektes bez. vordefinierter Kriterien überwacht werden kann.

Mit dem Observer Mechanismus, wie er hier vorgestellt wurde, können Objekte innerhalb eines Prozesses miteinander kommunizieren, während es der *Conference Bus* [MJ 95] ermöglicht die Kommunikation zwischen Objekten, die innerhalb separater Prozesse ausgeführt werden, zu realisieren [Rom 98]. Der Einsatz eines Conference Bus ermöglicht zudem die Korrelation von Ereignissen multipler Objekte. Dieser Ansatz wird im Rahmen dieser Arbeit nicht untersucht, könnte aber Bestand von Folgearbeiten sein.

4.4.3 Template Repository

Das *Template Repository* verwaltet die angelegten Interaktionsmuster einer Entität. Im Prinzip handelt es sich dabei um ein einfaches Datenbank-managementsystem. Das Attribut *Name* wird als Primärschlüssel verwendet. Als Sekundärschlüssel dient das Attribut *Trigger*.

1. **Das neue Interaktionsmuster in das *Template Repository* eintragen.**
2. **Existiert der Trigger des Interaktionsmusters noch nicht, so wird er an der *Observable API* registriert.**

Abbildung 4-9: Algorithmus zur Bearbeitung eines neuen Interaktionsmusters

Das Template Repository implementiert folgende Funktionen: Interaktionsmuster hinzufügen, löschen, auflisten, auswählen und das auszuführende Interaktions-muster an den *Intern Executer* weitergeben. Es besteht auch die Möglichkeit, existierende Interaktionsmuster zu deaktivieren, d.h. die assoziierten Interaktionen werden auch dann nicht ausgeführt, wenn der Trigger erfüllt wurde. Natürlich können deaktivierte Interaktionsmuster auch wieder aktiviert werden.

Soll ein neues Interaktionsmuster an einem Dienstendpunkt angelegt werden, so werden die Schritte aus Abbildung 4-9 ausgeführt. Falls sich die *Observable API* mit einem erfüllten Trigger zurückmeldet, sucht das Template Repository alle Interaktionsmuster mit identischen Trigger-Bedingungen. Diese werden dann zur weiteren Evaluation an den *Intern Executer* weitergegeben.

4.4.4 Intern Executer

Die zentrale Komponente des *Interaction Organizers* ist der *Intern Executer*, da dieser sowohl für die Ausführung der Interaktionen als auch für die Evaluierung

von Interaktionsmustern und CAPs zuständig ist. In diesem Abschnitt werden die jeweiligen Algorithmen beschrieben.

1. **Falls die Interaktion kein Event enthält, gehe zu Schritt 2; anderenfalls liefert der Intern Executer die Event Bedingung an den Intern Event Monitor. Wenn die Auswertung des Events positiv ist, gehe zu Schritt 2, anderenfalls wird die Auswertung der Interaktion abgebrochen.**
2. **Wenn die Interaktion eine lokale Methode bezeichnet, wird sie bei der *Observable API* aufgerufen.**
3. **Falls es sich bei der Interaktion um eine CAP-Erzeugung oder Interaktionsmuster-Erzeugung handelt, wird sie direkt beim *Intern Executer* bearbeitet; als Resultat wird das konstruierte CAP bzw. Interaktionsmuster zurückgeliefert.**
4. **Das konstruierte CAP wird an den *Router* und das Interaktionsmuster an das *Template Repository* weitergeleitet.**

Abbildung 4-10: Algorithmus für die Interaktionsausführung

Ausführung einer Interaktion

Das Konzept der Interaktion bildet einen Oberbegriff für lokale Methoden, die CAP-Erzeugung und die Interaktionsmuster-Erzeugung (vgl. 4.2.2). Der Prozess der Ausführung wird durch den jeweiligen Interaktionstypen bestimmt. Lokale Methodenaufrufe werden direkt an die *Observable API* weitergeleitet, während die CAP-Erzeugung und Interaktionsmuster-Erzeugung innerhalb des Intern Executers ausgeführt werden (*Create Cap/Template)*. Falls eine Interaktion ein Event enthält, muss das Auswertungsergebnis des Events in die Ausführung der Interaktion einbezogen werden. Abbildung 4-10 beschreibt die Ausführung einer Interaktion im Überblick.

Evaluierung von CAPs und Interaktionsmustern

Neben der Verarbeitung von Interaktionen ist der *Intern Executer* auch zuständig für die Evaluierung von eingehenden CAPs und validen Interaktionsmustern (*Trigger*==true).

Die Evaluierung von CAPs erfolgt wie in CAPEUS; siehe Kapitel 3. Ein Interaktionsmuster wird evaluiert, indem die inkludierten Interaktionen sukzessive nach dem in Abbildung 4-10 abgebildeten Algorithmus ausgeführt werden. Zudem verwaltet der *Intern Executer* den Lebenszyklus eines Interaktionsmusters. *One-Shot-Interaktionsmuster* werden nach ihrer Auswertung gelöscht, während *Multi-Shot-Interaktionsmuster* im *Template Repository* verbleiben.

4.4.5 Intern Event-Monitor

Der *Intern Event-Monitor* bezieht Daten, d.h. Informationen von der Umgebung mittels Sensoren, welche im Dienstendpunkt integriert sind. Grundsätzlich entspricht der *Intern Event-Monitor* der Funktion des *Event Monitors* der CAPEUS Architektur, welcher zum Matching von *Context Constraints* mit aktuell gemessenen Sensordaten eingesetzt wird.

Bevor der *Executer* eine Interaktion ausführt, welche ein Event enthält, wird der Event zum *Intern Event-Monitor* zur Überprüfung weitergeleitet. Der Event-Monitor implementiert sowohl *Pull-* als auch *Push-Modi* zum Abgleich von Sensordaten. Diesbezüglich wird das in Kapitel 5 vorgestellte Kontext-fusionsmodell angewendet.

4.4.6 Intern Router

Die letzte zu behandelnde Komponente des *Interaction-Organizers* ist der *Intern Router*. Die Konzeption des *Intern Routers* erweitert die von CAPEUS implementierten Routingprozesse. Während innerhalb von CAPEUS die Zustellung von CAPs immer über mindestens ein intermediäres Routerelement (Service Interaction Proxy) erfolgt, kann die Kommunikation basierend auf Interaktionsmustern unmittelbar zwischen Dienstendpunkten erfolgen. Das heißt, dass Teile der Routingfunktionalität auf die Dienstendpunkte verlagert werden. Die beiden Routingstrategien können kombiniert eingesetzt werden. CAPEUS erlaubt die assoziative Zustellung von CAPs über verschiedene Subnetzwerke hinweg, wohingegen der hier vorgestellte lokale Routingprozess innerhalb eines Subnetzwerkes angewendet werden kann, ohne dass ein explizites Routerelement erforderlich ist. Insbesondere bietet sich der Ansatz des lokalen Routings in *Personal Area Networks* (PANs) an, wo individuelle Dienstendpunkte mittels einer drahtlosen Netzwerktechnologie miteinander verbunden werden (z.B. Bluetooth [Blue]).

Im Rahmen von CAPEUS wird eine Ziel-Entität durch die in einem CAP inkludierten *Context Constraints* bestimmt. Dabei ermittelt der *Router* zunächst eine Menge von in Frage kommenden Dienstendpunkten; anschließend wählt die *Matcher-Komponente*, entsprechend der vorgegebenen Entitätsbeschreibungen und Relationen, eine konkrete Ziel-Entität aus.

Der *Intern Router* vereinigt diese beiden Evaluationsschritte. Das gewählte Routingverfahren ist abhängig vom Beschreibungstyp der im CAP enthaltenen Ziel-Entität (abstrakt, konkret, oder sitzungsabhängig). Der *Intern Router* verwaltet eine zweispaltige Tabelle, eine für die SessionID, die andere für die konkrete Zieladresse (z.B. IP-Adresse). Das Attribut SessionID kann die zugehörige Zieladresse eindeutig bestimmen; *SessionID* ist ein Primärschlüssel. Es

ist zu beachten, dass jedem Interaktionsmuster nur genau eine *SessionID* zugeordnet werden kann.

1. **Service Discovery initiieren (vgl.** 2.4**).**
2. **Evaluieren der Ergebnisse aus Schritt 1 bez. Relationen des CAPs.**
3. **Die Ausgabe von Schritt 2 nach einen vordefinierten Kriterium sortieren (z.B. räumlicher Abstand). Das erste Ergebnis wird dann als Ziel selektiert, die verbleibenden Kandidaten werden im Meta Constraint des CAPs zwischengespeichert.**
4. **Zieladresse uns SessionID in die Routing-Tabelle eintragen.**
5. **Falls die Ziel-Entität nicht ansprechbar ist oder das CAP wegen anderer Gründe (z.B. Netzfehler) nicht zustellbar ist, wird der nächste Kandidat den Meta Constraints entnommen; die Routing-Tabelle wird entsprechend modifiziert. Dieser Prozess wird solange wiederholt, bis keine weiteren Kandidaten mehr verfügbar sind.**

Abbildung 4-11: Routing-Algorithmus für abstrakte Entitäten

Die Aufgabe des Routingverfahrens ist es, die Zieladresse eines CAPs zu determinieren. Der Ablauf des Routingverfahrens kann nach den Typen der Entitätsbeschreibungen klassifiziert werden.

- **Konkrete Entitäten:** Die Zieladresse ist bereits im zu routenden CAP enthalten und kann somit direkt an die Ziel-Entität gesendet werden. Die *SessionID* und Zieladresse werden in die *Session-Table* eingetragen.
- **Abstrakte Entität:** Bei abstrakten Entitätsbeschreibungen wird die Zieladresse durch die in den *Context Constraints* angegebenen Dienstanforderungen bestimmt, es erfolgt also eine kontextadaptive Adressierung. Der Routing-Algorithmus wird in Abbildung 4-11 dargestellt.
- **Sitzungsabhängige Entitäten:** Bei sitzungsabhängigen Entitäten erhält man die Zieladresse durch Abfragen der SessionID, die vorher entweder durch den Routing-Prozess einer konkreten oder abstrakten Entität determiniert wurde. Das CAP wird an die Ziel-Entität weitergeleitet.

Die oben erläuterten Algorithmen fundieren die in 4.2.3 formulierten Kompositionseinschränkungen für Interaktionen eines Interaktionsmusters.

Ohne den Eintrag der Routing-Tabelle kann das Ziel einer sitzungsabhängigen Entität nicht bestimmt werden (siehe oben). Die Zieladresse einer Sitzung wird nur dann in die Routing-Tabelle eingetragen, wenn die Adresse einer konkreten oder abstrakten Entität innerhalb eines Routing-Prozesses determiniert wurde. Aus

diesem Grund ist es unzulässig, eine Sitzung mittels einer sitzungsabhängigen Entität zu definieren, weil ihre Angaben zur Bestimmung des Ziels nicht ausreichend sind. In Konsequenz dürfen die Entitäten eines initialen CAPs nur durch abstrakte oder konkrete Beschreibungen identifiziert werden.

Sollte das zweite (oder irgendein späteres) CAP einer Sitzung konkrete oder abstrakte Entitäten enthalten, so würde ihre Zieladresse unter derselben SessionID in die Routing-Tabelle eingetragen werden. Die Zieladresse des ersten CAP, welche das Ziel einer Sitzung definiert, würde dadurch überschrieben. Deshalb muss das zweite CAP einer neuen Sitzung zugeordnet werden. Das ist der Grund, weshalb alle nachfolgenden CAPs sitzungsabhängig sein müssen.

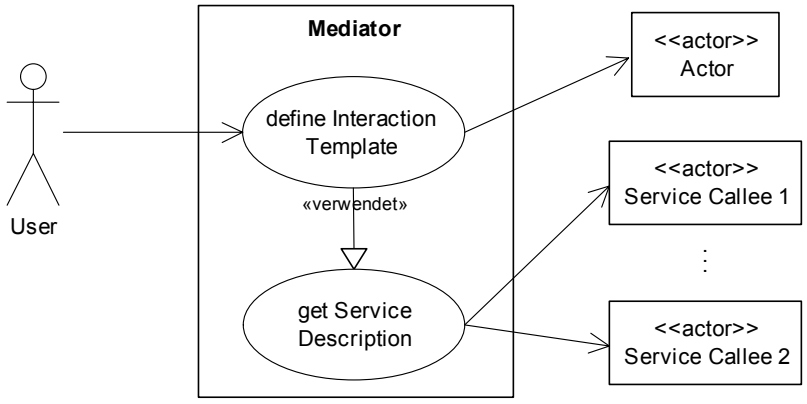

Abbildung 4-12: Anwendungsdiagramm Mediator

4.5 Tool-basierte Konstruktion von Interaktionsmustern

Die vorigen Abschnitte haben die Spezifikation und Evaluation von Interaktionsmustern beschrieben. Nun soll der Entwurf eines Toolkits vorgestellt werden, welches die Konstruktion von Interaktionsmustern unterstützt.

Zentral für die toolgestützte Komposition von Interaktionsmustern ist die *Mediator* Komponente. Abbildung 4-12 zeigt das Anwendungsdiagramm für diese Komponente. Der *Mediator* ermöglicht dem Benutzer die Definition von Interaktionsmustern vermittels einer grafischen Benutzeroberfläche. Der Konstruktionsprozess lässt sich in folgende Phasen untergliedern:

- Bestimmung beteiligter Dienstendpunkte (*Service Callees*)
- Definition von Interaktionsmustern
- Dekomposition und Distribution der Interaktionsmuster

Zunächst wird die Menge der an einem Prozess beteiligten Dienstendpunkte bestimmt, von denen der *Mediator* die Dienstschnittstellen anfordert. Auf diesen Dienstschnittstellen erfolgt die Definition der Interaktionsmuster. Nach erfolgreicher Erstellung der Interaktionsmuster kann es erforderlich sein, dass diese aufgrund verletzter Kompositionsvorschriften noch zerlegt werden müssen. Schließlich erfolgt dann eine Distribution der Interaktionsmuster auf die beteiligten Dienstendpunkte. Die erforderlichen Kommunikationsschritte erfolgen via CAPs.

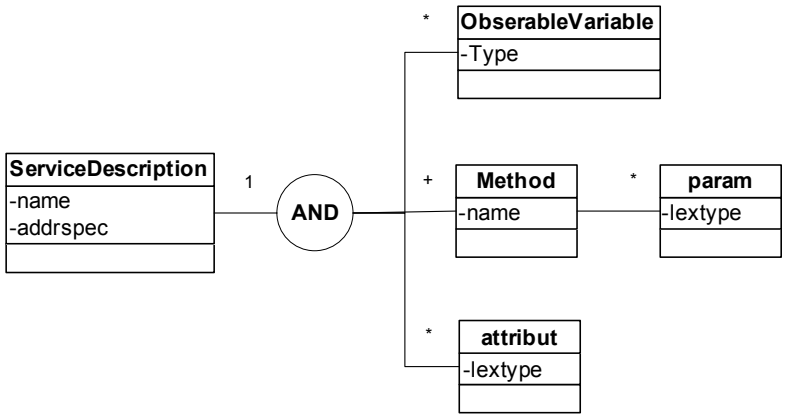

Abbildung 4-13: Strukturdiagramm einer Dienstbeschreibung

Der *Mediator* erlaubt nicht nur die Konstruktion neuer Interaktionsmuster, sondern auch die Modifikation bereits bestehender Interaktionsmuster. Abschnitt 4.6.3 präsentiert die prototypische Implementierung des *Mediators*.

4.6 Prototypische Implementierung

Die Implementierung der im letzten Kapitel beschriebenen Konzepte ist Bestand der nachfolgenden Abschnitte.

4.6.1 Repräsentation von Dienstbeschreibungen

Die hier verwendeten Dienstbeschreibungen erweitern das Dienstbeschreibungskonzept von CAPEUS um Eigenschaften, welche durch den Einsatz von Observable APIs (siehe 4.4.2) hinzukommen. Es handelt sich um einen neuen Dienstbeschreibungstyp, welcher in die vorhandene Hierarchie eingegliedert werden kann.

Analog zu CAPEUS macht die Dienstbeschreibung Angaben über die Eigenschaften und die verfügbaren Schnittstellen eines Dienstes. Zusätzlich

deklariert sie aber auch die so genannten beobachtbaren Variablen (*observable variables*) eines Dienstes. Schließlich werden auch Name, Adresse, und Eigenschaften eines Diensterbringers formuliert.

Die entstehende Dienstbeschreibung besteht im Wesentlichen aus drei Grundkonstrukten: *Attribut, ObservableVariable* und *Method* (siehe Abbildung 4-13). Eine vollständige *Document Type Definition* (DTD) findet sich in [Mi 01].

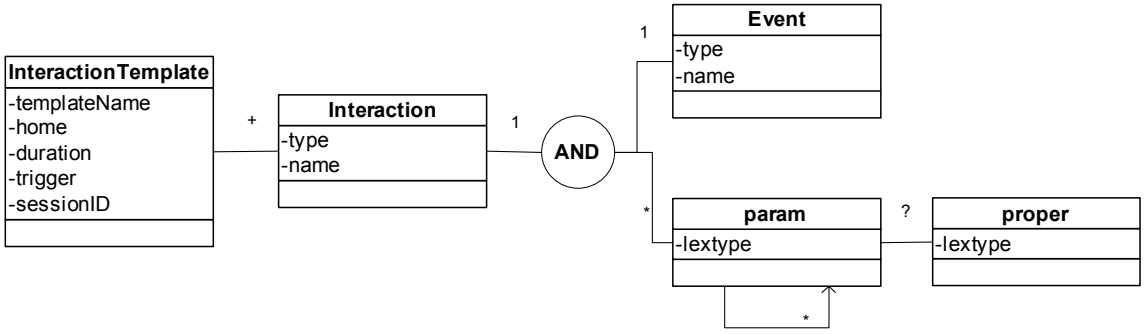

Abbildung 4-14: Strukturdiagramm eines Interaktionsmusters

4.6.2 Repräsentation von Interaktionsmustern

Das Interaktionsmuster ist eine zentrale Datenstruktur des in diesem Kapitel vorgestellten Systems. Es definiert eine Sequenz von Interaktionen an einem Dienstendpunkt, welche durch eine Trigger-Bedingung ausgelöst werden. Ein Interaktionsmuster kann sowohl lokale Methodenaufrufe spezifizieren als auch Kontextadaptive Interaktionen und weitere Interaktionsmuster. Auch hier wurde XML [Bou 99] für die Repräsentation gewählt. Abbildung 4-14 zeigt die Grammatik eines Interaktionsmusters im Überblick.

4.6.3 Implementierung des Interaction Organizers

Der *Interaction Organizer* erweitert die CAPEUS-Architektur um die kontextadaptive Dienstintegration in Ubiquitous Computing Umgebungen. Die Implementierung des *Interaction Organizers* wird in den folgenden Abschnitten erläutert. Im Kontrast zu CAPEUS – welches mittels Java [Java] entwickelt wurde – wurden die hier vorgestellten Komponenten in der Programmiersprache *Python* [Py] implementiert. Die Begründung für die Entscheidung, Python als Implementierungssprache zu wählen, erfolgt im nachfolgenden Abschnitt.

Die Programmiersprache Python

Python ist eine objektorientierte, interpretierte Sprache, die als freie Software für nahezu alle gängigen Systemumgebungen verfügbar ist (UNIX, Microsoft Windows™, Apple Macintosh™, PalmOS™). Entwickelt wurde Python Ende 1989 von dem niederländischen Mathematiker Guido von Rossum. Mittlerweile besteht ein reicher Bestand an Literatur zur Einführung in die Sprachkonzepte und begleitende Referenzdokumentationen [Py]. Neben der Plattformunabhängigkeit, der Ausnahmebehandlung und Objektorientierung, durch die sich auch Java auszeichnet, bietet Python für den Einsatz in dieser Arbeit ausschlaggebende Vorteile: direkte Realisierung der Mehrfachvererbung, dynamische Generierung und Interpretation von Quelltext zur Laufzeit, schnelle Entwicklung von Prototypen und eine gut integrierte XML-Anbindung.

Im Gegensatz zu Java implementiert Python einen Vererbungsmechanismus, mit dem das Prinzip der Mehrfachvererbung direkt abgebildet werden kann. Somit können die Unterklassen die Attribute und Methoden multipler Oberklassen erben. Dieses Konzept wird insbesondere bei der Implementierung der *Observable API* (siehe 4.4.2) eingesetzt.

Als interpretierte Sprache bietet Python Konzepte, die in übersetzten Sprachen in der Regel nicht möglich sind. So ist es beispielsweise möglich, zur Laufzeit die Objektstruktur zu modifizieren, so dass Objekte dynamisch aufgebaut werden können. Dieser Aspekt kann insbesondere bei der Verarbeitung von Dokumenten, wie auch XML-Dokumenten, gewinnbringend eingesetzt werden. Zudem unterstützt Python aufrufbare und ausführbare Objekte (*callable objects*) und mittels den Funktionalen *eval* und *exec* besteht die Möglichkeit, dynamisch generierte Codefragmente zur Laufzeit auszuwerten. Innerhalb des *Interaction Organizers* wird *eval* zur dynamischen Auswertung von Trigger-Bedingungen herangezogen, welche mittels Python Codefragementen repräsentiert werden.

Python hat sich insbesondere auch für das *Rapid Prototyping* bewährt, da es sich um eine so genannte Integrationssprache handelt [Ous 98], die es erlaubt, bereits bestehende Komponenten mittels einfacher Sprachkonstrukte zu einem Gesamtsystem zu komponieren.

Implementierungsübersicht

Im Folgenden soll die Struktur und Funktion der einzelnen Softwarekomponenten des *Interaction Organizers* vorgestellt werden. In diesem Abschnitt werden Implementierungspakete in Kurzform vorgestellt. Pakete bzw. inkludierte Klassen, welche für das Verständnis der Architektur einen wesentlichen Beitrag leisten werden nachfolgend in gesonderten Abschnitten detailliert erörtert.

Paket *contentObj*: Dieses Paket enthält alle Klassen, welche die konkreten Inhaltsobjekte beschreiben. Als Inhaltsobjekte werden hier CAPs, Interaktionsmuster und ihre Teilstrukturen verstanden. Dazu gehören beispielsweise Relationen, Entitäten und Interaktionen. Jede Klasse, die ein Inhaltsobjekt verkörpert, enthält die Methode self.toXML(), so dass Inhaltsobjekte jederzeit wieder in XML-Dokumente umgewandelt werden können.

Paket *util*: Das Paket *util* enthält nur eine Klasse: *XML_Objectify* [Mer 00]. Diese Klasse erlaubt, basierend auf DOM [Woo 98] eingehende XML-Dokumente zur Laufzeit in korrespondierende Python-Objekte umzuwandeln.

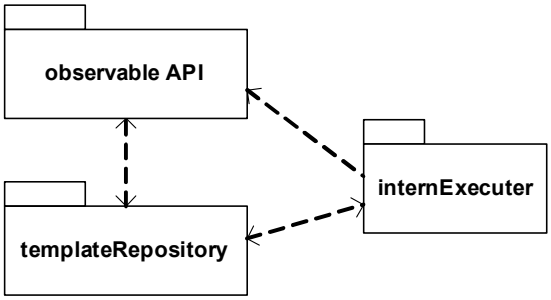

Abbildung 4-15: Abhängigkeiten der Unterpakete von *device*

Die Pakete *contentObj* und *util* implementieren lediglich Grundfunktionalitäten des *Interaction Organizers*. Die Hauptfunktionalität wird innerhalb des Paketes *device* definiert (s.u.).

