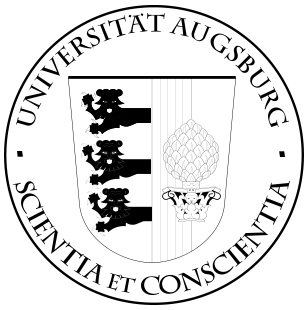

Reflektive mobile Agenten in ubiquitären Systemen

Dissertation

zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften

der Fakultät für Angewandte Informatik

der Universität Augsburg

eingereicht von

Dipl. Inform. Faruk Bagci

Erstgutachter: Prof. Dr. Theo Ungerer
Zweitgutachter: Prof. Dr. Bernhard Bauer

Tag der mündlichen Prüfung: 9. Dezember 2005

Zusammenfassung

Ubiquitäre Systeme werden als die nächste Generation der Computer gepriesen. Zukünftige Rechner werden unsichtbar und unaufdringlich in alltägliche Objekte integriert sein. Sie werden dem Menschen ungeahnte neue Dienste zur Verfügung stellen, indem sie Information aus der Umgebung sammeln und diese automatisch zur Unterstützung des Menschen verarbeiten. Um eine optimale Anpassung und damit einen angemessenen Mehrwert für den jeweiligen Benutzer bieten zu können, bedarf es auch persönlicher Informationen und der Information über den Aufenthaltsort des Benutzers. Mit neuen Arten von Geräten, wie Sensoren und *wearables*, und neuen Formen der Information wird zwangsläufig auch die Menge der Daten rapide ansteigen. Die effiziente Verarbeitung der Daten durch ein zentralisiertes System wird kaum mehr möglich.

Das Paradigma der mobilen Agenten bietet eine komfortable Methode, persönliche Interessen des Benutzers und die Anforderungen eines ubiquitären dezentralen Systems zu vereinen. Mobile Agentensysteme bilden von Grunde auf die höchste Stufe der Dezentralisierung. Persönliche Informationen, wie Präferenzen, Vorlieben und Gewohnheiten, können von mobilen Agenten gekapselt und für ortsbasierte Dienste im Namen des Benutzers eingesetzt werden.

Die Idee, die in dieser Arbeit vorgestellt wird, ist, dass der Benutzer von einem virtuellen Abbild in Form eines mobilen Agenten (so genannter *reflektiver Agent*) in der ubiquitären Umgebung begleitet wird. Der reflektive Agent kommt in einem neu entwickelten ubiquitären Agentensystem, dem Ubiquitous Mobile Agent System (UbiMAS), zum Einsatz. UbiMAS implementiert umfangreiche Sicherheitsfunktionen, die den reflektiven Agenten und damit die sensiblen persönlichen Daten des Benutzers schützen. Als ubiquitäre Umgebung wurde ein flexibles Bürogebäude mit elektronischen Türschildern, den *Smart Doorplates*, aufgebaut und für verschiedene Anwendungen der reflektiven Agenten eingesetzt.

Im praktischen Einsatz und verschiedenen Präsentationen hat die Implementierung Funktionsfähigkeit gezeigt. UbiMAS wurde sowohl auf rechenstarken PCs, als auch auf mobilen PDAs eingesetzt, die drahtlos über WLAN oder Bluetooth kommunizierten. Allerdings ist die volle Sicherheitsfunktionalität von UbiMAS zu dem aktuellen Stand der Technologie nur auf PCs realisierbar.

Vorwort

Diese Arbeit entstand in den Jahren 2002 bis 2005 am Lehrstuhl für Systemnahe Informatik und Kommunikationssysteme der Universität Augsburg. Die Visionen der ubiquitären Systeme weckten auch bei mir ein sehr großes Interesse. Die Erfahrungen über mobile Agenten aus dem Bereich der parallelen Verarbeitung resultierte in der Idee, zwei Paradigmen, das Paradigma der mobilen Agenten und das Paradigma des Ubiquitous Computing, zu kombinieren. In einer konspirativen Sitzung in der Villa Maison Blanche in Sion/Schweiz, wie sie jährlich vom Lehrstuhl organisiert wurde, entstand die Idee der *Smart Doorplates*. Somit war auch eine Basis für ubiquitäre Anwendungen geschaffen, auf denen mobile Agenten eingesetzt werden konnten. In erfolgreicher Zusammenarbeit konnte ein Prototyp der Smart Doorplates auf der *Nacht der Wissenschaft* im Juli 2003 vorgeführt werden. Die aktuelle Version ist im Flur des Lehrstuhls fest installiert, wobei die Arbeiten an neuen Anwendungen weiterlaufen. Die mobilen Agenten bilden einen essentiellen Bestandteil der Smart Doorplate Anwendungen und bieten bereits verschiedene Dienste an.

Ich möchte mich an dieser Stelle bei Herrn Prof. Dr. Theo Ungerer für seine erstklassige Betreuung und die vielen richtungsweisenden Diskussionen und Anregungen bedanken, ohne die diese Arbeit nicht möglich gewesen wäre. Bei Herrn Prof. Dr. Bernhard Bauer möchte ich mich für die Übernahme des Zweitgutachtens bedanken. Mein Dank geht auch an meine Kollegen Jan Petzold und Wolfgang Trumler für die gemeinsame Entwicklung und Umsetzung der Smart Doorplates Idee und für die hilfreichen Anregungen, die die Entwicklung dieser Arbeit vorangetrieben haben. Bedanken möchte ich mich auch bei Herrn Holger Schick, der mit seiner Diplomarbeit einen wertvollen Beitrag zu dieser Arbeit geleistet hat. Ein großer Dank geht auch an meine Frau Serpil Bagci für die mentale Unterstützung und das Korrekturlesen, wofür ich auch meinem Bruder Ahmet Bagci Dank schulde.

Augsburg, Oktober 2005 *Faruk Bagci*

Inhaltsverzeichnis

1 Einführung

Die Rechnerevolution hat bereits mehr als ein halbes Jahrhundert hinter sich gelassen. Dabei durchlebte der Computer mehrere Generationen. Vergleicht man die Entwicklung des Rechners mit anderen ausschlaggebenden Erfindungen des 20. Jahrhunderts, so fällt die mannigfaltige Wandlung des Computers sofort ins Auge. Die erste Zeit war durch die revolutionären Fortschritte im Computerdesign und der Maschinenprogrammierung geprägt. Neben der Hardware stellte sich schnell auch die Wichtigkeit der Datenverarbeitung hervor. Hard- und Software zusammen mündeten in einer neuen Technologie, der Informationstechnologie. Viele andere Bereiche wurden stark durch den Rechner beeinflusst. In der Medizin, der Auto-Entwicklung, dem Bankwesen und in der Unterhaltungselektronik ist der Computer nicht mehr wegzudenken.

Es bahnt sich nun eine neue Rechner-Ära an, die ubiquitären Systeme. Die Anzahl der Mikrochips im Umfeld des Menschen nimmt mehr und mehr zu. Bereits heute ist der Mensch von intelligenten Geräten umgeben, wie Handy, PDA, Navigationsgerät im Auto. Die Größe und Leistung der Mikrochips macht einen vielseitigen Einsatz möglich. Die Zahl wird sich höchstwahrscheinlich in den kommenden Jahren vervielfachen.

Zukünftige Computer werden in alltägliche Objekte integriert sein. Der Benutzer wird die Existenz solcher Chips in seinem Alltag nicht direkt wahrnehmen. Dies ist ein Ziel der ubiquitären Systeme, unsichtbar und unauffällig zu sein. Die gewohnte Umgebung wird sich nur bedingt ändern. Sicherlich werden mit fortschreitender Technologie neue Geräte entwickelt werden, die sich in den menschlichen Nutzbereich eingliedern werden. Touchscreens, intelligente Projektoren, elektronisches Papier oder intelligente Kleidung bzw. am Körper angebrachte Computer (so genannte *wearables*) sind hier nur einige Beispiele. Damit sich das ubiquitäre System bestmöglichst selbstständig an die aktuellen Begebenheiten anpassen kann, müssen Informationen aus dem Verhalten des Menschen und seiner Umgebungssituation automatisch erfasst werden. Sensoren können einen Teil dieser Informationen liefern. Es beschäftigen sich zahlreiche Projekte mit der Entwicklung von intelligenten Einzel- oder Multisensorboards, die untereinander drahtgebunden oder drahtlos kommunizieren können. Zusätzlich zu den Daten, die direkt aus der Umgebung gewonnen werden können, spielen persönliche Informationen, wie Profile, Präferenzen, Vorlieben und Gewohnheiten eine große

Rolle und sind daher unerlässlich.

Bei der Vielzahl der Rechner im ubiquitären System stellt sich das Problem, wie all diese Daten gespeichert und verarbeitet werden können, und auf welche Art diese Daten verschiedenen Dienste zur Verfügung gestellt werden können. Erschwerend kommt hinzu, dass der Mensch naturgemäß mobil ist, und die nötigen Daten immer am aktuellen Ort vorliegen sollten. Diese Tatsachen führen dazu, dass ein klassischer zentralistischer Aufbau des ubiquitären Systems nahezu unmöglich wird. Auch wenn nur einzelne Bereiche ubiquitär ausgestattet wären, wie zum Beispiel die Wohnung, würde der zentrale Ansatz bei der großen Anzahl an Knoten eines solchen *Smart Environment* fehleranfällig und unsicher sein. Der Server könnte bei der Vielzahl an Clients und Diensten, die auf dem ubiquitären System laufen, leicht zum Flaschenhals werden. Ferner könnte ein Fehler im Server zum Ausfall des gesamten Systems führen. Außerdem drängt sich bei einem Ein-Server-System ein Big-Brother-Szenario auf, wo die Gefahr bestünde, dass bei einem unbefugten Zugriff auf den Server alle persönlichen Informationen und Alltagsabläufe ausgespäht werden könnten.

Die bessere Alternative stellt ein dezentral aufgebautes System dar. Sowohl Dienste als auch Daten können hierbei dezentral auf viele Knoten verteilt werden. Ein dezentrales System ist zudem weniger fehleranfällig, da bei einem Ausfall eines Knoten, die anderen Knoten ihre Funktionalitäten aufrecht erhalten können. Das Problem des unautorisierten Zugriffs auf Daten besteht auch auf dem dezentralen System. Dies könnte jedoch weniger Folgen haben, wenn sich nur Datenfragmente auf verschiedenen Knoten befinden würden, die wenig Informationen preisgeben.

Beim Wechsel des Aufenthaltsorts stellt sich auch bei einem dezentralen System die Frage, wie die persönlichen Informationen an die richtige Stelle gelangen. Hierbei gibt es die Möglichkeit, dass der Benutzer die Daten mit sich trägt und bei Bedarf automatisch an das ubiquitäre System in der Umgebung weiter gibt. Der Benutzer muss dazu ein elektronisches Gerät, wie Laptop, Tablet PC, PDA oder Wearables mit sich führen. Solche Geräte sind gewöhnlich an eine einzige Person gebunden. Sicherheit und Geheimhaltung liegen daher in der Verantwortung dieser Person. Der Nachteil dieser Methode ist, dass Menschen oft aus Unbequemlichkeit die Geräte nicht mit sich tragen oder achtlos herumliegen lassen. Eine alternative Methode ist hier, die Verantwortung für die Speicherung und Übertragung der persönlichen Information an das ubiquitäre System weiterzugeben, wobei das System für die Sicherheit der Daten vor unbefugten Zugriff Sorge tragen muss. Der Benutzer wird also quasi begleitet durch ein mobiles virtuelles Abbild in der ubiquitären Umgebung.

Führt man den dezentralen Ansatz und die Idee des virtuellen Abbildes zusammen, bietet sich eine aus anderen Bereichen der Informatik bekannte Technologie an, die geradezu ideal auf dieses Szenario passt: das Paradigma der mobilen

Agenten. Allgemein versteht man unter einem Agenten, ein Programm, genauer eine Softwareeinheit, die im Auftrag eines Nutzers oder eines anderen Agenten bestimmte Dienste ausführt und dabei eine gewisse Autonomie besitzt, um an Stelle des Nutzers zu handeln. Mobile Agentensysteme charakterisieren sich zusätzlich durch die Fähigkeit der Agenten autonom zu migrieren, d.h. sie können Programmcode, Daten und Ausführungszustand an einen entfernten Rechner transferieren, um mit dem Programmablauf dort fortzufahren. Neben dieser physikalischen Mobilität haben mobile Agenten auch die Möglichkeit, miteinander zu kommunizieren, um Informationen auszutauschen.

Betrachtet man mobile Agenten im ubiquitären Umfeld, so stellt der Agent ein virtuelles Abbild des Benutzers dar, der persönliche Informationen mit sich trägt. Auf Grundlage dieser Daten kann der reflektive Agent selbständig im Namen seines Benutzers verschiedene Dienste ausführen. Zudem kann der mobile Agent über geeignete Schnittstellen Informationen über die Umgebung erhalten, die von dem lokalen ubiquitären System angeboten werden. Agentensysteme können plattformunabhängig programmiert werden, so dass der Einsatz auf heterogenen Systemen gewährleistet ist.

Mobile Agenten können verschiedenste intelligente Funktionen mit sich führen. Diese können dazu benutzt werden, auch Sicherheitsfunktionen im Agentensystem und in den Agenten zu implementieren. So können persönliche Daten im reflektiven Agenten gekapselt werden, um sie vor unbefugtem Zugriff zu schützen.

Genauso wie die Anhänger der mobilen Agenten diese Technologie anfänglich als revolutionär preisten und deren Stellenwert ähnlich der objekt-orientierten Programmierung stuften, genauso gibt es Skeptiker, die keine Vorteile in dem Einsatz von mobilen Agenten sehen. Diese Skepsis wurde auch durch die fehlende Sicherheit der entwickelten Agentensysteme vor allem im Bereich des Internets und des E-Commerce untermauert. Sicherheit ist jedoch in einem grundsätzlich offenen System wie dem Internet immer noch ein ungelöstes Problem. Zugegeben es wurden dieselben Fehler gemacht, wie das WAP als Internet für das Mobiltelefon zu vermarkten. Tatsache ist, dass das mobile Agentenparadigma eine innovative Methode für die Entwicklung verteilter Systeme darstellt. Die Vorteile sind viel offensichtlicher in einem geschlossenen System zu erkennen, wo auch die Sicherheit einfacher in Griff zu bekommen ist.

Die vorliegende Arbeit greift die Idee der reflektiven Agenten auf und untersucht deren Einsatz in einer realen ubiquitären Umgebung. Hierzu wurde ein ubiquitäres Büroumfeld geschaffen, wo jede Tür mit intelligenten Türschildern ausgestattet ist. Ein ubiquitäres mobiles Agentensystem, *UbiMAS* genannt, wurde entwickelt, welches als Plattform für die reflektiven Agenten dient. Neben den reflektiven Agenten, die die Bürobewohner repräsentieren, wurden Service-Agenten implementiert, die verschiedene Dienste anbieten und von den reflektiven

Agenten in Anspruch genommen werden können.

Ziele der Arbeit sind die Anwendbarkeit und Implementierung des Paradigmas der reflektiven mobilen Agenten in ubiquitären Systemen (genauer Smart Environment) zu untersuchen durch prototypischen Entwurf eines ubiquitären Agentensystems. Als wesentlich hat sich eine geeignete Sicherheitsarchitektur herausgestellt. Sicherheit spielt in einem ubiquitären Umfeld eine sehr große Rolle, wo es um Daten geht, die direkt aus persönlichen Verhalten, Gewohnheiten, Vorlieben und Profilen des Menschen gewonnen werden. Diese Informationen bedarf es vor unauthorisiertem Zugriff zu schützen, ohne die Bewegungsfreiheit des Benutzers einzuschränken. Im Fokus des in dieser Arbeit entwickelten Systems stehen Sicherheitsmechanismen zum Schutz von Agenten und Agentenknoten.

Das nächste Kapitel stellt die Visionen der ubiquitären Systeme vor und beschreibt das Paradigma der ubiquitären Systeme mit ihren Anforderungen und Voraussetzungen. Kapitel 3 beschäftigt sich mit den mobilen Agenten, indem zunächst der Begriff des Agenten abgegrenzt wird, um dann die Entstehung des mobilen Agenten zu erklären. In diesem Kapitel wird auch die Idee des reflektiven Agenten in ubiquitären Systemen eingeführt. Kapitel 4 behandelt die Sicherheit in mobilen Agentensystemen, indem zunächst Sicherheitsanforderungen untersucht werden und eine umfangreiche Bedrohungsanalyse durchgeführt wird. Das darauf folgende Kapitel beschreibt verwandte Arbeiten und stellt diese dem in dieser Arbeit entwickelten Agentensystem UbiMAS nach festgelegten Kriterien entgegen. Kapitel 6 beschreibt das Ubiquitous Mobile Agent System Framework und eine Implementierung einer ubiquitären Middleware. In Kapitel 7 wird die eigentliche UbiMAS-Host Architektur detailliert beschrieben und die implementierten Sicherheitsmechanismen vorgestellt. Im darauf folgenden Kapitel werden ubiquitäre Anwendungen auf Basis von UbiMAS innerhalb des Smart Doorplate Projektes beschrieben und Ergebnisse von Leistungsmessungen präsentiert. Die Arbeit schließt mit einer Zusammenfassung und einem Ausblick ab.

2 Ubiquitäre Systeme

Dieses Kapitel beschreibt die Vision der ubiquitären Systeme und zählt Paradigmen auf, die sich während der Forschung im Ubiquitous Computing herauskristallisiert haben. Es beginnt mit einer kurzen Geschichte, die einen Auszug aus dem Leben eines Menschen in der Zukunft schildert, um die visionären Ideen besser veranschaulichen zu können.

2.1 Ära der ubiquitären Systeme

Ein ubiquitärer Morgen
Der Intervallton wollte einfach nicht nachgeben und entriss ihn langsam von seinen Träumen. Er öffnete blinzelnd seine Augen und sah in seinem Blickfeld die Uhrzeit, die der Wecker an die Wand projizierte. Einen kurzen Moment wunderte er sich, dass er so früh geweckt wurde. „Es wird schon einen Grund haben“, dachte er sich. Sein Gedächtnis machte ihm in seinem hohen Alter immer mehr Probleme. Da war er froh, dass er Hilfe aus seiner Umgebung bekam. Er stand langsam auf und der Weckton wurde durch eine angenehme leise Musik abgelöst. Die Jalousien öffneten sich spaltbreit und das Licht der aufgehenden Sonne erhellte den Raum. Er ging wie jeden Morgen mit schleifenden Schritten ins Bad, wo es schon wohlig warm war. Die Dusche spülte den letzten Rest Schlaf aus seinem Körper und als er danach die Küche betrat, begrüßte ihn schon der milde Kaffeegeruch. Er nahm sich gleich eine Tasse und scrollte auf dem digitalen Papier zur Spalte „Aktuelles“, wo er sich immer die aktuellen Nachrichten beim Frühstücken durchlas. An erster Stelle konnte er auch schon den Grund seines frühen Aufstehens lesen: Sie haben heute um 8 Uhr einen Termin beim Arzt. Wahrlich, das hatte er vergessen. „Gut, dass mein Wecker dran gedacht hat!“, flüstete er vor sich hin.

Es war nicht immer so einfach mit den Computern. Er wusste das nur allzu gut, denn er war einer der wenigen Menschen, die jede der drei Computerären aus direkter Nähe miterlebten. Sein erster Job war die eines Operators. Damals füllten Computer auch Häuser, aber es

> waren nicht tausende pro Haus, sondern nur ein einziger Computer, der in einen größeren Saal hineinpasste. Mit seinem ersten Lohn kaufte er sich gute Sportschuhe, da er als Operator viel auf den Beinen war. Schnell musste er sein. Bei der Einstellung hatte man auf seine Sportlichkeit geachtet. Die Schnelligkeit des Großrechners hing nicht selten von seiner Schnelligkeit ab. Wenn da mal ein Band gewechselt werden musste und der Drucker einen Papierstau hatte, musste er schon einen 50-Meter-Sprint hinlegen. Er war verantwortlich für die korrekte Ausführung der Programme von mehreren hundert Programmierern, die alle auf den einen Rechner angewiesen waren. Dies änderte sich aber rasant. Innerhalb von nur 20 Jahren wurden die Rechner so klein und günstig, dass sie sich jeder leisten konnte. Das machte den Umgang mit dem Computer auch nicht einfacher.
>
> „Und nach dem Arztbesuch kann ich auch gleich einkaufen“, dachte er sich beim Blick in den Kühlschrank. Der Kühlschrank könnte ihm das Einkaufen zwar ersparen, indem er die fehlenden Waren automatisch bestellte, doch das wollte er bewusst nicht. Er schloss wieder die Kühlschranktüre und war sich sicher, dass der Kühlschrank ihm eine Einkaufsliste schicken würde, den er auf dem Display des Einkauswagens sehen könnte. Er zog sich Schuhe und Mantel an und als er sich der Tür näherte, hörte er Donner- und Regengeräusche. Unbewusst griff er zum Regenschirm und trat nach Draußen. Überrascht hielt er kurz inne, denn es war ein strahlender Frühlingsmorgen mit nur wenigen Wolken am Himmel. Mit einem leichten Lächeln auf den Lippen ging er kopfschüttelnd die Straße hinab und bemerkte gar nicht mehr die Werbetafel, die ihn mit dem Namen begrüßte.

Diese kleine Geschichte beschreibt eine Vision, die sich in Teilen vielleicht in wenigen Jahren verwirklichen wird. Es beschreibt einen Ausschnitt aus dem Tagesablauf eines Menschen im Zentrum eines ubiquitären Systems. *Ubiquitär* bedeutet allgegenwärtig. Es steht für ein Computersystem, wo sich im Umfeld eines einzigen Menschen eine große Anzahl von Mikroprozessoren befindet. Die heutige Zeit ist die Anfangsphase dieser anstehenden Computer-Ära. Schaut man sich um, so sieht man sich heute schon von mehreren Rechnern umgeben. PCs, Laptops, PDAs, Mobiltelefone und Navigationssystem im Auto sind nur einige Bespiele davon. Wie in der Geschichte angedeutet, weist die Historie des Computers verschiedene Evolutionsphasen auf. Die erste Phase ist die der Großrechner, die riesige Räume füllten, und sich nicht selten auch über mehrere Etagen ausbreiteten. Die Rechnerdinosaurier waren sehr teuer, so dass sie sich nur große Institutionen leisten und betreiben konnten. Operatoren wurden eingestellt, die diese Computer bedienten. Die Benutzer waren fast ausschließlich Programmierer.

Mit der Zeit wurden die Computer immer kleiner und billiger. Anfang der 90er Jahre nahm dann der Personal Computer, auch unterstützt durch die schnelle Verbreitung des Internet, Einzug in fast jedes private Arbeitszimmer. Dies läutete die zweite Rechner-Ära an, wo auf jeden Benutzer mindestens ein Computer fiel. Die Arbeit des Operators wurde automatisiert und durch das Betriebssystem ersetzt. Auch der Benutzerkreis weitete sich aus. Es waren jetzt nicht mehr Programmierer die typischen Benutzer, sondern Privatpersonen, die den Rechner ganz und gar nicht zur Programmierung nutzten. Die modernen Benutzer spielten Computerspiele, surften im Internet, erledigten ihre Bankgeschäfte, schrieben Briefe, schauten sich Filme an oder hörten Musik. Der Computer wurde zum alltäglichen Gebrauchsgegenstand. Die Beziehung Rechner-Benutzer hatte sich damit auch gewandelt. Der Benutzer stand nun im direkten Kontakt zum Computer. Auch die Kommunikation der Rechner machte Fortschritte. Neben dem weit verbreiteten Internet ermöglichten neue Funk-Technologien den Bentzern mobil und immer erreichbar zu sein. Neue Technologien eröffneten die Türe für neue Dienste. Längst wurde der potentielle Benutzer nicht nur mit Hardware gelockt, sondern auch mit neu entwickelter Software. Was aber blieb war die Tatsache, dass der Computer zwar intelligenter, aber nicht selbständiger wurde.

Die fortschreitende Entwicklung im Bereich der Hardware und Software brachte Forscher auf neue Visionen für zukünftige Computer-Technologien. Das Moore'sche Gesetz vom exponentiellen Wachstum der Leistung von Mikroprozessoren scheint ungebrochen seine Gültigkeit aufrecht zu halten. Die steigende Leistung geht auf dem Markt einher mit niedrigeren Preisen für Mikroprozessoren. Es gibt neben der Leistung auch Fortschritte in der Größe der Chips. Die Miniaturisierung der Hardware ist jetzt schon so weit, dass Prozessoren in andere Gegenstände *eingebettet* werden können. Moderne Autos sind heutzutage ein gutes Beispiel für solche eingebetteten Systeme, wo verschiedene Funktionen wie der Bremsassistent oder die Einspritzsteuerung des Motors von eigenständigen Mikroprozessoren gesteuert werden. Die Vision der ubiquitären Systeme beschreibt auf dieser Entwicklungsgrundlage die nächste Rechner-Ära, in der Computerleistung und Zugang zu digitalen Kommunikationsmedien überall verfügbar wird, genauer hinter all den Alltagsgegenständen verborgen, hinter den man es heute gar nicht vermuten würde. Damit ist auch die Entscheidung der Namensgebung *ubiquitär* begründet. Dieser Begriff wurde 1991 von Mark Weiser, dem damals leitenden Wissenschaftler am Forschungszentrum von XEROX in Palo Alto geprägt [Wei91]. Er definierte das Ziel des Ubiquitous Computing wie folgt:

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

Die Unsichtbarkeit des Computers wird auch die Beziehung von Mensch zum Rechner verändern. Die Kommunikation mit dem Computer ist heutzutage nicht sehr intuitiv. Meist sind Programme für den Durchschnittsmenschen, der nicht aus der IT-Branche kommt, nur sehr schwer zu verstehen und zu bedienen. Der Computer bietet bisher dem Menschen nur beschränkt Unterstützung in seinem Alltagsleben. Mit der Mobiltelefonie sind zwar Dienste zum Schreiben, Lesen und Versenden von Nachrichten, die auch Multimedia-Komponenten beinhalten können, erschienen, jedoch sind diese Dienste noch weit von der selbstständigen Erfassung von und Anpassung an Benutzerwünschen und -gewohnheiten entfernt. Der Computer reagiert immer noch nur auf direkte Eingaben des Menschen, und wenn diese nicht eindeutig sind, ist der Konflikt vorprogrammiert. Daher empfinden viele Menschen den Umgang mit dem Computer eher als Belastung und nicht als Hilfe.

Ubiquitäre Systeme versprechen dies zu ändern. Der Computer soll selbstständiger werden und von sich aus auf Ereignisse reagieren können, ohne dass der Mensch explizit eingreifen muss. Dies alles muss in der realen Welt geschehen, d.h. der Mensch soll nicht in eine virtuelle Welt hineingezogen werden, die nur im Computer existiert. Um die selbstständige Reaktion von Rechnern realisieren zu können, bedarf es neuer Technologien, die Informationen aus dem Verhalten des Benutzers und aus seiner Umwelt gewinnen und diese interpretieren können. Klassische Eingabegeräte wie Tastaturen, Mäuse oder Touch-Screens, reichen für solche Szenarien nicht aus und sind nicht intuitiv. Besser sind hier sprachgesteuerte Systeme, allerdings erwarten solche Systeme auch Kommandos vom Menschen und verstoßen daher gegen das Selbstständigkeitsprinzip des Ubiquitous Computing.

2.2 Paradigmen im Ubiquitous Computing

Es haben sich vier fundamentale Paradigmen im Ubiquitous Computing herauskristalisiert [HNS01]:

- Dezentralisierung
- Divesifizierung
- Konnektivität
- Simplizität

Dezentralisierung begann schon in der Mainframe-Ära als die entfernten Terminals eingerichtet wurden. Die PC-Generation führte eine weitergehende Dezentralisation mit dem Client-Server-Prinzip ein. Ubiquitäre Systeme verteilen

Aufgaben, Funktionen und Entscheidungen auf viele kleine Computer und gehen damit noch einen Schritt weiter. Informationen, die zur Erfüllung der Aufgaben notwendig sind, werden nicht mehr zentral gespeichert, sondern liegen weit verteilt im dezentralen System. Dies bringt jedoch auch neue Probleme mit sich, die im Ubiquitous Computing als Herausforderung angenommen werden wie die Synchronisation der Informationen und das Management der verteilten Applikationen.

Das zweite Paradigma wirkt sich auf die Funktionalität von Computersystemen aus. Heutzutage gleicht ein Computer einer Universalmaschine. Die Einsatzgebiete desselben Rechners können stark variieren. So kann ein Computer, der fast ausschließlich zur Textverarbeitung genutzt wird, zugleich einen Industrieroboter steuern. Ubiquitous Computing definiert eine Distanzierung von dieser Art Universalmaschine. Statt dessen sollen viele verschiedene Geräte hoch spezialisierte Aufgaben übernehmen. Computer in ubiquitären Systemen sind daher auf eine streng abgegrenzte Funktionalität beschränkt. Die Soft- und Hardware wird genau auf die gewünschte Aufgabe justiert. Die Herausforderung besteht darin, Anwendungen zu entwickeln, die sich automatisch diesen unterschiedlichen Geräten anpassen, die auch über unterschiedliche Ressourcen und Fähigkeiten verfügen, wie z.B. unterschiedliche Bildschirmgröße oder Prozessorleistung.

Die extreme Verteilung in ubiquitären Systemen setzt eine hohe Konnektivität der einzelnen Rechnerknoten voraus. Für die Kommunikation werden Medien wie Glasfaser, Infrarot und Funk eingesetzt. Um einen problemlosen Datenaustausch der heterogenen Knoten über verschiedenartige Medien zu gewährleiten, sind neue standardisierte Protokolle, spezialisierte Middleware und neuartige Frameworks notwendig.

Simplizität ist das letzte Paradigma der ubiquitären Systeme. Die hohe Anzahl der Computer im Umfeld des Menschen wird zu einem viel höheren Nutzerverhältnis im Vergleich zu heutigen PCs führen. Um die Akzeptanz von ubiquitären Systemen zu steigern, müssen Computer einfacher und intuitiver zu bedienen sein. Der Benutzer sollte sich im ubiquitären System natürlich zurechtfinden können, ohne einen Bedarf an zusätzlichen Schulungen zu benötigen. Das Hauptziel eines ubiquitären Systems ist dabei immer die selbständige Unterstützung des Menschen in seinem Alltag. Das System muss daher automatisch Entscheidungen treffen, wo es nur geht. Dies ist nicht immer einfach, da bei Fehlentscheidungen der Ärger des Menschen groß sein wird und die Akzeptanz dadurch sinken wird. Entscheidungen werden auf Grundlage der Umgebungsbedingungen und dem Verhalten der Person gefällt. Um Fehler zu vermeiden, ist eine genaue Erfassung dieser Informationen notwendig.