Paket *device*: Wie bereits erwähnt, wird die Hauptfunktionalität innerhalb dieses Paketes implementiert. Das Paket ist in mehrere Unterpakete untergliedert worden: *observableAPI, templateRepository* und *internExecutor* (siehe auch Abbildung 4-15).

Die Teilkomponenten *Intern Router* und *Intern Event-Monitor* sind im Rahmen dieser Arbeit nicht implementiert worden, da die entsprechenden Hardwarevoraussetzungen zum gegebenen Zeitpunkt nicht erfüllt werden konnten (z.B. Bluetooth-fähiges Endgerät). Im weiteren Verlauf werden die Eigenschaften der oben genannten Teilkomponenten erläutert.

Paket *observableAPI*

Zur Realisierung der *observableAPI* wurden die Software-Techniken Mehrfachvererbung und das erweiterte *Observer* Designmuster [GHJV 95] angewandt. Die Erläuterung der Implementierung erfolgt bez. des Szenarios aus Abschnitt 4.3.2; siehe auch Abbildung 4-7. Folgende Klassen dienen zur Realisierung der *Observable API*: *Telephone*, *ObservableService* und *ObservableAPI*. Die

Telephone Klasse spezifiziert die native Dienstschnittstelle des Telefons. *Telephone* unterscheidet die folgenden Attribute zur Spezifikation des internen Zustandes: isRinging, isWaiting und isTalking. Neben den Attributen definiert die Klasse noch folgende Methoden: **`ring(self), pickUp(), ringOff(), activateSpeaker()`** und **`getService-Description()`**, welche die entsprechenden Dienste implementieren. Der Aufruf der Methode **`getServiceDescription()`** liefert die Dienstbeschreibung des Telefonapparates als XML-Dokument; Abschnitt 4.6.1 beschreibt das Format.

Die ObservableService-Klasse realisiert das erweiterte *Observer*-Designmuster, welches es erlaubt, den Status einer Komponente bez. registrierbarer Kriterien zu überwachen und die erfüllten Kriterien bekannt gibt. Die *ObservableService*-Klasse ist nicht dienst- oder gerätespezifisch. Sie kann bei allen zu überwachenden Dienstendpunkten eingesetzt werden. Sie enthält zwei Attribute: criterList, templateManager. Das Attribut citerList ist eine Liste, die alle registrierten Kriterien zusammenfasst. Das Attribut templateManager verweist auf den Interessenten für ein bestimmtes Kriterium. Zur Verwaltung der zu überwachenden Kriterien definiert *ObservableService* die folgenden Methoden:

- **`__init__()`**: der Konstruktor erzeugt ein neues Objekt des beobachtbaren Dienstes
- **`addCriterion(criter)`**: registriert das übergebene Kriterium criter, indem es in die Liste criterList hinzugefügt wird
- **`deleteCriterion(criter)`**: löschen eines registrierten Kriteriums
- **`getCriterions()`**: gibt alle registrierten Kriterien aus
- **`Notify(criter)`**: teilt dem templateManager das erfüllte Kriterium mittels self.templateManager.fire(criter) mit
- **`checkAll()`**: überprüft alle registrierten Kriterien und meldet erfüllte Kriterien an templateManager
- **`check(criter)`**: überprüft mittels eval(criter) das angegebene Kriterium criter und liefert das Ergebnis zurück

Die *Observable API* ist die erweiterte API und vereinigt die Funktion der nativen Dienstschnittstelle eines Dienstendpunktes mit Überwachungsfunktionen. Das heißt, die Klasse *ObservableAPI* erhält die Funktionalitäten der beiden oben beschriebenen Klassen; sie wird daher mittels Mehrfachvererbung der beiden Klassen realisiert; siehe Abbildung 4-16.

Die abgeleitete Klasse *ObservableAPI* besitzt nun alle Attribute und Methoden der beiden Oberklassen. Die Überwachung der registrierten Kriterien wird dadurch realisiert, dass nach jedem Aufruf jeder Methode, die eine Änderung des internen

Zustandes verursachen kann - ring(), pickUp(), ringOff() und activateSpeaker() –, alle registrierten Kriterien überprüft werden. Die Methoden, die in der *ObservableAPI* implementiert werden, sind also die Methoden, die den internen Zustand des Dienstendpunktes alternieren können. Davon ausgeschlossen ist beispielsweise getServiceDescription(), welche eine Lese-Operation durchführt. Es ist zu bemerken, dass auch nichtöffentliche Methoden Bestandteil der *ObservableAPI* sein können.

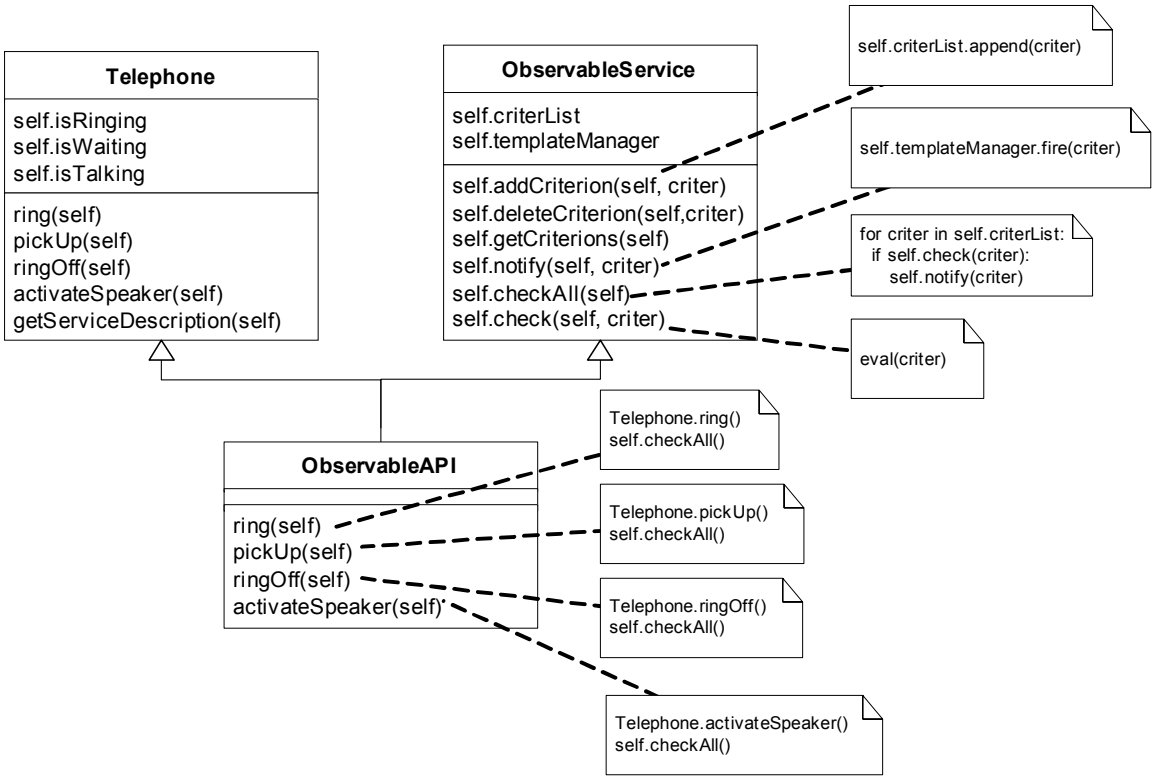

Abbildung 4-16: Klassendiagramm für observableAPI

Die Implementierung der *ObservableAPI* ist offensichtlich abhängig von dem in Frage kommenden Dienstendpunkt. Sie muss deshalb zum Einsatz bei anderen Dienstendpunkten entsprechend angepasst werden.

Paket *TemplateRepository*

Anders als die *Observable API* ist die Implementierung des *Template Repositories* bzw. des anschließend erläuterten *Intern Executers* nicht spezifisch für den betrachteten Dienstendpunkt.

Die Teilkomponente *TemplateRepository* des *Interaction Organizers* wird innerhalb des gleichnamigen Paketes implementiert. Es enthält nur die Klasse

TemplateManager, die über Verwaltungsfunktionen der Interaktionsmuster verfügt.

Neu generierte Interaktionsmuster werden via des *Intern Executers* angelegt (vgl. Abbildung 4-9). Die Klasse *TemplateManager* hat drei Attribute: **`observableAPI`**, **`templates`** und **`dispatcher`**. Das Attribut **`observableAPI`** bezieht sich auf eine Instanz der Klasse *ObservableAPI*, bei welcher die Trigger eines Interaktionsmusters registriert werden. Das Attribut **`templates`** verweist auf ein *Python Dictionary* [Py]. Hier werden die zu verwaltenden Interaktionsmuster abgespeichert. Die Trigger-Bedingung ist Primär-Schlüssel. **`dispatcher`** ist eine Instanz der Klasse *Dispatcher* dem Paket *internExecuter* angehörend. Die Interaktionsmuster, deren Trigger erfüllt sind, werden an diese Instanz zur Bearbeitung weitergegeben. Die Klasse *Dispatcher* wird im nächsten Abschnitt behandelt.

Im Folgenden werden die Methoden der Klasse *TemplateManager* kurz vorgestellt, sie dienen der Verwaltung der in **`templates`** gespeicherten Interaktionsmuster. Zudem wird die Konsistenz der Daten bei jeder Operation überprüft.

- **`__init__(observableAPI, dispatcher)`**: dieser Konstruktor legt die mit dem Objekt interagierenden *ObservableAPI*- und *Dispatcher* Instanzen fest
- **`getTemplateString(templateName)`** liefert das Interaktionsmuster als String zurück
- **`addTemplate(template, trigger, fileName)`** speichert das Interaktionsmuster als Datei und registriert den Trigger bei der Observable API.
- **`removeTemplate(trigger, templateName=none)`** löscht das durch den Namen spezifizierte Interaktionsmuster. Falls kein Name angegeben wird, werden alle Interaktionsmuster mit dem angegebenen Trigger gelöscht.
- **`activateTemlate(trigger, templateName=none)`** aktiviert das durch den Namen spezifizierte Interaktionsmuster. Falls kein Name angegeben wird, werden alle Interaktionsmuster mit dem angegebenen Trigger aktiviert.
- **`deactivateTemlate(trigger, templateName=none)`** deaktiviert das durch den Namen spezifizierte Interaktionsmuster. Falls kein Name angegeben wird, werden alle Interaktionsmuster mit dem angegebenen Trigger deaktiviert.

- **`isActive(trigger, fileName)`** liefert zurück, ob das spezifizierte Interaktionsmuster aktiv ist.
- **`listTemplate(trigger)`** listet alle Interaktionsmuster des angegebenen Triggers auf.
- **`fire(trigger)`**: gibt alle Interaktionsmuster des angegebenen Triggers an den **`dispatcher`** zur Bearbeitung weiter.

Paket *internExecutor*

Der *Intern Executor* realisiert die Evaluation von Interaktionen (CAP, Interaktionsmuster). Der Grobentwurf des *Intern Executors* wurde durch zweifache Anwendung des *Mediator* Designmusters [GHJV 95] verfeinert. Abbildung 4-17 zeigt die Architektur des *Intern Executors* in einem Überblick.

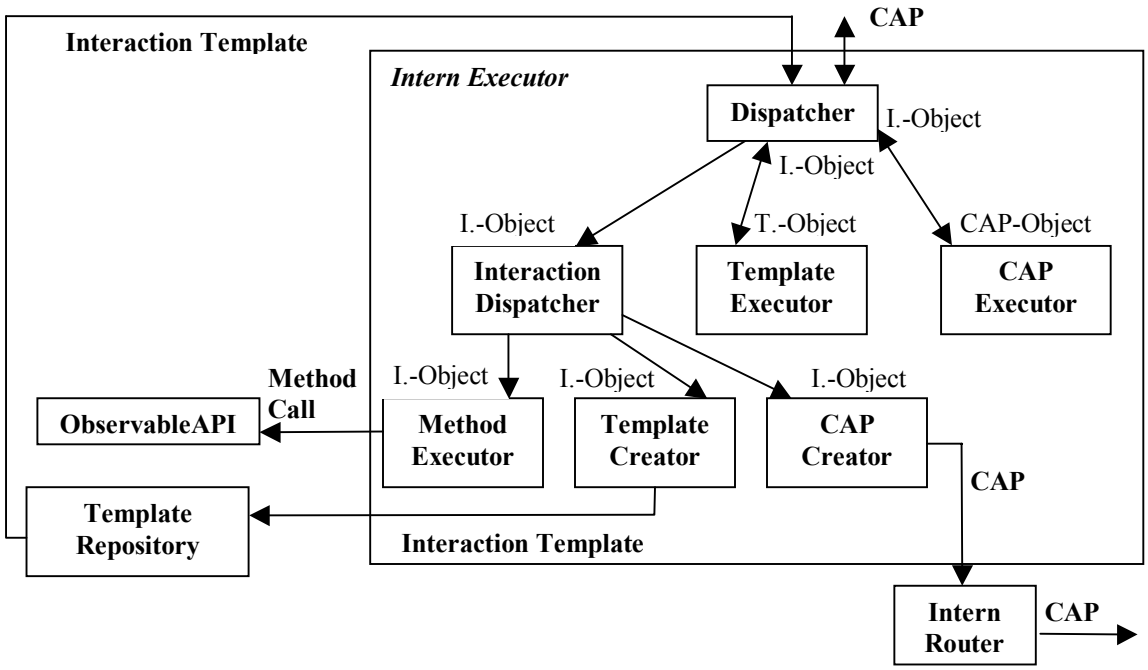

Abbildung 4-17: *Intern Executor* Architektur

Wie bereits beschrieben, bildet die Interaktion einen Oberbegriff für unterschiedliche Interaktionstypen: lokale Methodenaufrufe, Interaktionsmuster und Kontextadaptive Interaktionen. Zur Evaluierung der einzelnen Interaktionstypen definiert der *Intern Executor* folgende Klassen: *MethodExecutor*, *TemplateCreator* und *CAPCreator*. Außerdem wird die Klasse *Interaction-Dispatcher* definiert, welche die Funktion der oben genannten Klassen koordiniert.

Neben der Evaluation von Interaktionen ist noch die Ausführung von CAPs und Interaktionsmustern zu bewerkstelligen. Zu diesem Zweck dienen die Klassen *TemplateCreator* und *CAPExecutor*. Analog zu der Klasse *Interaction Dispatcher* koordiniert die Klasse *Dispatcher* das Zusammenspiel dieser Klassen. Der Entwurf der beiden *Dispatcher-Klassen* orientiert sich am *Mediator* Designmuster [GHJV 95].

4.6.4 Implementierung des Mediators

Wie bereits in Abschnitt 4.5 erwähnt, implementiert der *Mediator* ein Tool, welches zur Konstruktion von neuen Interaktionsmustern dient. Der Mediator kann auf einem CAPEUS-fähigen Endgerät (z.B. PDA oder Laptop) ausgeführt werden. Die erforderlichen Kommunikationsschritte erfolgen mittels Kontext-adaptiver Interaktionen bzw. CAPs.

Zunächst werden innerhalb eines Dienstvermittlungs-Prozesses (*Service Discovery*) die beteiligten Dienstendpunkte bestimmt; dann fordert der *Mediator* die Dienstbeschreibungen dieser Dienstendpunkte an. Das gewünschte System-verhalten kann der Benutzer mittels Erstellung von entsprechenden Interaktions-mustern definieren. Zu diesem Zweck wird ein Tool mit grafischer Benutzeroberfläche bereitgestellt (s.u.).

Nachdem der Benutzer die Prozessmodellierung beendet hat, erzeugt das Tool Interaktionen, welche die Generierung der entsprechenden Interaktionsmuster spezifizieren. Diese Interaktionen werden innerhalb von CAPs gekapselt und zu den entsprechenden Dienstendpunkten gesendet. Dort werden diese Interaktionen dann ausgeführt und in Konsequenz die Interaktionsmuster installiert. Diesen Vorgang veranschaulicht der obere Teil von Abbildung 4-18.

Zur Realisierung des Beispielszenarios aus 4.3.2 (Telefon-Stereoanlage) wird zunächst am Telefon ein mehrmaliges (*multi shot*) Interaktionsmuster benötigt, welches ein CAP an die Stereoanlage sendet, um die Lautstärke zu verringern, wenn das Telefon klingelt. Direkt danach konstruiert das Interaktionsmuster ein einmaliges (*one shot*) Kind-Interaktionsmuster, das beim Auflegen des Telefons ein weiteres CAP an die Stereoanlage sendet, um die Lautstärke wieder zu erhöhen (siehe unterer Teil von Abbildung 4-18).

Das *Mediator* Paket realisiert den Anwendungsfall „*define Interaction Template*“ (vgl. 4.5) und enthält folgende Applikationsklassen: *MediatorGUI* und *HandleFrame*. Die Klasse *MediatorGUI* implementiert die GUI-Oberfläche, mit der man die zu erzeugenden Interaktionsmuster definieren kann. Die Klasse *HandleFrame* analysiert die eingegebenen Informationen und konstruiert in Folge ein CAP, welches eine Interaktion des Typs „Interaktionsmuster-Erzeugung“ enthält. Zudem überprüft diese Klasse, ob die definierten Interaktionsmuster den

in 4.2.3 formulierten Kompositionseinschränkungen genügen. Sollten sie diesen Bedingungen nicht genügen, so werden sie entsprechend zerlegt.

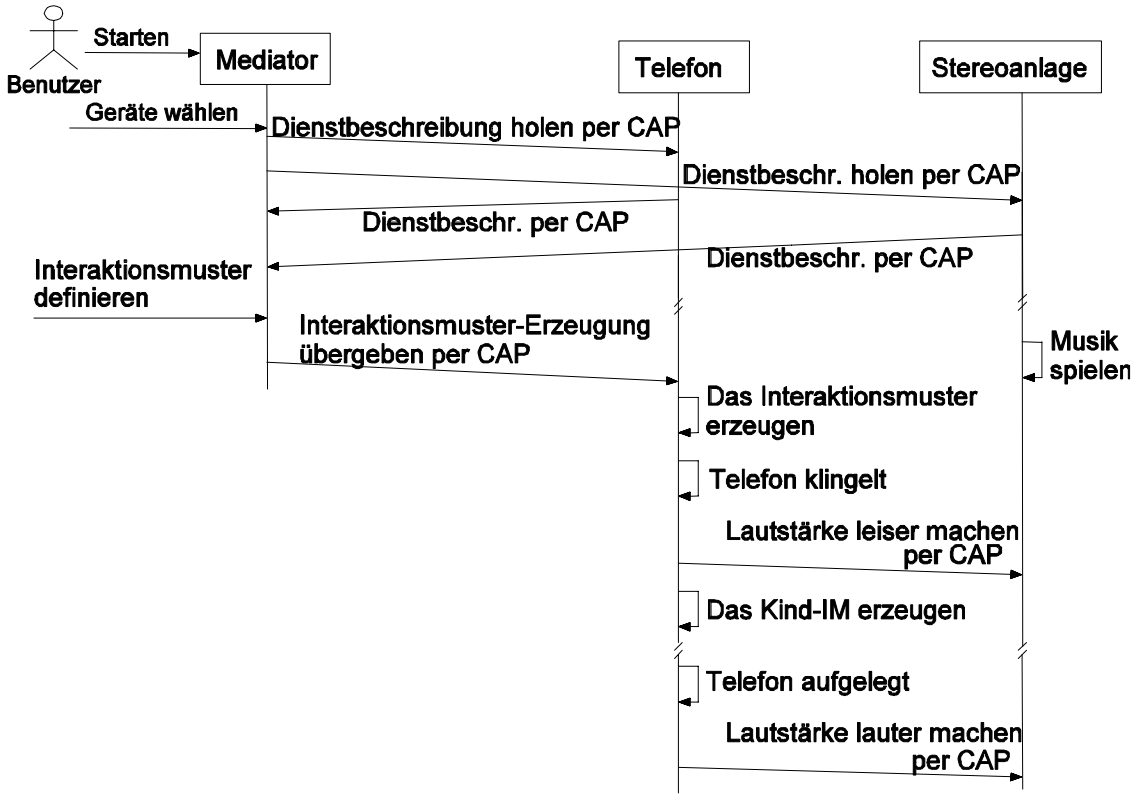

Abbildung 4-18: Sequenzdiagramm *Mediator*

Abbildung 4-19 zeigt einen Screenshot des *Mediators* für das Beispielszenario. Die Struktur der Oberfläche wird in Reaktion zu Benutzereingaben hierarchisch erweitert. Der *Mediator* erlaubt die Organisation von Interaktionsmustern, mit einer beliebigen Anzahl untergeordneter Kind-Interaktionsmuster, Kontext-adaptiven Interaktionen und lokalen Methodenaufrufen. Natürlich können Interaktionsmuster rekursiv geschachtelt werden. Die Definition von Interaktionsmustern erfolgt auf dem so genannten *Template-Frame*, während für die Festlegung von CAPs und lokalen Methoden das *Interaction Frame* verwendet wird (siehe Abbildung).

Eine ausführliche Anleitung zur Benutzung des *Mediators* findet sich in [L 01].

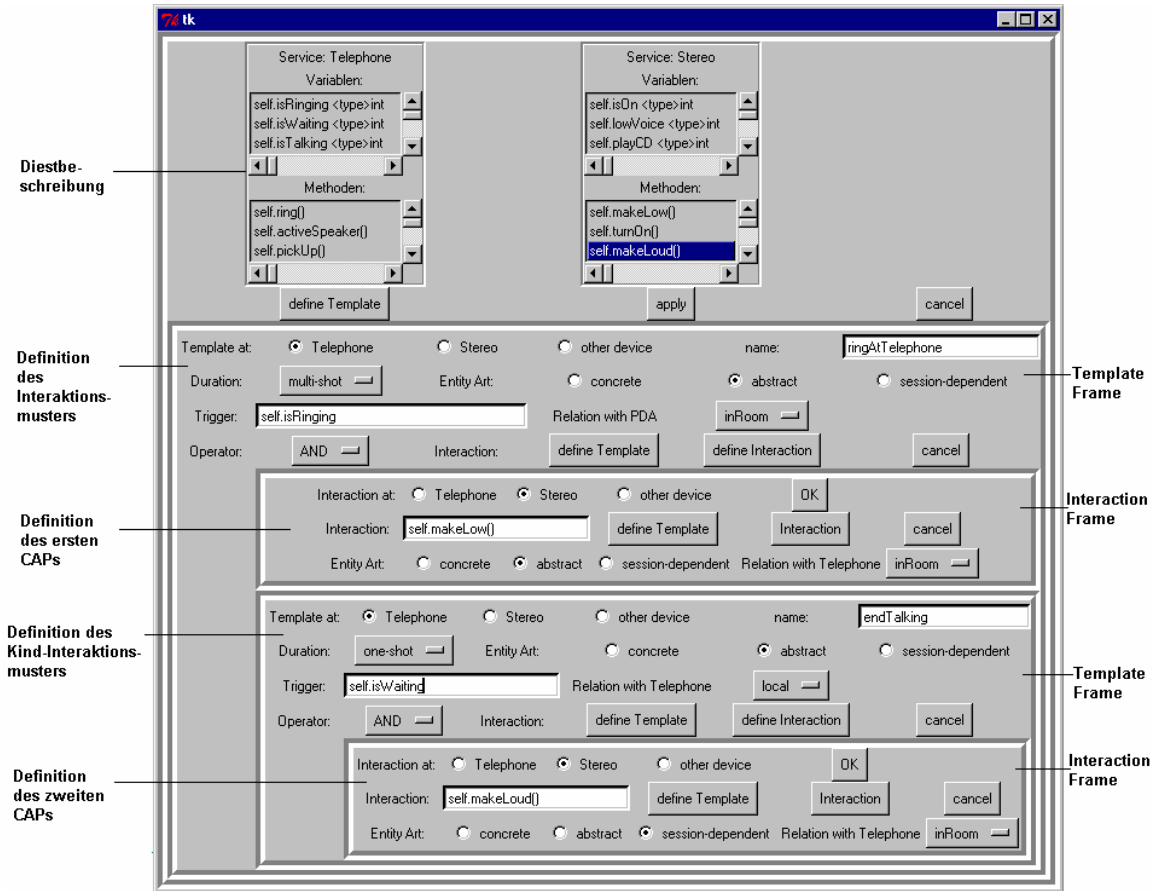

Abbildung 4-19: Die Benutzeroberfläche des *Mediators*

4.7 Verwandte Arbeiten

Die tool-gestützte Modellierung von Prozessen innerhalb von Ubiquitous Computing Umgebungen bildet einen noch jungen Forschungsbereich. Dennoch lassen sich einige Ansätze und Projekte identifizieren, welche für den hier gewählten Ansatz relevant sind. Da die in diesem Kapitel vorgestellten Konzepte direkten Bezug zu CAPEUS haben, ergänzen die in Abschnitt 3.6.1 erwähnten Arbeiten die hier folgenden.