Um Informationen direkt aus der Umwelt zu erhalten, muss die Umwelt mit intelligenten Fühlern ausgestattet werden. Sensorik ist hier das richtige Stichwort.

Sensoren nehmen einen Zustand oder eine Zustandsänderung aus dem physikalischen, realen Umfeld auf und stellen diese Information elektronisch zur Verfügung, die vom ubiquitären System effizient verarbeitet werden muss. Hierzu bedarf es einer Modellierung der Umgebungsdaten. Der Abschnitt 6.3.2.4 führt einen solchen Modellansatz ein.

3 Mobile Agenten

Der Begriff *Agent* ist wohl eines der meist gebrauchten Termini in der Informatik. Dabei unterscheiden sich die Bedeutungen stark je nach Anwendungsgebiet wie die Künstliche Intelligenz, das Netzwerkmanagement, das E-Commerce oder die Verteilten Systeme. Eine einheitliche Definition des Agenten bereichsübergreifend ist daher fast nicht möglich.

Der Grund für die Beliebtheit des Begriffs könnte in der ursprünglichen Bedeutung des menschlichen Agenten und seiner Aufgaben liegen. Agenten stehen im Dienste anderer Personen und repräsentieren deren Interessen. Sie sind im gewissen Sinne Manager, die einem Menschen lästige aber unerlässliche Arbeiten abnehmen, für die man sich neben dem Berufs- oder Privatgeschäft kümmern müsste.

3.1 Geschichte der Agenten

Im Zuge der Industrialisierung tauchte schon sehr früh der Gedanke auf Maschinen in die Dienste von Menschen zu stellen. Romane und Theaterstücke beschrieben in den 20er Jahren *Roboter*, die menschenähnlich waren und den Menschen dienten.

Alan Turing beschäftigte sich 1948 mit formaler Logik auf der die Intelligenz von Maschinen basiert. Er hatte die Vision, dass Maschinen nach dem Vorbild des menschlichen Gehirns arbeiten werden und maschinelle Entscheidungen und Vorschläge gleichsam beachtet werden wie die eines Menschen [Tur50].

Alan Kay, ein großer Anhänger der Agententechnologie, beschreibt den Ursprung des Software-Agenten wie folgt:

„The idea of an agent originated with John McCarthy in the mid-1950's, and the term was coined by Oliver G.Selfridge a few years later, when they were both at the Massachusetts Institute of Technology. They had in view a system that, when given a goal, could carry out the details of the appropriate computer operations and could ask for and receive advice, offered in human terms, when it was stuck. An

agent would be a 'soft robot' living and doing its business within the computer's world. " [Kay84]

Die Agentenforschung begann ursprünglich im Bereich der Verteilten Künstlichen Intelligenz (Distributed Artificial Intelligence), der sich in drei Teilbereiche gliedert: Multi-Agenten-Systeme, Verteiltes Problemlösen und Parallele Künstliche Intelligenz. Software-Agenten wurden wie der Name schon andeutet in dem ersten Teilgebiet eingesetzt, was bis zur Entstehung dieses Teilgebiets zurückverfolgt werden kann. Carl Hewitt definierte 1970 das „Actor"-Modell, welches interaktive Objekte mit eigenem, internem Zustand beschrieb, die auf Nachrichten ähnlicher Objekte reagieren konnten [Hew77].

Ende der siebziger Jahre beschäftigte sich die Forschung mit deliberativen bzw. smarten Agenten, die ein explizites, symbolisches Modell der Welt abbildeten und Entscheidungen mittels symbolischer Argumentation fällten [Nwa96]. Die Arbeiten konzentrierten sich hierbei um die Zusammenarbeit und die Kommunikation zwischen einzelnen Agenten, die Aufgabenverteilung, Koordination und Konfliktlösung durch Argumentation. Diese Ansätze wurden als „Makro"-Aspekte verstanden, die die Gemeinschaft der Agenten in den Vordergrund stellten. Forschungen, die den Agenten als Individuum betrachteten, wurden als „Mikro"-Aspekte zusammengefasst. Mit Beginn der 90er Jahre häuften sich neue Forschungen, die den Lernfaktor des einzelnen Agenten hervorstellten.

Mit der Entwicklung des Internet und des World-Wide-Web (WWW) als globales Informationsportal entstand auch ein neuer Agententyp - der Internetagent. Es wurden diverse Varianten diesen Typs entwickelt wie z. B. Suchagenten, Informationsagenten, Shoppingagenten etc. Unzählige Agentensysteme wurden programmiert, die untereinander nicht kompatibel waren und fremde Agenten nicht akzeptierten. Die Foundation for Intelligent Physical Agents (FIPA) [Con03] und die Object Management Group (OMG) [Gro98] wurden gegründet, um Standards im Bereich der Agententechnologie zu schaffen.

Mobile Agenten sind aus der Zusammenführung von Techniken des mobilen Codes und der intelligenten Agenten entstanden. Sie bilden also eine Teilgruppe der Software-Agenten. Bevor näher auf das Thema der mobile Agenten eingegangen wird, ist es notwendig Agenten genauer zu klassifizieren.

3.2 Definition des Agenten

Wie schon oben erwähnt, ist eine einheitliche Definition des Software-Agenten nicht möglich. [FG97] betrachtet verschiedene Agentensysteme aus verschiedenen Bereichen der Informatik und versucht aus Gemeinsamkeiten der Agenten-

definitionen und Agenteneigenschaften eine einheitliche Klassifizierung zu finden. Software-Agenten haben demnach bestimmte Mindesteigenschaften zu erfüllen (siehe auch [BWZ98], [HR95], [VOO95] und [JW98]):

- **Umgebungseigenschaft:** Sie befinden sich innerhalb einer Umgebung oder sind selbst Teil von dieser.
- **Reaktivität:** Der Agent nimmt seine Umgebung wahr, arbeitet innerhalb deren Grenzen und reagiert auf seine Umwelt. Entweder verfügt der Agent über Sensoren, die ihm die Umweltinformationen liefern oder es handelt sich um einen deliberativen Agenten, der ein Modell der Umwelt besitzt.
- **Autonomie:** Die wichtigste Eigenschaft eines Agenten ist jedoch die Autonomie in seinem Handlungsablauf. Der Agent definiert anhand der Informationen aus seiner Umgebung bestimmte Teile seiner Agenda selbst, d.h. nicht jeder Ablauf ist durch den Auftraggeber bestimmt. Dabei kann der Auftraggeber eines Agenten ein menschlicher Benutzer sein oder ein anderer Agent. Die Autonomie hat natürlich auch Grenzen. Geht es um kritische Entscheidungen wird eine Rückfrage an den Benutzer des Agenten gestellt.
- **Fortlaufender Prozess:** Agenten sind fortlaufende Prozesse, d.h. sie müssen nicht ständig neu gestartet werden. Sie entscheiden selbstständig, was sie als nächstes tun und wann sie sich beenden.
- **Proaktivität:** Die Proaktivität geht über die Reaktivität hinaus. Agenten reagieren nicht nur auf Geschehnisse der Umwelt, sondern sind fähig zielgerichtet zu planen und zu handeln. Sie ergreifen also selbst die Initiative.

Eine Software muss also mindestens diese Eigenschaften erfüllen, um Agent genannt werden zu können. Agentensysteme auf dieser Grundlage können aber auch weitere Eigenschaften haben. Ein White-Paper von IBM definiert eine weitergehende Klassifizierung in Agententum, Intelligenz und Mobilität [GAA+95]. Abbildung 3.1 stellt diese Charakterisierung graphisch dar.

Agententum definiert dabei den Grad an Agenten-Autonomie. Der Agent repräsentiert den Benutzer gegenüber Computersystemen, Anwendungen und anderen Agenten. Die Bandbreite an Agenten geht dabei von fest-codierten Agenten bis hin zu Agenten als intelligente Werkzeuge.

Intelligenz beschreibt, in welchem Umfang ein Agent selbständig handeln, planen und lernen kann. Auch hier existiert eine breite Palette an Agenten, die von einfacher Intelligenz aufbauend auf logischer Kodierung bis hin zur künstlichen Intelligenz mit komplexen Lern- und Adaptionsmethoden reichen.

Mobilität stellt den Grad an Beweglichkeit des Agenten durch ein Netzwerk dar. Kann der Agent seinen Aufenthaltsort nicht wechseln, so wird er statisch oder

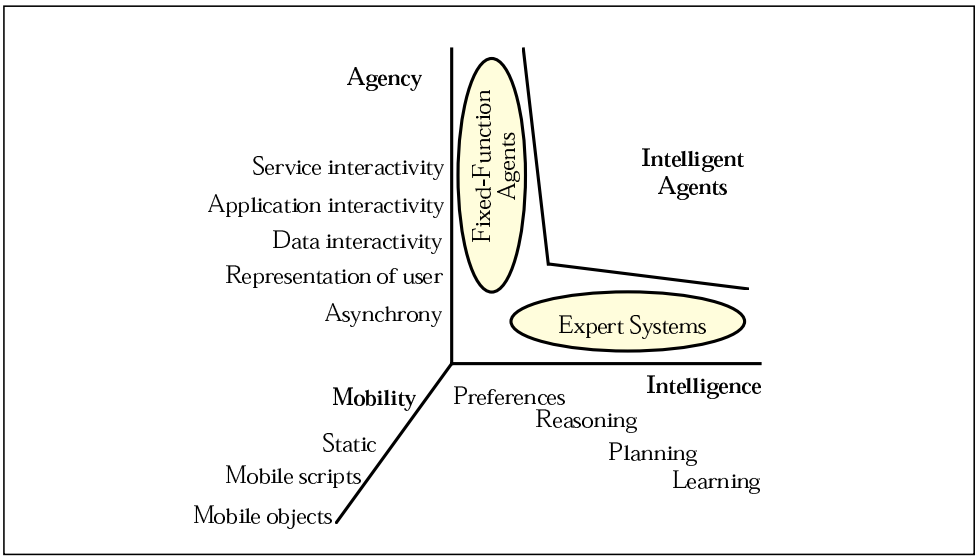

Abbildung 3.1: Bereich der Intelligenten Agenten

stationär genannt. Mobilität besitzt mehrere Stufen. Wird Code auf einer Maschine erzeugt und auf einem anderen Rechner ausgeführt, handelt es sich um ein mobiles Skript. Die letzte Stufe der Mobilität ist der mobile Code. Mobile Agenten bilden eine Teilmenge von diesen. Mobilität erhöht die Komplexität eines Agenten, da damit auch Aspekte wie die Sicherheit des Agenten, der Agentenknoten und Agentendaten, fehlerfreie Übertragung des Agenten, Vermeidung von Agenten-Klonen etc. zu bedenken sind.

3.3 Mobiler Code

Die Technologie des mobilen Code bildet die Basis für mobile Agenten. Ohne die Gewährleistung des Sendens und Ausführens von einem Stück Code auf einem entfernten Rechner könnten Agenten ihre Mobilität gar nicht verwirklichen. Die Programmiersprache Java demonstrierte in den letzten Jahren den Erfolg dieser Methode. Java bietet mit den Java Applets eine häufig benutzte Form, Code von einem Rechner zu einem anderen zu transportieren. Dabei ist die Idee der Code-Mobilität gar nicht so neu.

Die Job Control Language (JCL) [Bro98] aus den 60er Jahren ist die erste bekannte Realisierung dieses Konzeptes. Mit JCL konnten kleinere Rechner Aufgaben an einen zentralen leistungsfähigeren Großrechner senden, der diese Anfragen dann optimierte und ausführte. Kommandos in JCL enthielten Informationen, die zur richtigen Verarbeitung der Jobs beim Großrechner nötig waren. Auch die aus

dem Betriebssystem UNIX bekannte `rsh` (remote shell) [LMKQ02] ist ein Verfahren, der die Code-Mobilität ausnutzt. Mit `rsh` ist es möglich einen Skript zu einem entfernten Rechner zu senden, wo es dann ausgeführt wird mit Zugriff auf dort angeschlossene lokale Ressourcen wie Drucker oder Dateien auf der lokalen Festplatte des entfernten Rechners. Ein weiteres sehr bekanntes Beispiel für den erfolgreichen Einsatz von mobilem Code ist SQL (Structured Query Language) [GW89]. SQL wurde Ende der 70er Jahre entwickelt und 1986 zum ersten Mal standardisiert. Sie wird im Bereich der Datenbanken eingesetzt. Eine Anfrage wird in eine Nachricht gepackt und an einen entfernten Datenbank-Server gesendet, der die Anfrage intrepretiert und mit komplettem Zugriff auf die Daten verarbeitet. Das Ergebnis wird ensprechend gefiltert und an den anfordernden Client zurück gesendet. Ein weiteres Beispiel, bei dem es auf dem ersten Blick nicht ersichtlich ist, dass es sich auch um mobilen Code handelt, ist das aus der Büroumgebung bekannte Dokumentenformat PostScript [Inc99]. PostScript beschreibt in einer standardisierten Sprache, wie das zu druckende Dokument auszusehen hat. Jeder PostScript-fähige Drucker, auf den eine PostScript-Datei gesendet wird, interpretiert die Formatangaben in der gleichen Weise, baut die Seite entsprechend auf und druckt sie immer mit dem gleichen Aussehen aus.

Auch im Bereich der verteilten Systeme entdeckte man die Vorteile der Code-Mobilität. Hierbei war der Gedanke interessant, einen laufenden Prozess von einem Rechnerknoten auf einen anderen zu verlagern. Dies steigerte die Effizienz und eröffnete neue Möglichkeiten die Last in einem verteilten System besser und dynamischer auszugleichen. Traditionelle Methoden der Softwareentwicklung sind bei hoch skalierbaren Anwendungen für verteilte Systeme, bei denen es um Code-Mobilität und dynamische Rekonfiguration von Softwarekomponenten geht, nicht ausreichend. Die Behandlung von Softwareeinheiten auf demselben Rechnerknoten unterscheidet sich stark von der Behandlung von Komponenten auf entfernten Rechnern in Bezug auf Latenz, Informationszugriff, Fehlerbehandlung und Nebenläufigkeit. Hinzu kommt, dass entfernte Rechner nicht unbedingt homogen in Bezug auf Hardware, Betriebssystem und installierter Software sein müssen.

Die Einbeziehung von Mobilität führte in den 90er Jahren zu einer neuen Denkweise in der Programmentwicklung. Mobile Agenten kristallisierten sich als neues Paradigma für verteilte Systeme, in der Agenten als Objekte im Sinne einer objekt-orientierten Programmiersprache innerhalb eines heterogenen Rechnetzes migrierten. Die innovative Idee bei mobilen Agenten im Gegensatz zu anderem mobilen Code liegt in der Selbständigkeit des Agenten weiter zu migrieren, d. h. der mobile Agent initiiert seinen Transport selbst. Mobiler Code wie zum Beispiel ein Java Applet besitzt einen weiteren Unterschied: es migriert von der Quelle zum Ziel und verlässt diesen nicht. Dagegen migriert ein mobiler Agent normalerweise mehr als nur einmal.

Die Migrationsfähigkeit stand in den meisten Entwicklungen im Vordergrund, so

dass in vielen Entwicklungen Eigenschaften wie Autonomie oder Reaktivität bei der Definition von mobilen Agenten stark eingeschränkt waren oder sogar ganz hinausfielen. So war die Autonomie in vielen Projekten allein auf die autonome Migration des Agenten bezogen.

Um das Konzept des mobilen Codes von jeder Art per Interpreter ausgeführten höheren Programmiersprache abzugrenzen, unterscheidet man beim mobilen Code Ansatz zwischen Diensterbringern und Dienstbenutzern. Der Diensterbringer bietet eine Leistung an, die der Dienstbenutzer in Anspruch nehmen kann. Es haben sich drei Ansätze bei mobilem Code verbreitet [Vig98], die sich wie folgt charakterisieren lassen:

- **Remote Evaluation (REV):** Dieser Ansatz betrachtet den Fall, in dem ein Rechnerknoten A zwar das Service Know-How besitzt, aber nicht die Ressourcen hat, um diesen Dienst auszuführen. Der Dienst wird an einen entfernten Rechner B übertragen, der die nötigen Ressourcen aufweisen kann und dort ausgeführt. Die Ergebnisse werden dann an Rechner A zurückgesendet. Beispiel für REV ist rsh (Remote Shell) in UNIX, wo der Benutzer ein Skript-Code auf einem entfernten Rechner ausführen kann.
- **Code on Demand (COD):** In dem COD Ansatz besitzt Rechner A schon Ressourcen, benötigt aber das Know-How diese Ressourcen anzusprechen. Dieses Zugriffswissen fordert er von Rechner B an. Hier werden also Programme oder Teile davon bei Bedarf geladen. Typisches Beispiel für COD sind Java-Applets.
- **Mobile Agenten (MA):** Der Fokus bei REV und COD liegt beim Transfer von Code zwischen Rechnerknoten. Mobile Agenten unterscheiden sich hierbei in der Eigenschaft, dass nicht nur der Code übertragen wird, sondern auch Ausführungszustand und Daten. Der mobile Agent bringt also sich selbst und sein Wissen mit zur Ressource.

In [GV97] werden diese drei Ansätze mit dem Client-Server-Prinzip verglichen und dazu eine Semantik entwickelt. Der Code wird hierbei als Know-How, die Daten als Ressource bezeichnet. Die verschiedenen Architekturen unterscheiden sich in der Verteilung des Codes und der Daten in einem System mit mehr als einem Rechnerknoten.

3.4 Mobile Agenten

Obwohl sehr früh mit mobilem Code gearbeitet wurde, tauchte der Begriff *mobiler Agent* erstmals 1994 in dem White-Paper von General Magic Inc. auf [Whi94].

Die Idee war so reizvoll, dass es eine Welle an neuen Forschungen in der ganzen Welt auslöste.

Erste mobile Agentensysteme waren in verschiedenen Programmiersprachen geschrieben. General Magic bot sogar eine eigene Sprache für die Programmierung von mobilen Agenten an, das sogenannte *Telescript*. Die Verbreitung von Java als plattformunabhängige Sprache machte sich auch unter den Anhängern der mobilen Agenten beliebt, so dass heutzutage die meisten Agentensysteme in Java geschrieben sind. Abbildung 3.2 stellt die Entwicklungen in den Bereichen der Software Agenten und dem mobilen Code, die letztendlich in der Entstehung der mobilen Agenten mündeten.

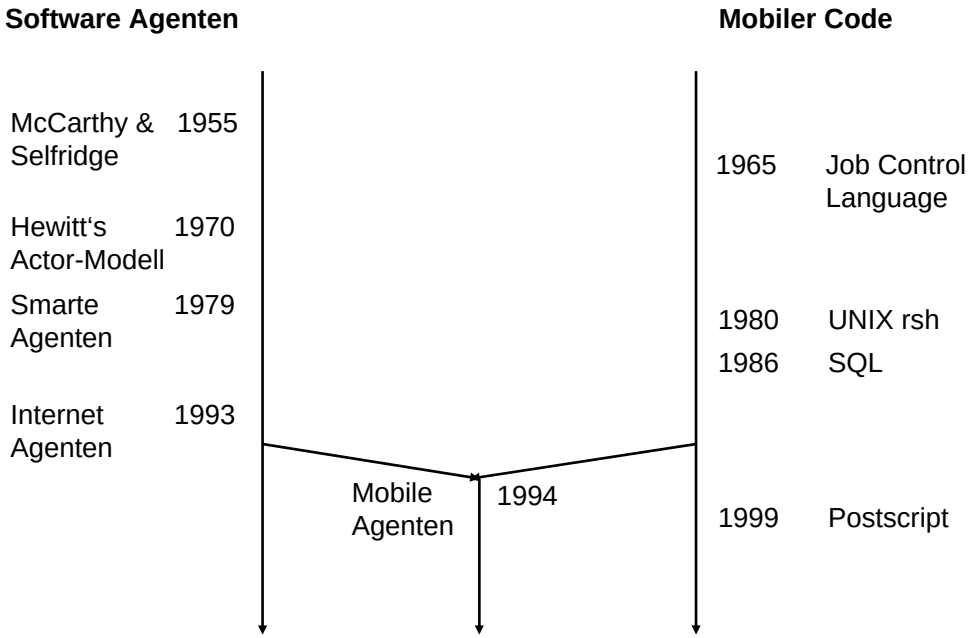

Abbildung 3.2: Zeittafel der Entwicklungen im Bereich Software Agenten, Mobiler Code und mobile Agenten

3.4.1 Eigenschaften von mobilen Agenten

Mobile Agenten bieten sehr interessante Eigenschaften, die für die anfängliche Euphorie sorgten. Das Paradigma der mobilen Agenten

- **unterstützt asynchrone und autonome Operationen,**
  Nach der Migration zur Informations- und Ressourcenquelle benötigt der mobile Agent keine aktive Verbindung zu seinem Besitzer oder anderen

Knoten. Eine Verbindung wird nur immer dann aktiviert, wenn der Agent seinen Auenthaltsort ändern möchte. Der Agent arbeitet also asynchron. In den meisten Client-Server-Ansätzen ist eine permanente Verbindung wegen eingehender Anfragen zwischen den Kommunikationspartnern vorgeschrieben, was bei mobilen Geräten jederzeit zu einem Problem werden kann. Das mobile Agentenparadigma bietet hier eine komfortable Alternative, da durch die autonome Operation des Agenten neue Anfragen direkt vor Ort gestellt und Resultate direkt bearbeitet werden können. Die Ergebnisse liegen dann bei der nächsten Verbindungsphase vor, ohne dass die verbindungslose Zeit untätig gewartet wurde.

- **ermöglicht dynamische und flexible Anpassung an sich verändernde Umgebungen,**
  Da der mobile Agent seine Funktionalität mitbringt, kann es sich je nach Umgebungsverhältnissen entsprechend verhalten. Ist ein bestimmter Dienst, den der Agent für seine Weiterarbeit bräuchte, auf dem aktuellen Knoten nicht vorhanden, kann er selbstständig entscheiden zu einem passenderen Knoten zu migrieren.

- **reduziert Speicherplatz,**
  Der mobile Agent trägt alles was er braucht mit sich. Bei der Migration werden alle Daten an den nächsten Knoten weitergeleitet und der Agent beendet, so dass er kein Speicher mehr belegt. Statische Server dagegen haben alle möglichen Dienste permanent geladen. In einer verteilten Umgebung, die ausschließlich aus statischen Servern besteht, müssen diese Dienste auf allen Servern geladen werden. Sie belegen daher fortwährend Platz auf allen Maschinen.

- **ermöglicht die Reduktion von Kommunikationskosten,**
  In den häufigsten Fällen der Datenkommunikation werden auf kurze Anfragen eine große Menge an Daten zurückgesendet. Diese Daten werden dann verarbeitet und meistens bleibt eine sehr kleine Resultatsmenge übrig. Diese Art von Kommunikation belastet das Netz sehr stark mit unrelevanten Daten. Mobile Agenten verarbeiten die Daten direkt an der Quelle und bringen nur die Ergebnisse zurück. Es ist aber hier zu beachten, dass der Overhead den der Code und die agenteneigene Daten hervorrufen, geringer sein sollte als die Rohdaten, die bei einer direkten Anfrage gesendet würden. Ist der Overhead zu groß, wirkt sich der Einsatz von mobilen Agenten nachteilig aus. Sei i die Anzahl der Interaktionen, d_i die Größe der Daten bei Kommunikation i, m die Größe der Migrationsdaten und r die Größe des Ergebnisses. Dann sind die Kommunikationskosten bei mobilen Agentensystemen geringer, wenn $\sum_{i=1}^{n} d_i > 2 * m + r$ gilt.

- **ermöglicht die Benutzung von verschiedenen Protokollen,**
  Im Internet werden je nach Medium und Applikation verschiedene Protokolle zur Kommunikation benutzt. Viele Schnittstellen sind zwar standardisiert, doch stellt sich der protokollübergreifende Austausch von Daten immer wieder als Problem dar. Mobile Agenten können so programmiert werden, dass sie je nach Kommunikationsschnittstelle das passende Protokoll benutzen, vergleichbar mit einem Dolmetscher, der viele verschiedene Sprachen beherrscht.

- **unterstützt fehlertolerante Ausführung,**
  Mobile Agenten können sich dynamisch an unerwartete Situationen anpassen. Diese Eigenschaft kann dazu benutzt werden, robuste und fehlertolerante Systeme aufzubauen. So können bei Überlastsituationen die Agenten selbst eine Balancierung des Systems einleiten, indem sie zu weniger frequentierte Knoten ausweichen. [PG03] untersucht die Lastverwaltung auf Basis von mobilen Agenten.

- **bietet eine Methode zur einfachen Entwicklung von neuen Diensten,**
  Der aufwendigste Teil ist die Entwicklung der Agentensystems an sich. Ist einmal ein Framework vorhanden, können neue Dienste mittels mobiler Agenten in kurzer Zeit programmiert werden. Neue Dienste können zudem im laufenden Betrieb installiert werden oder bestehende Applikationen erweitert und ergänzt werden.

Es gibt natürlich nicht nur Vorteile in der Nutzung mobiler Agenten. Einige Nachteile werden im Folgenden erläutert.

- Mobile Agenten brauchen eine Agentenplattform, um auf entfernten Rechnern ausgeführt werden zu können. Ohne eine ausreichende Infrastruktur von Agentenknoten ist eine Effizienzsteigerung gegenüber klassischen Client/Server Applikationen nicht zu erreichen.

- Bei zu einfachen Anfragen kann der Code des mobilen Agenten, der zu jedem Rechnerknoten übertragen werden muss, den der Agent besucht, auch leicht zu einer Netzbelastung führen. Agentencode ist in der Regel größer als eine einfache Client-Server Nachricht.

- Neben dem Code trägt der Agent Daten mit sich, die er unterwegs gesammelt hat und die vielleicht auf dem aktuellen und den zukünftigen Knoten nicht relevant für seine Ausführung sind, aber für das Gesamtergebnis unverzichtbar sind. Die Übertragung dieser Daten belastet den Netzverkehr zusätzlich.

- Da der Agent sensible Informationen mit sich trägt, bedarf es den Schutz dieser Daten vor dem Plattform-Rechner, auf den der Agent migriert.
- Auf der anderen Seite muss auch der Plattform-Rechner vor bösartigen Agenten geschützt werden. Dies wird meist durch die Ausführung des Agentenprogramms innerhalb einer Sandbox gewährleistet.

[CHK95] weist darauf hin, dass alles, was mit mobilen Agenten realisiert werden kann, auch mit anderen Methoden verwirklicht werden kann. Es wird aber betont, dass dies nicht in der Kombination und mit der Flexibilität möglich ist, wie es das mobile Agentenparadigma bietet. Dies ist vergleichbar mit der objekt-orientierten Programmierung. Alles, was in einer objekt-orientierten Programmiersprache geschrieben ist, kann auch in einer modularen oder funktionalen Programmiersprache nachprogrammiert werden. Allerdings dürfte dies viel schwieriger und komplizierter und mit mehr Zeitaufwand verbunden sein.

3.4.2 Infrastruktur für mobile Agenten

Damit mobile Agenten auf entfernten Rechnerknoten ausgeführt werden können, bedarf es einer Infrastruktur, die das mobile Agentenparadigma unterstützt. Es handelt sich dabei um ein verteiltes System, wo auf jedem einzelnen Rechnerknoten mindestens eine Agentenplattform, auch Agenten-„Host“genannt, gestartet werden muss. Der Zusammenschluss mehrerer Agentenplattformen bildet im Ganzen ein *Agentensystem*.

Ein Agenten-Host ist ein Dienst, der dafür zuständig ist, Agenten zu starten, auszuführen, zu verschicken und ankommende Agenten anzunehmen. Der Agenten-Host läuft meist als Prozess oberhalb des Betriebssystems und der eigentlichen Hardware. Es existiert noch eine Zwischenschicht zwischen Betriebssystem und Agentenplattform, die so genannte Middleware-Schicht. Die Middleware bietet einheitliche Schnittstellen für die Kommunikation verschiedenartiger Komponenten im System. Oft implementiert die Agentenplattform auch gleichzeitig diese Schnittstellen, so dass die Middleware-Schicht in die Agentenplattform einfließt.

Der Host ist auch für die Kommunikation von Agent mit Hosts oder anderen Agenten zuständig und steuert gegebenenfalls den Zugriff von Agenten auf System-Ressourcen. Abbildung 3.3 stellt die Schichten eines typischen Agenten-Frameworks dar.

Agenten-Hosts bilden ein Netzwerk von Plattformen, auf der mobile Agenten existieren und zwischen denen sie migrieren können. Abbildung 3.4 visualisiert die Agenten-Migration.

[BHR97b] unterscheidet bei der Migration zwei Arten:

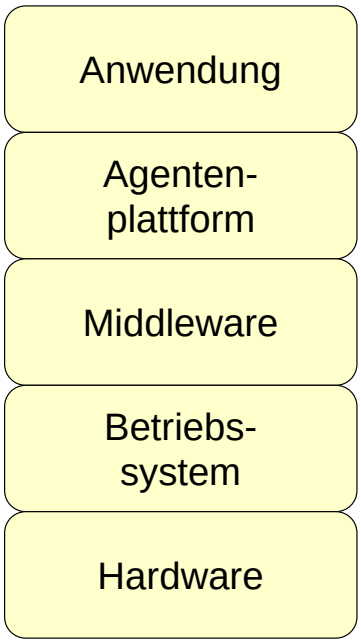

Abbildung 3.3: Schichten-Modell eines typischen Agenten-Frameworks

- *Schwache Migration*
  Die schwache Migration beschreibt den Zustand, dass der mobile Agent und seine aktuellen Daten zum Ziel-Host übertragen werden. Die Ausführung des Agenten auf dem Zielknoten beginnt entweder immer an derselben Stelle im Code oder an vordefinierten Methoden, die der Agent vor seiner Migration wählen kann.

- *Starke Migration*
  Starke Migration schließt bei der Übertragung den Ausführungszustand mit ein. Der Agent wird exakt an derselben Codestelle gestartet, an der die Migration auf dem Quell-Host angefordert wurde.

Die Migration stellt grundlegende Anforderungen an die Infrastruktur auf, die unabhängig von der Art der Migration realisiert werden müssen. Die Übertragung des Agenten und seiner Daten muss fehlerfrei geschehen. Sollte der Agent bei dem Transfer beschädigt werden oder gar verloren gehen, muss das Agentensystem für eine erneute Übertragung sorgen. Bei lose gekoppelten Netzen mit mobilen Knoten, auf denen die Agenten-Hosts laufen, kann dies zu Problemen führen. Knoten sind dann meist nicht erreichbar oder die Verbindung geht während der Übertragung verloren. Es können Verzögerungen auftreten, die zu unerwünschten Verdopplungen von Agenten führen können. Die meisten dieser Fälle sind schon aus den Kommunikationssystemen bekannt, wo es um Datentransfer zwischen

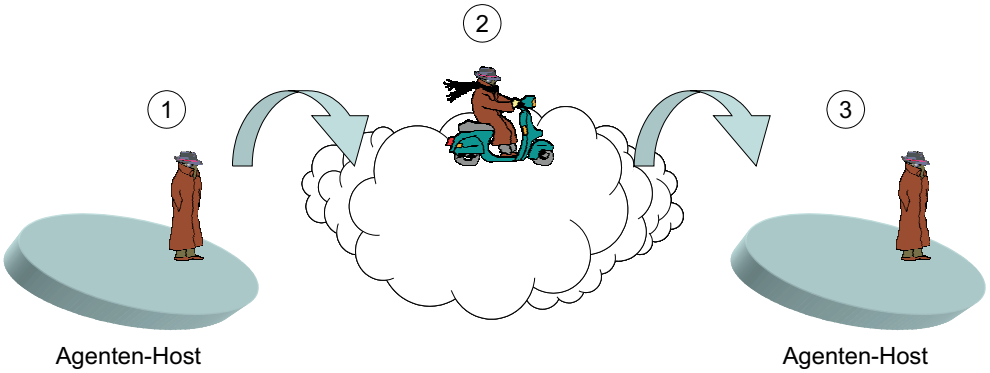

Abbildung 3.4: Migration des mobilen Agenten

zwei Partnern geht. Das Agentensystem kann bei der reinen Übertragung auf bekannte Methoden aus dem Bereich der Datenkommunikation zurückgreifen. Es müssen aber weitere Protokolle realisiert werden, die beispielsweise für die Übertragung der Agentenklassen notwendig sind.

Auch die Kommunikation der Agenten mit anderen Agenten oder Hosts, sollte dies überhaupt gewünscht sein, stellt Anforderungen an das Agentensystem. Der mobile Agent kann zusätzlich mit Diensten, welche auch in Form von Agenten auftreten können, mit Gruppen von Agenten oder mit dem Benutzer kommunizieren [BHR+97a].

3.5 Mobile Agenten in ubiquitären Systemen

Betrachtet man die Anforderungen für ubiquitäre Systeme und die Eigenschaften von mobilen Agentensystemen, so sind einige Übereinstimmungen deutlich zu erkennen. In dem Abschnitt 2.2 haben wir bereits die geforderten Merkmale ubiquitärer Systeme kennengelernt. Anhand dieser Merkmale sollen nun Anforderungen an mobile Agenten gezielt in ubiquitären Systemen definiert werden.

Anforderungen an mobile Agenten in ubiquitären Systemen

Das erste Paradigma des Ubiquitous Computing heißt Dezentralisierung. Ubiquitäre Systeme sind also in der Regel dezentralisierte und verteilte Systeme. Viele Projekte in diesem Bereich erfüllen leider schon diese Anforderung nicht. Ubiquitäre Geräte, Sensoren und Aktuatoren werden meist zentral über einen einzigen leistungsfähigen Server gesteuert. Der Grund dafür liegt daran, dass dieser Forschungsbereich noch recht jung ist und ubiquitäre Umgebungen nur im kleinen Rahmen realisiert werden können. Sollte in nächster Zukunft die Anzahl

an elektronischen Geräten rapide zunehmen, wird der zentrale Ansatz kaum mehr realisierbar sein.

Die Forderung nach Dezentralisation wird durch den Einsatz von mobilen Agentensystemen in ihrer grundlegenden Struktur schon erfüllt. Mobile Agenten bilden die höchste Stufe der Dezentralisierung, wo Informationen und Dienste auf entfernte Knoten verteilt werden. Die Mobilität der Agenten unterstützt auch die Mobilität des Benutzers in der ubiquitären Umgebung. Dadurch kann der Benutzer immer die richtigen Dienste und Informationen in seiner näheren Umgebung finden.

Die Konnektivität der ubiquitären Knoten setzt auch den Einsatz von drahtlosen Medien voraus. Dies definiert eine erweiterte Anforderung an die Migration des Agenten. Die Agentenplattform sollte daher auch auf mobilen Geräten ausführbar sein und sollte die Kommunikation über drahtlose Verbindungen erlauben. Da solche Geräte meist beschränkt in ihrer Speicher- und Energiekapazität sind, sollten mobile Agenten auch leichtgewichtig sein. Dies ist meist nicht leicht einzuhalten, da mit steigender Funktionalität des Agenten auch sein Gewicht zunimmt.

Eine weitere Anforderung, die sich durch die Verteilung ergibt, ist die Ad-Hoc Kommunikation. Personen und Geräte können in der ubiquitären Umgebung ihren Aufenthaltsort wechseln. Dabei kann es vorkommen, dass sie ihre Kommunikationsumgebung verlassen und in eine neue eintreten. Die Infrastruktur der mobilen Agenten sollte diese Anforderung in dem Sinne erfüllen, dass die Möglichkeit für eine dynamische Organisation der Agenten-Hosts gewährleistet sein sollte.

Die nächste Eigenschaft ubiquitärer Systeme ist die Diversifikation, d.h. die Beschränkung eines ubiquitären Knotens auf eine streng abgegrenzte Funktionalität. Da dabei Hard- und Software stark unterschiedlich sein können, sollten Applikationen plattformunabhängig konzipiert werden. Da heutige Systeme aus diesem Grund meist in der Programmiersprache Java realisiert sind, liegt es nahe, dass das Agentensystem auch in dieser Sprache implementiert wird. Der Focus liegt hierbei in der Plattformunabhängigkeit; die Wahl der Sprache Java ist nur Mittel zum Zweck.

Leider fehlen zur Zeit Bemühungen die Schnittstellen solcher unterschiedlichen Geräte zu standardisieren, so dass die Bedienung problematisch sein kann. Agenten können auch hier Vorteile bieten, indem sie eine Vermittlerrolle zwischen Diensten und spezialisierten ubiquitären Geräten übernehmen. Agenten sollten entweder selbst verschiedene Zugriffsmechanismen implementieren, um Informationen von unterschiedlichen Geräten wie Sensoren, zu erhalten oder die nötigen Protokolle kennen, um mit Diensten, die diese Informationen bereitstellen, zu kommunizieren.

Ein wichtiger Bereich des Ubiquitous Computing ist die Kontextsensitivität. Die

Umgebung soll sich in hohem Maße an den Benutzer anpassen. Dafür bedarf es zunächst der Erfassung der Informationen aus Umgebungssensoren. In direktem Zusammenhang mit Sensorinformationen steht der *Kontext*. Ein Kontext beschreibt in ubiquitären Systemen den Zustand, der durch die Auswertung von Informationen über die Umgebung eines Benutzers oder Systems entsteht [AEH+02].

Im Bereich der ubiquitären Systeme existieren viele Ansätze zur Modellierung des Kontextes. Kontextinformationen können einzelne Zustandsinformationen wie „An“ oder „Aus“ im Falle einer Lampe betreffen, aber auch Zustände mit vielen komplexen Informationen beschreiben [Bei02].

In den meisten Fällen wird der Begriff Kontext zur Beschreibung der physischen Umgebung des Benutzers oder Systems verwendet (physical context) [DRD+99]. Andere Ansätze berücksichtigen zusätzlich einen situationsbezogenen Kontext (situation context), welches die Situation beschreibt, in der sich der Benutzer oder das System gerade befinden [GSB02, SBG99]. Die reine Erfassung physischer Daten reicht für diese situationsbezogenen Informationen nicht aus. Es müssen weitergehende Informationen aus verschiedenen Sensordaten extrahiert werden.

Eine weitere Anforderung für mobile Agenten ergibt sich genau hieraus. Agenten müssen Kontexte erkennen können und dementsprechend handeln. Die Kontextdarstellung spielt dabei eine große Rolle, denn nur über eine einheitliche Darstellung ist die Erkennung von Kontexten auf fremden Knoten überhaupt möglich. Der Abschnitt 6.3.2.4 wird später eine solche Kontextdarstellung in XML einführen.

Die letzte Anforderung ist zugleich der wichtigste in Bezug auf die Durchsetzbarkeit und Akzeptanz des ubiquitären Systems: die Sicherheit. Ubiquitous Computing beschäftigt sich fast ausschliesslich mit personenspezifischen Informationen. Diese Daten müssen vor unauthorisiertem Zugriff geschützt werden. Da mobile Agenten Einheiten sind, die diese Informationen speichern und verarbeiten, muss das mobile Agentensystem Sicherheitsmechanismen vorweisen, die Agenten und Hosts vor bösartigen Angriffen schützen.

3.6 Vision der reflektiven Agenten

In dem Abschnitt 3.5 wurden bereits die Anforderungen an mobile Agentensysteme im Bereich des Ubiquitous Computing beschrieben. Von technologischer Seite scheinen sich die mobilen Agenten gut für ubiquitäre Systeme zu eignen. Die Idee, dass Agenten nicht nur in der virtuellen Welt Dienste ausführen, sondern Dinge in der realen Umgebung des Menschen gezielt steuern können, eröffnet unerwartete Möglichkeiten für innovative Anwendungen.

Der Mensch wird heutzutage überflutet mit Informationen. Daneben steigt auch die Anzahl der elektronischen Geräte im Umfeld des Menschen stetig. Wo man vor 20 Jahren nur das Radio und den Fernseher zu bedienen hatte, leidet man heute unter Massen von Fernbedienungen für die verschiedensten Geräte. Und die ubiquitäre Vision verkündet noch schlimmeres. Da ist man zu Recht froh, wenn einem jemand mit Rat und Tat zur Seite steht.

Genau dieser Jemand könnte ein mobiler Agent in der ubiquitären Umgebung des Menschen sein. Wenn in der Zukunft alltägliche Dinge Intelligenz vorweisen können und miteinander reden können, dann kann auch ein virtueller Helfer diese Dinge erfassen und steuern, und damit die Realität beeinflussen, als sei er in der realen Welt. Genau so wie die ubiquitäre Vision alltägliche Dinge in die elektronische Welt zieht, so drückt sie elektronische Programme tiefer in die reale Welt. Nicht die reale Welt um den Menschen herum wird auf eine virtuelle Ebene abgebildet werden, sondern der Mensch in der realen Welt wird abgebildet auf die existierende ubiquitäre Umgebung.

Virtuelles Abbild

Eine ubiquitäre Umgebung ist immer um den Menschen herum konzipiert. Der Mensch steht im Zentrum und alle Dienste im ubiquitären System zielen darauf ab, ihm Unterstützung im alltäglichen Leben zu geben. Die ubiquitären Elemente der Umgebung passen sich adaptiv dem Menschen an. Diese Anpassung erfordert sehr viel Kenntnis über Geschehnisse in der realen Umgebung und über den Menschen. Der Mensch hat Präferenzen, Vorlieben und Gewohnheiten. Diese Informationen müssen dem ubiquitären System zunächst einmal bekannt sein, damit es Anpassungen durchführen kann. Im ersten Schritt ist es unerlässlich, dass die persönlichen Daten vom Menschen selbst an das System übergeben werden. Im weiteren Betrieb kann das ubiquitäre System dieses Wissen durch Beobachtung der Person erweitern.

Nun ist der Mensch von Natur aus mobil. Ändert der Mensch seinen Aufenthaltsort, müssen die persönlichen Daten zum aktuellen ubiquitären Ort weitergeleitet werden. Hier gibt es zwei Möglichkeiten, wie diese Daten weitertransportiert werden können. Im ersten Fall überlässt man den Transport dem Menschen. Die Informationen könnten auf tragbaren Geräten wie PDAs, Handys, Laptops oder gar wearables gespeichert sein, die die Person immer bei sich hat. Diese Geräte könnten dann über ad-hoc aufgebaute Verbindungen, die persönlichen Daten an das umgebende System weiterschicken.

Diese Methode hat jedoch einige grundsätzliche Nachteile. Zum Einen neigen Menschen oft dazu, dass sie persönliche Dinge nicht immer mit sich tragen, wenn sie sich örtlich nicht weit von diesen entfernen. So nehmen die Wenigsten ihr Mobiltelefon immer zu sich, wenn sie kurz aus dem Büro zum Drucker gehen. Für die

ubiquitäre Anwendung, die zum Beispiel die Gewohnheiten des Bürobewohners aufzeichnet und für zukünftige Adaption lernt, gehen diese Lokationsdaten verloren bzw. werden erst gar nicht bemerkt. Das Tragen der mobilen Geräte oder Wearables wird heute noch als sehr unbequem empfunden.

Ein weiterer Nachteil ist, dass die Verantwortung für das Sichern der persönlichen Daten, in den Händen des Menschen liegt. Die Erfahrung aus dem Bereich des Internet zeigt leider hier, dass diese Aufgabe nicht ernst genommen wird oder den Benutzer überfordert. So könnte die Verbreitung mancher Viren im Internet leicht unterbunden werden, wenn die Benutzer Anti-Virenprogramme installieren oder ihre vorhandenen Anwendungen aktualisieren würden. Obwohl bei solchen Programmen schon fast alles automatisch geht, kostet es eine große Überwindung, sich mit der Sicherheit der eigenen Daten zu beschäftigen. Der Grund liegt darin, dass die Benutzer meist Angst haben, etwas Falsches oder Unwiederkehrbares zu tun, was wahrscheinlich aus dem Unverständnis der Schutzsysteme resultiert.

Der nächste Nachteil rührt aus einem technischen Problem: der Ad-hoc Verbindung und Weitergabe der Daten. Obwohl viele Forschungsprojekte sich mit diesem Problem beschäftigen, ist bisher noch keine alltagstaugliche Lösung gefunden. Die kabellose Verbindung kann über verschiedene Medien wie Funk, Infrarot oder Ultraschall gehen, wobei für jedes Medium verschiedene Protokolle existieren. Die Wahl des Mediums hängt meist von der Hardware der mobilen Geräte ab. Auch wenn heutige Endgeräte hybride Technologien nutzen und somit mehrere Medien unterstützen, sind wir weit entfernt von einem Ad-hoc Verbindungsaufbau zum umgebenden System ohne jegliche Benutzerinteraktion.

Die zweite Methode, wie die persönlichen Daten zum richtigen Ort kommen, ist die, dass das umgebende ubiquitäre System diese Daten speichert, sichert und zum aktuellen Auenthaltsort des Benutzers sendet. Die Aufgaben des Transportes und der Sicherheit sind also völlig losgelöst von der Verantwortung des Menschen. Man kann diese persönlichen Informationen als eine Art virtuelles Objekt im ubiquitären Raum betrachten. Bewegt sich die Person fort, so folgt ihm sein mobiles virtuelles Objekt wie ein elektronischer Schatten. Diese Methode setzt allerdings voraus, dass die Personen in der ubiquitären Umgebung mit Hilfe eines Location-Tracking-Systems geortet und verfolgt werden können. Ist dies gegeben, so lassen sich ortsbasierte Dienste auf persönliche Profile der Benutzer anpassen.

Betrachtet man mit dieser Idee im Hinterkopf das Paradigma der mobilen Agenten, so wird deutlich, dass dies sehr gut auf das virtuelle Objekt Szenario passt. Der mobile Agent stellt dabei ein virtuelles Abbild des Benutzers dar. Er reflektiert die Interessen seines Benutzers. In diesem Zusammenhang möchten wir diesen Agenten als *reflektiven Agenten* bezeichnen, der persönliche Informationen seines Benutzers kapselt und mit Hilfe dieser verschiedene Dienste für den Benutzer nutzt. Der reflektive Agent kann also viel mehr als ein virtuelles Ob-

jekt. Er kann seinen Informationsraum erweitern, indem er Umgebungsdaten vom ubiquitären System anfordert. So kann er Lokationsinformationen bezüglich des Benutzers abfangen und seinen Weitertransport samt seiner Daten selbst initiieren. Anhand geeigneter Schnittstellen kann der Agent auch die Geräte im Umfeld des Benutzers gezielt steuern. Die Sicherheit muss bei dieser Methode zwar auch das ubiquitäre System anbieten, einen Teil des Schutzes kann aber auch der mobile Agent übernehmen. Der Agent kann die persönlichen Informationen des Benutzers auf fremden Knoten schützen, um ein Ausspähen der Daten zu verhindern.

Der reflektive Agent muss in erster Linie mit Kontexten, sowohl Umgebungs- als auch Personenkontexte, in der ubiquitären Umgebung umgehen. Das Kontextbewusstsein ist ein großer Forschungsbereich im Ubiquitous Computing. Aus Sicht des Agenten ist eine standardmäßige Darstellung des Kontextes sehr wichtig, denn dies ermöglicht einen konformen Zugriff auf Informationen unabhängig vom aktuellen System.

4 Sicherheit in mobilen Agentensystemen

Die Sicherheit eines Softwaresystems ist mindestens genauso wichtig wie ihre Korrektheit, Zuverlässigkeit und Effizienz. Die fehlende Anerkennung der existierenden mobilen Agentensysteme ist nicht selten auf die fehlenden Sicherheitselemente zurückzuführen. Weltweit ist die Forschung mit Lösungen für sicherheitskritische Probleme von mobilen Agenten beschäftigt. Die aktuelle Technologie-Entwicklung hat jedoch immer noch nicht eine Stufe erreicht, die einen einheitlichen und sicheren industriellen Einsatz mobiler Agentensysteme ohne Bedenken garantiert.

Die Sicherheit in mobilen Agentensystemen kann man unterteilen in die Sicherheit

- vor Bedrohung,
- vor Verlust der Daten,
- bei der Migration und
- vor Fehlbedienung.

Sicherlich sind die einzelnen Punkte untereinander verflochten. So sind Sicherheitsmaßnahmen, die den Verlust der Daten verhindern, gleichzeitig Mechanismen, die die Sicherheit vor Bedrohung betreffen. Der Hauptaugenmerk soll jedoch hier bei der Sicherheit vor Bedrohungen liegen.

Sicherheit sollte nicht als eine einfache Zweiteilung charakterisiert werden, wie das System ist entweder sicher oder unsicher. Bei der Entwicklung von Software-Systemen ist eher die Frage wichtig, gegen welche Sicherheitsprobleme oder Arten von Angriffen das System gesichert ist oder sein soll. Man muss also die Bedrohung, vor der man sich schützen möchte kennen und definieren können. Nur gegen relativ konkret spezifizierten Bedrohungsszenarien wird man sich schützen können und auch davor nur dann, wenn sie korrekt eingeschätzt wurden und adäquate Gegenmaßnahmen implementiert wurden. Einen entscheidenden Einfluss üben hier Kosten-/Nutzen-Abwägungen aus. Mit genügend großen Investitionen kann man sich beliebig stark gegen beliebig viele Bedrohungen absichern, aber egal

wie viel Aufwand betrieben wird, vollständige Sicherheit lässt sich niemals erreichen. Kein wie auch immer geartetes Sicherheitskonzept ist in der Lage, ein System ganz allgemein, gegen alle Bedrohungen zu schützen. Diese Einsicht mag schockieren, sie ist aber weder auf mobile Agentensysteme beschränkt, noch auf Computer überhaupt. Diese Tatsache ist seit langem bekannt und wurde schon oft folgenschwer ignoriert.

4.1 Sicherheitsanforderungen

In diesem Abschnitt werden die verschiedenen Sicherheitsaspekte eines mobilen Agentensystems strukturiert dargestellt. Diese Aspekte beruhen auf Sicherheitsanforderungen für Computersysteme im Allgemeinen, die zunächst näher erläutert werden.

4.1.1 Authentizität

Authentizität ist eine wichtige Anforderung und die Grundlage vieler Sicherheitslösungen. Es fordert, dass innerhalb einer Kommunikation jeder Partner die Identität seines Gegenüber nachweisen kann. Wenn *Alice* sich gegen *Bob* authentifiziert, bedeutet das, dass Alice ihre Identität an Bob preisgibt und damit Bob die Entscheidung bietet, Alice zu trauen. D.h. Authentizität ist die nachweisbare Offenlegung der Identität als Vorstufe des Vertrauens.

Wenn man sich in der realen Welt umschaut, erkennt man schnell, dass es dort nicht anders ist. Um einen Kredit bei der Bank zu bekommen, muss ein Kunde sich zunächst ausweisen und seine finanzielle Situation offen legen. Der Bankangestellte bildet auf diesen Informationen ein Vertrauensverhältnis, wonach er dem Kunden den Kredit gewährt oder nicht.

Die Authentifizierung in der Computerkommunikation ist ungemein schwerer. Die Kommunikationspartner stehen sich meist nicht gegenüber, sondern sind kilometerweit entfernt. Hinzu kommt, dass nicht nur Menschen sich authentifizieren müssen, sondern auch Computer oder Programme, die miteinander kommunizieren. Eine gängige Authentifizierungsmethode ist die Registrierung und Abfrage von Benutzernamen und Passwort, was nur menschliche Benutzer angeht.

Das elektronische Pendent sind Zertifikate und elektronische Signaturen. Diese Methoden der Authentifizierung beruhen auf Geheimnisse, wie z.B. ein privater Schlüssel, welcher dazu verwendet wird, ein Dokument elektronisch zu unterschreiben oder zu verschlüsseln. Dabei gibt es die Unterscheidung zwischen symmetrischen und asymmetrischen Verfahren. Der Unterschied liegt bei der Art

der Schlüssel. Bei der symmetrischen Verschlüsselung existiert nur ein Schlüssel, mit dem ver- und entschlüsselt wird. Dieser muss jedem Kommunikationspartner bekannt sein. Der Nachteil bei diesem System ist die sichere Verteilung der Schlüssel. Die asymmetrische Verschlüsselung basiert auf privaten und öffentlichen Schlüsseln. Daten, die mit dem öffentlichen Schlüssel verschlüsselt wurden, können nur mit dem privaten Schlüssel entschlüsselt werden. Da der öffentliche Schlüssel frei zugänglich für Jedermann ist, bereitet der Austausch keine Probleme. Allerdings sind asymmetrische Verfahren sehr rechenaufwendig. Die Kenntnis der Schlüssel reicht schon aus, um Partner nach beiden Verfahren zu authentifizieren.

Authentifizierung ist auch für mobile Agentensysteme unerlässlich. Ein mobiler Agent muss sich auf jedem besuchten Agenten-Host authentifizieren, damit der Host entscheiden kann, ob dieser Agent vertrauenswürdig ist oder nicht. Nach diesem ersten Sicherheitsschritt kann der Host anhand der Identität des Agenten Zugriffsrechte definieren, die dem Agenten erlauben, auf Ressourcen zuzugreifen. Aber auch umgekehrt muss der Host sich gegenüber dem Agenten authentifizieren, um dem Agenten die Sicherheit zu geben, dass er sich auf einem *korrekten* Host befindet. Da der Host die volle Kontrolle über den Agenten hat, kann ein bösartiger Agenten-Host die sensiblen Agentendaten missbrauchen oder manipulieren. Diese Tatsache hebelt auch die Methoden der elektronischen Signaturen aus, da es nicht möglich ist, dass der Agent ein Geheimnis, z.B. in Form eines privaten Schlüssels, vor einem bösartigen Host schützen kann, da eine völlige Offenlegung für seine Ausführung unumgänglich ist.

4.1.2 Vertraulichkeit

Vertraulichkeit von Information bedeutet, dass nur derjenige die Daten lesen kann, der dazu vom Eigentümer der Daten berechtigt wurde. Die praktische Realisierung von Vertraulichkeit basiert auf Verschlüsselung. Dabei können wie bei der Authentifizierung symmetrische oder asymmetrische Verfahren angewendet werden. Die Daten werden mit entsprechenden Schlüsseln gesichert, so dass nur die Instanz diese Nachricht lesen kann, die dafür befugt ist.

Vertraulichkeit spielt auch bei mobilen Agenten eine große Rolle. Stellt man sich einen Einkaufsagenten vor, der verschiedene Online-Shops besucht, um den niedrigsten Preis für ein Produkt zu ermitteln, so müssen die Informationen über bereits besuchte Shops und ermittelte Preise vor unbefugten Zugriff geschützt werden. Genau wie im vorigen Abschnitt beschrieben, ist dies nicht ohne weiteres möglich, da der Host die volle Kontrolle über die Daten des Agenten besitzt.

4.1.3 Integrität

Integrität beschreibt die Sicherheit, dass eine Information zu keinem Zeitpunkt verändert wurde. In der Realität werden für solche Zwecke Vertrauenspersonen wie Notare hinzugenommen, die garantieren, dass nichts an dem Originaldokument verändert wurde. Spätere Änderungen müssen vor dem Notar durch alle Partner durch Unterschrift bestätigt werden.

Das digitale Äquivalent hierzu ist die digitale Signatur. Dabei kann die Authentifizierung und die Integrität in einem einzigen Schritt geprüft werden. Modifiziert man nämlich die asymmetrische Verschlüsselung so, dass der Sender eine Nachricht diesmal mit seinem eigenen privaten Schlüssel verschlüsselt, kann der Empfänger, der im Besitz des öffentlichen Schlüssels ist, die Nachricht entschlüsseln. Vollzieht sich die Entschlüsselung ohne Fehler, hat der Empfänger die Authentizität des Senders und gleichzeitig die Integrität der Nachricht bewiesen. Allerdings ist hierbei natürlich die Vertraulichkeit völlig vernachlässigt, da jeder, der im Besitz des öffentlichen Schlüssels des Senders ist, die Nachricht lesen kann. Aus diesem Grund wird die Integrität oft in einem separaten Schritt vollzogen. So genannte *digest* oder *Hash-Werte* sind gängige Verfahren. Hash-Funktionen sind Einweg-Verfahren, die Daten beliebiger Länge auf einen Datenblock fester Länge (Hash-Wert) abbilden. Es ist dabei garantiert, dass die Originaldaten nicht aus dem Hash-Wert abgeleitet werden können und zwei gleiche Hash-Werte garantiert vom selben Original abstammen.

Mobile Agenten dürfen auf ihrer Reise nicht verändert werden. Die Integrität sowohl der Daten, als auch des Ausführungscodes des Agenten müssen gesichert werden. Dabei ist es schon schwierig Mechanismen zu finden, die eine absichtliche oder technisch bedingte Integritätsverletzung überhaupt zu erkennen.

4.1.4 Verantwortlichkeit

Verantwortlichkeit bedeutet, dass jede Instanz verantwortlich für ihre ausgeführten Aktionen ist und diese nicht im Nachhinein abstreiten kann. Dies kann in der Realität durch Verträge realisiert werden, die von allen Partnern unterschrieben werden. In der digitalen Welt kann dies äquivalent mit digital signierten Verträgen vollzogen werden.

Da ein mobiler Agent meist im Namen einer Person handelt, sollte diese Person auch für die Aktionen des Agenten verantwortlich gemacht werden können. Genau dieser Umstand hemmt den Gebrauch des mobilen Agenten bei sensiblen Transaktionen, bei dem es um reale materielle Dinge geht. Das Vertrauen auf ein Stück Software, welche in der Lage ist, das Geld seines Eigentümers auszugeben, ist einfach nicht gegeben. Technisch gesehen ist die Verantwortlichkeit genauso

problematisch, denn ein Agent kann nicht wie in den Fällen davor einen privaten Schlüssel in sich sichern. Dieser würde aber für eine digitale Signatur eines Vertrages unabdinglich.

4.1.5 Verfügbarkeit

Verfügbarkeit ist die Forderung, den Zugriff auf einen Dienst so zu sichern, dass er zu jedem Zeitpunkt von jeder Instanz erreichbar ist. Die Verfügbarkeit wird meist durch Redundanz erreicht.

Im Fall der mobilen Agentensysteme müssen sowohl der Host als auch die Agenten vor Überlastung oder Blockade geschützt werden.

4.1.6 Anonymität

Anonymität steht in Kontrast zur Authentizität. Die Identität soll in diesem Fall geheim gehalten werden. In der realen Welt ist es nicht immer notwendig sich zu identifizieren. So kann man anonym einkaufen gehen, indem man alles bar bezahlt. Die Authentifizierung wird dabei durch die Transaktion des Geldes vor Ort geregelt. In der digitalen Welt ist dies nicht möglich, da kein einheitliches digitales Zahlungsmittel existiert.

Mobile Agenten können zwar auch nicht migrieren ohne sich zu authentifizieren, jedoch bieten sie eine Möglichkeit die Identität des Eigentümers geheim zu halten, für den sie Informationen sammeln oder Dienste ausüben, solange es nicht zu Verantwortlichkeitsproblemen kommt.

4.2 Bedrohungsanalyse

Um eine vollständige Umsetzung von Sicherheitsmechanismen für mobile Agentensysteme zu gewährleisten, wird in diesem Abschnitt eine vollständige Bedrohungsanalyse durchgeführt. Zuerst werden Annahmen betrachtet, die im Gegensatz zu herkömmlichen Softwareparadigmen, für mobilen Code nicht mehr zutreffen. Anschließend werden alle potenziellen Angreifer eines Agentensystems und deren Gefahren für das mobile Agentensystem beschrieben.

4.2.1 Allgemeingültige Annahmen

[Che98] zählt vier Annahmen auf, die für bisherige Softwareparadigmen vorausgesetzt werden können, die aber für das Softwareparadigma mobiler Code, zu dem die mobilen Agenten gehören, nicht mehr gelten. Anhand dieser Annahmen werden die einzelnen Aspekte für ein sicheres Agentensystem verdeutlicht.

Zuordnung von Personen zu Programmen

Die wichtigste Annahme, die mobile Code Systeme verletzen, ist die Identitätsannahme.

> **Immer wenn ein Programm eine Aktion ausführt, kann eine Person identifiziert werden, der diese Tätigkeit zugeordnet werden kann, und es ist sicher anzunehmen, dass diese Person die Aktion beabsichtigt.** [Che98]

Die meisten Sicherheitssysteme basieren auf dieser Annahme. Aus der Identitätsannahme wird abgeleitet, dass jedes Programm die Rechte des Benutzers hat, der es ausführt. Dies impliziert, dass ein Programm nur dann ausgeführt werden darf, wenn der Benutzer dem System bekannt ist.