Ein wesentlicher Aspekt der auf Interaktionsmustern basierenden Kommunikation zwischen Endgeräten ist die *Objektüberwachung*. Interne Zustände der Endgeräte werden als Kontextinformationen verwendet und beeinflussen die Kommunikation zwischen Dienstendpunkten. In Abschnitt 4.4.2 wurde in diesem Zusammenhang bereits CAMEO [Woo 96] erläutert, welches den Entwurf des hier vorgestellten Systems maßgebend beeinflusst hat. Dazu gehören auch die Systeme *K-Edit* [KE] und *Elvin* [SAS 01], die in den folgenden Abschnitten skizziert werden.

K-Edit

K-Edit [KE] bietet eine vergleichbare Funktionalität zu CAMEO [Woo 96], verfolgt aber einen allgemeineren Ansatz. K-Edit modelliert ein dezentrales System mit Punkt-zu-Punkt-Kommunikation. Knoten des Systems werden innerhalb eines hierarchischen Graphen organisiert. Ein Knoten innerhalb von K-Edit ist eine abstrakte Struktur, die entweder ein Datenobjekt oder eine Aktion repräsentiert. K-Edit wurde insbesondere für die Repräsentation von Metainformationen entwickelt. Im Folgenden soll auf die Basisoperationen von K-Edit eingegangen werden:

- *ls(node):* diese Operation liefert alle Kind-Knoten des Knotens *node* zurück.
- *stat(node)* dient zur Bestimmung der Statusinformationen eines Knotens.
- *set/get(node)* implementieren das Auslesen bzw. die Manipulation von Daten eines Knotens.
- *exec(node)* füht die mit dem Knoten *node* assozierte Aktion aus.
- *notify(node,mask)*. Mittels dieser Operation können sich Knoten für bestimmte Events des Knotens *node* registrieren. *mask* dient zur Spezifikation von den in Frage kommenden Event-Typen.

Die mittels *notify* spezifizierbare Objekbeobachtung erlaubt u.a. die Reaktion auf folgende Operationen: das Löschen eines Knotens, Lese- und Schreiboperationen, Ausführung einer Aktion. Diese Prinzipien der Objektbeobachtung (*notify*), der Inspektion (*ls*,*stat*) und der Manipulation (*set*, *get*, *exec*) entsprechen den Konzepten von CAMEO.

Elvin

Wie bereits erwähnt, realisiert Elvin [SAS 01] einen inhalts-bezogenen Nachrichtendienst in Verteilten Systemen. Nachrichten in Elvin werden durch Tuple repräsentiert. Jeder Eintrag eines Tuples wird bestimmt durch einen Datentyp und einen zugehörigen Wert. Die Kommunikation mittels Elvin erfolgt über einen zwischengeschalteten Server; siehe Abbildung 4-20. In Abhängigkeit vom Inhalt eines eingehenden Nachrichtentupels routet der Server diese an entsprechende Konsumenten (*Consumer*). Produzenten (*Producer*) versenden Nachrichten mittels *notify*. Konsumenten bekunden ihr Interesse an bestimmten Nachrichten mittels einer Subskription, die durch einen logischen Ausdruck repräsentiert wird (z.B. item==“gum“ && time > 800). Subskriptionen werden vom Server verwaltet und ihre Gültigkeit wird bez. aller eingehenden Nachrichten

verifiziert. Erfüllt eine Nachricht die Bedingungen einer Subskription, so wird die Nachricht an den entsprechenden Konsumenten weitergeleitet.

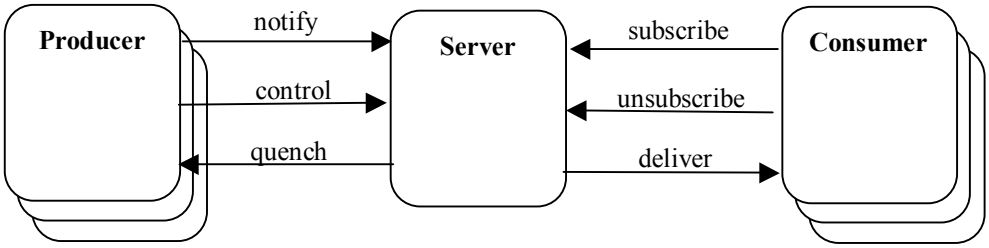

Abbildung 4-20: Elvin Protokoll-Architektur

Um nun den aufkommenden Datenverkehr zwischen Produzenten und Server zu reduzieren, verfolgt Elvin den Ansatz, entstehende Nachrichten bereits seitens des Produzenten zu filtern. Zu diesem Zweck müssen die Subskriptions-Terme auf die entsprechenden Produzenten verteilt werden. Dieses Verfahren wird als *Quenching* bezeichnet. Ähnlich wie CAMEO und K-Edit ist es auch hier erforderlich, dass der Zustand der Produzenten überwacht werden kann. Im Unterschied zu CAMEO und K-Edit erlaubt Elvin auch die Auswahl von Nachrichten bez. logischer Ausdrücke. Dieses Konzept wurde zur Auswertung von Interaktionsmustern übernommen.

4.8 Zusammenfassung

In diesem Kapitel wurde das Konzept der kontextadaptiven Dienstintegration beschrieben. Im Folgenden sollen die wichtigsten Aspekte bzw. die resultierenden Ergebnisse dieses Ansatzes zusammengefasst werden:

- Dynamische Rollenverwaltung und kontextadaptives Verhalten eines Dienstendpunktes.
- Tool-basierte Modellierung von kontextadaptiven Anwendungen durch Definition von Interaktionsmustern (z.B. mit dem *Mediator*, 4.5)
- Kontextadaptive Generierung von *Context-Aware Packets* (CAPs); Interaktionsmuster dienen als Vorlagen.

Von zentraler Bedeutung für die Realisierung der kontextadaptiven Dienstintegration ist die Konzeption des *Interaktionsmusters*. Zusammenfassend lässt

sich feststellen, dass Interaktionsmuster genutzt werden können, um Dienstendpunkten bzw. Endgeräten ein bestimmtes Verhalten vorzuschreiben. Das Verhalten wird bez. von Kontexten spezifiziert. Interaktionsmuster definieren insbesondere Dienstinteraktionen, die in Abhängigkeit von kontextuellen Einschränkungen ausgeführt werden. Die Komposition von Interaktionsmustern und die dynamische Parametrisierung von assoziierten Dienstinteraktionen erlaubt die Spezifikation von komplexen Kommunikationsmustern (*Communication Patterns*).

Die Interpretation und Verwaltung von Interaktionsmustern, die einem individuellen Dienstendpunkt zugeordnet sind, obliegt dem so genannten *Interaction Organizer* (4.4). Ein wesentlicher Bestandteil des *Interaction Organizers* ist das Konzept der *Observable API*. Die Observable API erweitert die native Dienstschnittstelle eines Dienstes insbesondere um die Möglichkeit der flexiblen Beobachtung von internen Statusänderungen einer Dienstinstanz. Die Reaktion auf interne Zustandsänderungen eines Endgerätes kann für die Formulierung von kontextadaptiven Anwendungen wesentlich sein, da sie das Repertoire verfügbarer Kontextinformationen in bedeutender Weise vergrößern kann. So lässt sich beispielsweise das Szenario: „Wenn der Benutzer ein Photo erstellt hat, dann soll es auf dem PDA angezeigt werden“ problemlos mittels eines Interaktionsmusters beschreiben, welches sich lediglich auf eine Zustandsänderung der Kamera bezieht.

Ein Ausblick auf zukünftige Forschungsarbeiten im Bereich der kontextadaptiven Integration erfolgt in 6.1.2.

Kapitel 5

Konstruktion und Komposition von Kontext-informationen

Ein essentieller Aspekt bei der Entwicklung von kontextadaptiven Anwendungssystemen, wie sie in Kapitel 3 und 4 beschrieben wurden, ist die Gewinnung von Kontextinformationen. Hier soll die Konstruktion von Kontextinformationen innerhalb von Ad-hoc-Sensornetzwerken untersucht werden. Diese Netzwerke werden durch autonome Sensorknoten gebildet, welche zum einen Umgebungsparameter wie etwa die Lautstärke oder den Lichteinfall bestimmen können und zum anderen die Möglichkeit bieten, diese Messwerte zu verarbeiten und über ein drahtloses Medium zu verbreiten. Durch Kooperation multipler Sensoren können so kontextuelle Informationen im Rahmen eines Datenfusions-Prozesses [A 92] sukzessive ermittelt werden.

Es werden hier Sensoren betrachtet, die an alltägliche Einrichtungsgegenstände gebunden sein können, so etwa an Stühle, Tassen oder ähnliches [Smart]. Da sich die Position dieser Objekte jederzeit verändern kann und die Netzwerkreichweite aufgrund von Energiesparmaßnahmen [RMGS] meist begrenzt ist und Knoten jederzeit hinzugefügt bzw. entfernt werden können, unterliegt die Netzwerktopologie ständigen Fluktuationen.

Netzwerke, die diesen Randbedingungen entsprechen, werden gemeinhin als Ad-hoc-Sensornetzwerke [ASSC 02] bezeichnet. Bisherige Forschungsarbeiten [WINS] fokussieren insbesondere effiziente Routing-Verfahren und Energiesparmechanismen. Diese Aspekte sollen auch in der hier erörterten Konzeption berücksichtigt werden, wobei jedoch der Schlüsselpunkt des hier vorgestellten Beitrages ein neues, daten-zentrisches (vgl. 3.1.3) Protokoll ist, welches die Konstruktion und Komposition von Kontextinformationen nativ unterstützt. Die Gewinnung von Kontext erfolgt selbstorganisierend und dezentral.

Grundlage für das hier vorgestellte Protokoll ist ein uniformes Datenaustauschformat, welches als *Smart Context-Aware Packet* (SCAP) [MS 01] bezeichnet wird. Es dient sowohl dem Datenaustausch als auch der Koordination von Sensorknoten; es ist eine Weiterentwicklung der in Kapitel 3 vorgestellten Konzeption des *Context-Aware Packets* (CAPs).

Das auf SCAPs basierende *Smart Stack Routing* (SSR) Protokoll zur Gewinnung von Kontextinformationen wird in Abschnitt 5.4 erläutert. Zunächst erfolgt eine Übersicht über den Themenkomplex der Datenfusion.

5.1 Architekturen zur Datenfusion

Arbeiten im Bereich der Datenfusion innerhalb von verteilten Sensor-Systemen wurden ursprünglich durch militärische Anwendungen motiviert. Insbesondere Fragestellungen der Objektüberwachung [TS 81,CV 86] wurden untersucht. Mittlerweile finden diese Techniken aber auch zusehends Anwendung in zivilen Projekten [KZK 97, Tenn 00]. Dazu zählen offensichtlich auch kontextadaptive Applikationen, wie sie im Rahmen dieser Arbeit behandelt werden.

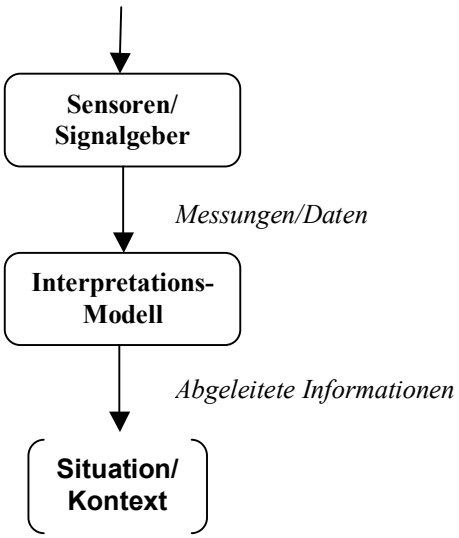

Abbildung 5-1: Der Prozess der Datenfusion im Überblick

Im Weiteren soll das Konzept der Datenfusion näher erläutert werden. Diesbezüglich werden zunächst einige Grundzüge der Datenfusion vermittelt. Im Anschluss erfolgt eine Diskussion von Architekturansätzen.

5.1.1 Datenfusion

Abbildung 5-1 zeigt den Prozess der Datenfusion im Überblick. Zunächst werden gemessene Zustandsdaten der Umwelt mittels Sensoren bzw. Signalgebern ermittelt. Diese Daten werden dann anhand eines *Interpretationsmodells* analysiert und bewertet. Die daraus abgeleiteten Informationen geben dann Auskunft über den Kontext oder die Situation des beobachteten Objektes bzw. der Umwelt. Angegliederte Applikationen können diese Informationen dann benutzen, um ihr Verhalten sitiuationsabhängig anzupassen. Die in einer Datenfusion eingesetzten

Sensoren und deren gemessene Quantitäten müssen immer in Bezug zur Applikation stehen. Ein Sensor wird als angemessen für einen bestimmten Datenfusions-Prozess bezeichnet, wenn die gemessene Quantität des Sensors in Bezug steht zur Situation oder zum Zustand der in Frage kommenden Umgebung. Die Relevanz dieser Bezogenheit ist auch abhängig davon, wie gut die gemessenen Daten interpretiert werden können. Oft sind diese Faktoren aufgrund unzulänglicher Kenntnisse über die Umwelt bzw. den Regelungsprinzipien von eingesetzten Signalgebern [MDW 94] nur unvollständig zu bestimmen.

Des weiteren sind gemessene Quantitäten inhärent mit Unsicherheiten behaftet, welche zeitlich variieren können. Messfehler können ebenfalls dazu führen, dass die ermittelten Daten eines Sensors unbrauchbar sind. In Kombination mit eventuellen Hardwareausfällen kann die Relevanz von gemessenen Sensordaten beträchtlich gemindert werden. Schließlich bewirken physikalische Einschränkungen eines einzelnen Sensors (z.B. Abstand zu einem beobachteten Objekt), dass mittels eines einzelnen Sensors nur partielle Informationen gewonnen werden können.

In Konsequenz der oben aufgeführten Einschränkungen kann festgestellt werden, dass für die meisten Anwendungen ein *einzelner* Sensor zur Bewerkstelligung einer Datenfusion nicht in Frage kommt. Diese Überlegung führte zu so genannten *Multi-Sensor-Systemen*, welche es erlauben, die Einschränkungen eines Ein-Sensor-Systems abzuschwächen. Dafür sind insbesondere folgende Eigenschaften eines Multi-Sensor-Systems verantwortlich:

- **Redundanz**: Redundanz bezeichnet den Einsatz einer oder mehrere Sensoren zur Beobachtung desselben oder überlappender Raumsegmente. Es ist hinreichend bekannt, dass Redundanz die Unsicherheit verringert, da die Messdaten mehrerer Sensoren miteinander korrelieren, während Störungen der Sensoren nicht miteinander korrelieren.
- **Kombination:** Die Kombination von mehreren verschiedenen Sensoren kann auch dazu dienen, die Qualität von Sensordaten zu verbessern. Indem z.B. für die Datenfusion bez. eines Raumsegments Sensoren an unterschiedlichen Orten zur Anwendung kommen. Die Sensoren brauchen dabei nicht vom gleichen Typ zu sein. Für einige Anwendungsfälle kann es von Nutzen sein, Sensoren, die unterschiedliche physikalische Größen messen, in einem System zu kombinieren [A 92].

In Anlehnung an die Aspekte *Redundanz* und *Kombination* kann die Datenfusion in einem Multi-Sensor System alternativ durch folgende Klassen der Sensorkonfiguration beschrieben werden [T 97]: *Complimentary Sensors*, *Competitive*

Sensors und *Cooperative Sensors*. Im Folgenden werden diese Klassen kurz erläutert:

- **Complimentary Sensors**: In diesem Fall sind die Sensordaten nicht direkt voneinander abhängig. Die Daten können aber kombiniert werden, um ein vollständigeres Bild der Umgebung zu gewinnen. So können beispielsweise verschiedene Radarstationen geographische Regionen überwachen, welche disjunkt voneinander sind.
- **Competitive Sensors**: Diese Konfiguration entspricht dem klassischen Konzept der Redundanz, wo identische Sensortypen parallel zum Einsatz kommen und äquivalente Daten liefern. Bei dieser Konfiguration ist vor allem das Problem des Konfliktes zu beachten, der eintritt, falls verschiedene Sensoren unterschiedliche Messdaten ermitteln (*conflicting readings*).
- **Cooperative Sensors**: Dieser Fall beschreibt den Fall, in der eine Situation nur mittels der Kooperation unterschiedlicher Sensoren zu bestimmen ist. Es wäre also nicht möglich, dieselbe Information mit Hilfe eines einzelnen Sensors abzuleiten.

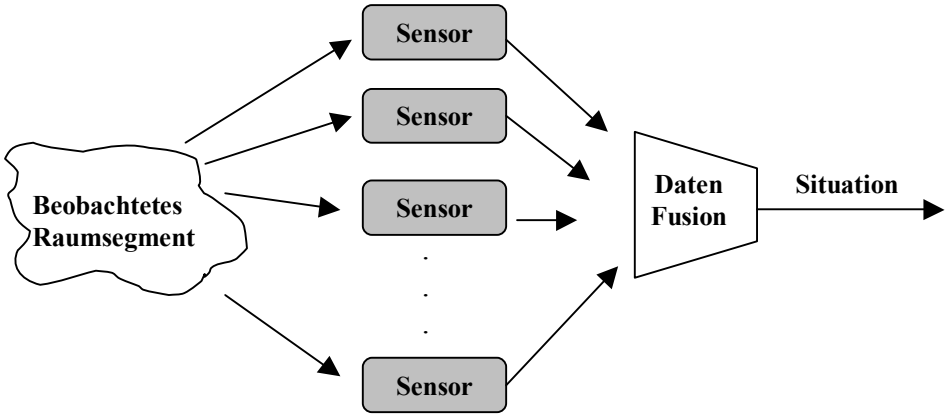

Abbildung 5-2: Datenfusion in einem Multi-Sensor System

Insbesondere der Fall *Cooperative Sensors* wird im Rahmen dieser Arbeit zur Ermittlung von Kontextinformationen behandelt.

Abbildung 5-2 veranschaulicht den Prozess der Datenfusion innerhalb eines Multi-Sensor Systems. Die Bestimmung einer Situation bez. eines beobachteten Raumsegements erfolgt über die Integration multipler Sensordaten.

Bisher ist der Prozess der Datenfusion anhand einzelner Phasen der Verarbeitung vorgestellt worden. Dazu gehören zunächst die Wahl geeigneter Sensoren und

deren Positionierung. Dann folgt der eigentliche Messvorgang, die Interpretation und die Weiterleitung zu entsprechenden Applikationen. Abschließend soll hier noch eine generelle Definition für die Datenfusion in Multi-Sensor Systemen gegeben werden [A 92]:

> *Datenfusion behandelt die synergetische Kombination von Informationen diverser Informationsquellen, wie etwa Sensoren, mit der Absicht, ein besseres Verständnis von der beobachteten Szenerie oder Umgebung zu gewinnen.*

5.1.2 Architekturansätze

Das Konzept der Datenfusion in Multi-Sensor Systemen erfordert Ansätze zur Organisation, d.h. wie werden Sensoren in einem System miteinander vernetzt, wie kommunizieren diese miteinander und soll der Prozess der Datenfusion zentral oder dezentral erfolgen. Diese Gesichtspunkte müssen bez. der Anwendung und der gegebenen Systemumgebung untersucht werden und entsprechend der zu erzielenden Performanz bzw. Zuverlässigkeit ausgewählt werden.

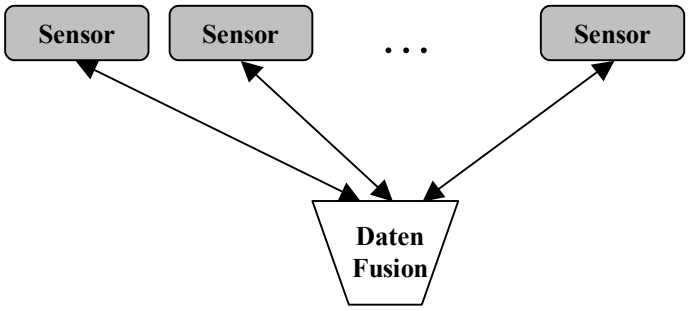

Abbildung 5-3: Zentrale Datenfusion

Laut [MDW 94] unterscheidet man zwischen folgenden Architekturansätzen: *zentral*, *hierarchisch* und *dezentral*. Jeder dieser Ansätze ist gekennzeichnet durch ein spezifisches Anwendungsprofil und kann so je nach Applikation entsprechend gewählt werden. Zudem ist zu beachten, dass die jeweiligen Architekturansätze entstanden sind, um spezifische Nachteile vorhergehender Ansätze zu beheben.

In einer *zentralen* Architektur ist eine zentrale Instanz zuständig für die Kollektion, die Verarbeitung und Interpretation von ermittelten Sensordaten.

Ein Vorzug des *zentralen* Architekturansatzes ist, dass Algorithmen zur Datenfusion, welche ursprünglich für Einzel-Sensor Systeme entwickelt wurden und als solche weniger komplex sind, konzeptionell auch in diesem Fall angewendet werden können. Da der zentrale Prozessor immer Zugriff auf alle eingehenden Sensordaten hat, lässt sich die Verwaltung eines solchen Systems

effektiv gestalten, etwa um fehlerhafte Daten eines defekten Sensors im Rahmen einer Validierung auszuschließen.

Neben diesen positiven Aspekten hat diese Architektur den Nachteil der Inflexibilität. Änderungen einer existierenden Applikation lassen sich wenn überhaupt nur umständlich realisieren. Hinzu kommen Probleme, die für zentrale Architekturansätze kennzeichnend sind, so etwa dass der zentrale Prozessor einen Flaschenhals bildet und dass das System bei Ausfall dieser Instanz nicht mehr einsetzbar ist. Aus diesem Grund verfügen solche Systeme oft über einen so genannten Backup-Prozessor. Sollte der Haupt-Prozessor ausfallen, werden alle Aufgaben auf den Backup-Prozessor transferiert.