In mobilen Code Systemen trifft diese Annahme nicht zu. Die Person, die das Programm ausführt, ist im Allgemeinen dem System, auf dem das Programm ausgeführt wird, nicht bekannt. Das bedeutet, dass das System nicht weiß, welche Rechte das Programm hat bzw. welche Rechte es ihm geben darf. Die Vergabe von Rechten für mobilen Code ist ein wesentlicher Bestandteil der Sicherheitsmechanismen. Die Problematik, die sich hier stellt, ist:

> Welche Rechte benötigt das Programm, um seine Aktionen ausführen zu können, und welche Rechte dürfen dem Programm gewährt werden, ohne ein Sicherheitsrisiko für das System zu verursachen?

Trojanische Pferde sind selten

In der heutigen Zeit von Sober, Sasser und Phatbot, kann diese Annahme eigentlich nicht mehr als gültig vorausgesetzt werden. Zwar sind Trojaner und Würmer nicht mehr selten, doch die Schutzmechanismen sind häufig nicht in dem eigentlichen System (z.B. Microsoft Windows) sondern werden zusätzlich eingerichtet (Virenscanner, Firewalls). Die Annahme kann deswegen weiterhin übernommen werden.

Die Identitätsannahme war noch nie korrekt. Trojanische Pferde und Fehlfunktionen in Programmen haben diese Annahme verletzt. Durch die Annahme der seltenen Trojanischen Pferde kann dieser Aspekt bei der Betrachtung der Gültigkeit für die Identitätsannahme vernachlässigt werden. [Che98] führt eine weitere These ein, die besagt, dass ein Programm aus leicht zu identifizierenden Quellen stammt. Folglich kann der Entwickler leicht ausfindig gemacht und zur Verantwortung gezogen werden. Aus diesem Grund wird ein Entwickler kaum absichtlich Funktionen in sein Programm einbauen, die unerwünschte Aktionen ausführen. Allerdings schützen die rechtlichen Bestimmungen nicht nur den Benutzer vor dem Entwickler, sondern sie gelten auch umgekehrt. Ein Benutzer darf ein Programm nicht gegen die Lizenzbestimmungen benutzen. Solche Schutzvorkehrungen werden als soziale Sicherheitsmechanismen bezeichnet.

In mobilen Code Systemen ist es unmöglich, den Code einer bestimmten Person zuzuordnen und diese für den Schaden zu belangen. Das heißt, die sozialen Mechanismen greifen in diesem Systemen nicht mehr. Dadurch steigt die Wahrscheinlichkeit, dass dies ausgenutzt wird und Programme nicht gewollte Aktionen durchführen. Während in herkömmlichen Systemen zur Verbreitung der schädlichen Programme (Würmer, Viren, Trojaner) häufig Sicherheitslücken benutzt werden, impliziert das Paradigma des mobilen Codes diese Verbreitung.

In letzter Zeit hat gerade die Verbreitung von Würmern den Schutz vor diesen in den Mittelpunkt des allgemeinen Interesses gerückt. Mittlerweile kennt nicht nur der Experte diese Gefahr, sondern auch dem normalen Privatanwender ist diese Gefahr bewusst. Aus diesem Grund ist für die Akzeptanz und Verbreitung des Systems die Lösung dieses Problems ausschlaggebend.

Ursprung der Angriffe

Aus den bereits erwähnten Thesen kann geschlossen werden, dass die Bedrohungen von Angreifern kommen, die Programme benutzen, um unberechtigten Zugriff zu erlangen. Dies kann auf zwei verschiedene Arten passieren:

- Annehmen einer falschen Identität
- Umgehen der Sicherheitsvorkehrungen

Die meisten Sicherheitssysteme basieren auf Benutzerauthentifizierung und benutzer-basierter Zugriffskontrolle. Bei mobilem Code können diese Sicherheitsvorkehrungen nicht umgesetzt werden. Eine sichere Authentifizierung kann nicht immer erfolgen, da zum Beispiel der Benutzer nicht bekannt ist. Selbst wenn der Benutzer bekannt wäre, könnte es sein, dass das Programm Aktionen ausführt, die dem Benutzer unbekannt sind. Das bedeutet, ein Benutzer kann ein System unabsichtlich angreifen, indem er unbewusst ein schädliches Programm benutzt. Deshalb dürfen die Sicherheitsmechanismen nicht vom Benutzer abhängig sein.

Aufenthalt des Programms ändert sich nicht

Die Annahme, dass sich der Aufenthalt des Programms nicht ändert, gilt für die meisten Programme. Allerdings besteht die wesentliche Eigenschaft von mobilen Code Systemen darin, die Transformation und Ausführung des Codes auf verschiedenen Rechnern zu ermöglichen. Des Weiteren wird davon ausgegangen, dass ein Programm immer auf demselben Betriebssystem läuft und dieses für die Sicherheit verantwortlich ist. Mobile Code Systeme müssen auf unterschiedlichen Betriebssystemen laufen und deswegen kann das Betriebssystem nicht für die Sicherheit verantwortlich gemacht werden [Lei03].

4.2.2 Potenzielle Angreifer

Jede beteiligte Instanz des mobilen Agenten Systems kann als potenzieller Angreifer betrachtet werden. Im Folgenden werden die einzelnen Teilnehmer genauer betrachtet. Es wird jeweils auf die Aufgabe der Instanz sowie die Gefahr als potenzieller Angreifer näher eingegangen. Eine genauere Klassifizierung und Untersuchung der Angriffe ausgehend von einem mobilen Agentensystem wird im hierauf folgenden Abschnitt durchgeführt. Abbildung 4.1 stellt die beteiligten Instanzen eines mobilen Agentensystems graphisch dar.

Agent

Die Aufgabe des Agenten ist die Bereitstellung von Diensten und die Ausführung dieser für seinen Benutzer. Die Angriffsmöglichkeiten des Agenten sind sehr vielseitig und können sich gegen unterschiedliche Instanzen richten. So kann der Agent z.B. das zugrunde liegende Rechnersystem angreifen. Eine weitere Gefahr, die von Agenten ausgeht, ist der Angriff gegen den Benutzer. Der Agent muss die Aufgaben für seinen Benutzer korrekt erledigen. Ein Angriff wäre beispielsweise die Weitergabe von vertraulichen Informationen, die der Benutzer dem Agenten anvertraut. Die wichtigste Rolle bei diesen Gefahren geht vom Autor (siehe unten) des Agenten aus. Weitere mögliche Angriffe, die von Agenten ausgehen, werden im Abschnitt 4.2.3 näher erläutert.

Agentenplattformen

Die Agentenplattform ist für die Ausführung der mobilen Agenten verantwortlich und hat somit Zugriff auf den kompletten Code. Ein möglicher Angriff stellt das unautorisierte Klonen von Agenten dar. Da die Agentenplattform in der Regel Sender und Empfänger ist, kann sie außerdem den Migrationspfad eines Agenten

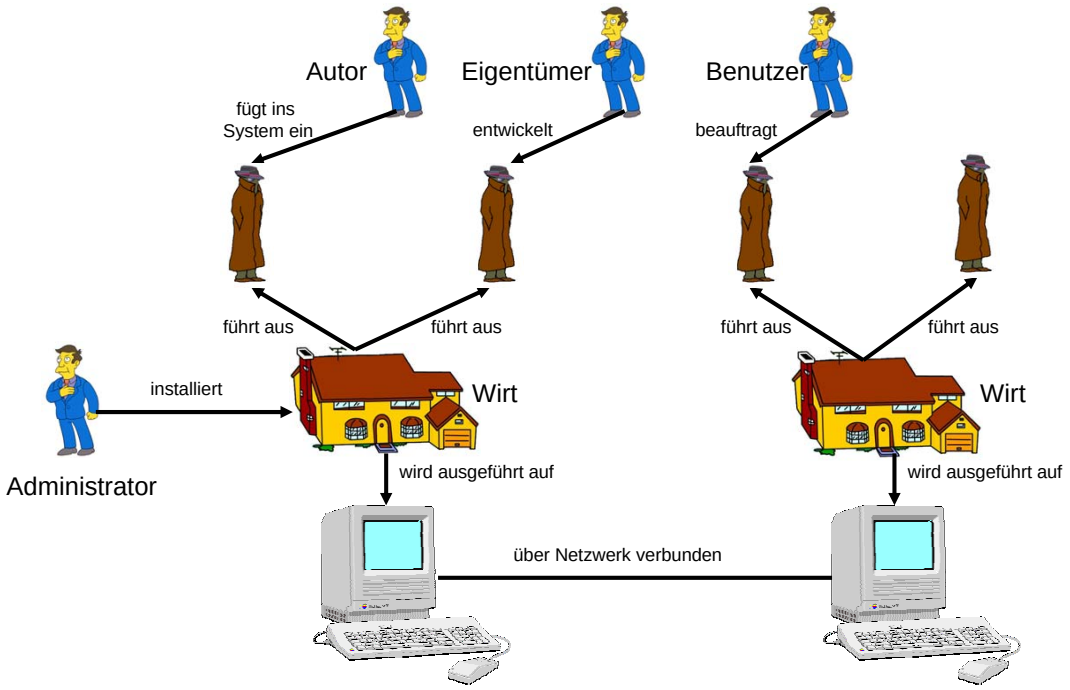

Abbildung 4.1: Beteiligte Instanzen

manipulieren. Wie der Agent kann auch die Plattform das zugrunde liegende Rechnersystem angreifen.

Netzwerk (Kommunikation/Migration)

Das Netzwerk ist die Grundlage für die Kommunikation und Migration in einem mobilen Agentensystem. Das heißt, Angriffe über das Netzwerk auf den Netzwerkverkehr zielen auf die Kommunikation und Migration ab. Auch dies ist ein bereits bekanntes Problem aus verteilten Systemen und stellt keine besonderen Anforderungen an die Sicherheitsmechanismen.

Rechner

Auf dem Rechner werden die Agentenplattform und die Agenten ausgeführt. Die Gefahr geht hier von Angriffen gegen den Rechner aus. Diese Angriffe können auch das mobile Agentensystem bedrohen.

Administrator

Der Administrator ist für die Installation und Konfiguration des mobilen Agentensystems verantwortlich. Bei diesem Vorgang kann der Administrator das System beliebig manipulieren. Diese Problematik besteht allerdings genauso bei herkömmlicher Software und wird deswegen nicht mehr genauer betrachtet.

Autor

Der Autor ist der Entwickler des Agenten. Er stellt den Agenten dem Eigentümer zu Verfügung. Die Angriffsmöglichkeiten des Autors entspricht denen eines Trojanischen Pferdes. Der Autor kann Funktionen einbauen, die dem Benutzer nicht bekannt sind. Im Prinzip kann er alle Angriffsmöglichkeiten des Agenten nutzen.

Eigentümer

Der Eigentümer führt den Agenten in das System ein. Ein Eigentümer könnte bewusst böswillige Agenten in ein System einführen, um diesem zu schaden.

Benutzer

Der Benutzer kann versuchen, durch die Inanspruchnahme vieler Agenten und deren Dienste, einen Denial-of-Service-Angriff auf das Agentensystem zu starten.

Nachdem hier die potentiellen Angreifer vorgestellt wurden, soll der nächste Abschnitt die möglichen Angriffe, die von mobilen Agenten ausgehen, näher erläutern.

4.2.3 Mögliche Angriffe

Grundlegend sind drei Arten von Angriffen zu unterscheiden: Angriffe auf das Rechnersystem, auf dem die Agentenplattform läuft, Angriffe auf die verwendete Technologie sowie Angriffe auf Hosts und Agenten.

4.2.3.1 Angriffe auf das Rechnersystem

Agenten wie auch Agenten-Hosts werden als Prozesse in einem gemeinsamen Prozess oder in unterschiedlichen Prozessen ausgeführt und sind somit Bestandteil dieses Rechnersystems. Das bedeutet, dass alle Angriffe gegen den Rechner auch Angriffe auf das Agentensystem darstellen. Ein Beispiel hierfür ist ein Denial-of-Service-Angriff, der sich gegen das zugrunde liegende Rechnersystem richtet. Ist der Angriff erfolgreich und führt er zum Absturz des Systems, ist zugleich auch das Agentensystem betroffen. Ein weiteres Beispiel wäre ein Angriff auf den

Adressbereich von Prozessen. Ist es einem Angreifer trotz Schutz des Betriebssystems möglich, auf den Adressbereich einzelner Prozesse zuzugreifen, kann dieser natürlich auch auf den Bereich des mobilen Agentensystems zugreifen und so beispielsweise an vertrauliche Daten des Agenten gelangen. Auch wenn sich solche Angriffe nicht ausdrücklich gegen das Agentensystem richten, sondern gegen einzelne Dienste des Rechnersystems, wie zum Beispiel das Betriebssystem, ist die Agentenplattform als Komponente des Gesamtsystems dadurch betroffen. Da diese Angriffe sich aber nicht durch die Existenz eines Agentensystems ergeben, wird auf Angriffe gegen das Rechnersystem nicht genauer eingegangen. Allerdings muss berücksichtigt werden, dass eventuell einige Sicherheitsmechanismen des Agentensystems auf der Annahme eines sicheren Rechnersystems beruhen und so durch solche Angriffe außer Kraft gesetzt werden können.

4.2.3.2 Angriffe auf die verwendete Technologie

Eine weitere Klasse von Angriffen, die sich nicht direkt gegen das mobile Agentensystem richtet, sind die Angriffe auf die verwendeten Technologien. Für die Implementierung eines Agentensystems werden eine Programmiersprache sowie eventuell weitere Technologien benötigt. Beinhaltet die verwendete Technologie Sicherheitslücken, treten diese auch in der Anwendung auf. Dadurch wird das Agentensystem angreifbar. Java ist eine häufig verwendete Programmiersprache für mobile Agentenplattformen. Weist eine Implementierung von Java Sicherheitslücken auf, so können diese zum Angriff genutzt werden.

In [BR02] wird auf die Problematik der Implementierung des Byte Code Verifiers näher eingegangen. Bevor ein Programm in die Java Virtuell Machine geladen wird, überprüft der Verifier, ob es sich um ein gültiges Java Programm handelt. Leider akzeptieren mehrere aktuelle Java-Implementierungen auch Byte Code, der kein gültiges Programm wiedergibt. Dadurch kann es einem Angreifer gelingen, unter Ausnutzung der Sicherheitslücke des Verfiers einen Angriff auf das Agentensystem durchzuführen. Da sich diese Sicherheitsproblematik wiederum nicht aus der Existenz einer mobilen Agentenplattform ableitet, wird auf eine genauere Beschreibung der Problematik verzichtet.

4.2.3.3 Angriffe auf Agenten-Hosts und Agenten

Das mobile Agentensystem besteht aus zwei grundlegenden Komponenten - den Hosts und den Agenten. Direkte Angriffe auf das Agentensystem richten sich immer gegen eine dieser Komponenten - oder beide. Der Schutz der Hosts und Agenten ist der wesentliche Bestandteil einer Sicherheitsarchitektur für ein Agentensystem. [Gra96] teilt die Angriffe in vier Kategorien ein:

- Angriffe gegen die Agentenplattform
- Angriffe gegen Agenten
- Angriffe von Agenten gegen Agenten
- Angriffe von Agenten gegen eine Gruppe von Agentenplattformen

Diese Unterteilung wird in vielen Arbeiten wie beispielsweise in [ACFT97] und [Lei03] übernommen. In [Hoh01] wird ein weiterer Aspekt eingeführt. Es wird zwischen lokalen und entfernten Angriffen unterschieden. Entfernte Angriffe werden von Parteien außerhalb einer Agentenplattform auf die Plattform oder seinen Agenten ausgeführt, während lokale Angriffe auf einer Agentenplattform erfolgen. Aus den oben genannten Kategorien ergeben sich folgende Angriffsszenarien:

- entfernte Angriffe
  - gegen den Agenten
  - gegen die Agentenplattform
- lokale Angriffe
  - Agent gegen Agent
  - Agent gegen die Agentenplattform
  - Agent gegen eine Gruppe von Agentenplattformen
  - Agentenplattform gegen Agent

Entfernte Angriffe gegen das mobile Agentensystem

Ein wesentlicher Bestandteil eines mobilen Agentensystems ist das zugrunde liegende Netzwerk. Das Netzwerk wird zur Kommunikation der Agenten-Hosts und Agenten sowie zur Migration der Agenten benutzt. Für den Schutz der Hosts und Agenten können entweder Sicherheitsanforderungen an das Netz gestellt werden oder die Agentenplattform muss den Schutz in einem unsicheren Netz gewährleisten. Die erste Variante führt zu einer starken Einschränkung der Benutzung und Verbreitung der Agentenplattform und stellt keine zufrieden stellende Lösung dar. Eine Plattform für mobile Agenten sollte keine Sicherheitsanforderungen an das Netzwerk stellen, sondern selbst Schutzmechanismen zur Verfügung stellen. Das in dieser Arbeit entwickelte Agentensystem ist einer von mehreren Diensten, die zusammen auf einer gemeinsamen Middleware basieren. In diesem Fall ist es sinnvoll, die Anforderungen für eine sichere Netzwerkkommunkation an die Middleware zu stellen, da sonst jeder Service eine eigene Sicherheitsarchitektur für Netzwerkkommunikation implementieren muss.

Entfernte Attacken gegen den Agenten können sich gegen die Kommunikation des Agenten mit Parteien außerhalb des ausführenden Agenten-Hosts richten. Der Agent kommuniziert mit anderen Komponenten des mobilen Agentensystems über das Netzwerk. Ein potenzieller Angriff beispielsweise ist das Abhören des Kommunikationskanals. Dadurch können vertrauliche Daten ausspioniert werden. Das Gleiche gilt für Angriffe auf die Migration eines Agenten. Die Migration ist wie die Kommunikation ein einfaches Senden von Daten über das Netz. Attacken gegen Datenübertragungen sind aus bestehenden Programmierparadigmen für verteilte Systeme bereits bekannt und mit bestehenden Sicherheitskonzepten einfach zu verhindern. Ein schwerwiegender Angriff ist eine Denial-of-Service-Attacke gegen den Agenten. Angriffe gegen den Agenten sind Datenflüsse, die der Host als reguläre Kommunikation wahrnimmt, und an den Agenten weiterleitet. Denial-of-Service-Angriffe sind ebenfalls bereits bekannt, stellen aber auch in konventionellen verteilten Systemen ein großes Problem dar.

Bis auf die Attacken gegen die Migration des Agenten treffen für die Agentenplattform die gleichen Angriffsarten wie für den Agenten zu. Der Schutz des Hosts kann genauso wie beim Agenten durch bekannte Sicherheitsverfahren für Verschlüsselung, Authentifizierung und Integritätssicherung realisiert werden. Denial-of-Service-Attacken, für die es auch in verteilten Systemen nicht immer eine Lösung gibt, stellen somit bei den entfernten Angriffen die einzige schwierige Aufgabe dar. Soweit herkömmliche Verfahren in verteilten Systemen vor diesen Angriffen schützen, können diese auch für mobile Agentenplattformen benutzt werden.

Lokale Angriffe gegen das mobile Agentensystem

Neben den Angriffen über das Netzwerk sind bei mobilen Agentensystemen die Attacken der einzelnen Komponenten auf einer gemeinsamen Agentenplattform untereinander zu betrachten. Betrachtet man den Agenten-Host als Betriebssystem und die Agenten als darauf ausgeführte Applikationen fallen einem sehr schnell viele Sicherheitsaspekte ein, die ein mobiles Agentensystem erfüllen muss. Zum Beispiel:

- Schutz der Ressourcen
- Aufteilung der Ressourcen
- Schutz der Anwendungen untereinander
- Schutz des Betriebssystems vor böswilligen Anwendungen

Da diese Problematiken im Kontext des Betriebssystems bereits längst bekannt und umgesetzt sind, stellt dies eine große Hilfe für die Umsetzung in mobilen

Agentensystemen dar. Allerdings wird durch diesen Vergleich auch die Achillesferse eines mobilen Agentensystems verdeutlicht. Eine Annahme, die im Gegensatz zu Betriebssystemen im Allgemeinen nicht getroffen werden kann, ist die einer vertrauten Plattform. Während bei einem Betriebssystem davon ausgegangen wird, dass ihm vertraut werden kann, trifft dies für Agenten-Hosts nicht zu.

In Betriebssystemen entspricht der Schutz von Agenten vor anderen Agenten dem Schutz von verschiedenen Prozessen. Verschiedene Sicherheitsdomänen sind dafür zuständig, den direkten Zugriff von einem Prozess auf den internen Zustand eines anderen Prozesses zu regeln. Wegen den verbundenen Sicherheitsrisiken ist der direkte Zugriff in Agentensystemen nicht erwünscht. Aus diesem Grund muss ein sicheres Kommunikationsmodell für die Agenten entwickelt werden. Die Kommunikation in einem Agentensystem geht über den einfachen Informationsaustausch in einem Transaktionsmodell hinaus [ACFT97].

Der Schutz der Agentenplattform vor böswilligen Agenten spielt in mobilen Agentensystemen eine entscheidende Rolle. Garantiert der Wirt keinen ausreichenden Schutz vor bösartigen Agenten wird kein Betreiber diesen Wirt auf seinem System zulassen. Dieses Problem ist ein wesentlicher Bestandteil für die Akzeptanz eines mobilen Agentensystems. Die Schutzmechanismen ähneln sehr denen eines Betriebssystems und so duplizieren und erweitern mobile Agentensysteme diesen Schutz [GBH98].

Auch bedingt durch die benutzte Programmiersprache können Angriffe durch bösartige Agenten auf die Agentenplattform möglich werden. Betrachtet man die Sprache Java, ist es Programmen erlaubt, eigene Threads zu starten. So laufen in den meisten Agentensystemen die Agenten als Threads. Gleichzeitig erlaubt Java den synchronisierte Zugriff auf Klassen. In [BR05] ist folgendes Beispiel aufgeführt, welches ein korrektes Java-Code beschreibt, und einen möglichen Denial-of-Service-Angriff darstellt:

```
public void run() {
    synchronized( Thread.class )
    {
        while( true );
    }
}
```

Ein Agent, der diesen Code ausführt, sperrt das Objekt `Thread.class` und führt eine Endlosschleife durch, so dass andere Agenten, die auch als Threads laufen, für immer blockiert sind.

In bestimmten Fällen ist die isolierte Betrachtung eines Agenten-Hosts nicht mehr möglich. Beispielsweise könnte ein Agent durch Sammeln von Informationen auf

verschiedenen Wirten an mehr Informationen gelangen als er darf oder, wenn ein Agent innerhalb eines Agentensystems nur eine bestimmte Anzahl an Ressourcen benutzen darf [Lei03]. [Gra96] verdeutlicht dieses Problem an einem Agenten, der auf jeder Agentenplattform zwei Kinderagenten startet, die wiederum zwei Agenten starten. Der Agent kann so sehr schnell ein System mit seinen Kinderagenten fluten, obwohl jeder Agent einzeln gesehen nur zwei Agenten erzeugt.

Wie bereits erwähnt ist die Achillesferse des Agentensystems der Schutz des Agenten vor einem böswilligen Host. Einige Ansätze stellen diesen Schutz unter bestimmten Rahmenbedingungen bereit, aber im Allgemeinen ist der Schutz schwer zu realisieren bzw. nicht möglich. Während bei allen vorher aufgeführten Angriffen reine Softwarelösungen betrachtet wurden, gibt es hier auch Hardwareansätze, um die Sicherheitsanforderungen zu erreichen. [Hoh01] unterscheidet die Kategorien die nur einen Gesamtschutz und einen Teilschutz bieten. Neben den Hard- und Softwarelösungen nennt [Hoh01] auch organisatorische Ansätze. Hier ist vor allem der Ansatz des vertrauenswürdigen Betreibers interessant. Je nach Anwendung ist die Annahme einer vertrauenswürdigen Agentenplattform keine Einschränkung für das Agentensystem.

Das folgende Kapitel betrachtet mobile Agentensysteme aus unterschiedlichen Bereichen und versucht diese mit Hinblick auf die ubiquitären Anforderungen zu vergleichen.

5 Verwandte Arbeiten

Seit Bekanntgabe der Idee des mobilen Agenten sind mit der ersten Euphorie bis Ende der neunziger Jahre zahlreiche Agentensysteme entwickelt worden. Die Meisten sind aus einer Anwendungsidee entstanden und sind speziell auf die daraus resultierenden Bedürfnisse implementiert. Die vorliegende Arbeit ist eine der Ersten, die den Einsatz von mobilen Agenten im Bereich der ubiquitären Systeme untersucht. Es gibt bisher kein Agentensystem, das speziell für ubiquitäre Umgebungen entwickelt wurde. Verwandte Projekte, die in letzter Zeit begonnen wurden, beschäftigen sich mit dem Einsatz schon vorhandener Agentensysteme aus anderen Bereichen in ubiquitären Szenarios. Da ein Agentensystem an sich eine Art Middleware darstellt, können durch Erweiterungen bestehender Komponenten neue Anforderungen erfüllt werden. Einige solcher Agentensysteme sollen hier kurz vorgestellt und anhand bestimmter Kriterien verglichen werden.

5.1 Vorhandene Agentensysteme

5.1.1 Agent Factory

Das erste Agentensystem, welches hier vorgestellt werden soll, trägt den Namen *Agent Factory* und wurde 1996 an der Universität Dublin in Irland entwickelt [CROO00]. Es stellt ein geschlossenes System dar, das anhand strukturierter Methoden die Entwicklung und den Einsatz von agenten-orientierten Anwendungen unterstützt. Diese Methoden basieren auf dem sogenannten Belief-Desire-Intension (BDI) Prinzip, welches die Entscheidungsfindung des Agenten abhängig macht von:

- Ziele des Agenten (Intention)
- Annahmen des Agenten (Belief)
- Vorhaben des Agenten (Desire)

BDI-Agenten passen ihr Verhalten an, indem sie einen mentalen Zustand, basierend auf einer Aggregation von verschiedenen mentalen Einstellungen, explizit modellieren.

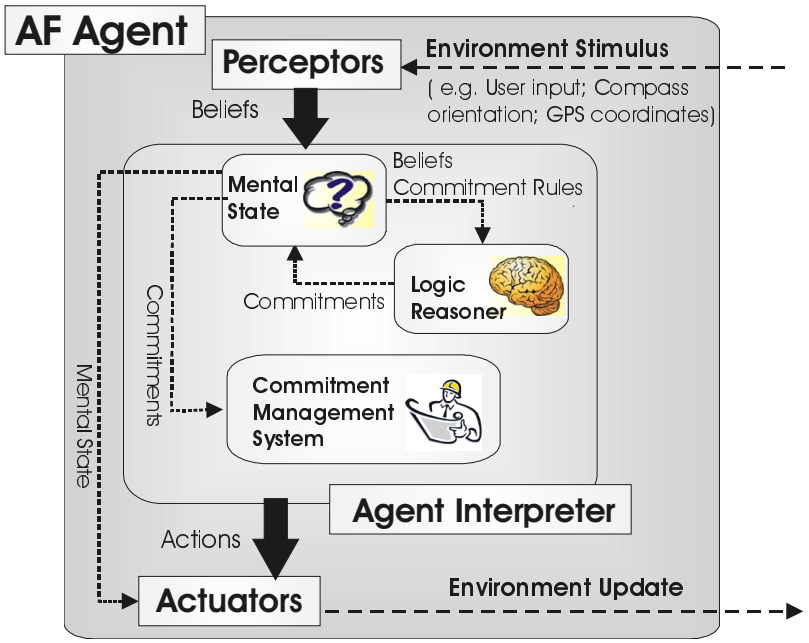

Abbildung 5.1: BDI-Agent in Agent Factory

Agent Factory bietet zwei Umgebungen an: die Agent Factory Entwicklungsumgebung und die Agent Factory Laufzeitumgebung. Die Erstere beinhaltet Computer-Aided Software Engineering (CASE) Tools, die den Agenten-Fabrikationsprozess unterstützen sollen. Die Laufzeitumgebung stellt die Infrastruktur der Agenten-Hosts dar.

Ein Agent besteht aus einem mentalen Zustand (*mental state*), Handlungregeln (*commitment rules*), Vorschriften (*perceptors*) und Aktuatoren (*actuators*). Der mentale Zustand beschreibt ein Modell der aktuellen Umgebung des Agenten in Form von Annahmen. Die Handlungsregeln repräsentieren das Verhalten des Agenten und definieren Situationen, in denen der Agent auf eine gegebene Handlung reagiert. Die Aktionen werden anhand von Aktuatoren ausgeführt. Mit Hilfe der Aktuatoren kann der Agent seine Umgebung beeinflussen. Die Vorschriften stellen funktionale Einheiten dar, mit denen der Agent seine Umgebung modellieren kann. Abbildung 5.1 stellt die Komponenten des Agenten grafisch dar.

Die erste Version von Agent Factory wurde in Smalltalk-80 programmiert. Der Einfluss der Programmiersprache Java überzeugte die Entwickler, so dass sie später eine weitere Version in Java anboten. Im Jahre 2000 wurde auch die Mobilität der Agenten unterstützt. Es wurden einige Anwendungen auf der Basis von Agent Factory entwickelt, die hauptsächlich aus den Bereichen E- und M-

Commerce stammen, wobei aktuelle Projekte auch im Ubiquitous Computing Bereich eingeordnet wurden [LOO02] [OO03] [KO03].

5.1.2 JADE

JADE ist eine Multiagenten-Plattform und Entwicklungsumgebung, die von einer Gruppe von Universitäten und industriellen Forschungseinrichtungen entwickelt wurde [BPR99]. Angeführt wird die Gruppe von der Telecom Italia Lab (TILAB), wobei auch Firmen wie Motorola und Siemens aktiv an der Entwicklung beteiligt sind. JADE steht für *Java Agent DEvelopement Framework* und bietet Systemkomponenten konform zur FIPA Spezifikation an wie der Naming-Service, der Yellow-Page Service, der Message-Transport und Parsing Service. Zugleich steht dem Benutzer eine Bibliothek von FIPA Interaktionsprotokollen zur Verfügung.