Hierarchische Architekturen beheben einige Schwächen zentraler Systeme, indem der Verarbeitungsprozess der Datenfusion auf mehrere Instanzen verteilt wird; siehe auch Abbildung 5-4.

In dieser Architektur werden die Daten schrittweise bearbeitet und an die nächsthöhere Ebene weitergeleitet. Schließlich fließen die bereits vorbearbeiteten Daten in einer zentralen Instanz zusammen. Dort wird der Prozess der Datenfusion abgeschlossen.

Obwohl dieser Architekturansatz einige Schwächen zentraler Ansätze beseitigt, können beim hierarchischen Ansatz einige Nachteile beobachtet werden, die auch für zentrale Ansätze kennzeichnend sind. Hinzu kommen folgende Nachteile, die in dem hierarchischen Aufbau begründet liegen:

- Die Architektur erfordert zwei unterschiedliche Algorithmentypen. Einer realisiert die lokale Vorbearbeitung, der andere die globale Fusion.
- In Abhängigkeit davon, wie der Datenfusionsprozess implementiert wurde, kann eine bidirektionale Kommunikation zwischen den einzelnen Ebenen erforderlich sein. Infolgedessen könnte es zu einem Flaschenhals bez. der Kommunikation kommen.

Dezentrale Architekturen können einige Probleme zentraler und hierarchischer Ansätze beheben. Innerhalb einer dezentralen Architektur kann jeder Sensor seine Messungen an andere Sensoren[10] weitergeben. Zudem kann jedes Sensorelement eigene oder Daten anderer Sensoren weiterverarbeiten. Eine dezentrale Architektur vereint alle Vorteile einer hierarchischen Architektur, ohne ein zentrales Rechenelement zu benötigen. Folgende Eigenschaften kennzeichnen eine dezentrale Architektur zur Datenfusion; siehe auch Abbildung 5-5:

- Die Performanz des Gesamtsystems ist nicht abhängig von einem individuellen Prozessor.

[10] Jeder Sensor verfügt über eigene Rechenkapazität und Kommunikationsmöglichkeiten.

- Die Performanz des Systems wird im Falle von Sensorausfällen nur graduell abnehmen, da der Ausfall eines einzelnen Sensors nicht zum Ausfall des gesamten Systems führt.
- Änderungen bez. der eingesetzten Sensortechnologie sind einfach zu implementieren, da solche Änderungen nur einen einzelnen Knoten betreffen.
- Aspekte der Kommunikation und Algorithmik werden in einer dezentralen Architektur zunehmend komplexer.

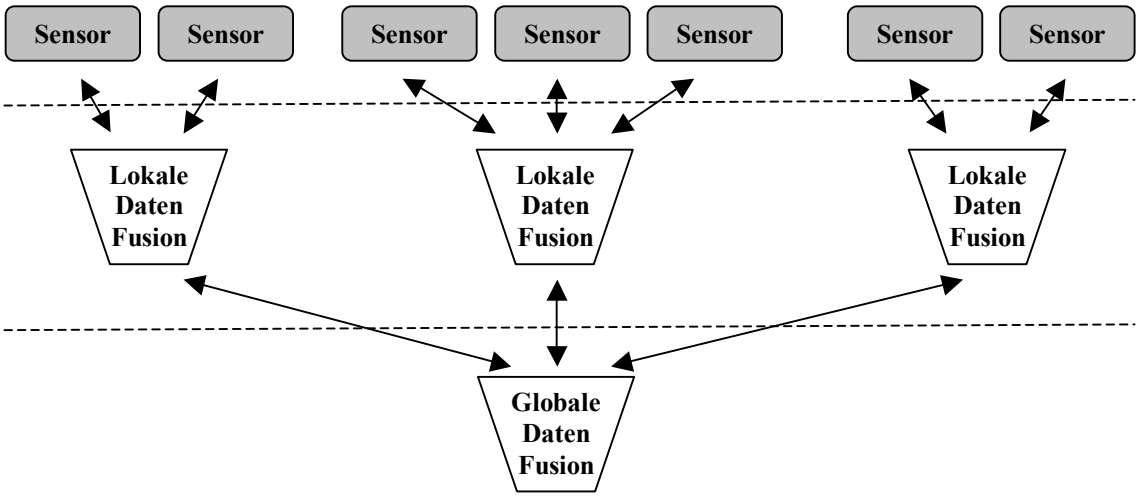

Abbildung 5-4: Hierarchische Architektur

Es ist zu beachten, dass der Kommunikationsaufwand in einem dezentralen System zur Datenfusion größer ist als für ein vergleichbares zentrales System, jedoch geringer als in einem hierarchischen System (vgl. [MDW 95]).

Wie in Abbildung 5-5 zu sehen, unterscheidet man dezentrale Architekturen bez. ihrer Vernetzung: *vollverbunden* und *partiellverbunden*.

Die Entwicklung eines dezentralen Systems, welches nur *partiellverbunden* ist, ist mit einigen zusätzlichen Problemen verbunden. So müssen unter anderem Fragen des Routings für diesen Fall gesondert behandelt werden. Auch die Implementierung des Datenfusions-Prozesses selbst muss an die Gegebenheiten der dezentralen Topologie angepasst werden.

In den folgenden Abschnitten wird die Datenfusion in drahtlosen Ad-hoc-Netzwerken untersucht, um die Gewinnung von Kontextinformationen zu realisieren. Diese Netze bilden ein dezentrales System, welches nach obiger Klassifikation als partiellverbunden zu bezeichnen ist. Hinzu kommen jedoch noch Aspekte, die für Netze dieser Art kennzeichnend sind. So kann sich während eines

Datenfusions-Prozesses die Topologie eines solchen Netzes ändern (etwa durch Ausfall oder vorübergehende Nicht-Erreichbarkeit eines Netz-Knotens). Diese Randbedingungen müssen in das Design der entsprechenden Datenfusions- und Routing-Algorithmen einfließen.

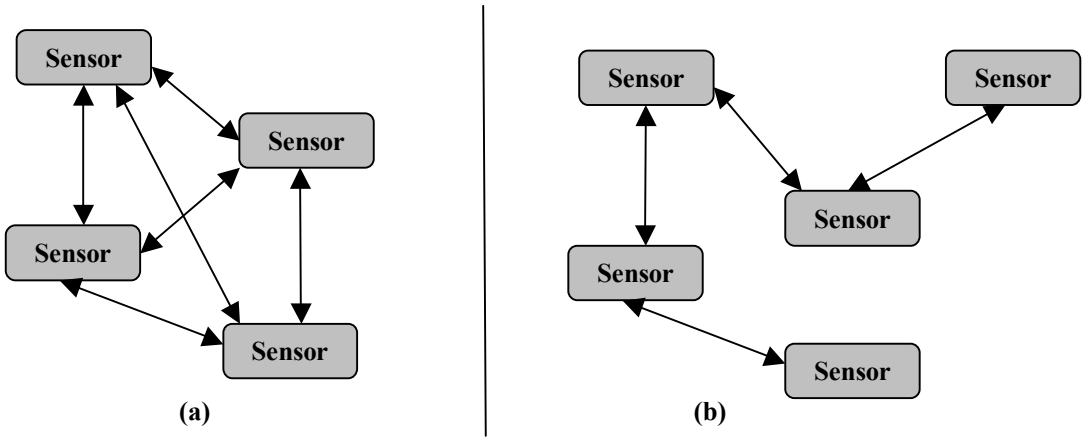

Abbildung 5-5: Dezentrale Architektur. (a) vollverbunden (b) partiellverbunden

5.2 Datenfusion in Ad-hoc-Sensornetzwerken

Im Folgenden wird der Prozess der Datenfusion zur Gewinnung von Kontextinformationen in Ad-hoc-Sensornetzwerken untersucht.

5.2.1 Architektur eines Sensorknotens

Innerhalb eines Sensornetzwerkes bezieht sich die Architektur von Sensorknoten jeweils auf die in Frage kommende Anwendung. In einem einzelnen Sensornetzwerk können zudem Sensorknoten mit abweichenden Charakteristika zum Einsatz kommen. So kann es in einem Sensornetzwerk Knoten mit einer übergeordneten Funktion geben, welche z.B. die gewonnenen Daten eines Sensornetzwerks über ein drahtloses Weitverkehrsnetz verbreiten oder auch Daten der anderen Sensorknoten archivieren. Diese Knoten müssen dann mit entsprechenden Eigenschaften ausgerüstet sein (mehr Speicher, höhere CPU-Leistung).

Da der Aufbau eines Sensorknotens sowohl von der Anwendung als auch von seiner Funktion im Sensornetzwerk bestimmt wird, können nur Aussagen über die generelle Architektur getroffen werden. Abbildung 5-6 zeigt den grundsätzlichen Aufbau eines Sensorknotens in einer Übersicht. Die Komponente Sensor realisiert den eigentlichen Messvorgang, der angeschlossene AC/DC Wandler wandelt die analogen Messwerte des Sensors in digitale Messdaten. Diese können daraufhin

mittels der CPU verarbeitet, gespeichert oder via eines Netzwerkmoduls zu anderen Sensorknoten übermittelt werden. Sensorknoten verfügen in der Regel über ein autonomes Energiesystem (vgl. 2.2.3). Als Energiequelle dient meist eine Batterie oder ein wiederaufladbarer Akku.

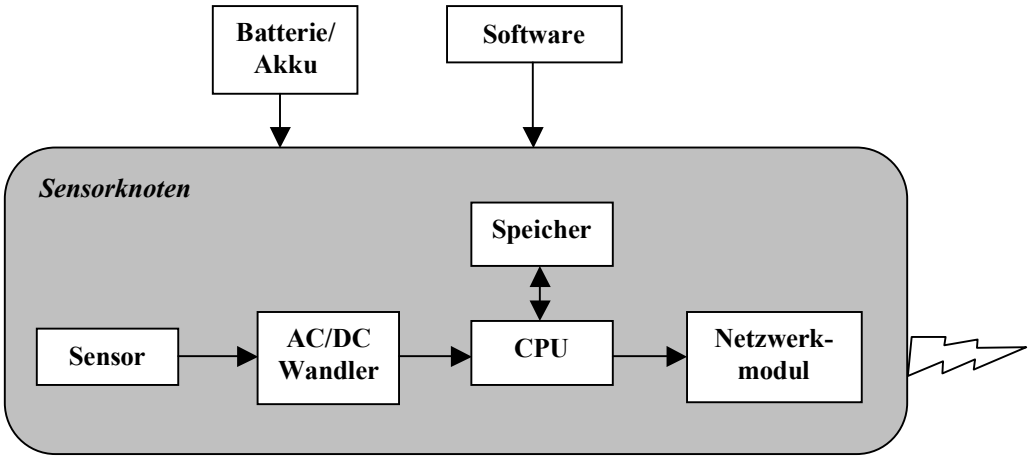

Abbildung 5-6: Architektur eines Sensorknotens

Es existieren auch Knoten, welche nicht über eine Sensor-Komponente verfügen, sondern lediglich über Rechenressourcen und Kommunikationsmöglichkeiten. Diese Knoten können insbesondere eingesetzt werden, um die Daten anderer Sensorknoten weiterzuverarbeiten. Diese Knoten werden im Rahmen dieser Arbeit als *Fusions-Knoten* bezeichnet.

5.2.2 Sensornetzwerke

Sensornetzwerke bilden eine neue Familie von drahtlosen Netzwerken [SGAP 00] und unterscheiden sich wesentlich von traditionellen Netzwerktypen, wie etwa zellulären Netzwerken oder *Mobile Ad-hoc-Networks* (MANETs). In traditionellen Netzwerken werden Aspekte der Organisation, des Routings und des Mobilitätsmanagements bez. Kriterien des Quality of Service (QoS) und der Erreichung hoher Bandbreiten ausgebildet. Zusammenfassend kann man feststellen, dass diese Netzwerktypen mit der Zielsetzung entwickelt wurden, einen hohen Datendurchsatz zu gewährleisten unter Berücksichtigung von Mobilitätseinflüssen (z.B. unterbrochenen Verbindungen). Der effiziente Einsatz von Energiereserven ist beim Entwurf dieser Netze nur von sekundärer Bedeutung, da die entsprechenden Energieressourcen (z.B. Batterien) zu jeder Zeit ausgewechselt werden können. Sensornetzwerke werden u.U. jedoch aus Hunderten bis Tausenden autonomen Netzwerkknoten gebildet, so dass die frequente Auswechslung von Energieaggregaten keine annehmbare Lösung darstellt. Weiterhin ist zu beachten, dass sich auch das Datenaufkommen in Sensornetzwerken von dem traditioneller Netzwerke unterscheidet. Während man in MANETs und

zellulären Netzwerken von einem hohen Datenaufkommen ausgeht, zeichnen sich Sensornetzwerke durch ein statistisches Datenaufkommen (*data bursts)* aus, deren Datenrate sich in einer Größenordnung von 1-100 kb/s bewegt. In Sensornetzwerken erfolgt der Datenverkehr vorwiegend unidirektional, also von multiplen Sensorknoten zu einer Senke.

Kennzeichnende Eigenschaften

Ein Sensornetzwerk lässt sich nach [EE 01] durch folgende Eigenschaften bzw. Ansprüche charakterisieren:

- Selbstorganisation der Vernetzungsstruktur in Abhängigkeit von der räumlichen Verteilung von Sensorknoten (*Ad-hoc-deployment*).
- Dynamische Adaption bez. Variationen des Netzwerkes (Ausfall von Knoten oder Nicht-Erreichbarkeit).
- Die autonome Operation eines Sensornetzwerkes erfordert Mechanismen zur automatischen Rekonfiguration.
- Minimierung von Kommunikationsvorgängen bzw. Schonung von Energiereserven.

Um den geforderten Eigenschaften zu entsprechen, müssen u.a. folgende Entwurfskriterien berücksichtigt werden: Da der Energieverbrauch gering gehalten werden soll, ist es erforderlich, energieeffiziente Algorithmen einzusetzen, welche insbesondere die Anzahl der erforderlichen Kommunikationsvorgänge reduzieren. Auch die Signalreichweite des Netzwerkmoduls eines Sensorknotens sollte möglichst gering sein, um den anfallenden Energieverbrauch zu verkleinern. Diese Entwurfsentscheidung mündet in einem so genannten *Multihop-Datenübertragungsschema* oder einem nachbarschaftlichen Schema. Insbesondere wird auch die lokale Vorverarbeitung von gemessenen Daten motiviert, da so das zu übertragende Datenvolumen reduziert werden kann. In diesem Zusammenhang soll folgendes Beispiel angeführt werden [SGAP 00]: Man benötigt eine Energie von 3 Joule, um ein Datenvolumen von 1 Kb über eine Strecke von 100 m zu übertragen. Eine gewöhnliche CPU mit einer Rechenleistung von 100 MIPS würde für die Ausführung von 300 Millionen Instruktionen etwa den gleichen Energiebedarf beanspruchen.

5.2.3 Kontextanfragen

Die Gewinnung von Kontextinformationen erfolgt in einem Prozess der Datenfusion, der durch eine *Kontextanfrage* initiiert wird (siehe Abbildung 5-7); eine Kontextanfrage spezifiziert die geforderten Kontexttypen (vgl. 2.3) einer

Anwendung. Im Rahmen der hier erörterten Konzeption sollen sowohl *Pull-* als auch *Push-Modi* Berücksichtigung finden [MS 01]. Das bedeutet, dass Kontextanfragen einerseits direkt durch einen Benutzer bzw. durch einen Benutzeragenten veranlasst werden können (*pull*) und andererseits auch von einem Sensorknoten ausgehen können (*push*). Im letzteren Fall ist die Initiierung einer Kontextanfrage an eine Bedingung geknüpft. So könnte beispielsweise eine Bedingung bez. der ermittelten Messdaten eines Sensors formuliert werden: „Wenn ein gemessenes Signal einen bestimmten Wert überschreitet, dann beginne die Bearbeitung der Kontextanfrage und sende die ermittelten Kontextinformationen zum Benutzer“.

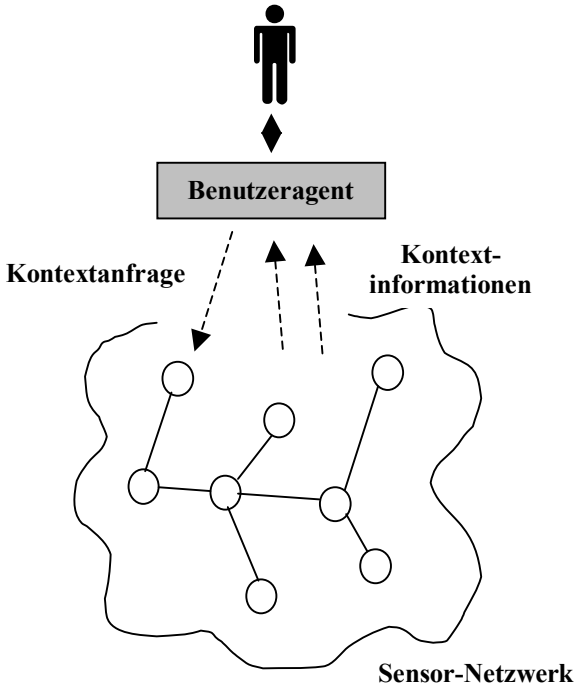

Abbildung 5-7: Kontextanfragen und Kontextinformationen

Das hier vorgestellte Verfahren zur Datenfusion geht von der Annahme aus, dass sich das betrachtete Sensornetzwerk durch eine partielle Verbindungsstruktur auszeichnet. Insbesondere wird von einer nachbarschaftlichen Vernetzung ausgegangen, d.h. Sensoren kommunizieren ausschließlich über einen Broadcast-Kanal mit allen erreichbaren Knoten. Die Erreichbarkeit eines Sensorknotens wird hauptsächlich durch die Entfernung bestimmt; in typischen Sensornetzwerken liegt die Signalreichweite zwischen 10 cm und 10 m.

Ein naiver Ansatz zur Auswertung von Kontextanfragen basiert auf dem Prinzip des *Flooding* [HKB 99]. Ähnlich dem Schneeballverfahren wird die Anfrage sukzessive an alle Knoten weitergeleitet, d.h. ein Knoten empfängt eine Anfrage,

sendet sie an alle erreichbaren Nachbarn und entsprechend wird das Verfahren fortgesetzt. Offensichtlich sollte dieses Verfahren möglichst vermieden werden, da es die ohnehin schon als beschränkt angenommenen Ressourcen eines Sensornetzwerkes stark beansprucht. Dennoch lassen sich das Prinzip des *Floodings* oder ähnliche nicht-deterministische Verfahren nicht immer vermeiden, da die Topologie eines Sensornetzwerkes zunächst nicht bekannt ist bzw. dynamischen Fluktuationen unterliegt. Ein initiales oder eingeschränktes Flooding könnte erforderlich sein, um zunächst Informationen über die Topologie eines Ad-hoc-Sensornetzwerkes zu gewinnen. Diese Informationen können dann genutzt werden, um die Bearbeitung zukünftiger Anfragen zu optimieren.

Der hier verfolgte Ansatz gewinnt Informationen über die Netzwerktopologie mittels Pfadaufzeichnungen der in das Netz gesendeten Pakete. Im Unterschied zu vielen anderen Ansätzen verlangt das hier beschriebene Verfahren nicht den stetigen Unterhalt von Routing-Tabellen seitens der Sensorknoten. Es handelt sich im Wesentlichen um ein zustandsloses Verfahren (*stateless protocol*).

5.3 Smart Context-Aware Packets

Das präsentierte Konzept zur Datenfusion in Ad-hoc-Sensornetzwerken basiert auf der Idee des *Smart Context-Aware Packets* (SCAPs). Es dient sowohl als uniformes Datenaustauschformat zwischen Sensorknoten als auch zur Koordination von Prozessen der verteilten Datenfusion. In den folgenden Abschnitten soll diese Konzeption erläutert werden. Ausgehend von der Idee des SCAPs wird in den nachfolgenden Abschnitten das *Smart Stack Routing* (SSR) Protokoll entwickelt.

5.3.1 SCAPs: Struktur

Nun soll die Struktur eines *Smart Context-Aware Packet* (SCAP) näher erläutert werden. Ein SCAP wird untergliedert in drei unterschiedliche Sektionen: *Retrieving Plan*, *Context Hypothesis* und *Packet Trace*. Der *Retrieving Plan* definiert einen Datenpfad (vgl. 3.2.1) innerhalb eines Sensornetzwerkes. Dieser Datenpfad wird definiert durch eine Sequenz von Sensortypen, deren Meßwerte zur Bestimmung einer bestimmten Kontextinformation zu ermitteln sind. Der *Retrieving Plan* kann zudem auch Regeln enthalten, die es ermöglichen, den assoziierten Datenpfad an die aktuell gemessenen Sensordaten anzupassen. Eine nähere Erläuterung des *Retrieving Plans* erfolgt im nachfolgenden Abschnitt.

Wie schon erwähn,t bezeichnet die *Context Hypothese* die auf einem Datenpfad bereits akkumulierten Sensordaten bzw. die davon abgeleiteten Kontextinformationen. Alle Sensordaten, die dem im *Retrieving Plan* spezifizierten Typen entsprechen, werden dem SCAP hinzugefügt. Fusions-Knoten können die Daten mehrerer Sensorknoten verarbeiten, um so abgeleitete Kontextinformationen zu gewinnen.

Die einzelnen Messwerte der Sensoren werden innerhalb der *Context Hypothesis* durch Tupel repräsentiert; siehe Abbildung 5-9.

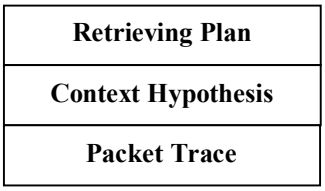

Abbildung 5-8: Smart Context-Aware Packet (SCAP)

Feature ID bezeichnet den Messdatentyp eines Sensors. Es kann sich beispielsweise um Lautstärke- oder Temperaturwerte handeln. *Feature Value* bezeichnet die Messdaten eines akkumulierten Eintrages. *Sensor Type ID* determiniert den Typ eines Sensorknotens. Der Typ bezieht sich nicht nur auf die ermittelten Messdaten, sondern auch auf die Ausstattung des Sensorknotens. *Sensor ID* dient der eindeutigen Identifikation eines Sensors. *Sensor Location* und *Timestamp* geben Auskunft darüber, an welchem Ort bzw. zu welcher Zeit ein bestimmter Messwert gewonnen wurde. Diese Angaben können für die weitere Verarbeitung der Daten, der Ableitung höherer Informationen, notwendig sein.