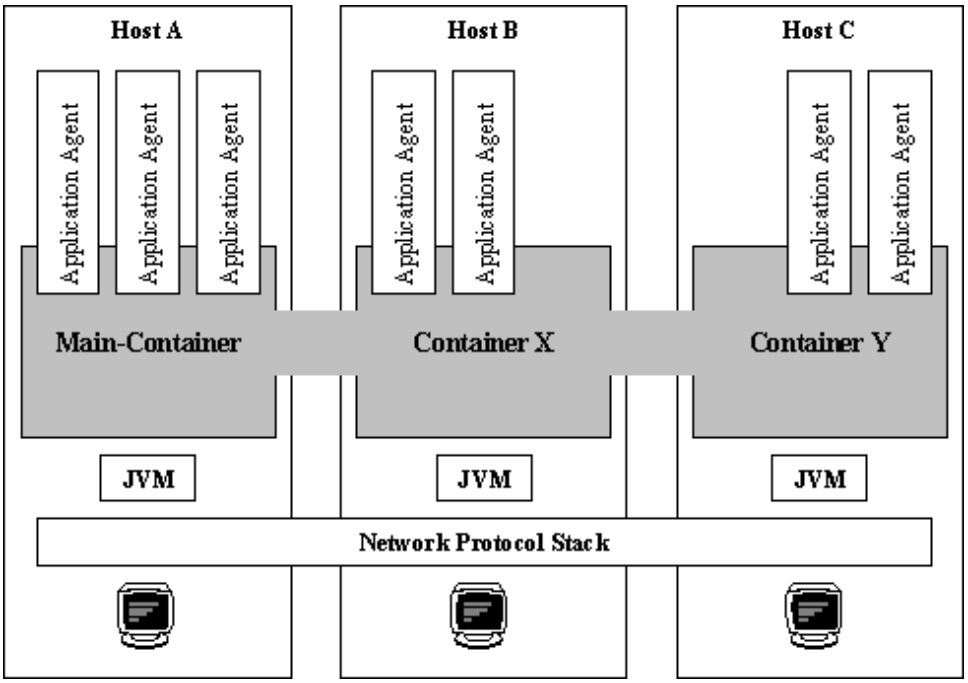

Abbildung 5.2: Verteilte Architektur der JADE-Agentenplattform

Jeder Agenten-Host besteht in JADE aus einem oder mehreren Containern, in denen die Agenten als Threads gestartet werden. Es muss mindestens ein Haupt-Container existieren, bei dem sich die anderen Container registrieren können. Der Haupt-Container verwaltet Tabellen für alle angeschlossenen Container und

alle im System laufenden Agenten. Abbildung 5.2 stellt die JADE-Architektur anschaulich dar.

Agenten kommunizieren miteinander durch den Austausch von Nachrichten, welche Java Objekte sind, die gemäß der FIPA Agent Communication Language (ACL) spezifiziert werden. Jeder Agent verwaltet selbst seine private Warteschlange von ACL-Nachrichten. Dabei können auch an fremde Agenten ACL Nachrichten gesendet werden, die auf anderen Plattformen als JADE laufen, aber FIPA konform sind.

Der Nachrichtentransfer innerhalb desselben Containers geschieht event-basiert, wogegen zwischen unterschiedlichen Containern die ACL-Nachricht mittels Java RMI (Remote Method Invocation) verschickt wird. Nachrichten zu fremden Plattformen können mittels Internet Inter-ORB Protocol (IIOP) oder HTTP verschickt werden. JADE wurde um einen Leightweight-Kernel, dem sogenannten JADE-LEAP erweitert, der den Einsatz auf mobilen Geräten ermöglicht. Die Agenten können zwischen JADE Containern migrieren, jedoch nicht zwischen JADE-LEAP Containern.

Projekte, die Anwendungen auf der Basis von JADE entwickeln, sind hauptsächlich aus den Bereichen E-Commerce und Mobile Computing [FAC00] [Mon00] [DIC00] [Sei05]. Im Bereich des Ubiquitous Computing wurde JADE bisher noch nicht eingesetzt.

5.1.3 Aglets Workbench

Die Aglets Workbench ist eine Java-basierte Entwicklungsplattform für autonome Agenten, die ursprünglich von IBM entwickelt wurde [KLO97]. IBM führte die Entwicklung nur bis zur Version 1.1 Beta 3 (lauffähig unter Java 1.1) fort und übergab den Quellcode an die Open-Source Gemeinde, welche schließlich neuere Versionen herausgab, die auch unter Java 1.3 laufen. Die Aglets Workbench setzt sich zusammen aus:

- **Aglets Framework:** Kernstück der Aglets Workbench ist das Aglets Framework, das Java um die agentenspezifischen Komponenten erweitert. Neue Agenten-Klassen können durch Ableitung aus der Aglets-Klasse erzeugt werden. Diese Basis-Klasse enthält bereits alle grundlegenden Elemente, die ein mobiler Agent benötigt, unter anderem einheitliche Namenskonvention, Migration, Kommunikations- und Kooperationsmechanismen usw.

- **Agent Transfer Protocol (ATP):** Dieses Protokoll dient dafür, den Agententransfer unabhängig von verwendeter Programmiersprache und Agentensystem zu ermöglichen.

- **JDBC Package:** Dieses Paket enthält Methoden, die es dem Agenten erlauben, auf Datenbanken zuzugreifen.

- **Tahiti:** Tahiti ist ein visuelles Tool, mit dem der Benutzer die Aktivität von Aglets auf seinem Rechner kontrollieren und beeinflussen kann.

- **Fiji:** Fiji dient dazu, Aglets in HTML-Webseiten zu integrieren.

Die Ausführungsumgebung von Aglets wird *Context* genannt. Der Context stellt Methoden zur Verfügung, um mit anderen Aglets, dem Host oder dem Benutzer über ein User Interface zu kommunizieren und schützt den Host vor direkten Zugriffen der Aglets. Um auch die Aglets vor direkten Zugriffen durch andere Agenten zu schützen, gibt es ein Proxy-Objekt, welches die privaten Methoden eines Aglets abschirmt. Außerdem fungiert ein Proxy als Vermittler für die Kommunikation zwischen Aglets, indem es als stationärer Stellvertreter den tatsächlichen Aufenthaltsort des Aglets versteckt.

Nachrichten, die Aglets austauschen, sind ebenfalls Objekte. Die Nachrichten können synchron oder asynchron übermittelt werden. Als Empfänger einer Nachricht fungiert immer der stationäre Proxy eines Aglets, der die Nachricht an das Aglet weiterreicht. Ein Aglet muss einen *Message Handler* implementieren, eine spezielle Methode, die beim Eingehen der Nachricht aufgerufen wird. Der Message Handler wird vom Proxy aktiviert.

Aglets wurden hauptsächlich im E-/M-Commerce Projekten eingesetzt [MB01]. Im ubiquitären Bereich gibt es auch hier keine Projekte.

5.1.4 Grasshopper

Das Grasshopper Agentensystem wurde von der IKV++ GmbH entwickelt und stellt die erste Agentenplattform dar, die sowohl OMG MASIF als auch FIPA konform ist. Das java-basierte Agentensystem beschreibt ein Modell mit den Komponenten *regions*, *places*, *agencies* und verschiedenen Typen von Agenten. Eine Region fasst Gruppen von Agencies, Places und Agenten zusammen. Eine Agency kann mehrere Plätze verwalten, auf denen sich Agenten aufhalten können. Für jede Region existiert ein Register, in dem alle Agencies dieser Region, zusammen mit jedem Place und allen Agenten, eingetragen sind. Abbildung 5.3 stellt das Distributed Agent Environment (DTE) von Grashopper grafisch dar.

Grasshopper unterscheidet zwei Arten von Agenten: stationäre und mobile Agenten. Die Ausführungsumgebung für Agenten ist die Agency, die aus einem Kernel und ein oder mehreren Plätzen besteht. Der Kernel bietet verschiedene Dienste zur Unterstützung der Agentenausführung, die im Folgenden beschrieben werden:

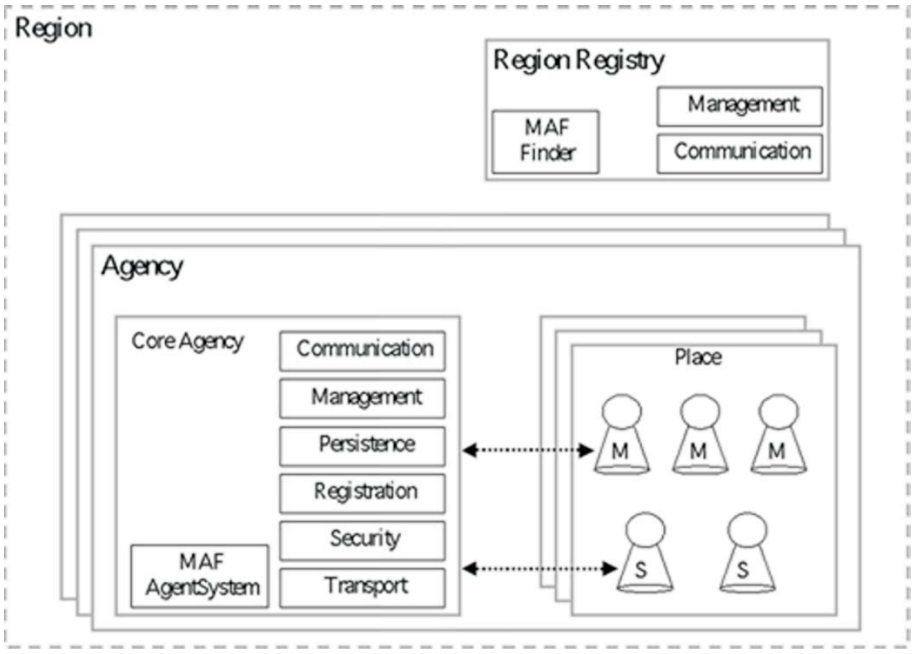

Abbildung 5.3: Distributed Agent Environment (DTE) von Grasshopper

- **Communication Service:** Dieser Service ist verantwortlich für alle entfernten Interaktionen, die zwischen den entfernten Komponenten von Grasshopper ablaufen wie die ortstransparente Agenten-Kommunikation, den Agententransport und die Agentenlokalisation. Die Kommunikation kann auf verschiedene Arten realisiert werden: CORBA IIOP, Java RMI oder schlichte Socket-Verbindungen.

- **Registration Service:** Der Registration Service verwaltet Informationen über Agenten, Agencies und Places. Über diesen Dienst kann jede Agency entfernte Agenten oder Places finden und Kontakt aufnehmen.

- **Management Service:** Dieser Dienst erlaubt das Monitoring und Kontrollieren von Agenten und Places einer Agency durch den (menschlichen) Benutzer. Agenten, Dienste und Places können kreiert, gelöscht, angehalten und wiedererweckt werden oder Listen über aktuelle Agenten oder Places ausgegeben werden.

- **Security Service:** Grashopper unterstützt zwei Sicherheitsmechanismen: externe Sicherheit und interne Sicherheit. Externe Sicherheit umfasst die Sicherung der Verbindungen zwischen entfernten Komponenten, welche durch Verwendung des SSL-Protokolls realisiert wird. Interne Sicherheit beschäftigt sich mit der Sicherung der Schnittstellen des Agenten-/Endsystems, was den unauthorisierten Zugriff auf Ressourcen verhindern

soll. Dies basiert hauptsächlich auf den Java Sicherheitsmechanismen.

- **Persistence Service:** Dieser Dienst erlaubt es Agenten und Places auf persistenten Medien zu sichern. Dies ermöglicht das Wiederherstellen des Systems nach einem Absturz.

Grasshopper wurde in verschiedenen Projekten aus dem Bereich E-Commerce, Management und Intelligent Networks eingesetzt [ANI00] [MAR99]. Projekte aus dem Bereich Ubiquitous Computing gibt es auch hier nicht.

5.1.5 Tracy Toolkit

Das Tracy Toolkit ist ein mobiles Agentensystem, das an der Universität Jena entwickelt wurde. Es ist ein modulares, komponent-orientiertes und erweiterbares Toolkit, welches in Java (Version 1.4) geschrieben ist. [BR05]

Tracy besteht aus einem Mikrokernel, das grundlegende Dienste anbietet wie die Ausführung und Kontrolle von stationären und mobilen Agenten. Oberhalb des Mikrokernels sitzt die Agency-Komponente, die alle aktuellen Agenten auf dem Host verwaltet. Zusätzlich muss die Agency verschiedene *Plug-Ins* managen. Ein Plug-In ist eine Softwarekomponente, die aus einem einzelnen Dienst besteht und die Funktionalität einer Agency erweitert. Plug-Ins können dynamisch geladen, im Fehlerfall gestoppt und später wieder gestartet werden. Tracy wird standardmäßig mit einigen Plug-Ins geliefert wie zum Beispiel die Inter-Agenten-Kommunikation, die Migration, einige Sicherheitskomponenten, Benutzermanagement und Zugriffsmanagement.

Bei der Entwicklung von Tracy galt der Hauptaugenmerk der effizienten Migration von Agenten. Es wurde ein neues Mobilitätsmodell, das sogenannte *Kalong*-Modell entwickelt, welches ein neues Migrationsprotokoll, das *Simple Agent Transport Protocol* (SATP) definiert. SATP arbeitet entsprechend einem einfachen Request/Reply-Schema und muss zusätzlich zu einem verbindungsorientierten Netzwerkprotokoll wie TCP benutzt werden.

Um die Migration zu verbessern, werden neben den Agencies sog. Code-Server betrieben, die den Code von Agenten zwischenspeichern. Bei optimaler Verteilung lassen sich dadurch die Transferzeiten bei der Migration minimieren. Wie oben erwähnt, werden auch Sicherheits-Plug-Ins angeboten, die Verfahren zum Schutz der Agenten und der Hosts vor Angriffen realisieren. Diese umfassen das Filtern der Klassen nach finalize-Methoden, um einen Angriff auf den Garbage-Kollektor zu verhindern, eine Agenten-Authentifikation und Methoden, die Datenobjekte als read-only markieren können oder die Daten verschlüsseln können, so dass sie nur auf einem bestimmten Host lesbar sind.

Tracy ist recht jung und hat noch keine Verwendung in externen Projekten gefunden, so auch nicht im Bereich ubiquitärer Systeme.

5.2 Gegenüberstellung

Betrachtet man die bestehenden mobilen Agentensysteme, so ist zu erkennen, dass ein speziell auf ubiquitäre Systeme zugeschnittenes Agentensystem nicht vorhanden ist. Die vorliegenden Systeme sind auf Anforderungen anderer Bereiche, wie E-Commerce oder Mobile Computing, angepasst oder sind bewusst generisch konzipiert, um ein breites Anwendungspotenzial zu erreichen. Ubiquitäre Systeme verlangen aber Eigenschaften, die spezialisierter sind, wie die Kontextsensitivität, personalisierte Inhalte oder Sicherheit. Das in dieser Arbeit entwickelte Framework, UbiMAS (Ubiquitous Mobile Agent System) Framework genannt, erfüllt im Gegensatz zu anderen Agentensystemen auch diese ubiquitären Eigenschaften. Die genaue Beschreibung von dem UbiMAS Framework folgt im nächsten Kapitel 6.

Die im vorhergehenden Abschnitt beschriebenen mobilen Agentensysteme und UbiMAS sollen hier anhand definierter Kriterien verglichen werden. Tabelle 5.1 stellt die Gegenüberstellung der mobilen Agentensysteme dar.

Auffällig ist, dass sich Java als Programmiersprache auch im Bereich der mobilen Agentensysteme etabliert hat. Alle Systeme sind in dieser Sprache implementiert. Dies liegt sicherlich an der Plattformunabhängigkeit von Java.

Die Wahl genau dieser Sprache beeinflusst jedoch die Migrationsart der Agenten. Der Hauptnachteil von Java ist die Tatsache, dass es unmöglich ist, den Ausführungszustand eines Threads an die Applikation weiterzugeben. Somit kommt nur die schwache Migration im Zusammenhang mit Java in Frage, wonach der Agent an dem Empfänger-Host nur in eine feste oder vorher angegebene Methode einspringen kann.

Als Migrationstechnik wird bei den meisten java-basierten Agentensystemen RMI verwendet. Einige Systeme bieten daneben alternative Protokolle an wie hier Aglets und Grasshopper, die zusätzlich IIOP, das CORBA Pendant zu RMI, realisieren. Der Nachteil von RMI liegt jedoch darin, daß ein zentrales Netzelement, der RMI-Registry-Server, vorhanden sein muß, damit zwei Java-Programme RMI nutzen können. Programme müssen in der Registry eingetragen werden, bevor sie per RMI benutzt werden können.

Das gleiche Problem besteht bei einer Lösung mit CORBA oder IIOP, damit Objekte (bzw. Agenten oder Java-Programme) miteinander kommunizieren können ist auch hier ein zentrales Element, der Object Request Broker (ORB) nötig,

der auch den Transport von Objekten (Agenten) zwischen zwei Plattformen ermöglicht. Ein Transport-Layer in Form von CORBA bietet jedoch den Vorteil, dass es sogar von der Programmiersprache unabhängig ist und damit auch Agenten aus anderen Programmierumgebungen verschickt werden können.

Einige wenige Agentensysteme implementieren ihr eigenes Protokoll. Dies hat den Vorteil, dass es direkt auf TCP/IP-Sockets aufsetzen kann und keine weiteren Layer wie RMI oder CORBA benötigt. Dadurch ist es zum einen möglich ganz spezielle Funktionen wie die effektive Verteilung der Agentenklassen einzubauen. Zum anderen kann ein solches Protokoll dem Ziel entsprechend schlank ausgeführt werden und genau auf das Problem der mobilen Agenten zugeschnitten werden. Nachteil ist hier, das zur Kommunikation des Agenten mit einer Zielplattform deren TCP/IP-Adresse bzw. die Portnummer des Kommunikationsports bekannt und offen sein muss, was gegebenenfalls zu Sicherheitsproblemen führen kann. Ein zentrales Verzeichnis wie die RMI-Registry gibt es hier ja nicht. ATP von Aglets und SATP vom Tracy Toolkit sind zwei Beispiele solcher Protokolle. UbiMAS geht hier einen ähnlichen Weg, setzt aber nicht direkt auf TCP/IP auf, sondern benutzt eine Peer-to-Peer-Middleware zur Übertragung von Nachrichten, die auch aus Agenten und deren Klassen bestehen können. Dadurch ist UbiMAS noch flexibler und unabhängig von der Transport- und Netzwerkschicht.

Alle Agentensysteme unterstützen autonome Agenten, d.h. die Agenten können ihren Handlungsablauf selbst anhand der Umgebungsinformationen definieren. Die Umgebung besteht dabei aus dem Host, anderen Agenten auf demselben Host und menschlichen Benutzer. Informationen werden über Nachrichten ausgetauscht. UbiMAS erweitert diese Umgebungssicht mit Informationen aus der realen Welt, die von Sensoren ins System eingespeist werden. Agenten in UbiMAS können also auch auf reale Ereignisse reagieren.

Um Namenskonflikte unter Agenten zu vermeiden, werden eindeutige Agenten-IDs benutzt. Die meisten Agentensysteme vergeben beim initialen Starten eines Agenten automatisch eine eindeutige ID oder leiten den Agenten-Namen aus eindeutigen Informationen ab wie z. B. die Netzwerkadresse des Erzeugerknotens. Andere Systeme wie das Tracy Toolkit verlagern die Vergabe der Agenten-IDs auf die Anwenderebene.

Agentenkommunikation ist ein sehr wichtiger Punkt beim Design von Agentensystemen. Um einen einheitlichen Nachrichtenaustausch zwischen unterschiedlichen Agenten und Agentensystemen zu gewährleisten, wurde von der FIPA die Agent Communication Language spezifiziert. Sie lehnt sich syntaktisch auf die aus der Künstlichen Intelligenz stammende *Knowledge Query and Manipulation Language* kurz KQML an, die von der amerikanischen Defense Advanced Research Projects Agency (DARPA) in Auftrag gegeben und finanziert wurde. In diesen Sprachen sind gemeinsame Fakten, Regeln, Bedingungen, Prozeduren und

Wissen formulierbar. JADE und Grasshopper implementieren ihre Nachrichten konform zur FIPA-ACL. Die anderen Agentensysteme unterstützen auch Agentenkommunikation, haben jedoch ihre eigenen Nachrichtentypen.

Das nächste Kriterium ist die Gruppenbildung. Hosts können sich aus Effizienz- oder Sicherheitsgründen zu Allianzen zusammenschließen. Lediglich JADE, Grasshopper und UbiMAS unterstützen diese Eigenschaft.

Personalisierte Inhalte sind an die persönlichen Profile der Benutzer angepasste Informationen. Die meisten mobilen Agentensysteme sind anwendungsgerichtet und nicht personengerichtet. UbiMAS unterstützt neben Service-Agenten auch Benutzer-Agenten, die benutzerspezifische Profile speichern. Diese mobilen Agenten können nicht nur personalisierte Informationen liefern, sondern die reale Umgebung an die Person anpassen, indem sie gezielt Geräte in der ubiquitären Umgebung steuern.

UbiMAS ist auch das einzige Agentensystem, welches kontext-sensitiv ist, d.h. UbiMAS Agenten können auf Sensorinformationen aus der Umgebung zurückgreifen und aus diesen Rohdaten komplexe Kontexte bilden. Der Abschnitt 6.3.2.4 schlägt eine Methode zur standardisierten Darstellung von Kontexten auf Basis von XML vor.

Die vielleicht wichtigste Eigenschaft, die ein ubiquitäres Agentensystem haben sollte, ist die Sicherheit. Agenten beinhalten sensible Daten, die nicht an unauthorisierte Instanzen weitergegeben werden dürfen. Das Agentensystem sollte Mechanismen gegen Angriffe anbieten, die sowohl gegen Hosts als auch gegen Agenten gerichtet sein können.

Die meisten Agentensysteme lassen diesen Punkt aus Angst vor Effizienzverlust unbeachtet. Agent Factory hat keinerlei Sicherheitsfunktionen implementiert. Die Grundversion von JADE bietet auch keine Sicherheitsmechanismen. Es wurde in einem späteren Stadium des JADE Projektes ein Sicherheitspaket herausgegeben, das Authentifizierung der Agenten und Plattformen einführt und eine Sicherung des Kommunikationskanals vornimmt. Aglets und Grasshopper beschränken sich auf sichere Kommunikation mittels SSL und dem Java Security Manager. Das Tracy Toolkit macht sich da schon mehr Gedanken und bietet Durchsuchung von Agentenklassen nach Unsicherheitmerkmalen und Kodierung von sensiblen Agentendaten. UbiMAS implementiert verschiedene Sicherheitsfunktionen, die in dem Abschnitt 6 detailliert beschrieben werden, um jeglichen Angriff von Seiten bösartiger Agenten abzublocken.

Tabelle 5.1: Gegenüberstellung der mobilen Agentensysteme

	Agent Factory	JADE	Aglets	Grass-hopper	Tracy Toolkit	UbiMAS
Software Technologie	Smalltalk-80/ Java	Java	Java	Java	Java	Java
Migration	schwach	schwach	schwach	schwach	schwach	schwach
Migrations-technik	RMI	IIOP/ RMI	ATP	IIOP/ RMI	SATP	P2P-Protokoll
Autonomie	√	√	√	√	√	√
Eindeutige Agenten-ID	?	√	√	?	–	√
Agenten-kommunikation	√	ACL	√	ACL	√	√
Gruppen-bildung	–	√	–	√	–	√
Personalisierte Inhalte	–	–	–	–	–	√
Context Sensitiv	–	–	–	–	–	√
Sicherheit	–	(√)	(√)	(√)	√	√

6 Das Ubiquitous Mobile Agent System Framework

Basierend auf den Anforderungen für ubiquitäre Agentensysteme und den Sicherheitsanforderungen wird in diesem Kapitel das Ubiquitous Mobile Agent System Framework (kurz: *UbiMAS Framework*) vorgestellt. Das UbiMAS Framework beschreibt ein Grundgerüst eines Agentensystems basierend auf einer ubiquitären Middleware [BPTU03a] [BPTU03c]. Es ist bewusst generisch konzipiert, um die Möglichkeit zu bieten, eine Vielzahl von Anwendungen nicht nur aus dem ubiquitären Bereich entwickeln zu können.

Die primären Ziele von dem UbiMAS Framework bestehen darin, den Anwendungen, speziell des mobilen Agentensystems, eine konforme und einfache Möglichkeit der Kommunikation zu bieten und eine vertrauensvolle und stabile Sicherheitsarchitektur zu realisieren. Wegen des ersten Zieles liegt eine bewusste Trennung der Agentenplattform von der Middleware-Schicht vor. UbiMAS-Hosts laufen oberhalb der Middleware als Dienste neben vielen anderen. Diese Trennung erlaubt es unabhängig von der eingesetzten Middleware eine einheitliche verteilte Anwendung auf Basis von UbiMAS-Plattformen zu realisieren. Die Middleware bleibt dadurch leicht austauschbar und kann je nach darunterliegender Netzwerktechnologie geeignet gewählt werden.

Zur Entwicklung einer geeigneten Sicherheitsarchitektur müssen einige Vorüberlegungen angestellt werden. Es muss untersucht werden, auf welcher Ebene welche Instanzen am passendsten sind, um Sicherheitsdienste zu implementieren. Auf Basis solcher Entscheidungen wurde die Sicherheitsarchitektur von UbiMAS entwickelt. Der erste Abschnitt dieses Kapitels stellt daher eine Bewertung der möglichen Konzepte dar, bevor im zweiten Abschnitt die Architektur des UbiMAS Frameworks vorgestellt wird. Die darauf folgenden Abschnitte gehen näher in die einzelnen Komponenten ein und beschreiben eine prototypische Entwicklung einer ubiquitären Middleware basierend auf Peer-to-Peer Technologie.

6.1 Bewertung möglicher Sicherheitskonzepte

Unabhängig von einzelnen konkreten Sicherheitsdiensten lässt sich ein System mobiler Agenten daraufhin untersuchen und bewerten, welche Instanzen im System geeignet und in der Lage sind, Sicherheitsdienste zu implementieren. Ausgehend vom Schichtenmodell für mobile Agentensysteme wie sie in Abbildung 3.3 vorgestellt wurde, werden die Beiträge und Grenzen der beteiligten Schichten im Folgenden untersucht. Jede Schicht wird unter der Annahme betrachtet, dass sie alle Sicherheitsdienste komplett realisieren muss. Durch diese Betrachtungsweise treten die Stärken, aber auch die Schwächen im Hinblick auf die Implementierung von Sicherheitsdiensten durch eine Schicht deutlicher hervor. Insbesondere wird dadurch auch klar, dass es Sicherheitsdienste gibt, die eine bestimmte Schicht für andere Schichten überhaupt nicht implementieren kann.

6.1.1 Sicherheitsdienste des Betriebssystems

Betrachtet man das Schichtenmodell von unten her, so ist die erste in Frage kommende Schicht das Betriebssystem. Es soll angenommen werden, dass das zum Einsatz kommende Betriebssystem, die Sicherheitsdienste bereitstellt. Das Agentensystem und die darauf ablaufenden mobilen Agenten erscheinen aus der Sicht des Betriebssystems als ein Prozess oder als eine Menge von Threads.

Ein solcher Ansatz hätte die Vorteile, dass bestehende und ausgereifte Sicherheitsdienste des Betriebssystems ohne Änderung übernommen und genutzt werden könnten. Die Implementierung der Sicherheitsarchitektur würde sich dabei auf eine geeignete Auswahl von Sicherheitsdiensten beschränken, die das Betriebssystem zur Verfügung stellt. Ein weiterer Vorteil bestünde zudem darin, dass sich ein Agentensystem nicht von einem „normalen“ Benutzerprozess unterscheidet. Dadurch lassen sich die Ressourcen des Betriebssystems völlig unabhängig vom verwendeten Typ des Agentensystems oder den mobilen Agenten sichern.

Das resultierende Sicherheitsmodell ist dadurch allerdings auch sehr starr. Die Dynamik des Systems mobiler Agenten kommt nicht zum Tragen und lässt sich in der Sicherheitsarchitektur auch nicht abbilden. Bei Betriebssystemen, bei denen Agentensystem und mobile Agenten als ein Prozess erscheinen, hat das Betriebssystem keine Kenntnis von den einzelnen Instanzen Agentensystem und mobiler Agent. Deshalb ist es auch nicht möglich, dass das Betriebssystem spezifische Sicherheitsanforderungen dieser Instanzen erfüllen kann. Das Betriebssystem kann in diesem Fall verschiedene mobile Agenten, die auf einem Agentensystem ablaufen, überhaupt nicht als eigenständige Komponenten erkennen. Im Allgemeinen muss davon ausgegangen werden, dass bei einer Implementierung der Sicherheitsarchitektur im Betriebssystem auch nur die Ressourcen des Betriebssystems, und

nicht mobile Agenten oder Agentensysteme, verlässlich geschützt werden können.

Bei Sicherheitsdiensten auf Betriebssystemebene ist eine dynamische Anpassung nur schwer möglich. Die sicherheitsrelevanten Entscheidungen werden größtenteils beim Start eines Prozesses festgelegt und können dann nicht mehr verändert werden, ohne den Prozess zu stoppen. Beispielsweise wird beim Start eines Prozesses festgelegt, wer für den Prozess verantwortlich ist und welche Rechte der Prozess besitzt. Mobile Agenten - die auf einem Agentensystem „kommen und gehen“ - für die verschiedene Organisationen verantwortlich sind, und denen dynamisch Rechte erteilt werden sollen, lassen sich damit nur sehr schwer oder überhaupt nicht realisieren.

6.1.2 Sicherheitsdienste des Agentensystems

Sollen die Sicherheitsdienste alleinig durch die Agentenplattform implementiert, überwacht und durchgesetzt werden, übernimmt der Agenten-Host als Laufzeitumgebung für mobile Agenten auch die Sicherung der Agenten. Vorteile einer solchen Sicherheitsarchitektur sind:

- Mobile Agenten können geschützt werden, unabhängig davon, ob sie eigene Sicherungsmaßnahmen implementieren. Die Entwicklung und Implementierung von mobilen Agenten wird dadurch vereinfacht.

- Der Agenten-Host, als Abstraktionsschicht oberhalb der heterogenen Betriebssysteme, bildet eine homogene Ausführungsplattform für mobile Agenten. Die Sicherheitsdienste, die der Agenten-Host zur Verfügung stellt, realisieren daher eine einheitliche Sicherheitsarchitektur, deren Dienste-Schnittstellen, auch über heterogene Systemgrenzen hinweg, einheitlich sind.

- Die Agentenplattform implementiert auch die Sicherheitsdienste, die zur Sicherung der Ressourcen des Betriebssystems verwendet werden. Auch diese Aufrufschnittstellen sind auf allen Betreibssystemen gleich und nicht vom unterliegenden Betriebssystem abhängig.

Bei der Realisierung der Sicherheitsdienste durch den Agenten-Host existieren auch Nachteile, die im Folgenden genannt werden sollen:

- Agenten müssen sich voll auf die Sicherheitsdienste des Agenten-Hosts verlassen und diesen auch vertrauen können, da sie selbst keinerlei eigene Schutzmaßnahmen implementieren.

- Auch die Betriebssysteme müssen sich auf die Sicherheitsdienste der Agentenplattform verlassen. Betriebssysteme sind zwar immer noch in der Lage, eigene betriebssystem-spezifische Sicherheitsdienste zu nutzen und z.B. den Zugriff des Hosts auf Ressourcen des Betriebssystems beschränken, müssen aber bei der Vergabe von Rechten die Interessen des Hosts wahren.

6.1.3 Sicherheitsdienste der mobilen Agenten

Bei diesem Ansatz werden alle Sicherheitsdienste durch den mobilen Agenten selbst implementiert, überwacht und durchgesetzt. Die Sicherheitsarchitektur wird durch die Gesamtheit der Sicherheitsdienste, die die verschiedenen mobilen Agenten implementieren, gebildet.

Der Vorteil dabei ist, dass ein Agent sich bezüglich seiner Semantik optimal selbst schützen kann. Das Wissen, welche Dienste und Ressourcen des Agenten zu schützen sind, ist bei der Implementierung des Agenten am umfangreichsten. Schutzdienste können zudem individuell und feingranular auf jeden einzelnen Agenten abgestimmt werden.

Nachteil eines solchen Sicherheitssystems wäre, dass Sicherheitsdienste von den verschiedenen Implementierern der Agenten zur Verfügung gestellt würden. Eine einheitliche Sicht auf die Schnittstellen lässt sich dabei nur schwer durchsetzen. Nicht jedem Implementierer kann vertraut werden. Die Sicherheitsdienste in ihrer Gesamtheit sind immer nur so gut wie die schlechteste Implementierung. Es würde eine Sicherheitsarchitektur entstehen, die voll auf Agenten basiert, die aber von verschiedensten Organisationen entwickelt wären. Dies würde implizieren, dass Agenten-Hosts und Betriebssystem dem möglicherweise fremden Agenten voll vertrauen müssten. Außerdem kann sich ein Agent nie vollständig schützen, da er unter der Kontrolle des Agentensystems ausgeführt wird.

6.1.4 Schlussfolgerung

Werden die Nachteile aus den vorhergehenden Abschnitten betrachtet, so wird klar, dass keine der drei Schichten Betriebssystem, Agentenplattform und mobiler Agent die vollständige Sicherheitsarchitektur implementieren kann. Dem Grundsatz nach sollte jede Schicht die Sicherheitsdienste implementieren, für die sie am besten geeignet ist. Das Betriebssystem kann seine Ressourcen (z.B. Dateien, Netzwerkverbindungen, u.ä.) selbst schützen. Dabei müssen aber das Agentensystem mit alle darauf laufenden Agenten als eine Einheit (ein Prozess) betrachtet werden. Die Rechte, die das Betriebssystem vergeben und kontrollieren kann, richten sich nach den Rechten desjenigen, der das Agentensystem startet.

Um feingranulare Sicherheitsmechanismen auf Basis von mobilen Agenten realisieren zu können, ist das Betriebssystem nicht geeignet. Deshalb wird im Agentensystem, als Mediator zwischen Betriebssystem und mobilem Agenten, der größte Teil der Sicherheitsarchitektur realisiert werden. Der Agenten-Host stellt eine Art „Agentenbetriebssystem“ dar und ist als solches auch am besten geeignet, um Sicherheitsdienste zu implementieren. Es gibt aber auch Fälle, in denen nur der mobile Agent „weiß“, welche seiner Ressourcen zu schützen sind. In diesem Fall muss der mobile Agent Entscheidungsregeln für den Zugriff auf seine Ressourcen festlegen. Für die technische Durchsetzung kann er sich auf Schnittstellen und Methoden des Agenten-Hosts abstützen.

Das UbiMAS Framework stellt ein Sicherheitsmodell vor, in der Sicherheitsaufgaben, die die knotenübergreifende Kommunikation betreffen, auf die Middleware verlagert werden. Die Dienste oberhalb der Middleware implementieren weitere Sicherheitsmethoden nach den jeweiligen Anforderungen des Dienstes. Der UbiMAS-Host kann sich also als Dienst auf eine sichere Kommunikation verlassen und muss nach den Anforderungen eines ubiquitären Agentensystems wie sie in Kapitel 4 beschrieben wurden, eine entsprechende Sicherheitsarchitektur realisieren. Im Abschnitt 7.5 wird diese Architektur näher erläutert.

6.2 Architektur des UbiMAS Frameworks

Dieser Abschnitt beschreibt das UbiMAS Framework und seine Komponenten mitsamt ihrer Funktionalitäten. Die Architektur des UbiMAS Frameworks besteht aus drei Schichten. Abbildung 6.1 stellt das UbiMAS Framework graphisch dar.

6.2.1 Betriebssystem und Netzwerkschicht

Die unterste Schicht des UbiMAS Frameworks ist als Betriebssystemschicht und Netzwerkschicht zusammengefasst. Das Betriebssystem besitzt alle Systemressourcen und verwaltet die Verteilung dieser auf die Anwendungen. Da eine Anforderung für das Agentensystem die Plattformunabhängigkeit ist, muss durch die geeignete Wahl einer Programmiersprache, die Schnittstelle zum Betriebssystem und seinen Ressourcen plattformübergreifend konform sein. Es wird hier nicht näher darauf eingegangen, wie die Schnittstellen zwischen den oberen Schichten und dem Betriebssystem genau auszusehen haben. Die Netzwerkkomponenten werden hier auch als Ressourcen des Betriebssystems angesehen. Der Zugriff auf das Netzwerk ist also auch über konforme Schnittstellen der Programmiersprache möglich.

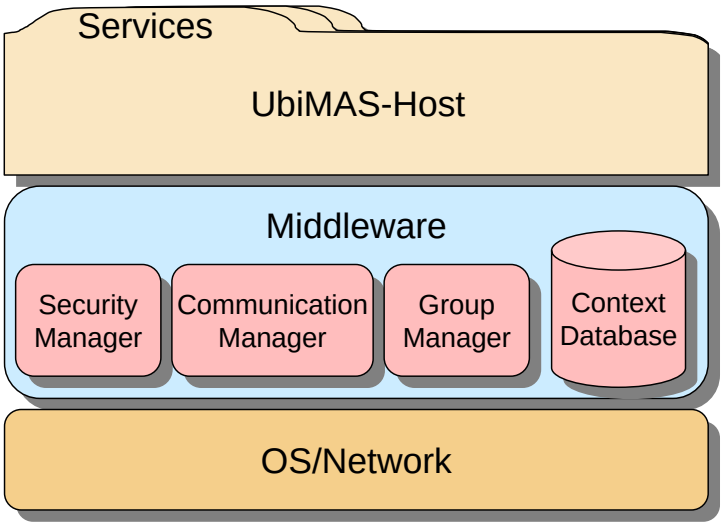

Abbildung 6.1: Das UbiMAS Framework

6.2.2 Middleware-Schicht

Die zweite Schicht des Frameworks ist die Middleware-Schicht. Middleware bezeichnet eine Softwareschicht zwischen dem Übertragungsnetzwerk und den Anwendungen. Ihre Aufgabe ist es, von den Eigenheiten und der Komplexität der verwendeten Infrastruktur zu abstrahieren und den Anwendungen eine reibungslose, standardisierte Interaktion zu ermöglichen.

Middleware stellt eine Ebene in einem komplexen Software-System dar, die als Dienstleister anderen ansonsten entkoppelten Softwarekomponenten die Kommunikation untereinander ermöglicht. Meist erfolgt diese Kommunikation mit Hilfe eines Netzwerkes, das durch die Middleware für die sie benutzenden Softwarekomponenten transparent gemacht wird.

Unter dem Begriff des Dienstes selbst ist eine Programmeinheit zu verstehen, die eine abgeschlossene Aufgabe erfüllt und von einem Diensteanbieter entwickelt, gewartet und angeboten wird. Das Spektrum möglicher Dienste reicht von systemnahen Diensten über komplexere, benutzernahe Anwenderdienste bis hin zu Standardwerkzeugen der Bürokommunikation. Dienste gliedern sich oft nahtlos in eine Middleware ein, die dafür sorgt, dass die Dienstnutzer einen Dienst finden und mit ihm interagieren können.

Eine Middleware stellt vor allem Dienste für Identifikation, Authentifizierung, Zugriff auf Systemressourcen, Nachrichtenaustausch und Sicherheit zur Verfügung, um so vernetztes Arbeiten zu erleichtern. Zusätzlich muss die ubiquitäre Middle-

ware das Erfassen und Speichern von Kontexten unterstützen und den Diensten eine geeignete Schnittstelle für den Abruf von Kontextdaten bieten. Entsprechend teilt sich die Middleware-Schicht im UbiMAS Framework in die Komponenten: Communication Manager, Group Manager, Security Manager und Context Database, auf die im Folgenden näher eingegangen wird.

6.2.2.1 Communication Manager

Der Communication Manager übernimmt die Aufgaben des Nachrichtentransfers zwischen Diensten entfernter Plattformen. In Richtung der oberen Schichten bietet die Middleware Standardmethoden zur Generierung, Modifizierung, Versendung und zum Empfang von Nachrichten. Empfängt die Middleware eine Nachricht von einem entfernten Knoten, so muss sie prüfen, an welchen Dienst die Nachricht gerichtet ist und sie an die entsprechende Komponente weiterleiten. Um die Middleware von der darunterliegenden Netzwerkschicht zu entkoppeln, kann der Communication Manager verschiedene Schnittstellen implementieren, die nach oben hin transparent sind. Dadurch kann eine optimale Anpassung an die verwendete Technologie auf der Netzwerkebene erreicht werden. Desweiteren bietet der Communication Manager einem Dienst die Möglichkeit, sich für Nachrichten anderer Dienste zu registrieren. Dadurch ist es einer UbiMAS-Agentenplattform möglich, Sensornachrichten, die von einem lokalen oder entfernten Dienst versendet werden, ebenfalls zu empfangen und an interessierte Agenten weiterzuleiten.

6.2.2.2 Group Manager

Je nach Anwendung kann es notwendig sein, dass geschlossene Gruppen auf der Ebene der Middleware gebildet werden müssen. Ein Location-Tracking Dienst könnte zum Beispiel über ein verteiltes Netzwerk von verschiedenartigen Sensoren, die jeweils von einem lokalen Dienst gesteuert werden, den Aufenthaltsort von Objekten innerhalb eines ubiquitären Gebäudes bestimmen. Die Bildung einer Gruppe unter den Lokations-Diensten hätte dabei den Vorteil, dass Nachrichten sicher innerhalb der Gruppe ausgetauscht werden könnten, ohne dass sie von außen stehenden Diensten bemerkt werden. Der Group Manager verwaltet mögliche Gruppen und unterstützt durch vordefinierte Methoden die Gruppenbildung, den Nachrichtenverkehr innerhalb einer Gruppe und die Kommunikation zwischen verschiedenen Gruppen. So erlaubt die Middleware neben Punkt-zu-Punkt-Verbindungen auch das Versenden von Broad- oder Multicast-Nachrichten zwischen den Knoten, die zu einer geschlossenen Gruppe angehören.

6.2.2.3 Security Manager

Der Security Manager ist für die Sicherung der Kommunikation zwischen Knoten und Gruppen zuständig. Die Sicherheitsmethoden basieren auf Public Key Verfahren. Es werden Funktionen bereitgestellt wie die Vergabe und Verwaltung der Schlüssel und Zertifikate, die Ver- und Entschlüsselung von Nachrichten unter Nutzung diverser Algorithmen, Authentifizierung der Knoten und Integritätsprüfungen der Nachrichten. Der Security Manager hat auch die Aufgabe, die Zugriffe auf Systemressourcen zu verwaltet. Da es sich um eine ubiquitäre Middleware handelt, können die Dienste auch auf Geräte in der ubiquitären Umgebung wie Sensoren oder Aktuatoren zugreifen. Der Zugriff muss über den Security Manager angefragt werden, der nach vordefinierten Policies die Entscheidung über das Gewähren oder Verweigern der Zugriffe fällt.

6.2.2.4 Context Database

Eine Anforderung von ubiquitären Systemen ist die Kontextsensitivität. Die ubiquitäre Middleware bietet eine Datenbank für Kontextinformationen in der lokale Kontexte erfasst werden. Lokal laufende Dienste können auf diese Informationen direkt zugreifen. Werden entfernte Kontexte benötigt, können diese über Nachrichten abgefragt werden. Wie und in welcher Struktur die Kontextinformationen erfasst und gespeichert werden, muss während der Implementierungsphase realisiert werden und wird daher nicht vom UbiMAS Framework spezifiziert. Es sollen hier lediglich die Kontext-Eigenschaften definiert werden, die für eine geeignete Umsetzung wichtig sind. Demnach besitzen Kontexte folgende Eigenschaften:

- *dynamisch*
  Kontexte sind in hohem Maße dynamisch. So ändert sich ein situationsbezogener Kontext bei jedem Ortswechsel oder jeder Änderung der Umgebungsbedingung. Kontexte dürfen daher nicht auf statische Werte festgesetzt werden, sondern müssen leicht modifizierbar sein.

- *zeitabhängig*
  Kontexte ändern sich mit der Zeit. Dabei lässt sich bei einigen Kontexten ein gewisser Zusammenhang zwischen Kontextwert und dem Zeitpunkt ableiten. Es gibt Kontexte, die periodisch die selben Zustände einnehmen und daher vom System vorhergesehen werden können. Dies trifft jedoch nicht auf alle Kontexte zu. So gibt es auch unberechenbare Kontexte, die willkürliche Zustände aufweisen.

- *ortsabhängig*
  Genauso wie die Zeitabhängigkeit spielt die Ortsabhängigkeit eine große

Rolle. Kontexte von mobilen Objekten sind stark ortsabhängig und nehmen je nach Ort verschiedene Zustände an.

- *untereinander korreliert*
  Verschiedene Kontexte beeinflussen sich gegenseitig, in dem Sinne, dass eine Änderung in dem einen Kontext auch eine Änderung des anderen Kontextes bewirken kann. Kontexte können also voneinander abhängig sein.

- *zusammensetzbar*
  Wie oben schon beschrieben können Kontexte aus vielen anderen Kontexten beliebig zusammengesetzt sein.

- *unscharf*
  Kontextwerte können binär, diskret, kontinuierlich oder aus einem bestimmten Intervall sein. Dies kann dazu führen, dass ein Zustandswert nicht genau definierbar ist. So kann ein situationsbezogener Kontext unscharf sein in dem Sinne, dass die Sensorwerte, welche die Situation umschreiben, ein großes Spektrum haben können.

6.2.3 Dienste

Die oberste Schicht im UbiMAS Framework bilden die Dienste. Ein UbiMAS-Host wird als Dienst oberhalb der Middleware gestartet. Daneben können viele verschiedene Dienste laufen, die auf derselben Middleware aufbauen. Dadurch ist eine nahtlose Kommunikation unter unterschiedlichen Diensten gewährleistet. Der UbiMAS-Host definiert eine Plattform für mobile Agenten und stellt Dienste zum Starten, Ausführen, Terminieren, zur Kommunikation und Migration von Agenten bereit. Dabei ist die Zahl der UbiMAS-Hosts auf einem Knoten nicht auf eins begrenzt. Es können mehrere Hosts parallel auf einem einzigen Knoten gestartet werden. Kapitel 6 beschreibt detailliert den Aufbau eines UbiMAS Agenten-Hosts. Im folgenden soll eine prototypische Implementierung der ubiquitären Middleware vorgestellt werden, die im Rahmen dieser Arbeit entwickelt wurde.

6.3 Ubiquitäre Middleware

Im Rahmen dieser Arbeit wurde das UbiMAS Framework in mehreren Phasen realisiert. In der ersten Phase wurde eine ubiquitäre Middleware implementiert, die aus den Grundkomponenten des UbiMAS Frameworks besteht. Als Kommunikationsinfrastruktur wurde das Peer-to-Peer-Konzept gewählt.

Mit Peer-to-Peer (P2P) Technologie bezeichnet man einen neuartigen Ansatz der Kommunikation in Computernetzwerken. In einer klassischen so genannten Client/Server Architektur stellt eine zentrale Instanz (Server) einer unbestimmten Anzahl von Systemen (Clients) einen bestimmten Dienst zur Verfügung. Ein entscheidendes Merkmal einer solchen Architektur ist, dass der Server einen so genannten „single point of failure“ darstellt: Fällt der Server aus, so kann der von ihm erbrachte Dienst von den Clients nicht mehr wahrgenommen werden.

Dies stellt genau den zentralen Gedanken der P2P Technologie dar. In einem P2P Netzwerk wird ein Dienst nicht von einer zentralen Instanz erbracht, sondern von einer Gruppe so genannter Peers. Dies hat den Effekt, dass beim Ausfall eines einzelnen Peers, der Dienst weiterhin von den verbleibenden Peers erbracht werden kann (wenn auch i.A. nicht in dem selben Umfang). Des weiteren ermöglicht dieser Ansatz höchste Skalierbarkeit [Wil02].

Außerdem kann in einem P2P Netzwerk jeder Peer sowohl Dienste wahrnehmen, als auch anbieten. D.h. in einem P2P Netzwerk hat jeder Peer sowohl die Funktion eines Servers, als auch die eines Clients. Charakteristisch für P2P Systeme ist, dass sie mit instabilen Netzwerkverbindungen und temporären Netzwerkadressen umgehen können. Die Knoten am Rande des Netzwerkes sind innerhalb eines P2P-Systems weitestgehend autonom.

Anwendungen der P2P Technologie finden sich besonders in den Bereichen Instant Messaging, File Sharing und Distributed Computing. Nachteilig an den heute weit verbreiteten P2P Anwendungen ist, dass jede eine eigene Implementierung des P2P Netzwerkes mit sich bringt, und sie nicht auf einem gemeinsamen Framework bzw. einer gemeinsamen Netzwerk-Schicht basieren.

Die hier vorgestellte ubiquitäre Middleware benutzt das Java P2P-System JXTA [Pro02]. Um das Verständnis für die Kommunikationabläufe in der Middleware zu erleichtern, soll das JXTA System im Folgenden vorgestellt werden.

6.3.1 Das JXTA P2P-System

Das Projekt JXTA ist ein von Sun Microsystems ins Leben gerufenes Entwickler-Team, das aus der Notwendigkeit heraus geformt wurde, eine solide und ausgereifte Basis für zukünftige P2P Entwicklungen zu haben. Gegründet wurde es Arpil 2001 unter der Leitung von Bill Joy und Mike Clary, die das Team bis heute führen.

JXTA fungiert mit seinen Protokollen als virtuelles Overlay-Netzwerk, das auf dem Internet und auf nicht IP-Netzen aufsetzt. Peers können somit unbeeinträchtigt durch Firewalls miteinander interagieren.

Das JXTA Projekt bietet eine Sammlung von P2P Protokollen öffentlich an, die jedem angeschlossenen Endgerät im Netzwerk erlaubt, als Peers miteinander zu kommunizieren und zu kollaborieren. Peers in JXTA können nicht nur leistungsstarke PCs oder Server sein, sondern auch kleinere Geräte wie Sensoren, Mobiltelefone oder PDAs. Für solche Kleinstgeräte, die auf Basis von Java 2 Micro Edition programmiert werden können, bietet die Projektgruppe das Packet JXME an. JXME ist eine „footprint“ Implementation von JXTA, d.h. es sind nicht alle Komponenten und Protokolle vorhanden wie in der Vollversion. Das eigentliche Handikap von JXME ist, dass es unbedingt einen JXTA Proxy benötigt, um kommunizieren zu können.

Jeder Peer arbeitet unabhängig und asynchron von allen anderen Peers und ist eindeutig identifiziert durch eine PeerID. Die Kommunikation unter den Peers geschieht durch den Austausch von Nachrichten, die im folgenden auch *Message* genannt werden. Die Message ist die Hüllklasse für den Datenaustausch. Die Daten, die verschickt werden sollen, werden der Message als *Message-Elemente* hinzugefügt. JXTA bietet verschiedene Arten von Message-Elementen an, wie beispielsweise *StringMessageElement* oder *ByteArrayMessageElement*.

Die Nachrichten werden über so genannte *Pipes* verschickt, welches ein asynchrones und unidirektionales Transportmechanismus darstellt. Pipes unterstützen unterschiedslos den Transfer von jeder Art von Objekten, sei es Binärcode, Daten-Strings oder andere java-basierte Objekte. Es gibt drei verschiedene Arten von Pipes:

- Input-Pipe: Dient zum Empfang von Messages.
- Output-Pipe: Dient zum Versenden von Messages.
- Propagate-Pipe: Propagate-Pipes verbinden einen Output-Pipe mit vielen Input-Pipes. Nachrichten können so an mehrere Empfänger gleichzeitig gesendet werden.

Eine bidirektionale Verbindung zwischen zwei Peers kommt zustande, indem jeweils die Input-Pipe des einen Peers an die Output-Pipe des anderen angeschlossen wird. Abbildung 6.2 stellt die JXTA-Pipes grafisch dar.

Alle JXTA Netzwerkressourcen sind in einem XML-Dokument, dem so genannten *Advertisement*, zusammengefasst. Advertisements sind sprach-neutrale Meta-Datenstrukturen, die von JXTA Protokollen benutzt werden, um Peer-Ressourcen zu beschreiben und zu veröffentlichen. Jeder JXTA Peer muss ein Peer-Advertisement definieren, welches den Peer-Namen, die Peer-ID und vorhandene Pipes beinhaltet. Abbildung 6.3 zeigt ein Beispiel für ein Peer-Advertisement in JXTA.

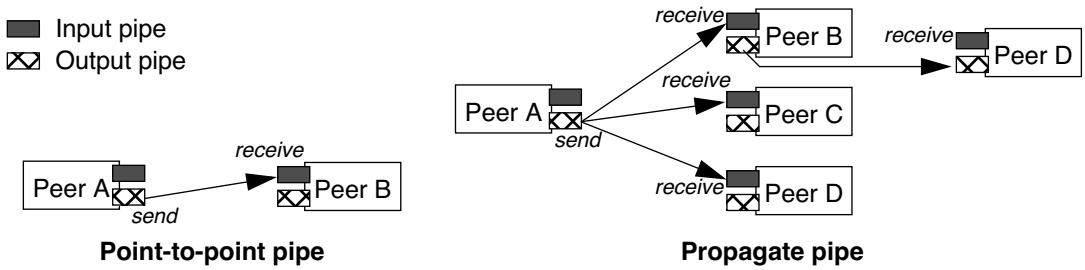

Abbildung 6.2: Pipes in JXTA

Jede Pipe wird ebenso durch ein Pipe-Advertisement spezifiziert. Ein Pipe-Advertisement besteht aus einer Id, einem Typ und einem Namen. Die Id dient zur Identifizierung der Pipe. Haben Input- und Output-Pipe die gleiche Id, kann über diese Pipes kommuniziert werden. Der Typ spezifiziert die Art der Pipe. Dabei wird zwischen Unicast- und Broadcast-Pipe unterschieden. Der Name dient nur zur Beschreibung und hat keinerlei Funktionalität.

Peergruppen werden auf Basis von Gruppen-Advertisements gebildet. Jeder Peer befindet sich im initialen Zustand in einer globalen Gruppe, die *Net-PeerGruppe* oder *Worldgroup* genannt wird. Darüber hinaus können Peers beliebig viele Peergruppen bilden oder Mitglied anderer Peergruppen werden.

```
<?xml version="1.0"?>

<!DOCTYPE jxta:PipeAdvertisement>

<jxta:PipeAdvertisement xmlns:jxta="http://jxta.org">
        <Id>
            urn:jxta:uuid-
59616261646162614E504720503250338E3E786229EA460DADC1A176B69B731504
        </Id>
        <Type>
            JxtaUnicast
        </Type>
        <Name>
            TestPipe.end1
        </Name>
</jxta:PipeAdvertisement>
```

Abbildung 6.3: Beispiel eines Peer-Advertisements in JXTA

Jede ID in JXTA wird eindeutig generiert und ist in Form von URNs [IET97] aufgebaut. Ein Beispiel für eine ID ist in Abbildung 6.3 zu sehen. Es gibt lediglich zwei reservierte IDs: die Null-ID und die ID für die Net-Peergruppe.

6.3.2 Prototypische Implentierung der Middleware

Um die Anforderung der Plattformunabhängigkeit einzuhalten, wurde die ubiquitäre Middleware in der Programmiersprache Java implementiert. Für leistungsstarke Knoten, wie PCs oder Workstations, wurde die Java 2 Standard Edition eingesetzt. Die Middleware lässt sich aber auch auf kleineren Geräten wie PDAs ausführen. Dafür wurde eine abgestufte Version in Java 2 Micro Edition implementiert.

Die Netzstruktur der ubiquitären Middleware entspricht der eines Peer-to-Peer-Netzes. Jeder Middleware-Knoten stellt einen Peer dar. Wie oben erwähnt befinden sich alle JXTA-Peers standardmäßig in der Net-Peergruppe. Die UbiMAS Middleware-Knoten verfügen über fest definierte Middleware-Advertisements und bilden schon nach dem Start zusammen eine geschlossene Peergruppe. Diese Middleware-Gruppe verfügt über Pipes, die nur innerhalb der Gruppe bekannt sind, so dass fremde Knoten keine Nachrichten empfangen können, die über diese Pipes verschickt werden.

Jeder Peer besitzt eine eindeutige ID, die einer JXTA Peer-ID entspricht. Diese ID wird beim ersten Erzeugen des Peers zufällig generiert und bleibt für die gesamte Lebensdauer gleich. Neben der ID besitzt jeder Peer einen Namen, der vom Entwickler beim Start angegeben wird. Bei einer großen Anzahl von verteilten Knoten wird es offensichtlich nicht möglich sein, die Eindeutigkeit von Namen zu garantieren, so dass die Adressierung allein über die IDs geschehen sollte.

Entsprechend des UbiMAS Frameworks ist die Middleware in die Komponenten Communication Manager, Group Manager, Security Manager und Context Database aufgeteilt. Die Implementierungen dieser Komponenten sollen im Folgenden genauer beschrieben werden.

6.3.2.1 Communication Manager

Der Communication Manager nutzt die JXTA Infrastruktur, um eine nachrichtenbasierte Kommunikation zwischen den Diensten zu ermöglichen. Dies umfasst in erster Linie Möglichkeiten zur Bearbeitung von Nachrichten und den Transfer der Messages.

JXTA und JXME unterscheiden sich in der Verarbeitung der Message-Elemente. So sind die Methoden für das Einfügen, Lesen und Löschen von Elementen in Nachrichten in beiden Versionen unterschiedlich benannt und umgesetzt. Um die Handhabung zu vereinheitlichen führt UbiMAS eine eigene Message-Struktur ein. Diese Struktur ist in der *UbiMessage*-Klasse implementiert. Die Elemente der Nachrichten werden in Form eines zweier Tupels erfasst, d.h. jedes Element besteht aus einem Schlüssel und den eigentlichen Daten in Form eines Byte-Arrays.

Jedes neue Element wird hinten an die Nachricht angehängt. Die Standardklasse von JXTA für Messages implementiert lediglich Methoden zum Anfügen und Löschen von Elementen. Die Eindeutigkeit der Schlüssel wird nicht überprüft. Es kann also vorkommen, dass zwei unterschiedliche Elemente mit demselben Schlüssel gespeichert werden. Beim Abrufen der Elemente würde JXTA das erste gefundene Elemente zurückgeben, d.h. es wäre nicht mehr möglich das nachfolgende Element zu erhalten ohne das Erste zu löschen. Die Klasse UbiMessage erweitert die JXTA Methoden, so dass die Eindeutigkeit der Schlüssel eingehalten wird. So kann man gezielt ein bestimmtes Element löschen oder überschreiben.

UbiMessage beinhaltet eine weitere Funktion, die das Erzeugen von Nachrichten betrifft. Es können nicht nur leere Nachrichten erzeugt werden, sondern Dienste können bei der Erzeugung eine schon existierende Message der Klasse UbiMessage mitgeben. Die Elemente der schon vorhandenen Nachricht werden einfach als Elemente der neuen Nachricht eingetragen. Damit lassen sich Nachrichten anderer Klassen bequem in eine UbiMessage umwandeln.

Da UbiMessage von der Standardklasse Message erbt, können Dienste beide Klassen zur Bearbeitung ihrer Nachrichten verwenden. Zusätzlich zu den Elementen, die die Dienste in die Nachricht eintragen, fügt der Communication Manager zu jeder Nachricht die Sender-PeerID, den Sender-Peernamen, den Typ des Senders und das Pipe-Advertisement der Input-Pipe des Senders hinzu. Mit diesen Informationen kann der Empfänger den Sender identifizieren und auf der richtigen Pipe antworten.

Grundsätzlich werden in UbiMAS drei Arten von Nachrichten unterschieden: eine Nachricht außerhalb von UbiMAS, innerhalb der Middleware-Peergruppe und innerhalb einer Service-Peergruppe, von welcher der Sender ein Mitglied ist. Der sendende Dienst kann dazu verschiedene Methoden zum Versenden der Nachricht aufrufen, je nachdem an welchen Peer oder welche Peergruppe die Message geschickt werden soll.

Nachrichten außerhalb von UbiMAS

Jeder Peer befindet sich innerhalb der Net-Peergruppe. Dienste können somit jeden JXTA-Peer über die Pipe dieser globalen Gruppe erreichen. Wird eine Broadcast-Nachricht an die Net-Peergruppe gesendet, erhält jeder Peer die Message, der in diesem Moment erreichbar ist. Kommunikation über die Net-Peergruppe ist daher sicherheitskritisch und sollte nur in wenigen begründeten Ausnahmen getätigt werden. Neben der fehlenden Sicherheit belasten solche Broadcast-Nachrichten unnötig das gesamte Netz. Peers außerhalb des UbiMAS Netzwerkes können auch einzeln und direkt angesprochen werden. Der Sender muss lediglich das Pipe-Advertisement des Empfängers kennen, auf den dieser nach eingehenden Nachrichten lauscht. Da JXTA zwar den Nachrichtenverkehr aber nicht die

Inhalte der Nachrichten und deren Weiterverarbeitung standardisiert, ist es für Peers fremder Anwendungen schwierig eine Kommunikationsbasis aufzubauen. Sie können zwar Nachrichten austauschen, die Analyse und Verarbeitung der Inhalte wird höchstwahrscheinlich Schwierigkeiten bereiten. Nichtsdestotrotz bietet der Communication Manager die Möglichkeit an alle Peers außerhalb UbiMAS Nachrichten zu senden.