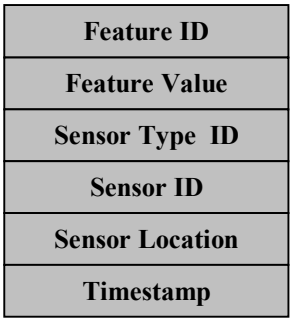

Abbildung 5-9: Repräsentation von Messdaten und Kontextinformationen

Packet Trace bildet die dritte Sektion eines SCAPs und dient der Verwaltung von Informationen bez. des Pfades eines SCAPs durch ein Sensornetzwerk. *Packet Trace* gliedert sich in zwei separate Datenstrukturen, die als *Stacks* bezeichnet werden. Der erste Stack repräsentiert zu jedem Zeitpunkt den bereits zurückgelegten Pfad eines SCAPs. Zu diesem wird die *Sensor ID* jedes besuchten Sensorknotens hinzugefügt. Komplementär dazu wird der zweite Stack benötigt, um das SCAP entsprechend eines vordefinierten Pfades durch das Netz zu routen. Demzufolge werden die zu besuchenden Knoten innerhalb dieses Stacks angezeigt. Ist dieser Stack bei einer initialen Kontextanfrage noch nicht besetzt

muss der Pfad noch mittels Flooding ermittelt werden und kann dann für nachfolgende Kontextanfragen eingesetzt werden.

Die in den Stacks gespeicherten konkreten Pfadinformationen geben Auskunft über die gegenwärtige Topologie eines Sensornetzwerkes. Mittels dieser Informationen kann vermieden werden, das Knoten mehrfach besucht werden und somit Schleifen entstehen. Informationen des Stacks beziehen sich jeweils auf eine bestimmte Kontextanfrage und deren Sub-Kontextanfragen. Da die Topologie eines Sensornetzwerkes zeitlich variiert, sind auch Pfadinformationen nur zeitlich begrenzt gültig.

Neben Pfadinformationen können Stacks auch zusätzliche Attribute enthalten. So können z.B. Angaben über die Energiereserven einzelner Knoten mit aufgezeichnet werden. Dann ist es möglich, die Evaluierung von Kontextanfragen nicht nur nach der Pfadlänge, sondern auch nach Energiekriterien zu optimieren.

Das Konzept des *Packet Trace* basiert auf dem in Abschnitt 5.4.1 präsentierten *Stack Routing Algorithmus* [WOS].

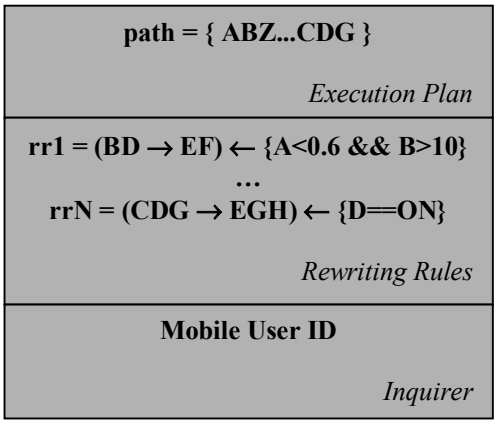

Abbildung 5-10: Struktur des *Retrieving Plans*

Retrieving Plan

Dieser Abschnitt erläutert die Semantik und Repräsentation des *Retrieving Plans*. Wie bereits erwähnt, definiert der *Retrieving Plan* eine Sequenz von Sensor-Typen, die zur Evaluierung einer Kontextanfrage zu besuchen sind. Der *Retrieving Plan* besteht aus drei Informationseinheiten (siehe auch Abbildung 5-10): *Execution Plan*, *Rewriting Rules* und *Inquirer*.

Der *Execution Plan* beschreibt einen abstrakten Datenpfad, der durch eine Sequenz von zu besuchenden Sensor-Typen definiert wird. *Rewriting Rules* können eingesetzt werden, um den durch den *Execution Plan* definierten Datenpfad an aktuell gemessene Sensordaten anzupassen. Die Anpassung des Pfades

kann pro besuchten Sensorknoten erfolgen. Die Semantik einer *Rewriting Rule* wird an einem Beispiel erläutert (vereinfachte Darstellung):

$$rr1 = (BD \rightarrow EF) \leftarrow \{ A<0.6 \;\&\&\; B>10 \}$$

{A<0.6 && B>10) definiert die Bedingung, unter der der Teilpfad BD durch den Teilpfad EF ersetzt wird. A und B bezeichnen hier Messwerte von Sensoren der Typen A und B. Konzepte der Termersetzung [BN 99] können zur Evaluation von *Rewriting Rules* angewendet werden. Zudem können auch Aussagen über die Korrektheit bzw. Termination eines *Retrieving Plans* getroffen werden.

Rewriting Rules werden auch eingesetzt um eine sensor-initiierte Kontextanfrage auszulösen (siehe 5.3.2).

Schließlich bezeichnet die Sektion *Inquirer* die Instanz, an welche die Ergebnisse einer Kontextanfrage geliefert werden sollen.

5.3.2 Evaluation von Kontextanfragen

Im Folgenden sollen Grundmechanismen der Datenfusion mittels SCAPs erläutert werden. Wie bereits erwähnt, wird die Gewinnung von Kontextinformationen als eine Menge von Kontextanfragen verstanden; im Rahmen der hier erläuterten Konzeption werden Kontextanfragen durch SCAPs repräsentiert. SCAPs erlauben die Formulierung von Kontextanfragen bez. der Angabe einer Sequenz von zu besuchenden Sensor-Typen. Diese Sequenz bildet in Analogie zu 3.2.1 einen Datenpfad.

Zunächst wird das SCAP an einen Knoten des Sensornetzwerkes gesendet. Falls dieser Knoten zum definierten Datenpfad gehört, werden die entsprechenden Sensordaten dem SCAP hinzugefügt (*Context Hypothesis*). Dann sendet dieser Knoten jeweils eine Kopie des SCAPs an alle Nachbarn. Dieses Verfahren wird kontinuierlich fortgesetzt, und jeder der besuchten Sensorknoten kann aus den bereits akkumulierten Sensordaten Schlussfolgerungen über den gegenwärtigen Kontext ableiten. In Übereinstimmung mit den im SCAP spezifizierten Regeln (*Rewriting Rules*) kann der Prozess der Datenfusion sukzessive fortgesetzt werden, indem das SCAP an Knoten weitergeleitet wird, welche die bisher abgeleiteten Kontextinformationen festigen (*Redundanz*) bzw. erweitern. Zusammenfassend kann festgestellt werden, dass der Prozess der Konstruktion von Kontext-informationen sich durch einen Prozess der kontinuierlichen Neuberechnung auszeichnet. Der resultierende Kontext wird schrittweise verfeinert. Sobald angeforderte Kontextinformationen vorliegen, werden sie zurück an den anfragenden Knoten/Benutzeragent (*User Agent*) gesendet.

Zur Verarbeitung von SCAPs wird vorausgesetzt, dass Sensorknoten über entsprechende Rechenleistung verfügen, welche die Verarbeitung von SCAPs erlauben (siehe 5.2.1). Abbildung 5-11 zeigt ein Beispiel.

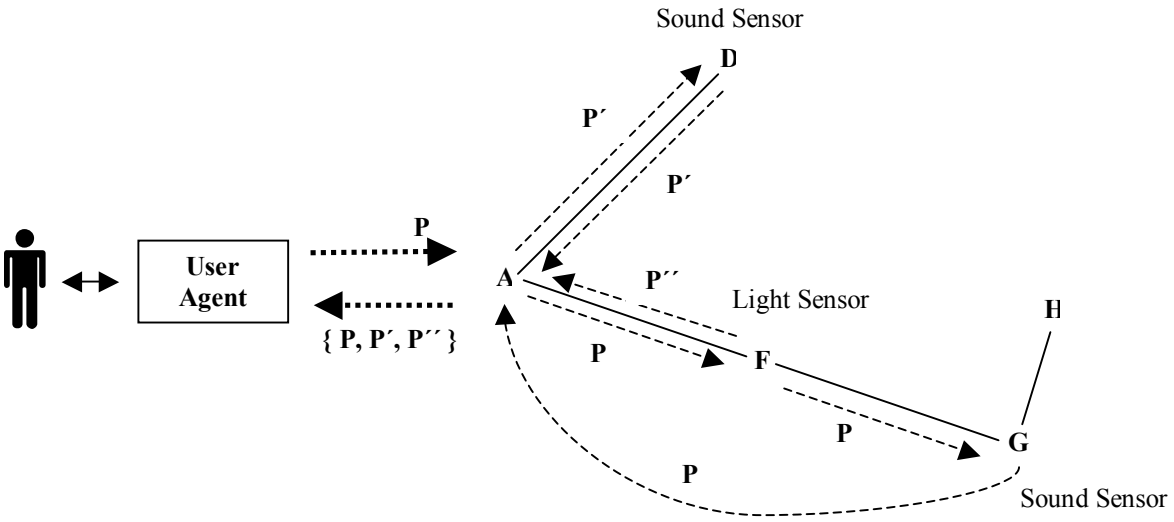

Abbildung 5-11: Anwendungsbeispiel für SCAPs

Der Benutzer ist interessiert an kontextuellen Informationen bez. des gemessenen Lichteinfalls und der Lautstärke. Diesbezüglich wird ein SCAP P mit entsprechenden Anforderungen in das umgebende Sensornetzwerk gesendet. Die durchgezogenen Linien zeigen die Verbindungsstruktur des Netzwerkes an. Zunächst wird das Paket von Knoten A empfangen. Da A weder über Licht noch über Akustiksensoren verfügt und somit den Anforderungen des SCAPs nicht genügt, wird das Paket geteilt: P wird an F gesendet und P' an D. Knoten D verfügt über einen Akustiksensor. Die aktuell gemessenen Daten werden eingelesen, dem Paket hinzugefügt und schließlich an den Benutzeragenten zurückgeschickt. Ähnlich ist die Situation bei Knoten F; hier können dem ankommenden Paket Sensordaten bez. des Lichteinfalls hinzugefügt werden. In diesem Fall wird eine Kopie des SCAPs (P´´) zurück zum Sender geschickt. Das ursprüngliche Paket P wird an Knoten G weitergeleitet. Knoten G verfügt über Messdaten bez. der Lautstärke; auch hier werden die Daten dem SCAP hinzugefügt, dann wird Paket P zurück zum Benutzer gesendet.

In Konsequenz empfängt der Benutzer drei Antworten auf seine Anfrage:

1. Paket P: verfügt über Messdaten des Lichteinfalls und der Lautstärke
2. Paket P´: Messdaten bez. der Lautstärke
3. Paket P´´: Messdaten bez. des Lichteinfalls

Da alle drei Pakete Informationen bez. des zurückgelegten Pfades liefern, können zukünftige Anfragen an die Knoten D, G und F direkt ausgeführt werden. In Netzen mit einer größeren Anzahl von Knoten ist davon auszugehen, dass zur Gewinnung bestimmter Kontextinformationen ein entsprechend größeres

Repertoire an möglichen Pfaden zur Verfügung steht. Die Auswahl eines Pfades kann dann beispielsweise durch folgende Kriterien bestimmt werden: durch Bestimmung des kürzesten Pfades. Auf diese Weise kann die Antwortzeit minimiert werden, und da nur wenige Kommunikationsvorgänge erforderlich sind, können die Energieressourcen des Netzwerkes geschont werden. Indes hat dieses Auswahlkriterium den Nachteil, dass die Energiereserven des Netzes u.U. einseitig verbraucht werden. Ein weiteres Auswahlkriterium sieht deshalb vor, zunächst eine Menge von möglichst kurzen, sich nicht überschneidenden Pfaden zu bestimmen. Diese werden dann in konsekutiven Kontextanfragen gleichmäßig verteilt genutzt. Mit diesem Schema kann eine gleichmäßige Auslastung (insbesondere Energieverbrauch) des Netzwerkes erreicht werden.

Die Verwaltung der in den SCAPs abgelegten Pfadinformationen (*Packet Traces*) erfolgt mittels *Smart Stack Routing* (SSR); siehe folgender Abschnitt.

Das angeführte Beispiel behandelt den Fall, dass die Anfrage von einem Benutzer-agent initiiert wird (*pull mode*). Komplementär dazu kann eine Kontextanfrage auch von einem Sensorknoten initiiert werden (*push mode*). Zur Umsetzung dieses Szenarios wird ein SCAP auf einem Knoten gespeichert. Die Ausführung dieses SCAPs ist an eine bestimmte Bedingung geknüpft. Hier ein Beispiel: Wird an einem Sensor A ein Messwert kleiner als 0.5 gemessen, so sollen auch Messwerte der Sensortypen B, C und D berücksichtigt werden. Dieses Verhalten kann durch folgende Vorgaben innerhalb des *Retrieving Plan*s (vgl. Abbildung 5-10) erreicht werden: *path* = ∅ und *rr1* = { ∅ → BCD} ← {A<0.5}. Die akkumulierten Daten werden an den Knoten zugestellt, der durch das *Inquirer* Feld bestimmt ist.

5.4 Smart Stack Routing

In diesem Abschnitt wird das *Smart Stack Routing* (SSR) [MSS 02] Protokoll basierend auf SCAPs vorgestellt. SSR implementiert die Datenfusion (vgl. 5.1) in hochdynamischen Sensor-Netzen mit einer nachbarschaftlichen Verbindungs-struktur. SSR basiert insbesondere auf dem in 5.4.1 vorgestellten *Stack Routing Protokoll* [WOS]. Demzufolge fließen die Ideen des *gerichteten* und *ungerichteten* Datenverkehrs des Stack Routing Protokolls mit ein. Auch das TTL-Feld, welches zur Begrenzung von Pfadlängen dient, findet Verwendung.

Zunächst wird SSR anhand von Erweiterungen bzw. Modifikation bez. des Stack Routing Protokoll eingeführt. Anschließend erfolgt eine Demonstration von SSR anhand eines einfachen Fallbeispiels.

5.4.1 Klassifikation von Routing-Protokollen

Nun sollen Routing-Protokolle untersucht werden, welche bez. des in diesem Abschnitt vorgestellten *Smart Stack Routing* (SSR) Protokolls relevant sind. Tabelle 5-1 klassifiziert die hier untersuchten Protokolle nach ihrem

Hauptanwendungsgebiet. Sie werden kategorisiert bez. der konstituierenden Netzwerknoten und der Vernetzungsstruktur. Zum einen wird unterschieden zwischen *nachbarschaftlich* und *vollvernetzten* Netzwerken. In einem vollvernetzten Netzwerk kann jeder Knoten direkt mit jedem anderen partizipierenden Knoten kommunizieren. Im Gegensatz dazu erlaubt eine nachbarschaftliche Verbindungsstruktur nur die Kommunikation mit Knoten in einem abgegrenzten Cluster. In dem hier betrachteten Anwendungsfall wird die Erreichbarkeit bzw. Nachbarschaft eines Knotens durch die Signalreichweite der eingesetzten Netzwerktechnologie bestimmt. Soll ein Kommunikationsvorgang zwischen Knoten verschiedener Nachbarschaften erfolgen, so muss zumindest ein intermediärer Knoten existieren, der beiden Nachbarschaften angehört.

	Statische Netzwerknoten	**Dynamische Netzwerknoten**
nachbarschaftlich vernetzt (everybody to subset)	Stack Routing Directed Diffusion	Smart Stack Routing
vollvernetzt (everybody to everybody)	Gnutella	Gnutella

Tabelle 5-1: Klassifikation von Routing-Verfahren

Zum anderen wird zwischen *statischen* und *dynamischen* Netzwerkknoten unterschieden. Statische Netzwerknoten bezeichnen Knoten, deren relative Position zum betrachteten Netzwerk fix ist; zudem wird angenommen, dass die Verfügbarkeit von Knoten hoch ist (wenige Verbindungsabbrüche, etc.). Dynamische Knoten zeichnen sich hingegen durch eine große Variabilität bez. der Verfügbarkeit aus. Insbesondere die hier betrachteten Ad-hoc-Sensornetzwerke zeichnen sich durch Knoten dieser Art aus.

Nachfolgend sollen die klassifizierten Protokolle kurz erläutert werden.

Gnutella

Gnutella [GNU] bezeichnet ein dezentrales P2P-Netzwerk [S 00], welches den Datenaustausch bzw. die Suche nach Daten implementiert. Insbesondere für Privatnutzer hat sich Gnutella in den letzten Jahren als Medium zum Austausch von Dateien und Informationen etabliert. Im Kontrast zum P2P-System Napster, welches insbesondere zur Distribution von MP3-kodierten Musikstücken verwendet wurde, verfügt Gnutella über keinerlei zentrale Instanz, die den Datenverkehr zwischen individuellen Peers koordinieren könnte. Zugang zu einem Gnutella-Netzwek erhält ein Nutzer durch eine spezielle Client-Software. Diese agiert – wie es für P2P charakteristisch ist – auch als Server, so dass angebotene Daten des Benutzers auch für andere Teilnehmer des Netzwerkes verfügbar und lokalisierbar sind. Bezüglich der oben angeführten Klassifikation von Routing-

Strategien kann Gnutella sowohl als Netzwerk mit *statischen* als auch *dynamischen* Netzwerknoten eingeordnet werden, da Klienten jederzeit das Netzwerk verlassen bzw. aus dem Netzwerk austreten können.

Bezüglich der Vernetzungsstruktur wird Gnutella als *vollvernetzt* eingestuft, einzelne Klienten werden über IP-Adressen identifiziert. Jeder Nutzer wird einem Cluster zugeordnet, der durch den *Time-to-Live* (TTL)-Parameter seiner initialen Suchanfrage determiniert wird. Jeder Klient verfügt über diese lokale Sicht. Werden aber Suchanfragen anderer Klienten akzeptiert und weitergeleitet, so entsteht eine vollvernetzte Netzwerkstruktur.

Im Kontrast zu drahtlosen Sensornetzwerken ist die Cluster-Struktur eines Gnutella-Netzwerkes eine Konsequenz aus Performanzüberlegungen bez. Suchoperationen und der Aufnahme neuer Klienten. Nach abgeschlossenen Suchvorgängen erfolgt die Kommunikation direkt (*peer-to-peer*), da das Netzwerk dann vollvernetzt ist.

Die Idee lokaler Routing-Tabellen, welche die letzten einkommenden Suchanfragen registrieren, wurde für das hier entwickelte Routing-Protokoll SSR berücksichtigt.

Stack Routing

Stack Routing [WOS] ist ein weiterer Ansatz eines P2P-Netzwerkes zum dezentralen Datenaustausch. Das Konzept des Stack Routings unterscheidet sich insbesondere von Gnutella, indem es im Sinne von 3.2.1 ein daten-zentrisches Protokoll definiert.

Innerhalb von Stack Routing wird der Datenverkehr in zwei Kategorien eingegliedert: *gerichtet* oder *ungerichtet*. *Ungerichteter Datenverkehr* bezieht sich nicht auf bestimmte Entitäten (die z.B. durch eine Adresse determiniert werden), sondern auf Entitäten, die vorgegebenen Anforderungen gerecht werden. Daher kann auch der Transport eines CAPs (vgl. 3.4) der Klasse des *Ungerichteten Datenverkehrs* zugeordnet werden. Im Kontrast dazu erfolgt der *Gerichtete Datenverkehr* nach einem konkreten Pfad, der die Folge der zu besuchenden Knoten beschreibt.

Stack Routing initiiert Suchanfragen mittels *Ungerichtetem Datenverkehr*. Dabei wird die besuchte Knotenfolge aufgezeichnet. Jeder Knoten ergänzt das Paket um seine Kennung. Zu diesem Zweck existiert eine spezielle Datenstruktur, die als *Stack* bezeichnet wird. Infolgedessen kann von jedem Knoten aus der initiierende Knoten einer Suchanfrage anhand des eingehenden Paketes bestimmt werden. Da durch dieses Verfahren ein sich zunehmend ausfächernder Suchbaum entsteht (in Abhängigkeit von der Verbindungsstruktur des Netzwerkes), ist es notwendig, die Ausbreitung zu begrenzen; siehe auch Abbildung 5-12. Dies geschieht durch *Time-to-Live* (TTL)-Felder in den jeweiligen Paketen, welche die maximale Länge

eines individuellen Pfades begrenzen. Wird eine Entität lokalisiert, die den Kriterien der Suchanfrage entspricht, so wird das Paket mittels der Methode des *Gerichteten Datenverkehrs* zurück zum Sender gesendet.

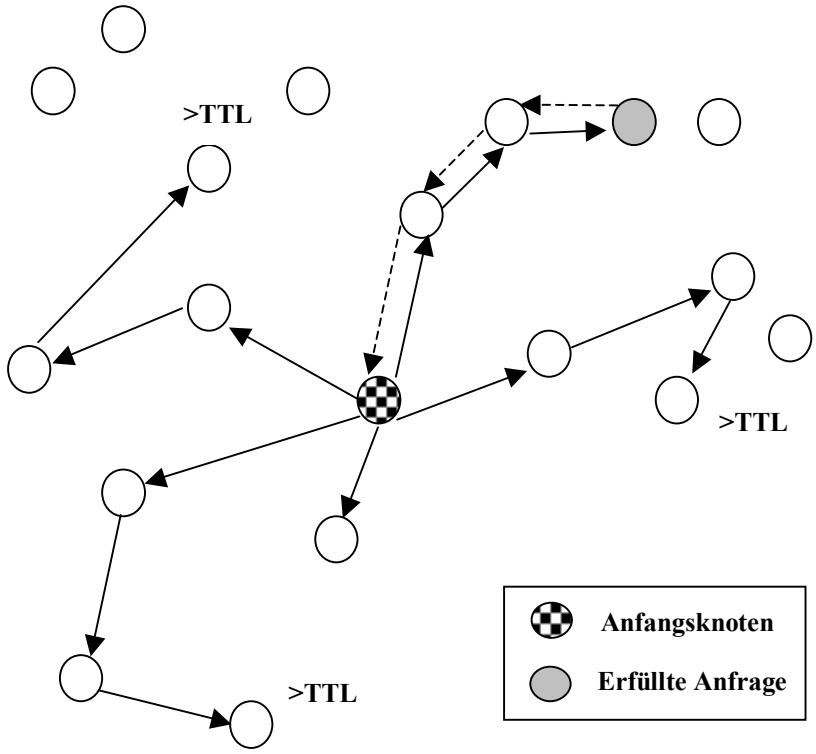

Abbildung 5-12: Stack Routing

Der Einsatz von zwei Stacks erlaubt die bidirektionale Kommunikation zwischen Knoten. Ein Stack wird – wie bereits beschrieben – benutzt, um den zurückgelegten Pfad aufzuzeichnen, dieser wird als *Push Stack* bezeichnet. Der zweite wird *Pop Stack* genannt. Dieser ist entweder leer, dann erfolgt die Kommunikation *ungerichtet*, oder er enthält den Rückpfad, dann erfolgt die Kommunikation *gerichtet.* Demzufolge führt das Vertauschen von *Pop* und *Push Stack* zur Richtungsumkehr des Paketes.

Obschon sich Stack Routing für Netze mit nachbarschaftlicher Verbindungsstruktur eignet, wird doch von *statischen* Netzwerknoten ausgegangen. Das heißt, verloren gegangene Netzwerkverbindungen bzw. ausfallende Knoten, wie sie für drahtlose Sensornetzwerke kennzeichnend sind, würden den erfolgreichen Einsatz von Stack Routing verhindern, da aufgezeichnete Pfadinformationen bei Ausfall eines Knotens ihre Gültigkeit verlieren würden.

Dennoch hatte das Verfahren des Stack Routings einen kennzeichnenden Einfluss auf das hier vorgestellte SSR Protokoll. Innerhalb von SSR werden auch Stacks

zur Speicherung von Pfadinformationen genutzt, aber das Konzept des *Gerichteten Datenverkehrs* wurde den Merkmalen eines drahtlosen Sensornetzwerkes angepasst (siehe 5.4).

Directed Diffusion

Directed Diffusion [EGHK 99] gehört derselben Klasse an, wie das im vorherigen Abschnitt beschriebene Stack Routing (vgl. Tabelle 5-1). Im Unterschied zu Stack Routing lässt sich Directed Diffusion auch in Sensornetzwerken mit einer hohen Anzahl von Sensorknoten einsetzen, da es entsprechende Skalierbarkeitsbedingungen erfüllt. Zudem erfordert der effektive Einsatz von Directed Diffusion einen stetigen Datenfluß.

Directed Diffusion ist ein daten-zentrisches Protokoll in der Hinsicht, dass jeder Sensor seine angebotenen Daten mittels einer Menge von Attributen deklariert, wonach andere Sensoren ihre Anfragen formulieren können. Anfragen werden durch das Netz propagiert und bilden Pfade. Sensoren, die der Anfrage genügen, schicken ihre Ergebnisse auf den inversen Pfad zurück zum anfragenden Sensor. Innerhalb von Directed Diffusion werden durch Anfragen Gradienten gebildet, welche eine gerichtete *Diffusion* von Daten ermöglichen. Mit anderen Worten konstruiert das Verfahren Directed Diffusion einen Pfad von einer Quelle zu einer Senke. Dabei ist zu beachten, dass die entsprechenden Anfragen periodisch wiederholt werden, um die Aktualität der assoziierten Gradienten zu gewährleisten.

Directed Diffusion passt sich auch Änderungen der Netzwerktopologie an, jedoch erfolgt die Reaktion zeitlich verzögert, da die im Netz verteilten Gradienten nur nach und nach korrigiert werden können. Aus diesem Grund eignet sich Directed Diffusion mehr für den kontinuierlichen Datentransport innerhalb von Sensornetzwerken, nicht aber für sporadische Anfragen.

Als nächstes werden die Basiskonzepte des hier vorgestellten Smart Stack Routing Protokolls erläutert. Im Anschluss erfolgt eine Demonstration des Protokolls anhand eines Fallbeispiels.

5.4.2 SSR Basiskonzepte

Der Einsatz von *Rewriting Rules* wird im Weiteren nicht berücksichtigt. Dennoch lassen sich die hier vorgestellten Konzepte auf diesen Anwendungsfall verallgemeinern. Da dieser aber keinen direkten Einfluss auf die Implementierung des SSR Protokolls hat, wird er hier nicht berücksichtigt.

Request ID

Request ID dient zur eindeutigen Identifikation einer individuellen Kontextanfrage. Der Hauptgrund für dieses zusätzliche Feld ist die Reduzierung

der im Umlauf befindlichen SCAPs. SCAPs mit derselben *Request ID* können sukzessive ausgefiltert werden. Die *Request ID* besteht aus zwei Anteilen: der Kennung des anfragenden Sensorknotens und einer durchgezählten Nummer (*Request Number)*; siehe auch Abbildung 5-13.

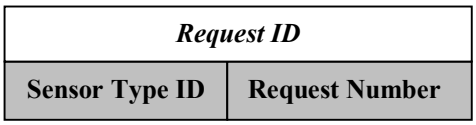

Abbildung 5-13: Request ID

Die *Request Number* dient der Unterscheidung konsekutiver Kontextanfragen eines einzelnen Knotens. Da die Lebenszeit eines SCAPs durch das TTL-Feld begrenzt ist, können bereits verwendete *Request Number* Einträge zu einem späteren Zeitpunkt wiederverwendet werden.

Die von einem SCAP induzierten Pakete werden als identisch bezeichnet, falls ihre *Request IDs* übereinstimmen.

Semi-gerichteter Datenverkehr

Zentral für die Konzeption des SSR-Protokolls ist die Idee des *Semi-gerichteten Datenverkehrs*, welche eine Synthese aus den Prinzipien des *Gerichteten-* und *Ungerichteten Datenverkehrs* bildet.

SSR ermöglicht das erfolgreiche Routen eines Paketes auch dann, wenn die benutzten Pfadinformationen ihre Gültigkeit verloren haben, z.B. wegen des Ausfalls eines Sensorknotens. Zu diesem Zweck gliedert sich der Routingprozess gemäß SSR in drei Phasen (siehe Abbildung 5-14): *Gerichteter Datenverkehr, Ungerichteter Datenverkehr* und *Semi-gerichteter Datenverkehr.*

Sollten während der Phase des *Gerichteten Datenverkehrs* die assoziierten Pfadinformationen nicht mehr gültig sein, so wechselt SSR in die Phase des *Semi-Gerichteten* Datenverkehrs (siehe Abbildung 5-15). Um anzuzeigen, dass sich SSR nun im *Semi-Gerichteten Modus* befindet, wird ein Markierungselement (S*) auf den Pop Stack gelegt. Nun wird mittels *Ungerichten Modus* ein Wiederaufsetzpunkt gesucht. Dieser Wiederaufsetzpunkt bezeichnet einen Knoten, der Teil des ursprünglichen Pfades ist. Zur Bestimmung des Wiederaufsetzpunktes wird bei jeder Bewegung der durch den Pop Stack definierte Pfad analysiert. Sobald eine Übereinstimmung gefunden worden ist, wechselt SSR wieder in den *Gerichteten Modus*. Auf dem Pop Stack werden alle Daten oberhalb der gefundenen Übereinstimmung gelöscht.

Während eines einzelnen Routingprozesses kann es zu mehreren Phasenwechseln dieser Art kommen. Nähert sich die Anzahl der Phasenwechsel der Pfadlänge, so degeneriert SSR zu einem Flooding Algorithmus. Dieser Fall kann auftreten, wenn

zwischen zwei konsekutiven Kontextanfragen die Netzwerktopologie wesentlich verändert hat.

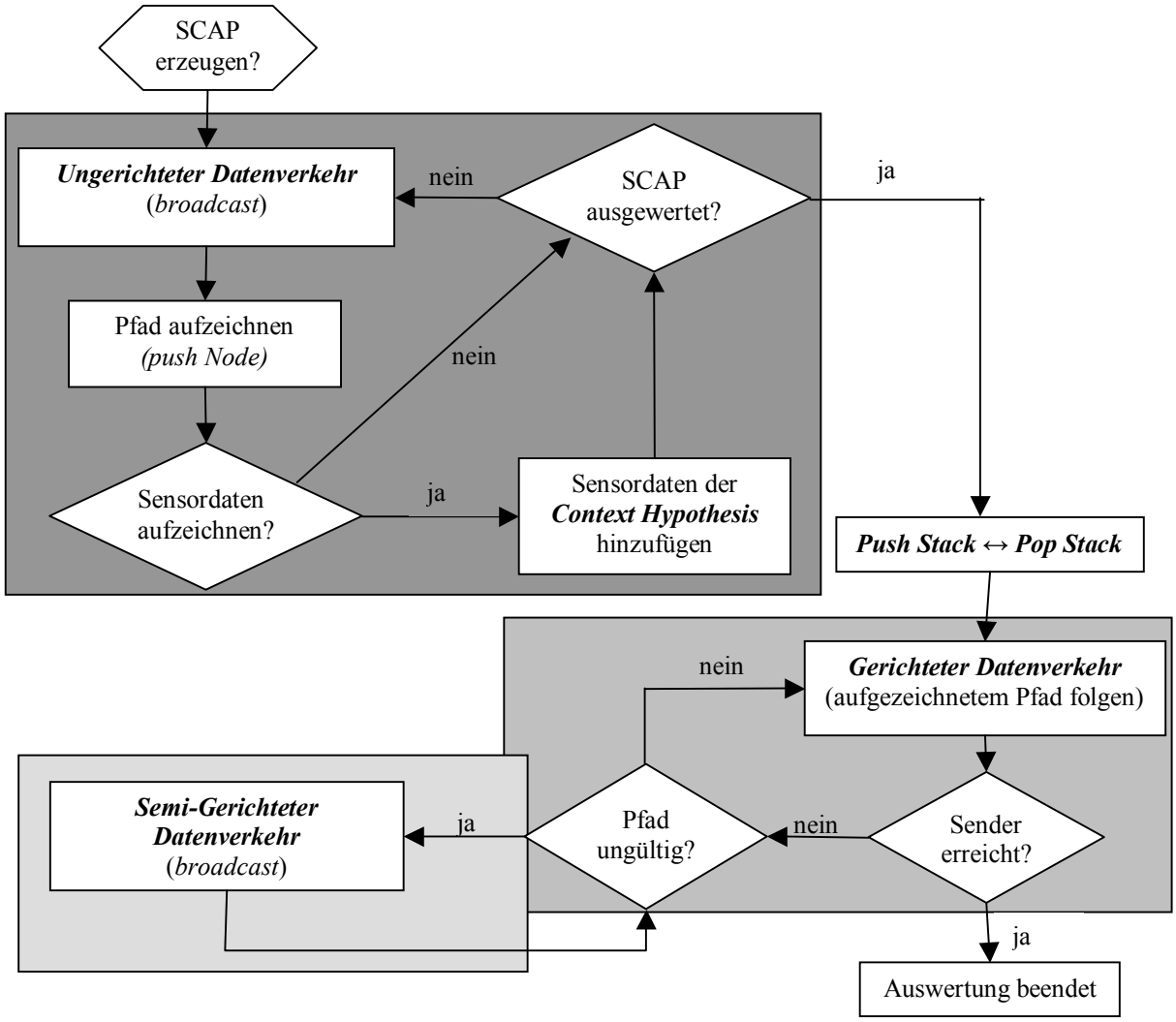

Abbildung 5-14: Flussdiagramm Smart Stack Routing (SSR)

Dead-End Erkennung

In einem drahtlosen Netzwerk kann es zu einem so genannten *Dead-End* kommen, d.h. ein Paket erreicht einen Knoten, in dessen Nachbarschaft nur der vorher sendende Knoten zu erreichen ist. Bei Erkennung einer Dead-End Situation sollte das entsprechende Pakete gelöscht werden. Da generell davon ausgegangen wird, dass ein Paket von einem Knoten immer an alle erreichbaren Nachbarn gesendet wird, macht es keinen Sinn, dieses Paket wieder zurückzusenden, da noch andere Kopien des initiierenden SCAPs im Umlauf sind.

Die Erkennung einer Dead-End Situation lässt sich einfach vollziehen. Sie tritt immer dann auf, wenn außer dem sendenden Knoten kein anderer mehr verfügbar ist. Das Paket wird gelöscht.

Receiving Table

Jeder Knoten verfügt über einen so genannten *Receiving Table*. Alle Knoten am Pfadende einer Kontextanfrage benutzen diese Tabelle, um festzustellen, ob sie bereits ein anderes Paket dieser Anfrage erhalten haben. Aufgrund dieser Informationen können dann redundante Pakete aus dem Netz entfernt werden. Die Größe der Tabelle ist abhängig vom TTL-Parameter der SCAPs.

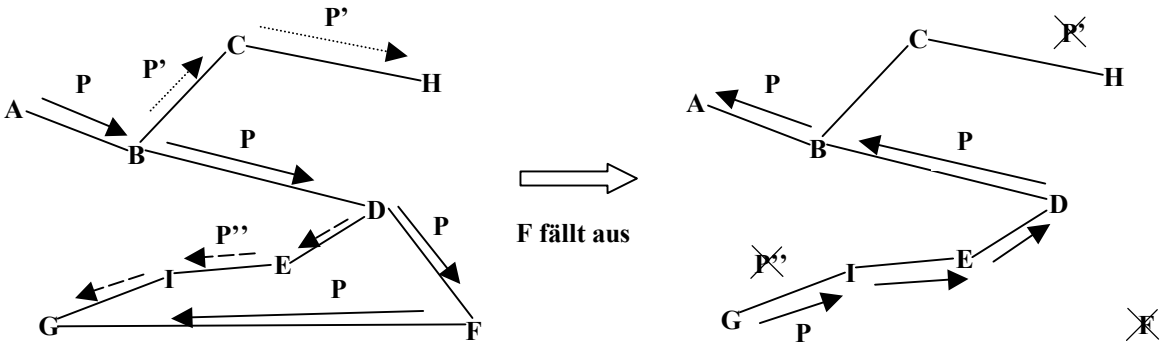

Abbildung 5-15: SCAP Evaluation in einem partiellverknüpften Netzwerk

5.4.3 Fallbeispiel

In diesem Abschnitt sollen die Konzepte von SSR anhand eines Fallbeispiels diskutiert werden. Zur Demonstration behandeln wir ein Sensornetzwerk bestehend aus den Knoten A, B, C, D, E, F, G, H und I. Die durchgezogenen Linien in Abbildung 5-15 bilden den Verbindungsgraph des betrachteten Sensornetzwerkes. Um die Dynamik eines drahtlosen Netzwerkes zu berücksichtigen, gehen wir davon aus, dass Knoten F während der Auswertung einer Kontextanfrage ausfällt. So ist es möglich die Reaktion des SSR Protokolls bez. Fluktuationen der Netzwerktopologie zu veranschaulichen.

Das Beispielszenario wird initiiert durch Knoten A, der eine Kontextanfrage bez. Sensordaten des Knotens G formuliert. Zu Anfang sind sowohl der Push als auch der Pop Stack leer.

Ausgehend von Knoten A existiert nur eine mögliche Verbindung. Bevor SCAP P an Knoten B gesendet wird, wird die Marke **A*** auf den Push Stack gelegt. **A*** identifiziert den Knoten, von dem die zu bearbeitende Anfrage ausgeht; siehe auch Tabelle 5-2. B empfängt das SCAP von Knoten A und stellt fest, dass es die Anfrage nicht erfüllen kann. Demzufolge wird der Rückpfad zu A aufgezeichnet,

indem **Ba** auf den Push Stack gelegt wird. Danach wird das Paket an die Nachbarn des Knotens B versendet. Da von B zwei Knoten erreichbar sind – C und D – wird von SCAP P ein weiteres Paket abgeleitet: P'. P' bewegt sich Richtung Knoten C (siehe Tabelle 5-3), P bewegt sich weiter in Richtung D (siehe Tabelle 5-2).

Mit Erreichen des Paketes P' bei Knoten H wird ein Dead-End festgestellt; das Paket wird entfernt (vgl. Tabelle 5-3). Währenddessen erreicht P Knoten D, der Rückpfad zu Knoten B wird auf dem Push Stack ergänzt. Von Knoten D bestehen wieder zwei Alternativen: E und F. In Konsequenz wird ein weiteres Paket abgeleitet: P''. P bewegt sich über F zum Zielknoten G. Da der von P'' gewählte Pfad über noch zwei weitere Knoten - I und E - verläuft, gehen wir davon aus, dass P vor P'' Zielknoten G erreicht.

Bei Erreichen von Paket P bei G fällt Knoten F aus und ist nicht mehr erreichbar. G überprüft die *Receiving Table* und stellt fest, dass die Anfrage noch nicht beantwortet wurde. Also werden die angeforderten Sensordaten der *Context Hypothesis* von SCAP P hinzugefügt.

Wie in Tabelle 5-2 zu beobachten, vertauscht G den Pop mit Push Stack von Paket P. Da die vorher aufgezeichneten Pfadinformationen aufgrund des Wegfalls von F nicht mehr gültig sind, erfolgt der Transport von P nun *semi-gerichtet*. Zunächst wird P an Knoten I gesendet.

Simultan[11] erreicht P'' Knoten G. G überprüft dann erneut seine *Receiving Table* und stellt fest, dass diese Anfrage schon beantwortet wurde. Demzufolge wird P'' gelöscht; vergleiche Tabelle 5-4. Währenddessen wird innerhalb von P der Pfad zwischen G und D auf dem Push Stack abgelegt. Da aber D teil des Originalpfades war, der auf dem Pop Stack lokalisiert werden kann, wechselt SSR von *Semi-gerichtetem-* zurück zu *Gerichtetem Datenverkehr*. Über B wird das SCAP dann zu Knoten A geleitet, welcher die durch die Kontextanfrage spezifizierten Daten nun auslesen kann.

Zudem kann A aus dem eingehenden SCAP den Pfad zu G extrahieren, so dass nachfolgende Anfragen an Knoten G optimiert werden können, indem statt des *Ungerichteten Datenverkehrs* der effizientere *Gerichtete Datenverkehr* eingesetzt werden kann.

[11] Es wird hier angenommen, dass etwaige Kollisionen durch tiefere Netzwerkschichten abgefangen werden. Dieses Problem ist nicht Bestand dieser Arbeit.

Knoten	Aktion	*Push Stack*	*Pop Stack*
A	push **A***; sende zu B;	**A***	
B	push Ba; sende zu D;	Ba **A***	
D	push Db; sende zu F;	Db Ba **A***	
F	sende zu G;	Fd Db Ba **A***	

- Knoten F fällt aus –

G	Daten hinzufügen; Vertausche Stacks; push **S***		**S*** Fd Db Ba **A***
E[12]	durchsuche Pop Stack; push Eg; semi-gerichtet zu D;	Ba **A***	**S*** Fd Db Ba **A***
D	Durchsuche Pop Stack; Eintrittspunkt gefunden; Korrektur Pop Stack: push De; pop Db; sende zu B;	Db Ba **A***	Ba **A***
B	pop Ba push Bd; sende zu A;	Fd Db Ba **A***	**A***
A	push Ab; pop A*; liefere Daten	Fd Db Ba **A***	

Tabelle 5-2: Packet Trace für Paket P

Knoten	Aktion	*Push Stack*	*Pop Stack*
B	push **A***; sende zu B;	Ba **A***	
C	push Cb; sende zu H;	Cb Ba **A***	
H	lösche Paket (dead-end);	Hc Cb Ba **A***	

Tabelle 5-3: Packet Trace für Paket P'

Knoten	Aktion	*Push Stack*	*Pop Stack*
D	push Db; sende zu E;	Db Ba **A***	
E	push Ed; sende zu I;	Ed Db Ba **A***	
I	push Ie; sende zu G;	Ie Ed Db Ba **A***	
G	terminiere Paket;	-	-

Tabelle 5-4: Packet Trace für Paket P''

5.5 Zusammenfassung

In diesem Kapitel wurde ein neues Konzept zur Gewinnung von Kontextinformationen in Ad-hoc-Sensornetzwerken beschrieben. Es ergänzt die in Kapitel 3 und 4 erläuterten Ansätze zur Realisierung kontextadaptiver Systeme.

Es wurde ein daten-zentrisches Verfahren konzipiert, welches auf dem uniformen Datenaustauschformat Smart Context-Aware Packets (SCAPs) beruht. SCAPs dienen der Akkumulation von Sensordaten und der Ermittlung von Topologie-informationen, welche sich auf zurückgelegte Pfade im Netzwerk beziehen. Einmal gewonnene Informationen können in nachfolgenden Kontextanfragen angewendet werden, um deren Evaluation zu optimieren.

Basierend auf der Idee des SCAPs wurde ein neues Verfahren entwickelt, welches als *Smart Stack Routing* (SSR) bezeichnet wurde. Es wurde inspiriert durch das *Stack Routing Protokoll*, welches in dynamischen Netzwerkumgebungen, wie sie hier betrachtet werden, nicht effizient einsetzbar ist. SSR unterscheidet zwischen *gerichteten* und *ungerichteten* Kommunikationsmodi. Um auf etwaige Knotenausfälle bzw. Erreichbarkeitsprobleme reagieren zu können, unterstützt SSR den dynamischen Wechsel zwischen beiden Kommunikationsmodi. Sollte beispielsweise während der gerichteten Weiterleitung ein noch zu besuchender Knoten ausgefallen sein, so wechselt das SCAP in den nicht-deterministischen, ungerichteten Modus. Nach der Lokalisierung eines Wiederaufsetzpunktes wechselt das SCAP wieder in den gerichteten Kommunikationsmodus.

Da das Volumen der vom SCAP mitgeführten Pfadinformationen linear mit der Pfadlänge ansteigt, ist das Verfahren nur begrenzt skalierbar. Da aber auch hier das Lokalitätsprinzip (vgl. 2.3.3, 3.1.3) angewendet werden kann, ist dieser Aspekt nicht von vorrangiger Relevanz. Dennoch kann der Einsatz in Sensornetzwerken mit einer wesentlich höheren Knotenanzahl empfehlenswert sein. So könnte ein solches Netzwerk in Zellen separiert werden. Innerhalb dieser Zellen könnte SSR eingesetzt werden, während zur Kommunikation zwischen Zellen ein skalierbares Protokoll, wie etwa Directed Diffusion, eingesetzt werden könnte. Diese Vorgehensweise hätte den Vorteil, dass die mächtigeren Primitiven von SSR zur Gewinnung von Kontextinformationen auch in einer solchen Umgebung eingesetzt werden können.

Ein Ausblick auf zukünftige Forschungsarbeiten bez. der Datenfusion in Ad-hoc-Sensornetzwerken erfolgt im nachfolgenden, sechsten Kapitel.

Kapitel 6

Abschlussbetrachtungen

In den folgenden Absätzen werden die wesentlichen Ergebnisse dieser Arbeit zusammengefasst. Außerdem erfolgt eine Erörterung von Forschungsausblicken.

6.1.1 Kontextadaptive Dienstnutzung

Die hier vorgestellten Arbeiten haben insbesondere die kontextadaptive Dienstvermittlung in Ubiquitous Computing Systemen zum Inhalt (siehe Kapitel 3). Vor diesem Hintergrund konnten die klassischen Ansätze der Dienstvermittlung neu interpretiert werden und bez. der neuen Randbedingungen erweitert werden. Aus dem Blickwinkel der Kontextadaptivität wird ersichtlich, dass sich die Prozesse der *Selektion* und *Ausführung* von Diensten nur als eine Ganzheit verstehen lassen, da sich die beiden Prozesse gegenseitig bedingen. Diese Schlussfolgerung findet ihren Ursprung im Wesen der Kontextadaptivität, sie bezieht sich sowohl auf räumliche als auch temporale Bezugspunkte. In vereinfachter Form lässt sich konstatieren, dass räumliche Bedingungen die Selektion eines Dienstes determinieren, während temporale Bedingungen den Zeitpunkt einer Dienstausführung bestimmen. In Berücksichtigung dieser Grundannahmen präsentiert die vorliegende Arbeit Ansätze, in denen die räumlichen und temporalen Bezugnahmen als notwendige Designkriterien eingeflossen sind. Räumlichen Aspekten wird durch Einbeziehen des Lokalitätsprinzips (3.1.3) Rechnung getragen. Die Interpretation dieses Prinzips führt zu einem Dienstmodell, das auf ortsbezogenen Domänen basiert (3.2.1). Temporale Aspekte finden durch Einbeziehung von Trigger-Konzepten Berücksichtigung.

Zur Realisierung der kontextadaptiven Dienstnutzung wurde ein Datenzentrisches Protokoll (3.1.3) entwickelt, welches nicht nur die kontextadaptive Selektion und Ausführung von Diensten gestattet, sondern – im Rahmen einer horizontalen Integration (3.1.3) – auch die Komposition von multiplen Diensten. Des Weiteren wurde aufgezeigt, dass eine geplante Dienstkomposition auch noch zur Laufzeit angepasst werden kann, um so beispielsweise auf Fehler reagieren zu können.