Nachrichten innerhalb der UbiMAS Middleware-Peergruppe

Sollen Nachrichten nicht an alle Peers, sondern nur an die Mitglieder der UbiMAS Middleware-Peergruppe gesendet werden, können die Dienste die Nachricht als Middleware-Message deklarieren. Auch hier gibt es die Möglichkeit eine Broadcast-Nachricht an alle Middleware-Knoten oder eine Unicast-Message an einen Einzigen zu senden. Die Knoten werden über ihre IDs angesprochen, d.h. der Sender muss die ID des Empfänger kennen und an den Communication Manager weitergeben.

JXTA bietet keinen zuverlässigen und gesicherten Nachrichtentransfer, d.h. es ist nicht garantiert, dass die geschickte Nachricht auch wirklich beim Empfänger ankommt und die Inhalte nicht von unautorisierten Peers gelesen werden. Der Communication Manager implementiert daher Protokolle, die die sichere Übermittlung der Nachrichten garantieren. Es wird dabei zwischen zwei Protokollen unterschieden: die nicht-gesicherte und die gesicherte Kommunikation.

Die Kommunikation innerhalb der Middleware-Peergruppe basiert auf einem nicht-gesicherten Kommunikationsprotokoll, welches den gesamten Ablauf der Kommunikation regelt. Nicht-gesichert bedeutet hier, dass die Nachrichten unverschlüsselt gesendet werden und von jedem anderen UbiMAS-Dienst gelesen werden können. Ist die Nachricht an einen einzelnen Peer gerichtet, wird sie vom Empfänger quittiert. Falls eine Quittung ausbleibt, sendet der Communication Manager die Nachricht nach einer gewissen Zeit erneut. Nachrichten, die an alle Middleware-Knoten gerichtet sind, werden dagegen nicht quittiert. In der Regel wird die nicht-gesicherte Kommunikation zum Auffinden von bereits existierenden Peergruppen benutzt. Der weitere Nachrichtenaustausch wird dann über die gesicherte Kommunikation realisiert.

Nachrichten innerhalb einer Peergruppe

Das gesicherte Kommunikationsprotokoll regelt den Nachrichtentransfer auf der Ebene der Peergruppen, die durch Dienste gebildet wurden. Der Hauptunterschied zum nicht-gesicherten Protokoll betrifft die Sicherheit. Dienste, die über das gesicherte Protokoll kommunizieren wollen, müssen sich mit einem Zertifikat als Mitglied der Peergruppe ausweisen. Nachrichten, die innerhalb der Gruppe versendet werden, sind mittels einem Gruppenschlüssel verschlüsselt und können

nur durch die Gruppenmitglieder gelesen werden. Der Communication Manager arbeitet hier mit dem Group Manager zusammen und bietet Methoden zum Senden von Nachrichten:

- an alle vorhandenen Peergruppen,
- an eine bestimmte Peergruppe,
- an einen einzelnen Peer einer Peergruppe oder
- die eigene Peergruppe an.

Wird eine Nachricht an eine Peergruppe gesendet, bekommen automatisch alle Mitglieder diese Nachricht.

Neben der nachrichten-basierten Kommunikation beinhaltet die Middleware auch eine event-gesteuerte Kommunikation. Die Nachrichten können nach Typen klassifiziert werden. Jeder Dienst kann dabei seinen eigenen Typ definieren. Verteilte Dienste wie das UbiMAS Agentensystem haben ihren eigenen Nachrichtentyp. Damit kann der Communication Manager die Nachrichten gezielt lokal verteilen. Bei dem Empfang von neuen Nachrichten werden die entsprechenden Dienste per Event vom Manager benachrichtigt. Der Communication Manager leitet gleichzeitig den Inhalt der Nachricht direkt an den entsprechenden Dienst weiter. Die weitere Verarbeitung der Nachrichten ist den Diensten selbst überlassen.

6.3.2.2 Group Manager

Der Group Manager stellt den Diensten Schnittstellen zur Verfügung, die das Bilden und Verwalten von Peergruppen ermöglichen. Jeder Dienst kann eine Peergruppe erzeugen oder anfragen, ob es einer Peergruppe beitreten kann.

Dienste können über eine Broadcast-Anfrage an die Middleware-Knoten alle existierenden Peergruppen erfragen. Diese Anfrage wird von dem jeweiligen Group Manager des anfragenden Dienstes über eine Middleware-Nachricht an alle Knoten weitergeleitet. Innerhalb der Anfrage-Nachricht gibt der anfragende Peer das Pipe-Advertisement seiner Input-Pipe an, wo er auf eingehende Nachrichten wartet. Dienste, die Erzeuger einer Peergruppe sind und Interesse an einem neuen Peer-Mitglied haben, antworten über die angegebene Pipe mit Informationen über ihre Peergruppe. Falls kein Interesse an einer Mitgliedschaft besteht, ignorieren die Dienste die Anfrage und senden keine Antwort.

Die Peergruppen-Information beinhaltet die ID, den Namen und den Typen der Peergruppe und das Pipe-Advertisement, das die Gruppe zur Kommunikation

benutzt. Darüber hinaus werden die ID, Name, Typ und Pipe-Advertisement der Input-Pipe des Erzeuger-Dienstes angefügt.

Die Peergruppen-Informationen werden nach dem Eintreffen per Event an den anfordernden Dienst weitergeleitet. Falls dieser in eine existierende Peergruppe eintreten möchte, sendet er über den Group Manager eine Nachricht, mit der Anfrage um Beitritt. Der Empfänger kann dem Beitrittswunsch nachkommen oder es ablehnen. Hierzu sendet er jeweils eine vorgefertigte Nachricht vom Group Manager zurück. Wird dem Beitritt zugestimmt, sendet der Group Manager des Erzeuger-Dienstes ein Zertifikat, mit dem sich der neue Dienst ausweisen kann.

Ein Dienst, der eine neue Gruppe ins Leben rufen möchte, muss zunächst eine neue Input-Pipe entsprechend eines neu erzeugten Advertisements öffnen. Über vordefinierte Methoden werden die nötigen Informationen über die Peergruppe an den Group Manager weitergegeben, der diese Gruppe dann registriert. Anfragen an die Peergruppe werden dann an den Erzeuger automatisch weitergeleitet. Der Group Manager speichert alle Informationen über Peergruppen, die lokal erzeugt wurden und deren Mitglieder in einer Tabelle. Für die Verwaltung der Peergruppen ist der Group Manager verantwortlich. Der Communication Manager greift auf die Gruppen-Tabellen zurück, wenn es um das Senden von Nachrichten an Mitglieder einer Peergruppe geht.

Zwei Erzeuger von Peergruppen können untereinander eine Partnerschaft vereinbaren, so dass auch zwischen diesen Gruppen jeder Peer mit jedem über das gesicherte Kommunikationsprotokoll kommunizieren kann.

Möchte ein Dienst aus einer Gruppe aussteigen, muss dieser es dem Erzeuger der Gruppe mitteilen. Der Group Manager des Erzeugerdienstes entfernt den Peer dann aus der Gruppenliste. Wenn ein Erzeuger die Gruppe schließt, werden alle Gruppenmitglieder vom Group Manager automatisch benachrichtigt und die Gruppe aufgelöst. Abbildung 6.4 stellt eine mögliche Konstallation von UbiMAS-Peergruppen graphisch dar.

6.3.2.3 Security Manager

JXTA bietet in der aktuellen Version keine Sicherheitsmöglichkeiten, so dass die Middleware geeignete Schnittstellen realisieren muss. Die Aufgaben des Security Managers betreffen die Sicherung der Kommunikation zwischen den Middleware-Knoten und den darauf laufenden Diensten. Die Sicherheitsmechanismen basieren auf *Public Key Infrastructure (PKI)* Methoden.

Public Key Umgebungen ermöglichen das verschlüsseln von Informationen oder ganzen Verbindungen durch den Austausch von Schlüsseln. Hierbei hat jeder Teilnehmer einen öffentlichen (*public key*) und einen privaten (*private key*) Schlüssel.

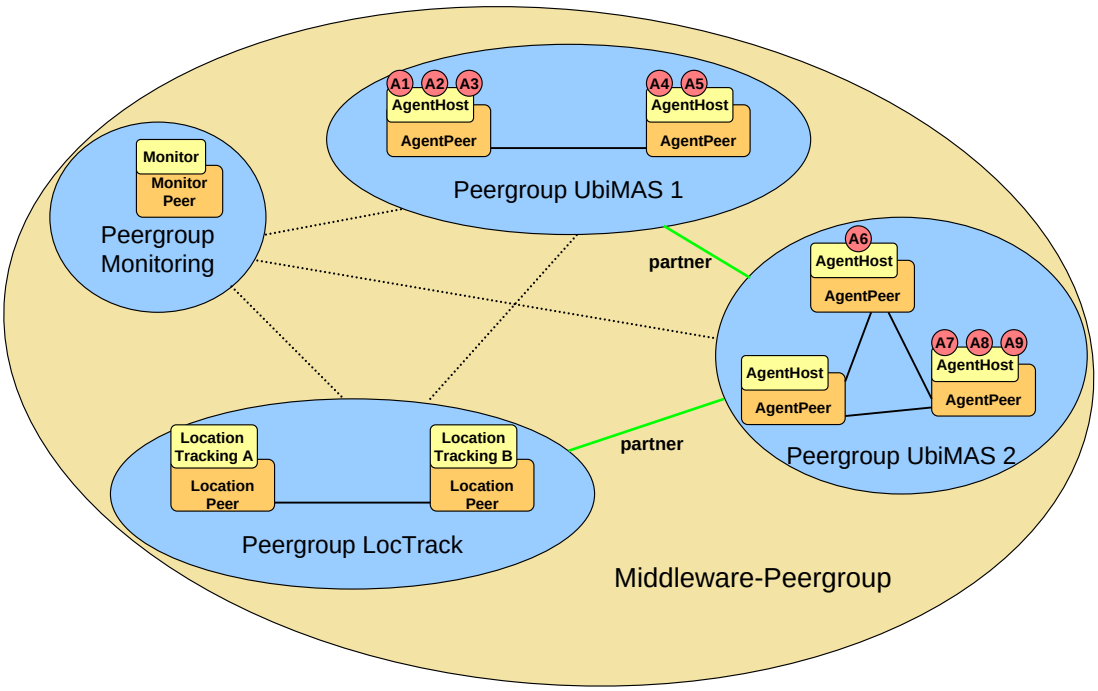

Abbildung 6.4: UbiMAS Peergruppen

Wollen nun zwei Partner auf Basis von PKI Informationen austauschen, so geben sie dem jeweils anderen Partner ihren öffentlichen Schlüssel. Dieser wird benutzt, um die Daten zu verschlüsseln, die sich nur mit dem privaten Schlüssel wieder entschlüsseln lassen. Somit ist immer gewährleistet, dass die Information nur demjenigen Partner zugänglich gemacht wird, für den sie auch bestimmt war. Andererseits können die Schlüssel nicht nur zum ver-/entschlüsseln, sondern auch zum signieren, d.h. sozusagen zum unterschreiben genutzt werden. Die Authentizität der signierten Daten kann unter Zuhilfenahme des öffentlichen Schlüssels gesichert werden. Hierbei kann ferner festgestellt werden, ob irgendwer sonst auf dem Weg die Daten manipuliert hat.

Wie im vorigen Abschnitt erwähnt, werden nur Nachrichten innerhalb der gesicherten Kommunikation verschlüsselt. Die nicht-gesicherte Kommunikation bietet keinerlei Sicherheit auf Basis der Middleware. Jeder Middleware-Knoten und optional auch Dienste besitzen einen öffentlichen und einen privaten Schlüssel. Die Inhalte der Nachrichten werden vor dem Versenden mit dem öffentlichen Schlüssel des Empfängers verschlüsselt.

Um die Manipulation der Nachrichten bei der Übertragung zu erkennen, werden die Nachrichten, wie oben beschrieben, vor dem Übertragen vom Sender signiert und die Signatur an die Nachricht angehängt. Beim Empfang der Nachricht wird vor ihrer Verarbeitung die Signatur überprüft.

Der Security Manager fordert nicht bekannte öffentliche Schlüssel vom Empfänger an und verwaltet die bekannten Schlüssel der Kommunikationspartner. Zudem verwaltet der Security Manager alle öffentlichen Schlüssel der lokalen Dienste. Der jeweilige public key des Senders wird bei der Übertragung der Nachrichten automatisch angefügt. Der Manager verschlüsselt die Nachrichten dann auf Wunsch mit dem jeweiligen public key des Empfängers. Die Dienste brauchen sich daher keine Gedanken über die Wahl der richtigen Schlüssel zu machen.

Die Kommunikation innerhalb von Peergruppen ist durch Gruppenschlüssel gesichert. Der Security Manager bietet den Diensten auch die komplette Verwaltung der Peergruppen-Schlüssel an. Beim Anlegen einer neuen Gruppe fordert der Group Manager ein neues Schlüsselpaar an, welches vom Security Manager generiert wird. Der Gruppenerzeuger bekommt dieses Schlüsselpaar und sendet sie an jeden neuen Dienst, der der Peergruppe beitretet, natürlich innerhalb einer verschlüsselten Nachricht, die nur der Empfänger lesen kann. Die Nachrichten, die an die Gruppenmitglieder gerichtet sind, werden mit Hilfe der Gruppenschlüssel verschlüsselt, so dass nur die Mitglieder sie entziffern können. Damit bietet die Middleware auch gesicherte Übertragung von Multicast-Nachrichten an.

Zur Verschlüsselung kommt eine Kombination von asymmetrischer (RSA) und symmetrischer (DES) Verschlüsselung zum Einsatz. Dabei wird vor der Übertragung ein Schlüssel erzeugt, womit die zu sichernden Daten per DES-Verfahren veschlüsselt werden. Die Daten können somit nur mit demselben Schlüssel entschlüsselt werden. Da dieser aber gerade erzeugt wurde, muss der Sender diesen an den Empfänger sicher übergeben. Der Schlüssel wird daher per RSA Verfahren mit dem public key des Empfängers verschlüsselt und der Nachricht angefügt. Damit ist sicher gestellt, dass nur der Empfänger den Schlüssel lesen kann und die verschlüsselten Daten in der Nachricht entschlüsseln kann.

Eine weitere wichtige Aufgabe des Security Managers ist die Vergabe von Zugriffsrechten auf Systemressourcen. Dienste können den Zugriff auf eine bestimmte Ressource beim Security Manager beantragen. Der Manager entscheidet dann, ob der Zugriff gewährt wird oder nicht. Da die Middleware in der Sprache Java implementiert ist, regelt der interne Security Manager von Java den Zugriff auf die Ressourcen. Dieser wurde aber im Zuge der Implementierung von UbiMAS erheblich erweitert, um den Anforderungen des Agentensystems zu genügen. Die Erweiterungen sind im Abschnitt 7.5 detailliert beschrieben.

6.3.2.4 Context Database

Die Kontextdatenbank wurde in der Implemtierung der Middleware in Form einer speziellen Datenstruktur umgesetzt, die es erlaubt sowohl einfachste als auch komplexe Kontexte effizient zu speichern. Die Datenstruktur basiert auf einem Kontext-Modell, welches XML als Darstellungssprache nutzt und im folgenden erläutert werden soll.

Kontext-Modell

In [BPTU03b] beschreiben wir ein Kontext-Modell, welches zunächst vom einfachen Kontext ausgeht und durch Zusammensetzung von diesen komplexe Kontexte aufbaut. Ein einfacher Kontext beschreibt eine einzelne Merkmalsinformation (z. B. „Licht An“). Ein Beispiel für einen komplexen Kontext stellt ein Raumkontext dar, der die Nachbarräume, Personen im Raum, Zustand des Lichtes usw. umfasst.

Gegeben sei eine Menge von Bezeichnern L sowie eine Menge von Zuständen S. Gesucht ist die Menge der Kontexte C. Ein Kontext $c \in C$ kann dann eine der folgenden Formen haben:

- Einfacher Kontext: $c := (L, S)$

  Beispiel: `Stromkontext = (Strom, an)`

- Zusammengesetzter Kontext: $c := (L, (L, S)^+)$

  Beispiel:
  ```
  Lichtkontext = (Licht, (Strom, an),
                         (Leistung, 60))
  ```

- Komplexer Kontext: $c := (L, C^+)$

  Beispiel:
  ```
  Raumkontext = (Raum, (Licht, (Strom, an),
                               (Leistung, 60)),
                       (Radio, (Strom, an),
                               (Lautstärke, 12),
                               (Sender, 3)))
  ```

Aufgrund dieser Modellierung, speziell des strukturierten Aufbaus von komplexen Kontexten, und der Anforderungen aus dem vorangegangenen Abschnitt wurde eine XML-basierte Umsetzung [Con02] dieses theoretischen Modells gewählt. Räume, Objekte und Zustände werden in Form von Elementen und Attributen in XML dargestellt. Das obige Beispiel sieht in der XML-Umsetzung wie folgt aus:

```
<room name="Raum">
```

```
  <device name="Licht">
    <function name="Strom" value="an"/>
    <function name="Leistung" value="60"/>
  </device>
  <device name="Radio">
    <function name="Strom" value="an"/>
    <function name="Lautstärke" value="12"/>
    <function name="Sender" value="3"/>
  </device>
</room>
```

Das Beispiel zeigt einen Raumkontext, wie er innerhalb eines Gebäudekomplexes für jeden Raum existieren könnte. Der Raumkontext setzt sich hierbei aus den Kontexten verschiedener Geräte zusammen. Raumkontexte können lokal auf Rechnern in den jeweiligen Räumen verfügbar sein und aufgrund der standardisierten Darstellung zwischen diesen potentiell heterogenen Rechner ausgetauscht werden.

Die Kontextinformationen werden von den ubiquitären Diensten (z.B. den im nächsten Abschnitt beschriebenen mobilen Agentensystem) genutzt. Möchte ein Dienst einen bestimmten Kontext erfragen, sendet dieser eine Nachricht an die Middleware. Innerhalb dieser Message kann der Dienst genauer spezifizieren, von welcher Instanz er den aktuellen Kontext haben möchte. Der Umfang der Kontextinformation hängt von der Größe der angegebenen Instanz ab. Wird z.B. ein Raum als Instanz angegeben, werden alle Kontexte von allen Geräten, die sich innerhalb dieses Raumes befinden zurückgegeben. Die Kontextinformationen werden dem Dienst wiederum in Form einer Nachricht geschickt.

Für die Behandlung von XML-Strukturen gibt es verschiedene Ansätze. Im allgemeinen wird die XML-Datei in eine Datenstruktur *geparst*, die sich dann im Speicher befindet und effizient bearbeitet werden kann. Als Java-basierte Lösung für das Lesen, Modifizieren und Zurückschreiben der XML-Informationen wurde das JDOM Softwarepaket [jdo02] verwendet.

7 UbiMAS-Host Architektur

Dieses Kapitel beschreibt die eigentliche Anwendung, das ubiquitäre mobile Agentensystem UbiMAS. Das Agentensystem besteht aus einem Netzwerk von entfernten Agenten-Hosts. Die UbiMAS-Hosts bieten den mobilen Agenten eine Ausführungsplattform und sind entsprechend dem UbiMAS Framework Dienste oberhalb der ubiquitären Middleware [BSP+05a] [BSP+05b]. Neben den Hosts können parallel so viele andere Dienste laufen wie die Middleware es erlaubt.

Die UbiMAS-Hosts bilden untereinander eine Peergruppe. Nach dem Starten versucht der Host bestehende Peergruppen zu finden, indem er über die Middleware eine Broadcast-Anfrage sendet. Kommen nach einer bestimmten Wartezeit keine Antworten, geht der Agenten-Host davon aus, dass er der Erste ist und erzeugt mit Hilfe des Group Managers eine neue Peergruppe. Für die sichere Kommunikation unter den UbiMAS-Hosts müssen neue Schlüssel und Zertifikate generiert werden. Der Erzeuger-Host einer UbiMAS-Peergruppe übernimmt dabei automatisch die Rolle einer Certification Authority (CA). Die CA erzeugt für jedes neue Mitglied jeweils einen neuen öffentlichen und privaten Schlüssel und vergibt Zertifikate, mit denen sich die Hosts authentifizieren können. Die genaue Abfolge und weitere Sicherheitskonzepte in UbiMAS werden im Abschnitt 7.5 näher beschrieben.

Die UbiMAS-Hosts sind modular aufgebaut und bestehen aus mehreren Komponenten. Hauptaufgabe des Hosts ist, eine Laufzeitumgebung für mobile Agenten anzubieten. Neben der Ausführung der Agenten ist der Host für die Verwaltung des Agentensystems, die Kommunikation, die Agentenmigration und die Sicherheit des Systems verantwortlich.

Die UbiMAS-Hosts basieren auf der abstrakten Klasse *AgentHost*. UbiMAS soll als Plattform für die verschiedensten Arten von Agentenanwendungen dienen. Aus diesem Grund regelt UbiMAS nur die grundlegendsten Aufgaben eines Agentensystems. Auch die Agentenklasse implementiert nur die notwendigsten Anforderungen.

Dabei ergibt sich die Schwierigkeit, dass einerseits die für ein Agentensystem benötigten Funktionalitäten wie Sicherheit, Kommunikation und Migration komplett integriert werden müssen und andererseits den Entwicklungen von eigentlichen Anwendungen keine Einschränkungen gemacht werden sollen. Daher wurde

bei der Entwicklung von UbiMAS auf geeignete Modularität geachtet, damit Erweiterungen durch Anwendungsentwickler schnell und einfach gestaltet werden können. Abbildung 7.1 stellt die UbiMAS-Host Architektur samt ihrer Komponenten graphisch dar. Die einzelnen Bausteine und ihre Aufgaben werden im folgenden detailliert beschrieben.

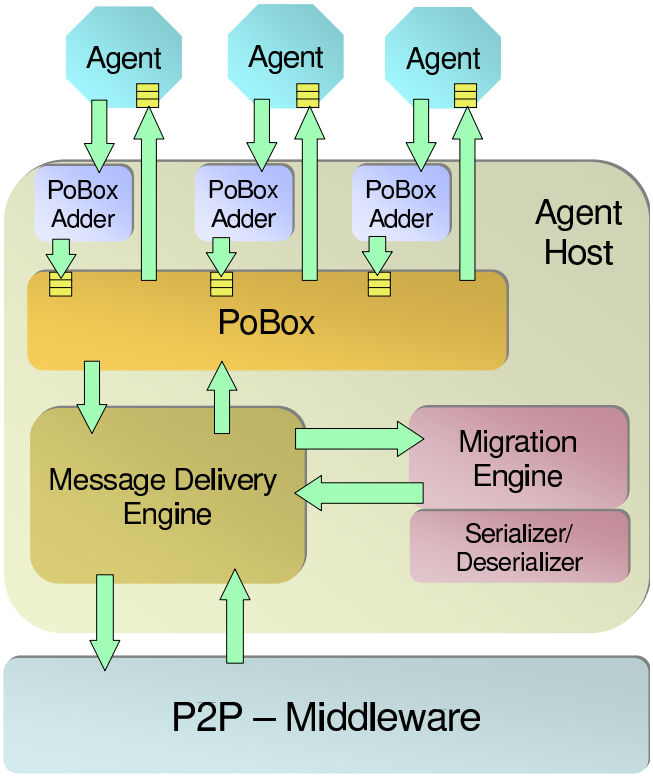

Abbildung 7.1: UbiMAS Host Architektur

7.1 Message Delivery Engine

Die direkte Schnittstelle zu der ubiquitären Middleware bildet die Message Delivery Engine. Diese Engine leitet ausgehende Nachrichten an die Middleware

weiter. Eingehende Nachrichten werden von der Middleware per Event an den Host weitergeleitet.

UbiMAS Hosts definieren ihren eigenen Nachrichtentyp. Bekommt die Middleware eine UbiMAS Host Message werden alle lokalen Hosts benachrichtigt. Die Message Delivery Engine prüft daraufhin, ob der eigene Host Empfänger der Nachricht ist. Falls dies zutrifft, wird die Nachricht weiter verarbeitet, sonst verworfen.

Als Empfänger einer UbiMAS Message kann entweder ein Host oder ein Agent eingetragen sein. Ist es der Host, entschlüsselt die Message Delivery Engine die Inhalte der Nachricht mit dem privaten Schlüssel des Agenten-Hosts und bearbeitet die Daten entsprechend der Inhalte weiter.

Neben den Nachrichten zur Bildung von Peergruppen, kommunizieren in der Grundversion von UbiMAS Hosts untereinander nur, um Nachrichten zwischen Agenten auf entfernten Hosts zu vermitteln, um den Aufenthaltsort eines Agenten ausfindig zu machen oder um Agenten-Code oder Klasse zu versenden.

Agenten migrieren von einem Host zum anderen, indem der Code und die Klasse des Agenten in eine Nachricht gepackt und verschickt werden. Der genaue Ablauf der Migration wird in dem Abschnitt 7.3 detailliert beschrieben. Die Aufgabe des Message Delivery Engines ist hier den Prozess des Packens bzw. Entpackens zu veranlassen und zu koordinieren. Der Migration Engine wird dafür beauftragt, den Agenten in eine Nachricht zu serialisieren und die nötigen Headerinformationen zu füllen. Die Nachricht wird dann wie andere UbiMAS-Messages über die Middleware gesendet.

Der Agenten-Host merkt sich jeden Agenten, der schon mal auf ihm ausgeführt wurde. Zieht der Agent weiter, wird zusätzlich zur ID des Agenten der zuletzt bekannte Aufenthaltsort gespeichert.

Eine Nachricht, die an einen lokalen Agenten adressiert ist, wird von der Message Delivery Engine, nachdem die Daten entschlüsselt worden sind, an die PoBox des Agenten-Hosts weitergereicht, die eine direkte Schnittstelle zum Agenten besitzt. Die PoBox kennt alle lokalen Agenten und übernimmt zugleich die Weiterleitung der Nachrichten an Agenten, die sich nicht auf dem aktuellen Host befinden.

Liegt eine Nachricht an einen entfernten Agenten vor, sendet die Message Delivery Engine diesen an den letztbekannten Host, wo der Empfängersagent vermutet wird. Sollte sich der Agent nicht mehr auf diesem Host befinden, kann die Message Delivery Engine eine Suchanfrage an alle UbiMAS-Hosts senden. Der Host, auf dem der Agent sich aktuell befindet, antwortet mit einer Anwesenheitsnachricht, so dass die Nachricht direkt an diesen weitergeleitet werden kann.

Kann eine Nachricht, aus welchen Gründen auch immer, nicht versendet werden,

wird der sendende Host oder Agent benachrichtigt. Das weitere Vorgehen ist dann dem Sender überlassen.

Möchte ein Entwickler die Funktionalität des Agenten-Hosts erweitern, braucht er lediglich eine Host-Klasse zu definieren, die von der Basisklasse erbt und einen speziellen Event implementiert. Wird eine Nachricht unbekannten Typs erhalten, wird diese dann nicht gleich gelöscht, sondern per Event an mögliche erbende Host-Klassen weitergeleitet.

7.2 Mobile Agenten

Neben dem Host ist der Agent die zweite Hauptkomponente des Agentensystems. Ein Agent muss in UbiMAS ein Java Objekt vom Typ *Serializable* sein oder eine Unterklasse dessen bilden, da bei der Migration der Agent in einen Byte-Array umgewandelt werden muss. Der Objektstatus beinhaltet alle Agentenattribute, welche auch serialisiert werden müssen.

UbiMAS Agenten haben eine allgemein eindeutige ID, welche sich während des Agentenlebens nicht ändert. Es handelt sich dabei um eine Pseudo-Zufallszahl, die bei der Erzeugung des Agenten einmal generiert wird. Allgemein eindeutig bedeutet, dass nie zwei Agenteninstanzen innerhalb einer UbiMAS-Peergruppe existieren dürfen. UbiMAS implementiert einen Algorithmus, bei dem ein 32-stelliges Byte-Array benutzt wird, um die Wahrscheinlichkeit, dass zwei identische Zufallszahlen erzeugt werden, sehr gering zu halten.

Mobile Agenten können zu sendende Nachrichten oder Daten, die der Agent beinhaltet, verschlüsseln. Für diesen Sicherheitsvorgang braucht der Agent ein Schlüsselpaar. Bei der Erzeugung des Agenten werden zwei Schlüssel, ein privater und ein öffentlicher Schlüssel, generiert. Zur Verschlüsselung implementiert die abstrakte Agentenklasse dieselbe Methode wie die ubiquitäre Middleware, die aus einer Kombination von asymmetrischer (RSA) und symmetrischer (DES) Verschlüsselung besteht. Der abstrakte Agent definiert Methoden, womit die Agenten die öffentlichen Schlüssel anderer Agenten anfordern können.

Der Agent ist über einen persönlichen PoBoxAdder mit dem lokalen Host verbunden. Nachrichten, die der Agent sendet, werden über den PoBoxAdder in die Warteschlange des Agenten-Hosts eingefügt. Für die eingehenden Nachrichten besitzt jeder Agent eine persönliche Warteschlange. Der lokale Host fügt Nachrichten, die an den Agenten adressiert sind, in die Warteschlange ein. Diese werden dann der Reihe nach von Seiten des Agenten verarbeitet.

Der Agent wird als eigener Thread gestartet. Auf einem einzigen UbiMAS Host können mehrere Threads parallel laufen, d.h. es können sich dort mehrere Agenten

gleichzeitig befinden. Jeder Agent bekommt beim Starten einen PoBoxAdder zur Verfügung, der die einzige Verbindung darstellt, über den der Agent mit dem Host kommunizieren kann. Der Agent stellt dem Host auch nur die Methoden zur Verfügung, die er für das Laden und Entfernen des Agenten benötigt. Die eigentliche Kommunikation findet über den Austausch von Nachrichten statt.

Der abstrakte Agent definiert hierzu grundlegende Methoden zum Erzeugen und Senden von Nachrichten. Der Agenten-Host dient hierbei als Vermittler. Nachrichten, die an einen lokalen Agenten gerichtet sind, werden über die lokale PoBox des Hosts direkt an den Empfänger weitergeleitet. Befindet sich der Agent auf einem entfernten Host vermittelt der lokale Host die Nachricht weiter an den entsprechenden Agenten-Host, der dann die Message an den Empfänger-Agenten übergibt.