Das entwickelte Protokoll basiert auf einem uniformen Datenaustauschformat, welches als *Context-Aware Packet* (CAPs) bezeichnet wird. Innerhalb eines CAPs werden Dienstanforderungen mittels so genannter *Context Constraints* formuliert. Context Constraints werden kontinuierlich mit ermittelten Kontextinformationen der Umgebung verglichen. Kann bei einem Vergleich eine Übereinstimmung der

durch die Context Constraints implizierten Bedingungen mit den aktuellen Kontextinformationen festgestellt werden, so wird der entsprechende Dienst ausgeführt. Abweichungen führen entweder zu einer Weiterleitung des CAPs oder zu einer verzögerten Ausführung eines selektierten Dienstes.

Ausblick

Zukünftige Arbeiten im Bereich der kontextadaptiven Dienstnutzung beziehen sich insbesondere auf folgende Themenbereiche:

- Untersuchung von Routing-Verfahren
- Entdeckung und Behandlung von Fehlern in einer Dienstsequenz

Die bisherige Implementierung basiert auf einem einfachen hierarchischen Routing-Verfahren (3.4). Weitere Arbeiten könnten alternative Mechanismen zur kontextadaptiven Weiterleitung von Datenfragmenten untersuchen. Dabei wäre es möglich, auch kombinierte Routing-Verfahren zu betrachten: Aspekte der Konstruktion von Kontextinformationen (Kapitel 5) und Aspekte der eigentlichen Datenvermittlung könnten in einem einheitlichen Ansatz zusammenfließen. Motiviert wird diese Idee durch den Umstand, dass die kontextadaptive Datenvermittlung immer nur hinsichtlich der verfügbaren Kontexttypen bzw. der Distributionsschemata (dezentral vs. zentral, push vs. pull) von Kontextinformationen verstanden werden kann.

Ein weiterer relevanter Gesichtspunkt betrifft die Entdeckung und Behandlung von Fehlern bei der Verarbeitung eines CAPs. Obschon die bisherigen Arbeiten eine prinzipielle Möglichkeit zur Fehlerbehandlung aufgezeigt haben (3.4.3), existiert zurzeit kein abgeschlossenes Modell. Im bisherigen Entwurf wurde insbesondere nicht der Ausfall von Komponenten des Systems berücksichtigt, die eine durch ein CAP definierte Dienstsequenz unterbrechen könnten.

6.1.2 Kontextadaptive Integration von Diensten

Die Arbeiten bez. der *kontextadaptiven Integration von Diensten* münden in einem System, in welchem die Rollen individueller Dienstendpunkte (z.B. mobile Endgeräte) dynamisch verwaltet werden können. Auf diese Weise konnten erforderliche Kommunikations- und Koordinationsvorgänge zwischen Dienstendpunkten kontextadaptiv formuliert werden. Zu diesem Zweck wird die Konzeption der kontextadaptiven Dienstnutzung (siehe voriger Abschnitt) um folgende Aspekte erweitert:

- Kontextadaptive Generierung von Context-Aware Packets
- Unterstützung von Sessions
- Lokales Routing

Während im Rahmen der kontextadaptiven Dienstnutzung vor allem die Evaluation von CAPs und die damit verbundenen Dienstinteraktionen untersucht wurden, wurde bez. der kontextadaptiven Integration von Diensten auch die systemgestützte Generierung von CAP-Dokumenten behandelt. Ein zentrales Ergebnis dieser Überlegungen war die Konzeptionisierung und Spezifikation des Interaktionsmusters (*Interaction Template*). Ein Interaktionsmuster ist eine Dateneinheit, welche die zu einer bestimmten Aufgabe gehörenden Interaktionen und ihre Ausführungsbedingungen abstrakt beschreibt, um den in Frage kommenden Entitäten ihre Rollen zuzuordnen. Es wurde gezeigt, dass die Komposition von Interaktionsmustern zudem die Abbildung komplexer Prozesse erlaubt. In diesem Zusammenhang war auch die Realisierung des *Interaction Organizers* (4.4.1 von ausschlaggebender Relevanz. Der Interaction Organizer erzeugt CAPs basierend auf den in den Interaktionsmustern formulierten Regeln.

Ein weiterer Beitrag bezieht sich auf die tool-gestützte Gestaltung von Interaktionsmustern basierend auf einer grafischen Benutzeroberfläche; diese Komponente wird als *Mediator* bezeichnet. Mittels des Mediators erfolgt die Erstellung von Interaktionsmustern in drei Schritten: (1) Identifikation von beteiligten Dienstendpunkten, (2) Definition des Rollenverhaltens bzw. den erforderlichen Interaktionen zwischen Dienstendpunkten und (3) die Verteilung der resultierenden Interaktionsmuster auf die entsprechenden Dienstendpunkte.

Ausblick

Die rollenbasierte Prozessmodellierung, wie sie hier vorgeschlagen wurde, realisiert einen relevanten Aspekt in Bezug auf Ubiquitous Computing Systeme. Insbesondere, weil sie es erlaubt, die vielfältigen Anwendungskontexte mobiler Endgeräte in angemessener Weise abzubilden.

Zukünftige Arbeiten könnten die bestehende Konzeption erweitern, indem sie den Prozess der Erstellung und Distribution von Interaktionsmustern um ein weiteres automatisieren würden. Somit wäre dem Anspruch der vereinfachten Bedienung im Höheren genüge getan. Ein bereits skizzierter Ansatz [S 02] sieht vor, die von einem Benutzer ausgelösten Aktionen in einer Ubiquitous Computing Umgebung auf Wunsch aufzuzeichnen. Basierend auf diesen Aufzeichnungen könnten dann automatisch Interaktionsmuster bzw. Codefragmente identifiziert werden, welche zu einem späteren Zeitpunkt, in Abhängigkeit von kontextuellen Einschränkungen, zur Ausführung gebracht werden könnten.

Die Grundidee dieses Verfahrens, welches im Umfeld von Desktop-Systemen, als Script- oder Macro-Recording (z.B. AppleScript [AS]) bekannt ist, lässt sich im Rahmen von Ubiquitous Computing Systemen und den damit assoziierten Randbedingungen neu interpretieren. So müsste beispielsweise berücksichtigt werden, dass sich die bestehenden Verfahren auf Einzel-Rechner-Architekturen

beziehen. Es wäre also u.a. erforderlich, das Konzept des Script-Recordings bez. verteilter Systemarchitekturen zu verallgemeinern.

6.1.3 Konstruktion und Komposition von Kontextinformationen

Schließlich präsentiert die Arbeit einen neuen Ansatz zur Konstruktion von Kontextinformationen (Kapitel 5), welcher für die Realisierung der oben beschriebenen Ansätze essentiell ist. Ausgehend von der Annahme, dass in einer Ubiquitous Computing Umgebung auch die Erfassung von Kontextinformationen an mobile Entitäten gebunden ist, wurde sie als ein Prozess der Datenfusion in einem drahtlosen Ad-hoc-Sensornetzwerk verstanden.

Zu diesem Zweck wurde auch hier ein Datenzentrisches Protokoll vorgeschlagen, in welches Grundkonzepte des in Kapitel 3 vorgeschlagenen Konzeptes zur kontextadaptiven Dienstnutzung miteingeflossen sind. Jedoch musste die Konzeption erweitert bzw. angepasst werden, um den dynamischen Aspekten eines Ad-hoc-Sensornetzwerkes zu genügen. So mussten insbesondere Strategien entwickelt werden, welche den Prozess der Datenfusion auch dann ermöglichen, wenn die Netzwerktopologie dynamischen Änderungen unterliegt. So können beispielsweise Netwerkknoten ausfallen oder nicht mehr erreichbar sein.

Das hier vorgeschlagene Protokoll wurde als *Smart Stack Routing* (SSR) bezeichnet. Es ermöglicht die Gewinnung von Kontextinformationen basierend auf Kontextanfragen, die den gewünschten Kontexttyp einer Anwendung bestimmen. Kontextanfragen werden durch *Smart Context-Aware Packets* (SCAPs) repräsentiert. SCAPs werden in ein Sensornetzwerk „injiziert“. Sie akkumulieren bzw. verarbeiten die gewünschten Daten und werden anschließend zum Sender zurückgesendet. SCAPs bestehen aus drei Teilen: *Retrieving Plan*, *Context Hypothesis* und *Packet Trace*. Der *Retrieving Plan* definiert einen Datenpfad. Anhand einer Sequenz von abzufragenden Sensor-Typen – nicht konkreten Knoten – wird die Kontextanfrage formuliert. Die schrittweise ermittelten Daten werden innerhalb der Datensektion *Context Hypothesis* akkumuliert. Zusätzlich können innerhalb des *Retrieving Plans* so genannte *Rewriting Rules* definiert werden, welche es erlauben, den Datenpfad in Anhängigkeit von gewonnen Messdaten anzupassen. Schließlich dient das Datenfeld *Packet Trace* eines SCAPs der Gewinnung von Topologieinformationen eines Sensornetzwerkes. Diese Informationen können eingesetzt werden, um die Bearbeitung von aufeinanderfolgenden Kontextanfragen zu optimieren, so etwa bez. der Anzahl der zu besuchenden Knoten oder eines gleichmäßig verteilten Energieverbrauchs. Bei der Evaluation einer initialen Kontextanfrage existieren keinerlei Informationen über die Topologie des in Frage kommenden Sensornetzwerks. In diesem Fall wird ein nichtdeterministisches Verfahren eingesetzt, dass als *Flooding* bezeichnet wird. In dieser Phase werden Informationen über die Topologie des Netzes gesammelt. Diese werden in Form von zurückgelegten Pfaden eines SCAPs im *Packet Trace*

abgelegt. Für zukünftige Kontextanfragen können diese Pfadinformationen wiederverwendet werden.

SSR unterscheidet zwischen einem so genannten *gerichteten* und *ungerichteten* Datenverkehr. Der gerichtete Datenverkehr kann dann eingesetzt werden, wenn gültige Pfadinformationen im zu transportierenden SCAP vorliegen. Für den Fall, dass die Pfadinformationen nicht mehr vollständig die gegenwärtige Struktur des Sensornetzwerkes reflektieren (es ist z.B. ein Knoten auf dem Pfad ausgefallen), kann vorübergehend auf die Methode des ungerichteten Datenverkehrs gewechselt werden. Hier wird nach einem Wiederaufsetzpunkt in der Nachbarschaft des zuletzt besuchten Knotens gesucht. Sobald dieser bestimmt ist, wird wieder in den gerichteten Modus gewechselt.

Die Konzeption von SSR basiert auf der Beobachtung, dass sich in einem hinreichend kurzen Zeitintervall die Topologie eines Sensornetzwerkes nur graduell ändert, und somit bereits bestehende Topologieinformationen kontinuierlich angepasst werden können.

Ausblick

Bisher wurde SSR nur qualitativ untersucht. Zur Festigung der bisher ermittelten Ergebnisse wäre insbesondere auch eine quantitative Analyse wünschenswert. Dann wäre es möglich, Variationen des Protokolls bez. der Effektivitätskriterien miteinander zu vergleichen, um so eine möglichst optimale Variante zu bestimmen. Zudem wäre der Vergleich mit bereits existierenden Protokollen von Interesse.

SSR impliziert die Annahme, dass sich akkumulierte Messungen immer auf ein einzelnes Phänomen beziehen. Diese Annahme kann jedoch nicht in jedem Fall bestätigt werden. Es besteht beispielsweise die Möglichkeit, dass multiple Sensoren demselben Netzwerk angehören, die von den Sensoren gemessenen Daten sich aber auf verschiedene Phänomene beziehen. So könnte ein Sensor A ein Phänomen in Raum A beobachten und ein Sensor B ein Phänomen in Raum B. SSR würde beide Messungen in Bezug zu einem einzelnen Phänomen in Raum A oder B interpretieren.

Ein vorstellbarer Lösungsansatz besteht in der Möglichkeit, dass ein Sensornetzwerk in eigenständige Domänen partitioniert wird. Die Ausbildung von Domänen würde bez. zeitlich korrelierter Messdaten erfolgen.

Referenzen

[AS 00] G. Abowd, J.P.G. Strebenz. Final Report on the Inter-Agency Workshop on Research Issues for Smart Environments. IEEE Personal Communications, October 2000.

[A 92] M.A. Abidi. Fusion of multi-dimensional data using regularization. *In Data fusion In Robotics and Machine Intelligence.* Academic Press 1992.

[AAHL+ 97] Abowd, G.D., Atkeson, C.G., Hong, J., Long, S., Kooper, R., and Pinkerton, M. Cyberguide: A Mobile Context-Aware Tour Guide. *ACM Wireless Networks* 3, 421-433. 1997

[Air] Airport Wireless Technology. http://www.apple.com/airport

[AS] AppleScript. http://www.apple.com/applescript/

[ASSC 02] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*. 38, 2002.

[Azu 97] R. T. Azuma. A Survey of Augmented Reality. *Presence*, August 1997.

[Badge] The Active Badge Project. AT&T Laboratories Cambridge. http://www.uk.research.att.com/ab.html

[BaMK 97] Rob Barrett, Paul P. Maglio and Daniel C. Kellem. How to personalize the Web. Conference Proceedings on Human factors in computing systems, 1997, pages 75 - 82

[BB 95] A. Bakre, B.R. Badrinath. Handoff and System Support for Indirect TCP/IP. Proceedings 2nd Usinix Symposium Mobile and Location-Independent Computing. Ann Arbor, MI. April 1995.

[BBG+ 00] G. Banavar, J. Beck, E. Gluzberg, J. Munson, J. Sussmann, D. Zukowski. Challenges: An Application Model for Pervasive Computing. In Proceedings of MobiCom 2000, Boston August 2000

[BC 95] K. A. Bharat, L. Cardelli. Migratory applications. In Proceedings of the 8th Annual ACM Symposium on User Interface Software and Technology, Pittsburgh, Pa., November 1995

[BE 00] F. Bry, N. Eisinger. Data Modeling with Markup Languages. http://www.pms.informatik.uni-

muenchen.de/forschung/datamodeling-markup.html, July 2000.

[BEK+ 00] Simple Object Access Protocol (SOAP) 1.1 World Wide Web Consortium (W3C), http://www.w3.org/TR/SOAP/

[BGI 99] J. Beck, A. Geaut, and N. Islam. Moca: A service framework for mobile computing devices. In Proceedings of the ACM International Workshop on Data Engineering for Wireless and Mobile Access (MobiDE), August 1999.

[BKAB+ 98] E.A. Brewer. R. H. Katz, E. Amir, H. Balakrishnan, Y. Chawathe, A. Fox, S. D. Gribble, T. Hodes, G. Nguyen, V. N. Padmanabhan, M. Stemm, S. Seshan, T. Henderson. A Network Architecture for Heterogeneous Mobile Computing. IEEE Personal Communications. October 1998.

[Blue] Bluetooth. http://www.bluetooth.com

[BMM+ 00a] M. Bauer, A. MacWilliams, F. Michahelles, C. Sandor, S. Riß, M. Wagner, B. Zaun, C. Vilsmeier, T. Reicher, B. Brügge, G. Klinker. DWARF: Requirements Analysis Document. Technical report, Technical University of Munich, 2000.

[BMM+ 00b] M. Bauer, A. MacWilliams, F. Michahelles, C. Sandor, S. Riß, M. Wagner, B. Zaun, C. Vilsmeier, T. Reicher, B. Brügge, G. Klinker. DWARF: System Design Document. Technical report, Technical University of Munich, 2000.

[BO 92] D. Batory, S. O´Malley. The Design and Implementation of Hierarchical Software Systems with Reusable Components. *ACM Transactions on Software Engineering and Methodology*, October 1992

[Bos 97] J. Bosak. XML, JAVA, and the future of the Web. Technical Report. SUN Microsystems, http://www.ibiblio.org/pub/sun-info/standards/xml/why/xmlapps.html, 1997

[Bou 99] R. Bourret. XML and Databases. http://www.rpbourret.com/xml/XMLAndDatabases.htm

[Bro 96] P.J. Brown. The Stick-e Document: a Framework for Creating Context-Aware Applications. In Proceedings of EP'96, Palo Alto, January 1996

[BroBo 97] P. J. Brown, J. D. Bovey, X. Chen. Context-aware applications: From the laboratory to the marketplace. *IEEE Personal Communications*. October 1997.

[Brow 98] M. Brown, Eugene Santos, Sheila B. Banks and Mark E. Oxley. Using explicit requirements and metrics for interface agent user

model correction. Proceedings of the second international conference on Autonomous agents , 1998, Pages 1 – 7

[BTA 99] J. Brotherton, K. Truong, G. Abowd. Supporting capture and access interfaces for informal and opportunistic meetings. GVU Center, Georgia Institut of Technology. January 1999. http://www.gatetech.edu/gvu/reports/1999

[BW 00] G. Borriello, R. Want. Embedded computation meets the world wide web. CACM. May 2000.

[CDK 02] G. Coulouris, J. Dollimore, T. Kindberg. Verteilte Systeme, Konzepte und Design. Addison-Weseley. 2002

[Cha 00] A. Chakraborty. A distributed Architecture for Mobile Location-Dependent Applications. Master's thesis. Massachusetts Institute of Technology, June 2000. http://nms.lcs.mit.edu/papers/achakra-thesis.pdf

[Che 97] P. Pin-Shan Chen. The Entity-Relationship Model-Toward a Unified View of Data. *ACM Transactions on Database Systems,* 1976

[CT 01] The context-toolkit. http://www.cc.gatech.edu/fce/contexttoolkit/. 2001

[CV 86] Z. Chair, P. K. Varshney. Optimal data fusion in multiple sensor detection systems. IEEE Transactions on Aerospace and Electronic Systems, 22:98--101, 1986.

[CZHK 99] S. Czerwinski, B. Zhao, T. Hodes, A. Joseph, R. H. Katz. An Architecture for a Secure Service Discovery Service. *Proceedings of The Fifth ACM/IEEE International Conference on Mobile Computing (MobiCom '99)*, Seattle, WA, August 1999.

[DA 00] A.K. Dey, G.D. Abowd. CyberMinder: A context-aware system for supporting reminders. http://www.cc.gatech.edu/fce/contexttoolkit/pubs/HUC2000.pdf. 2000

[DAPW 97] A. Dey, G. Abowd, M. Pinkerton, and A. Wood. CyberDesk: A Framework for Providing Self-Integrating Ubiquitous Software Services. Proc. ACM UIST'97, 1997

[DCME 01] N. Davies, K. Cheverst, K. Mitchel, A. Efrat. Using and Determinig Location in a Context-Sensitive Tour Guide. IEEE Computer. August 2001.

[Dey 00] Anind K. Dey. Providing Architectural Support for Building Context-Aware Applications. PhD thesis, College of Computing,

Georgia Institute of Technology, December 2000.

[DeyAb 00] A. K. Dey and G. D. Abowd. Towards a Better Understanding of Context and Context-Awareness *In the Workshop on The What, Who, Where, When, and How of Context-Awareness, as part of the 2000 Conference on Human Factors in Computing Systems (CHI 2000)*, The Hague, The Netherlands, April 2000.

[DL 02] R. Droms, T. Lemon. The DHCP Handbook. Macmillan Technical Publishing, Indianapolis, IN. 2002

[DWA 00] DWARF – Distributed Wearable Augmented Reality Framework, 2000. http://www.augmentedreality.de

[EE 01] J. Elson, D. Estrin. Time synchronization for wireless sensor networks. *Proceedings 15th International Parallel and Distributed Processing Symposium*. 2001.

[EGH 00] D. Estrin, R. Govindan, J. Heidemann. Embedding the Internet. Communications of the ACM. May 2000

[EGHK 99] D. Estrin, R. Govindan, J. Heidemann, S. Kumar. Next century challenges: Scalable Coordination in Sensor Networks. *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*. 1999

[EHAB 99] M. Esler, J. Hightower, T. Anderson, and G. Borriello. Next century challenges:data-centric networking for invisible computing. the portolano project at the university of washington. In Proc. of the Fifth ACM/IEEE Intl. Conf. on Mobile Networking and Computing, August 1999.

[EHC+93] S. Elrod, G.Hall, R. Costanza, M. Dixon, J. Des. Responsive Office Environments. *Communications of the ACM,* July 1993.

[F 94] O. Föllinger. Regelungstechnik. Einführung in die Methoden und ihre Anwendung. Hüthig, Hdlbg. 1994.

[FF 98] D. Franklin, J. Flachsbart. All gadget and no representation makes jack a dull environment, February 1998.

[FH 92] S. Fischer-Hübner. IDA (Intrusion Detection and Avoidance System): Ein einbruchsentdeckendes und einbruchsvermeidendes System. Dissertation am Fachbereich Informatik der Universität Hamburg, Juli 1992.

[Gel 00] H.-W. Gellersen. Ubiquitous Computing, Vorlesung im WS 00/01. http://www.teco.edu/lehre/#vorlesung. 2000

[GHJV 95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software.

Addison-Wesley, Reading, MA, 1995.

[GNU] Gnutella. Welcome to Gnutella. http://gutella.wego.com/

[Gog 86] J. A. Goguen. Reusing and Interconnecting Software Components. *IEEE Computer*. February 1986.

[GP 00] C. F. Goldfarb and O. Prescod. *The XML Handbook.* Prentice Hall PTR, 2000. ISBN 0-13-014714-1.

[Gutt 99] E. Guttman. Service Location Protocol : Automatic Discovery of IP Network Services. IEEE Internet Computing. July-August 1999.

[Har 99] E. R. Harold. *The XML Bible.* IDG Books, 1999.

[HHRB 92] P. Honeymann, L. Huston, J.Rees, D. Bachmann. The LITTLE WORK project. In *Proceedings of the 3rd Workshop on Workstation Operating Systems.* 1992.

[HK 99a] A Document-based Framework for Internet Application Control T. Hodes, R. H. Katz. *Proceedings of the Second USENIX Symposium on Internet Technologies and Systems (USITS '99),* Boulder, CO, October 1999

[HK 99b] Composable Ad hoc Location-based Services for Heterogeneous Mobile Clients T. D. Hodes, R. H. Katz. *ACM Wireless Networks Journal, special issue on mobile computing: selected papers from MobiCom '97*, October 1999.

[HKB 99] Adaptive protocols for information dissemination in wireless sensor networks. *Proceedings of the 5th annual ACM/IEEE international conference on mobile computing and networking.* 1999.

[HL 96] B. C. Housel, D. B. Lindquist. WebExpress: A system for optimizing Web browsing in a wireless environment. In *Proceedings of thr 2nd Annual International Conference on Mobile Computing and Networking* (MOBICOM'96). ACM Press. 1996.

[HMNS 01] U. Hansmann, L. Merk, M. S. Nicklous, T. Stober. Pervasive Computing Handbook. *Springer-Verlag*. 2001

[HNBR 97] Hull, Richard, Philip Neaves and James Bedford-Roberts (1997). Towards situated computing. In the Proceedings of the 1st International Symposium on Wearable Computers (ISWC'97), pp. 146-153, Cambridge, MA, IEEE. October 13-14, 1997.

[HNBR 97] R. Hull, P. Neaves, J. Bedrod-Roberts. Towards Situated Computing. In *Proceedings of the 1st International Symposium on*

Wearable Computers, ISCW '97, Cambridge MA, IEEE Press, October 1997.

[HoKa 98] Enabling ``Smart Spaces:" Entity Description and User Interface Generation for a Heterogeneous Component-based Distributed System. T. D. Hodes, R. H. Katz. *DARPA/NIST Smart Spaces Workshop, (unrefereed position paper) Gaithersburg, Maryland, July 1998.*

[How 97] T. Howes. The String Representation of LDAP Search Filters. IETF. RFC 2254. December 1997.

[HTML] http://www.w3.org/MarkUp/

[HTTP] HTTP (Hypertext Transfer Protocol). http://www.w3.org/Protocols

[HYBS 00] J. Lilley, J. Yang, H. Balakrishnan, S.Seshan. A Unified Header Compression Framework for Low-Bandwidth Links. *Proc. of the Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networking*, August 2000.

[IL 01] Diego López de Ipiña and Sai-Lai Lo. LocALE: a Location-Aware Lifecycle Environment for Ubiquitous Computing *Proceedings of the 15th IEEE International Conference on Information Networking* (ICOIN-15), Beppu City, Japan. January 31 - February 2, 2001

[INS 99] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, J. Lilley. The Design and Implementation of an Intentional Naming System. *In Proceedings of the ACM Symposium on Operating Systems Principles,* Charleston, SC. 1999.

[IrDA] The Infrared Data Association (IrDA). http://www.irda.org

[JAVA] JAVA, www.sun.com/java

[JHE 99] J. Jing, A. S. Helal, A. Elmagarmid. Client-Server Computing in Mobile Environments. ACM Computing Surveys, Vol. 31, No. 2. June 1999.

[JINI] JINI, http://java.sun.com/products/jini.

[JK 97] A. D. Joseph, M. F. Kaashoek. Building reliable mobile-aware applications using the Rover toolkit. *Wireless Networks 3*. 1997

[JXP] XJParser. http://xdev.datachannel.com/downloads/xjparser/.

[K 94] Alfred Kobsa. User modeling and user-adapted interaction. *Proceedings of the CHI'94 conference companion on Human factors in computing systems.* 1994

[K 95] T. Kaß. Anwendungsspezifische Autorisierungsstrategien in polymorphen persistenten Programmierumgebungen: Ein Bibliotheksansatz. Diplomarbeit Universität Hamburg. FB Informatik. 1995.

[Ka 00] Mari Korjea-aho. Context-Aware Application Survey. Technical Report. Department of Computer Science. Helsinki University of Technology. April 2000.

[Ka 94] R. Katz. Adaptation and mobility in wireless communication systems. IEEE Personal Communications. 1994.

[KE] The k-Edit System. http://koala.ilog.fr/k-edit.html.

[KLST 96] R. H. Katz, D. Long, M. Satyanarayanan, S. Tripathi. Workspaces in the Information Age. In *Report of the NSF Workshop on Workspaces in the Information Age*. Leesburg, VA. October 1996

[KOA+99] C.Kidd, R.Orr, G. Abowd, C. Atkeson, I. Essa, B. MacIntyre, E. Mynatt, T. Starner, W. Newstetter. The Aware Home: A Living Laboratory Ubiquitous Computing Research. In *Proceeding of the Second International Workshop on Cooperative Buildings – CoBuild'99*. 1999.

[KRA 94] M. Kojo, K. Raatikainen, T. Alanko. Connecting mobile workstations to the internet over a digital cellular telephone network. In *Proceedings of the Mobi-data Workshop*. Rutgers University Press. 1994.

[Krieg 98] D. Krieger, R.M. Adler. The Emergence of Distributed Component Platforms. IEEE Computer. March 1998

[KS 92] J. J. Kistler, M. Satyanarayanan. Disconnected Operation in the Coda File System. ACM Transactions on Computer Systems, vol. 10, no. 1. February 1992

[KSB 98] Kortuem, Gerd, Zary Segall and Martin Bauer (1998). Context-aware, adaptive wearable computers as remote interfaces to 'intelligent' environments. In the Proceedings of the 2nd International Symposium on Wearable Computers (ISWC '98), pp. 58-65, Pittsburgh, PA, IEEE. October 19-20, 1998.

[Kur 00] Roland Kurmann .Spontane Vernetzung, Dienstbeschreibung, service discovery. Fachseminar 2000. ETH Zürich. 2000.

[KZK 97] M. Kam, X. Zhu, P. Kalata. Sensor fusion for mobile robot navigation. IEEE Proceedings, 85(1):108--119, 1997.

[L 01] Y. Lin. Kontext-abhängige Dienstintegartion mittels kombinierbarer Interaktionsmuster. Diploma Thesis. Universitz

kombinierbarer Interaktionsmuster. Diploma Thesis. University of Munich, 2001.

[L 98] C. Linnhoff-Popien. CORBA – Kommunikation und Management. Springer-Verlag. 1998.

[LS 97] E. Lupu, M. Sloman. A Policy Based Role Object Model. In *First International Enterprise Distributed Object Computing Workshop* (*EDOC'97)*, Australia. October 1997

[LWT 94] C. Lindblad, D. Wetherall, D. Tennenhouse. The VuSystem: A programming system for visual processing of digital video. In *ACM Multimedia. 1994*

[LY 99] T. Lindholm, F. Yellin. The Java(TM) Virtual Machine Specification (2nd Edition). Addison-Wesley. 1999.

[M 87] P. Mockapetris. Domain Names-Implementation and Specification. IETF RFC 1035. October 1987

[M 95] G. Meszaros. Pattern: Half-Object Plus Protocol. http://www.bell-labs.com/topic/books/PLoPD1/manuscripts/HalfObjectPlusProtocol.html. 1995

[MaSc 96] M. Matthew, C. Schmandt. CLUES: dynamic personalized message filtering. Proceedings of the ACM 1996 conference on Computer supported cooperative work, 1996, pages 113 – 121

[McG 00] R. E. McGrath. Discovery and Its Discontents: Discovery Protocols for Ubiquitous Computing. 2000

[MDNS] Multicast DNS. http://www.multicastdns.org/

[MDW 94] J. Manyika, H. Durrant-Whyte. Data Fusion and Sensor Management: A Decentralized Information-Theoretic Approach. Ellis Horwood Series in Digital and Signal Processing. 1994

[Mer 00] D. Mertz. On the pythonic treatment of XML documents as objects. http://gnosis.cx/publish/programming/xml_matters_2.txt. 2000

[MES 95] L. B.Mummert, M. R. Ebling, M. Satyanarayanan. Exploiting Weak Connectivity for Mobile File Access. Proceedings 15th ACM Symposium Operating System Principles. Copper Mountain Resort, CO. December 1995

[MGKR+ 96] S. Markwitz, F. Gudermann, C. Kaiser, H. Röhrig, K. Römer, R. Farsi. AI-Trader. http://www.vsb.cs.uni-frankfurt.de/projects/aitrader/. 1996

[MGR+ 99] N. Minar, M. Gray, O. Roup, R. Krikorian, P. Maes. Hive: Distributed Agents for Networking Things, *ASA/MA* August 1999.

[Mi 01] F. Michahelles. Designing an Architecture for Context-Aware Service Selection and Execution. Diploma Thesis. University of Munich, 2001.

[MIP] IP Routing for Wireless/Mobile Hosts (mobileip) http://www.ietf.org/html.charters/mobileip-charter.html

[MJ 95] S. McCanne, van Jacobson. vic : A flexible framework for packet video. In *ACM Multimedia*. 1995

[MS 01] F. Michahelles and M. Samulowitz. Smart CAPs for Smart Its – Context Detection for Mobile Users. In Proccedings of the Third International Workshop on Human-Computer Interaction with Mobile Devices, Lille, France, September 2001

[MSS 02] F. Michahelles, M. Samulowitz, and Bernt Schiele. Detecting Context in Distributed Sensor Networks by Using Smart Context-Aware Packets. Architecture of Computer Systems (ARCS), 2002

[NCLL 01] H. F. Nielsen, E. Christensen, S. Lucco, D. Levin. Web Services Referral Protocol (WS-Referral). Web Link http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/wsreferspecindex.asp

[Nel 98] G. Nelson. *Context-Aware and Location Systems.* PhD thesis. University of Cambridge, Computer Laboratory, Cambridge. UK, January 1998.

[Nelson 98] G. Nelson. Context-Aware and Location Systems. PhD thesis, University of Cambridge, Computer Laboratory, Cambridge, UK, January 1998

[Nor 98] D. Norman. *The Invisible Computer.* MIT Press, 1998.

[NSN+ 97] B. D. Noble, M. Satyanarayanan, D. Narayanan, J. E. Tilton, J. Flinn, K.R. Walker. Agile application-aware adaptation for mobility. *ACM SIGOPS Operating Systems.* 1997

[NT 01] H. F. Nielsen, S. Thatte. Web Services Routing Protocol. Microsoft Coporation. October 2001.

[O 92] R. Oppliger. Computer-Sicherheit. Vieweg-Verlag. 1992.

[OMG 99] Object Management Group. CORBAServices: Common Object Services Specification. Object Management Group (OMG). 1999

[Orw 93] Jon Orwant. *Doppelgänger Goes To School: Machine Learning*

for User Modelling. PhD thesis. MIT Media Laboratory, Cambridge, September 1993.

[Ous 98] Scripting: Higher level programming for the 21st century. *IEEE Compute*. March 1998.

[P 82] Simple Mail Transfer Protocol (SMTP). RFC 281. http://www.freesoft.org/CIE/RFC/821/

[P2P 02] Peer-to-Peer Networking. http://www.openP2P.com

[Pas 00] Dynamic networking requires comprehensive service discovery. *HP Chronicle.* October 2000. http://www.serverworldmagazine.com/hpchronicle/2000/10/discovery.shtml

[Pas 98a] Pascoe, J. Adding Generic Contextual Capabilities to Wearable Computers. *2nd International Symposium on Wearable Computers*, 146-153, 1998

[Pas 98b] J. Pascoe. The Stick-e Note Architecture: Extending the Interface Beyond the User. Technical Report, University of Kent, Canterbury, 1998.

[Pen 93] A. Pentland, "Smart Rooms", Scientific American, April 1993

[Py] http://www.python.org

[R 95] B. R. Rao, Making the Most of Middleware.. Data Communications International 24, 12 (September 1995): 89-96.

[RDF 99] Resource Description Framework (RDF) Model and Syntax Specification W3C Recommendation 22 February 1999. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222

[Ref] Java Reflection API. http://java.sun.com/products/jdk/1.1/docs/guide/reflection/

[RFC 2165] J. Veizades, E. Guttman, C. Perlins. Service Location Protocol". IETF, RFC 2165. June 1997. http://www.rfc-editor.org/rfc/rfc2165.txt

[RFC 2608]] E. Guttman, C. Perkins, J. Veizades, M. Day. Service Location Protocol, Version 2. IETF, RFC 2608, June 1999. http://www.rfc-editor.org/rfc/rfc2608.txt

[Rho 97] Rhodes, B. J. The Wearable Remembrance Agent, in Proceedings of the 1st International Symposium on Wearable Computers, ISWC '97 (Cambridge MA, October 1997), IEEE Press, 123-128

[RMGS] P. Rentala, R. Musunri, S. Gandham, U. Saxena. Survey on Sensor Networks. Department of Computer Science. University

of Texas at Dallas.

[RMI] Java Remote Method Invocation (RMI). http://java.sun.com/products/jdk/rmi/

[RMS 92] H. Ritter, T. Martinez, K.Schulten. Neuronale Netze. Ein Einführung in die Neuroinformatik selbstorganiserender Netze. Addison-Wesley. 1992

[RMW 99] B. J. Rhodes, N. Minar, J. Weaver. Wearable Computing Meets Ubiquitous Computing - Reaping the best of both worlds. Appears in The Proceedings of The Third International Symposium on Wearable Computers (ISWC '99), San Francisco, CA, October 18-19 1999

[Rom 98] C. Romer. A Composable Architecture for Scripting Multimedia Network Applications. Technical report. University of California, Berkley, July 1998.

[Ronin] Ronin Agent Framework. http://gentoo.cs.umbc.edu/ronin/

[RoTo 99] E. M. Royer, C.K. Toh. A Review of Current Routing Protocols for ad hoc mobile wireless networks. *IEEE Personal Communications*, vol 6, no. 2, April 1999.

[RW 96] A. Rowstron and A. Wood. An Efficient Distributed Tuple Space Implementation for Networks of Heterogeneous Workstations. *Department of Computer Science, University of York*, Technical Report YCS-270, 1996.

[Rya 99] N. Ryan. ConteXtML: Exchanging Contextual Information between a Mobile Client and the FieldNote Server. Technical report. Computing Laboratory, University of Kent at Canterbury, CT2 7NF, UK. August 1999.

[S 00] C. Shirky. What is P2P … and What Isn't. http://www.openp2p.com/pub/a/p2p/2000/11/24/shirky1-whatisp2p.html?page=1

[S 02] M. Samulowitz. Context-Aware Service Provision. Siemens Research Days 2002.

[S 94] M. Sloman, Policy Driven Management For Distributed Systems, *Plenum Press Journal of Network and Systems Management*, vol 2, no. 4, Dec. 1994

[Sa 00] M. Samulowitz. Towards Context-Aware User Modeling. Third International IFIP/GI Conference, Trends Towards Universal Service Markets (USM 2000), Munich, Germany, September 2000, Springer-Verlag Lecture Notes in Computer Science

[SAHARA] SAHARA: Service Architecture for Heterogeneous Access, Resources, and Applications. http://sahara.cs.berkeley.edu/

[SAL 02] Salutation, 2002. http://www.salutation.org

[SAS 01] P. Sutton, R. Arkins, B. Segall. Supporting Disconnectedness - Transparent Information Delivery for Mobile and Invisible Computing. *CCGrid 2001 IEEE International Symposium on Cluster Computing and the Grid*, 15-18 May 2001, Brisbane, Australia

[Saty 01] M. Satyanarayanan, Pervasive Computing: Vision and Challenges. IEEE Personal Communications. August 2001.

[Saty 96] M. Satyanarayanan, Fundamental Challenges in Mobile Computing. Proc. 15th ACM Symposium Principles of Dist. Computing, Philadelphia, PA, May 1996.

[Saty 96a] M. Satyanarayanan. Accessing information on demand at any location. Mobile information access. *IEEE Personal Communications.* 1996

[SAW 94] B. Schilit, N. Adams, R. Want. Context-aware computing applications. *In the Proceedings of the 1st International Workshop on Mobile Computing Systems and Applications.* Santa Cruz, CA. December 1994

[SAX 00] SAX2/Java. The simple API for XML. Megginson Technologies Ltd., http://www.megginson.com/SAX/Java/index.html. May 2000.

[Sch 00] A. Schmidt. Implicit Human Computer Interaction Through Context. Personal Technologies Volume 4(2&3), June 2000.

[Sch 86] B. Schäfters. Grundbegriffe der Soziologie. UTB Leske. Opladen. 1986

[Sch 95] W. Schilit. *System Architecture for Context-Aware Mobile Computing.* PhD thesis. Columbia University. 1995

[SDA 99] D. Salber, A. K. Dey , G. D. Abowd. The context toolkit: Aiding the Development of Context-Enabled Applications *Proceeding of the CHI 99 conference on Human factors in computing systems : the CHI is the limit,* May 1999

[Sentient] Sentient Computing Project Home Page. http://www.uk.research.att.com/spirit/

[SG 01] A. Schmidt, H.-W. Gellersen. Nutzung von Kontext in ubiquitären Informationssystemen. *it+ti - Informationstechnik und technische Informatik, Sonderheft "Ubiquitous Computing -*

der allgegenwärtige Computer". 2001, Volume 43, Issue 02, p. 83-90.

[SG 01a] A. Schmidt, H.-W. Gellersen. Model, Architektur und Platform für Informationssysteme mit Kontextbezug. *Informatik Forschung und Entwicklung.* Springer-Verlag. 2001.

[SGAP 00] K. Soharbi, J. Gao, V. Ailawadhi, G.J. Pottie. Protocols for self-organization of a wireless sensor network. *IEEE Personal Communications*, Volume 7, Issue 5. 2000.

[SL 01] M. Samulowitz, Y. Lin. Interaction Templates for Pervasive Service Chains. In Proceedings ICII 2001, Beijing, China, October 2001

[SL 99] M. Sloman, E. Lupu. Policy Specifications for Programmable Networks. *In First International Working Conference on Active Networks (IWAN'99),* Berlin, June 1999.

[Slo 94] M. Sloman. Policy driven Management for Distributed Systems. *Journal of Network and Systems Management*, 1994

[Smart] Smart-Its. http://www.smart-its.org

[SmartC] Smart Card. http://www.citi.umich.edu/projects/smartcard/

[SMLP 01a] M. Samulowitz, F. Michahelles, C. Linnhoff-Popien. Adaptive Interaction for Enabling Pervasive Computing Services. In: 2nd ACM International Workshop on Data Engineering for Wireless and Mobile Access (MobiDE'01) in cooperation with ACM SIGMOD, Santa Barbara, California, USA, May 20, 2001

[SMLP 01b] M. Samulowitz, F. Michahelles, C. Linnhoff-Popien. CAPEUS: An Architecture for Context-Aware Selection and Execution of Services. Third IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems (DAIS'2001), Krakow, Poland, September 17-19, 2001

[SMR] T. Starner, S. Mann and B. Rhodes The MIT wearable computing webpage. http://wearables.www.media.mit.edu/projects/wearables.

[ST 94] Bill N. Schilit, Norman Adams, and RoyWant. Context-aware computing applications. *In Proceedings Workshop on Mobile Computing Systems and Applications*. IEEE, December 1994.

[Stev 94] W. Richard Stevens. The Protocols (TCP/IP Illustrated, Volume 1). Addison-Wesley. January 1994.

[Sto 95] G. W. Strong. New directions in human-computer interaction. ACM Interactions. January 1995.

[SvL 01] A. Schmidt, K. Van Laerhoven. How to Build Smart Appliances? *IEEE Personal Communications 8(4),* August 2001.

[SWvG 00] J. Seemann, J. Wolff von Gudenberg. Software Entwurf mit UML. Springer-Verlag 2000.

[Sync 02] SyncML Data Synchronization Specification, version 1.2. http://www.syncml.org/technology.html. 2002

[T 93] G. E. Thaller. Computersicherheit. Nummer 18 in DuD Fachbeiträge. Vieweg-Verlag. 1993.

[T 97] A. Tebo. Sensor fusion employs a variety of architecture, algorithms, and applications. http://www.spie.org/web/oer/august/aug97/sensor.html

[Tan 96] A. Tanenbaum. Computer Networks. Prentice Hall. ISBN 0-13-349945-6. 1996

[TD 92] C. D. Tait, D. Duchamp. An Efficient Variable-Consistency Replicated File Service. Proceedings USENIX File Systems Workshop. May 1992

[Te 95] D. B. Terry et al. Managing Update Conflicts in a Weakly Connected Replicated Storage System. Proceedings 15th ACM Symposium Operating System Principles, Copper Mountain Resort. December 1995.

[Tenn 00] D. Tennenhouse. Proactive Computing. Communications of the ACM. May 2000

[TS 81] R. R. Tenney,N. R. Sandell, Jr. Detection with distributed sensors. IEEE Transactions on Aerospace and Electronic Systems, 1981.

[UPnP] Universal Plug and Play (UPnP). http://www.upnp.org

[URL] Uniform Resource Locators. http://www.w3.org/Addressing/URL/Overview.html

[W 98] M. Weber. Verteilte Systeme. Spektrum Akad. Vlg., Hdg. 1998.

[WAP 02] The Wireless Application Protocol (WAP). http://www.wapforum.org

[Weis 91] Mark Weiser. The Computer for the Twenty-First Century. *Scientific American*, September 1991.

[Weis 93a] Mark Weiser, "Hot Topics: Ubiquitous Computing" IEEE Computer, October 1993.

[Weis 93b] M. Weiser. Some Compute science issues in ubiquitous computing. Communications of the ACM 36, 7, July 1993.

computing. Communications of the ACM 36, 7, July 1993

[Weis 94] Mark Weiser. The world is not a desktop. *Interactions*. January 1994.

[WG 00] Wang, Z, Garlan, D. Task-Driven. Technical Report, CMU-CS-00-154, School of Computer Science, Carnegie Mellon University, May 2000

[WH 99] J. Weatherall, A. Hopper. Predator. A Distributed Location Service and Example Applications.

[WHFG 92] R. Want, A. Hopper, V. Falcao, and J. Gibbons. The active badge location system. ACM Trans. On Info. Sys., vol. 10, no.1, pp. 91-102, January 1992

[WHK 97] M. Wahl, T. Howes, S. Kille. Lightweight Directory Access Protocol (v3). IETF RFC 2251. December 1997.

[WINS] The WINS Project. http://www.janet.ucla.edu/WINS/

[WJH 97] Ward, Andy, Alan Jones and Andy Hopper (1997). A new location technique for the active office. IEEE Personal Communications 4(5): pp. 42-47. October 1997.

[WJH 97] A. Ward, A. Jones, A. Hopper. A New Location Technique for the Active Office. In *The Proceedings of MobiCom'97. September 1997.*

[WO 98] J. Wing, J. Ockerbloom. Respectful Type Converters. IEEE Transations on Software Engineering, November 1998.

[Woo 96] A. Wood. CAMEO: Supporting Observable APIs. *Position Paper for the WWW5 Programming the Web Workshop.* April 1996.

[Woo 98] L. Wood. Document Object Model (DOM) Level 1 Specification. W3C Recommendation, October 1998.

[WOS] WorldOS. Routing – Directed vs. Undirected. http://worldos.com/technology/routing.php

[WS 01] R. Want, B. Schilit. Expanding the Horizons of Location-Aware Computing. IEEE Computer, August 2001.

[X4C] XML4C. http://www.alphaworks.ibm.com/tech/xml4c

[XMLPro] XML Protocol Abstract Model. W3C. http://www.w3.org/TR/xmlp-am/

[XMLUse] XML Protocol Usage Scenarios. W3C. http:// www.w3.org/TR/xmlp-scenarios/

[XRPC] UserLand Software, Inc. XML-RPC. http://www.xmlrpc.com/

[XSS] The XML Software Site. http://www.xmlsoftware.com/parsers/

[XSZD 99] Y. Xu, D. Sauquet, E. Zapletal, P. Degoulet. Using XML in a generic model of mediators. In *XML Europe 1999. Granada. Spain 1999.*

[XWN 00] D. Xu, D. Wichadakul, K. Nahrstedt. Resource-Aware Configuration of Ubiquitous Multimedia Service. In *Proceedings of IEEE International Conference on Multimedia and EXPO2000 (ICME 2000),* July 2000.

[ZERO 02] Zero Configuration Networking (zeroconf). http://www.ietf.org/html.charters/zeroconf-charter.html. 2002

Lebenslauf

Persönliche Daten

Name:	Michael Samulowitz
Anschrift:	Klugstr. 114 80637 München
Geburtsdatum:	27. Oktober 1971
Geburtsort:	Würselen
Familienstand:	ledig

Ausbildung

1978 – 1982	Grundschule Bischofstraße, Stolberg-Büsbach
1982 – 1991	Goethe-Gymnasium, Stolberg
1992 – 1998	Studium der Informatik an der RWTH Aachen
1999 – 2002	Promotion im Fachbereich Informatik an der Ludwig-Maximilians-Universität München

Wehrdienst

1991 – 1992	Grundwehrdienst in Gerolstein und Aachen

Berufliche Tätigkeiten

1994 – 1997	Studienbegleitende Tätigkeiten 1994 – 1996: Freie Mitarbeit für Hewatt GmbH 1996 – 1997: Studentische Hilfskraft, RWTH Aachen
1998 – 1999:	Freie Mitarbeit für Hewatt GmbH
1999 – 1999	Berater, Debis Systemhaus