Die Kommunikation über Nachrichten hat zwei wesentliche Vorteile:

- Sicherheit: Dadurch, dass die Agenten keine Referenz von dem Knoten haben, sondern nur den PoBoxAdder, können sie den Agentenknoten nicht manipulieren. Außerdem kann der Host auch keine beliebigen Funktionen des Agenten aufrufen.
- Erweiterbarkeit: Wird die Funktionalität des Agenten-Hosts oder des Agenten erweitert, muss nur das Kommunikationsprotokoll erweitert werden und nicht der bestehende Code.

Die Abbildung 7.2 stellt die einzelnen Zustände im Leben eines UbiMAS-Agenten graphisch dar. Die erste Stufe bildet die Erzeugung des Agenten. Agenten können von jedem beliebigen Host gestartet werden. Die Agenten-Klasse muss sich auf dem lokalen Dateisystem des Agenten-Hosts befinden oder kann von einem entfernten Knoten heruntergeladen werden. Bei der Erzeugung gibt der lokale Host seine ID und seinen öffentlichen Schlüssel an den Agenten weiter. Bevor der Agent gestartet wird, muss er vom Host registriert werden. Dabei wird der Agent in die aktuellen Tabellen des Hosts eingetragen.

Jedem Agenten auf dem Host ist ein *Notifier* zugeordnet. Dieser überwacht die Ausführung des Agenten und achtet darauf, dass der Agent sich nicht aufhängt. Nach dem Starten des Agenten befindet sich dieser in einem Wartezustand. Wenn nach einem bestimmten Timeout keine Nachricht an den Agenten eintrifft, wird der Agent vom Host über den Notifier informiert und führt die Methode *checkToDo* aus, die durch den Entwickler implementiert werden muss. Innerhalb dieser Methode kann der Agent je nach Programmierung zu einem anderen Host migrieren. Bleibt er auf dem aktuellen Host, kehrt der Agent in den Wartezustand zurück.

Ein Agent reagiert auf Nachrichten, d.h. er befindet sich in einer Schleife, wo er

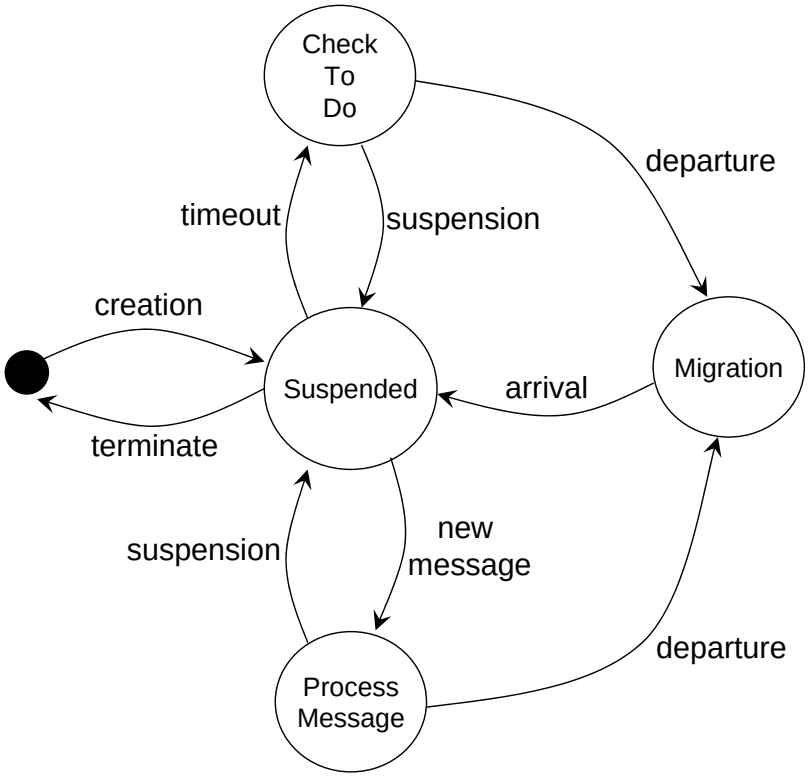

Abbildung 7.2: Lebenszyklus eines UbiMAS Agenten

auf eingehende Nachrichten wartet und dann entsprechend Aktionen ausführt. Treffen Nachrichten in die Warteschlange des Agenten ein, werden diese nacheinander verarbeitet. Dies ist mit dem Methodenaufruf *processMessage* realisiert. Der Entwickler kann durch Erweitern dieser Methode dem Agenten bestimmte Funktionen auftragen, die bei bestimmten Nachrichten ausgeführt werden. Innerhalb der *processMessage* Methode werden zunächst die Headerinformationen aus der Nachricht in lokale Variablen gespeichert. Hierzu gehören zum Beispiel die ID des sendenden Agenten, falls es eine Agentennachricht ist, die ID des sendenden Hosts und die jeweiligen öffentlichen Schlüssel des Senders, die der Nachricht angefügt wurden. Je nach Sendertyp, d.h. Host oder Agent, führt der Agent die Methoden *processHostMessage* oder *processAgentMessage* aus. Diese Trennung hat den Vorteil, dass Erweiterungen durch den Entwickler getrennt und unabhängig durchgeführt werden können.

Soll bei der Verarbeitung der Nachricht der Agent verschickt oder zerstört werden, wird das Attribut *Finished* gesetzt und der Thread gestoppt. Bevor ein Agent verschickt werden kann, muss er deregistriert werden. Anschließend werden die Daten des aktuellen Hosts gelöscht. Zum Abschluss wird er serialisiert, so dass er

nur als Byte-Array vorhanden ist und versendet werden kann.

Migriert der Agent nicht, werden alle Nachrichten verarbeitet und der Agent geht wieder in den Wartezustand, bis eine neue Nachricht hinzugefügt wird oder der Timeout abläuft. Falls der Nachrichtentyp dem Agenten nicht bekannt ist, ignoriert dieser die Message.

7.3 Migration Engine

Die Migration Engine ist wie der Name schon andeutet für die Migration der mobilen Agenten zuständig. Agenten werden innerhalb von gewöhnlichen Nachrichten zwischen Hosts verschickt. Das mobile Agentensystem UbiMAS verwendet die schwache Migration. Der Agent wird also am Ziel-Host nicht genau an der Code-Stelle ausgeführt, wo er unterbrochen wurde, sondern führt einen Methodenaufruf durch. Die Migration wird vom Agenten selbst initiiert. Entweder bekommt der Agent eine Anfrage von einem anderen Agenten oder Host, auf einen anderen Knoten zu migrieren, oder er entscheidet von sich aus zu migrieren.

Wenn ein Agent auf einen entfernten Host migrieren möchte, ruft er die Methode *sendAgent* auf, welche im abstrakten Agenten implementiert ist. Diese Methode generiert eine Nachricht mit dem Migrationswunsch, die an den aktuellen Host gesendet wird und stoppt daraufhin den Agententhread. Als Parameter kann der mobile Agent entweder eine Host-ID, welche dem Ziel-Host darstellt, oder eine Agenten-ID, welche einen entfernten Agenten repräsentiert, zu dem migriert werden soll, angeben.

Damit die Agenteninstanz in eine Nachricht gepackt werden kann, muss er vorher in einen Byte-Stream umgewandelt werden. Die Migration Engine verfügt hierzu über den Serializer. Von dem Zeitpunkt des Eintreffens der Migrationsnachricht bis zur Deregistrierung des Agenten kann der Agent noch Nachrichten empfangen, verarbeitet werden diese jedoch erst nach der Migration. Der Host ruft die Methode *unloadAgent()* mit dem zu sendenden Agenten als Übergabeparameter auf. Innerhalb dieser Methode werden die aktuellen Hostinformationen aus dem Agenten gelöscht, der Agent deregistriert und mit Hilfe des Serializers serialisiert. Der entstehende Agenten-Byte-Stream wird in eine spezielle Nachricht gepackt. Die in der Zwischenzeit erhaltenen Nachrichten an den Agenten werden an die erzeugte Message gehängt. Der Host füllt noch die Header der Nachricht mit den Senderinformationen, Nachrichtentyp und Sicherheitsschlüssel und sendet diese an den Ziel-Host.

Der Ziel-Host erkennt über den Nachrichtentyp, dass es sich um einen migrierenden Agenten handelt. Der Byte-Stream wird entpackt und über den Deserializer zurück in einen Agentenobjekt umgewandelt und mit Hilfe eines eigenen Clas-

sloaders geladen. Falls die richtige Klasse des Agenten nicht vorhanden ist, muss sie von dem Quell-Host angfordert werden. Dies geschieht mit einer vordefinierten Nachricht *unknownClass()*. Der Quell-Host schickt daraufhin die gewünschte Agentenklasse eingekapselt in einer neuen Nachricht an den Ziel-Host.

Ist der Agent geladen, werden die aktuellen Knoteninformationen gesetzt, der Agent registriert und abschließend gestartet. Jetzt verarbeitet der Agent alle Nachrichten, die er nach seiner Entscheidung zur Migration empfangen hat.

Der Ziel-Host bestätigt zum Abschluss der Migration den korrekten Empfang des Agenten, indem er eine Quittung sendet. Der Quell-Host quittiert seinerseits den Empfang der vorhergehenden Nachricht.

Diese Art der Migration entspricht fast einer starken Migration, obwohl der Agent nicht genau an der Stelle seines Ausführungszustandes nach dem Versenden gestartet wird. Betrachtet man den Zustand beim Versenden genau, ergibt sich dieser aus der Belegung der Variablen, dem Inhalt der Warteschlange und dem Zustand des Threads. Zum Zeitpunkt des Sendens ruht der Thread. Auch wenn technisch gesehen die Eigenschaften der starken Migration nicht gegeben sind, unterscheiden sich die Zustände vor dem Versenden und vor dem Starten auf dem Zielknoten nicht. Abbildung 7.3 veranschaulicht den Ablauf der Migration.

7.4 Die PoBox und der PoBoxAdder

Der Nachrichtenverkehr zwischen dem Host und den Agenten geht über die PoBox. Die PoBox ist nach einer typisch amerikanischen Post Office Box benannt, über die Briefe sowohl gesendet als auch empfangen werden können. Die PoBox repräsentiert somit den „Briefkasten“ des Agenten-Hosts. Alle Nachrichten von lokalen Agenten müssen in den Briefkasten eingeworfen werden. Als „Briefträger“ fungiert der PoBoxAdder. Jeder Agent bekommt, wenn er erzeugt oder nach der Migration geladen wird, seinen eigenen PoBoxAdder. So hat der Agent selber keinen direkten Zugriff auf den Agentenknoten. Außerdem fügt der PoBoxAdder auch die Absenderinformation den Nachrichten zu, so dass gefälschte Sendernamen erst gar nicht auftreten können, und passt auf, dass der Briefkasten des Hosts nicht überfüllt wird.

Die PoBox gibt die Nachrichten, die von den PoBoxAddern der lokalen Agenten, hinzugefügt werden, an die Message Delivery Engine weiter, falls der Empfänger sich nicht lokal auf dem selben Host befindet. Ist die Nachricht an den lokalen Host oder einen lokalen Agenten gerichtet, leitet die PoBox sie sofort an den Empfänger weiter. Die gesamte Kommunikation zwischen Host und Agent läuft über die PoBox und den PoBoxAdder. Außer der *addMessage*-Methode des PoBoxAdders gibt es keine Methodenreferenzen zwischen Agenten und Hosts. Abbildung 7.4

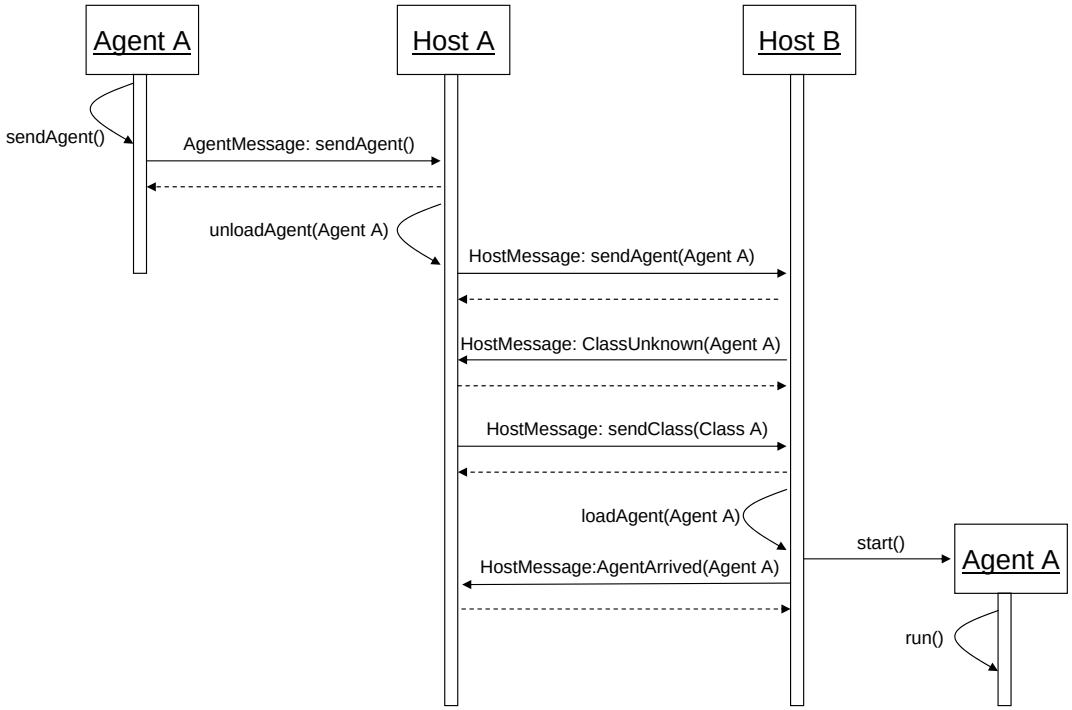

Abbildung 7.3: Ablauf der Migration

veranschaulicht den Nachrichtenaustausch zwischen Agent und PoBox.

Die PoBox installiert für jeden Agenten eine Warteschlange für eingehende Nachrichten. Auf der Gegenseite besitzt jeder Agent eine Warteschlange für Nachrichten, die von der PoBox kommen. Die Längen der Warteschlangen werden jeweils dynamisch von dem Besitzer der Warteschlangen gesetzt. Außerdem kann ein Timer gesetzt werden, der zwischen zwei Nachrichten abgewartet werden muss, so dass das Fluten mit Nachrichten unterbunden wird. Diese Methoden bieten verschiedene Sicherheitsmöglichkeiten, die in Abschnitt 7.5 näher erläutert werden.

7.5 Sicherheit in UbiMAS

Kapitel 4 fasste die Sicherheitsanforderungen von mobilen Agentensystemen sowie die Bedrohungen, die durch alle beteiligten Instanzen ausgehen, zusammen. Neben der Ubiquität legt UbiMAS einen großen Wert auf die Sicherheit des Systems, da dies ein essentieller Faktor für die Akzeptanz des Agentensystems bei

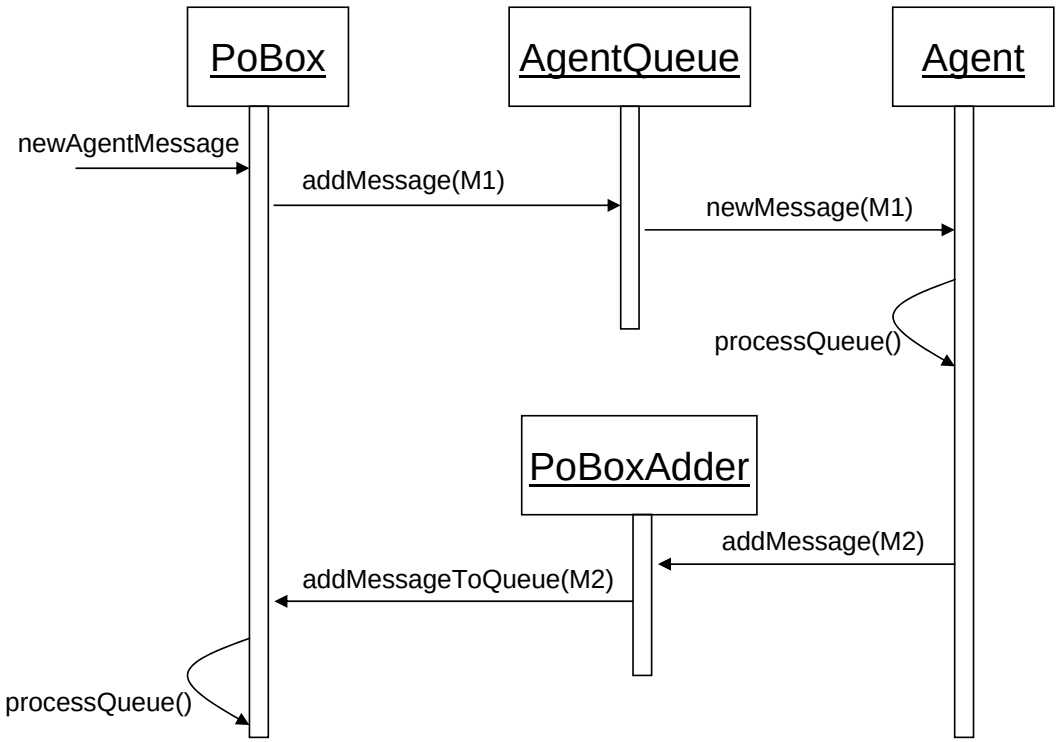

Abbildung 7.4: Nachrichtenaustausch zwischen Agent und PoBox

den Benutzern darstellt.

Wie in dem Abschnitt 6.1 erwähnt wurde, wird die knotenübergreifende Kommunikation durch die Middleware gesichert. Weitergehende Sicherheitsmechanismen müssen die Dienste jeweils selbst implementieren. UbiMAS-Hosts realisieren eine Sicherheitsarchitektur, die das Ziel bestrebt, sowohl Agenten als auch Agenten-Hosts gegen böswillige Angriffe zu schützen. Dieser Abschnitt beschäftigt sich mit den implementierten Sicherheitsmechanismen von UbiMAS, um den definierten Anforderungen zu genügen.

7.5.1 Authentisierung in UbiMAS

Die Authentisierung dient dazu, zweifelsfrei die Identität einer Instanz im System zu bestimmen. In UbiMAS kommen dabei die Instanzen Host und Agent in Frage. Sowohl die Identität der Hosts als auch der Agenten muss eindeutig festgelegt und beweisbar sein, damit Maskeraden nicht durchführbar sind. Die Authentisierung

der Hosts in UbiMAS geschieht durch die Bildung von sicheren Peergruppen.

Der Erzeuger einer UbiMAS-Peergruppe agiert gleichzeitig als Certification Authority (CA). Die CA, oft auch als Trust Center bezeichnet, erzeugt eine Datenstruktur mit der eindeutigen ID des neuen Mitglieds und mit dessen öffentlichem Schlüssel, welches auch durch die CA generiert wurde. Diese Datenstruktur wird dann mit dem privaten Schlüssel der CA digital signiert. Die CA fungiert dabei in der Rolle eines „Notars“ . Durch die digitale Signatur wird der Name untrennbar mit dem öffentlichen Schlüssel verbunden. Das Ergebnis dieses Prozesses wird als Zertifikat bezeichnet. Jedes neue Mitglied bekommt somit neben dem Schlüsselpaar ein Zertifikat, womit es sich authentifizieren kann. Ein anderes Mitglied, welches die Identität überprüfen möchte, kann das Zertifikat mit dem öffentlichen Schlüssel des Erzeuger-Hosts entschlüsseln und vergleichen, ob die ID mit der entschlüsselten ID übereinstimmt.

Die Authentisierung von Agenten läuft nach dem selben Schema ab. Wird ein Agent auf einem Host neu erzeugt, bekommt es ein Zertifikat, welches mit dem privaten Schlüssel des aktuellen Hosts signiert ist. Damit kann jede andere Instanz im System die Identität des Agenten über den erzeugenden Host überprüfen.

7.5.2 Schutz vor bösartigen Hosts

Der Schutz des Agenten vor bösartigen Hosts wird auch als Ausführungsintegrität bezeichnet, da dabei die Integrität des Mobilen Agenten während seiner Ausführung auf einem Agentensystem betrachtet wird. Bei der Ausführungsintegrität sind zwei Fälle zu unterscheiden:

- Die Integrität bezüglich Veränderungen durch den Gast-Host. Dabei stellt sich die Frage, wie eine Manipulation am mobilen Agenten erkannt oder verhindert werden kann.
- Die Integrität bezüglich der Ausführung selbst. Hierbei handelt es sich um die Frage, ob der Agent überhaupt ausgeführt wurde oder ausgeführt, aber vom Agenten-Host bewusst vorzeitig beendet wurde. Diese Art von Angriff wird auch *Denial of execution* genannt.

Zu diesen Punkten gibt es eine Vielzahl von Untersuchungen, die aber alle nur für, zum Teil exotische, Spezialfälle eine Lösung versprechen. Im Allgemeinen scheinen diese Probleme nicht lösbar zu sein, da der Agenten-Host die vollständige Kontrolle über den mobilen Agenten besitzt. Veränderungen an dem Agenten oder der Ausführung können technisch nicht wirksam verhindert werden.

Eine Lösung des Integritätsproblems kann nur auf organisatorischem Wege gefunden werden. Zum einen kann in einem ubiquitären System, im Gegensatz

zu einem offenen Szenario, auf die räumlich begrenzte Struktur zurückgegriffen werden. Des weiteren beziehen sich ubiquitäre Dienste nur auf einen engen Personenkreis. Die Verletzung der Integrität eines mobilen Agenten im ubiquitären System muss für einen Beteiligten so „teuer“ sein, dass sich die Manipulation für den Angreifer nicht „rechnet“. Das heißt, die beteiligten Personen müssen ein Sanktionssystem festlegen, das die Verletzung der Integrität unter Strafe stellt.

Aus diesem Grund nimmt UbiMAS an, dass die Agenten-Hosts vertrauensvoll sind. Die Hosts werden durch einen Administrator installiert und gestartet. Der Administrator und der Author der Agenten-Hosts werden als bekannt vorausgesetzt. Es ist daher anzunehmen, dass diese Personen im eigenen Interesse vertrauensvoll agieren werden. In diesem Sinne setzt UbiMAS beim Schutz vor bösartigen Hosts auf soziale Schutzmechanismen.

Die Infiltration von fremden Hosts ist nicht möglich, da für jeden Dienst Zertifikate und Schlüssel vorgeschrieben sind. Jede Peergruppe besitzt ihr eigenes Zertifikat, dessen Gültigkeit bei jeder Kommunikation überprüft wird. Möchte ein fremder Peer Mitglied einer Peergruppe werden, muss er eine Anfrage für einen Beitritt stellen. Die Mitglieder entscheiden dann über die Aufnahme des Peers. Zudem werden alle Nachrichten unter Nutzung von Public Key Infrastructure Mechanismen verschlüsselt. Der Schlüsselaustausch wird bei UbiMAS nur unter den Mitgliedern erlaubt und wird über sichere Kommunikationskanäle durchgeführt.

7.5.3 Schutz vor bösartigen Agenten

7.5.3.1 Schutz der Agentendaten

Der Schutz der Agentendaten bezieht sich darauf, dass mobile Agenten ihre sensiblen Daten vor anderen Agenten geheim halten können. Der Agenten-Host stellt eine allgemeine Laufzeitumgebung für mobile Agenten dar. Diese könnten versuchen, den Agenten-Host zu missbrauchen, um an Daten anderer Agenten zu gelangen oder andere Agenten auf dem Host zu manipulieren. Daher muss das Agentensystem auch Eigenschaften aufweisen, wie sie normalerweise zur Trennung von Prozessen auf Betriebssystemen verwendet werden, wie:

- Trennung der Speicherbereiche verschiedener Agenten (Speicherschutz)
- Trennung der Laufzeitumgebung der Agenten (Laufzeitschutz)
- Trennung der Namensräume

Das Agentensystem muss sicherstellen, dass die Speicherbereiche, die von einem Agenten genutzt werden, von keinem anderen Agenten gelesen oder beschrieben

werden können. Andernfalls könnte ein böswilliger mobiler Agent Daten von anderen mobilen Agenten auslesen oder, falls er schreibend auf den Speicher zugreifen könnte, sogar den mobilen Agenten verändern.

Werden Programmiersprachen verwendet, in denen es keine Pointer und keine Methoden zur direkten Speichermanipulation gibt, kann kein mobiler Agent direkt auf den Speicher und damit auf Speicherbereiche anderer mobiler Agenten zugreifen. Java ist eine Programmiersprache, die diesen Ansatz konsequent verfolgt [LY99]. Alle Systeme mobiler Agenten, die in Java implementiert sind, können sich bezüglich Speicherschutz auf die virtuelle Maschine abstützen. Da UbiMAS auch in der Programmiersprache Java geschrieben ist, besteht von dieser Seite ausreichender Schutz.

Neben den Methoden, die ein mobiler Agent selbst anbietet, existieren auch Methoden des Agenten-Host (z.B. suspend, stop) oder des Betriebssystems (z.B. setPriority), um den Status eines mobilen Agenten zu ändern. Laufzeitschutz ist dann gegeben, wenn ein mobiler Agent nicht durch Missbrauch des Agenten-Hosts oder des Betriebssystems andere mobile Agenten beeinflussen kann. Das Agentensystem muss deshalb den Zugriff auf eigene Schnittstellen, mit denen mobile Agenten manipuliert werden können, schützen. Bei Agentensystemen, die in Java implementiert sind, werden Mobile Agenten häufig als Threads realisiert.

Ein Thread ist ein eigener Kontrollfluss innerhalb eines größeren Anwendungskontextes (z.B. Prozess im Betriebssystem, Applet oder Application in Java). Threads besitzen einen Vater und nutzen gemeinsam dieselben Ressourcen. Standardmäßig können sich Threads auch gegenseitig beeinflussen (z.B. stoppen, unterbrechen, Prioritäten ändern, usw.). Werden mobile Agenten als Threads implementiert, muss deshalb auch in diesem Bereich ein Laufzeitschutz realisiert werden. In Java ist dies möglich, indem man mit Hilfe der Java Sicherheitsarchitektur (Policy Manager und Access Controller) den Zugriff auf Threads und Gruppen von Threads beschränkt. In UbiMAS bildet jeder mobile Agent mit allen seinen Sub-Threads eine eigene Thread Group. Dies hat den Vorteil, dass der Agent sehr einfach vor dem Zugriff von Threads aus anderen Thread Groups geschützt wird. Ein Low-Level Aufruf über Methoden zur Thread Manipulation kann damit verhindert werden.

Neben der Sicherheiten, die Java vom Grunde auf bietet, schützt sich UbiMAS vor der Gefahr externer unauthorisierter Methodenaufrufe von Seiten böswilliger Agenten, indem es Referenzen vermeidet. Dies bedeutet, dass kein mobiler Agent direkte Methodenreferenzen auf Hosts oder andere Agenten kennt. Der Agent kann also keine fremden Funktionen oder externe Daten aufrufen, außer die, die standardmäßig für den Nachrichtenaustausch definiert sind. Die einzige Schnittstelle zur Kommunikation stellt der PoBoxAdder dar, der wiederum geeignete Schutzsmechanismen gegen andere Arten von Angriffen beinhaltet, die

unten näher erläutert werden.

Eine Trennung der Namensräume ist insbesondere für Systeme, die eine dynamische Bindung von Laufzeitbibliotheken oder dynamisches Nachladen von Klassen ermöglichen, unabdingbar. Um Namenskonflikte zwischen verschiedenen Agenteninstanzen sowie zwischen Agenteninstanz und Agentensystem, die beim Instantiieren von Objekten und beim Aufruf von Methoden auftreten können, zu verhindern, ist eine Trennung der Namensräume erforderlich. Dies lässt sich in Java (ab Java 2 SDK) durch den Classloader realisieren. Der Classloader ist für das Laden und Instantiieren von Java Byte Code zuständig, von der Classloader Klasse können eigene Classloader Objekte abgeleitet und instantiiert werden. Alle Klassen, die von einem bestimmten Classloader Objekt geladen wurden, bilden einen Namensraum. Um Namenskonflikte, zu verhindern, wird für jeden mobilen Agent ein eigener Classloader verwendet. Zur Laufzeit werden Objekte nicht nur durch ihren Namen, sondern durch ihren vollständigen Package Name und den Classloader, der die entsprechende Klasse geladen hat, identifiziert. Der Classloader wird daher auch als Defining Classloader bezeichnet.

UbiMAS trennt zwischen den Standardmethoden, die im abstrakten Agenten implementiert sind und den zusätzlichen Klassen, die ein mobiler Agent in seiner Erweiterung definieren kann. Um die Standardmethoden nicht für jeden Agenten auf dem Host redundant zu laden, gibt es einen Standard-Classloader, der beim Laden des ersten Agenten die Methoden lädt. Alle übrigen Methoden werden im agenten-spezifischen Classloader geladen.

7.5.3.2 Schutz vor Maskeraden und DoS-Attacken

Die Gefahr bei Maskerade besteht darin, dass ein böswilliger Agent vortäuscht, ein anderer Agent zu sein, um unautorisiert Informationen zu erlangen. UbiMAS bietet als Lösung gegen die Vortäuschung der Identität den PoBoxAdder, welches mehrere Schutzmechanismen besitzt. Der PoBoxAdder trägt zu jeder Nachricht, die der Agent sendet, die ID des Senders ein. Auch wenn ein Agent eine falsche ID angibt, wird diese mit der richtigen überschrieben. So hat kein Agent die Möglichkeit sich als ein Anderer auszugeben.

Die meisten Angriffe auf Kommunikationssysteme sind Denial-of-Service Attacken. Eine bösartige Instanz versucht durch das Senden von ungemein vielen Nachrichten das System zu überlasten und zum Absturz zu bringen. Um sich vor DoS-Attacken zu schützen besitzt die PoBox eine beschränkte Warteschlange für eingehende Nachrichten.

Der Agenten-Host kann die Länge jeder Warteschlange dynamisch ändern. Agenten können nur solange Messages senden, wie noch Platz in der Warteschlange ist. Darüber hinaus kann der Host einen Timer für jeden Agenten setzen. Die Agen-

ten müssen dann nach jeder Message warten bis der Timer abgelaufen ist, bevor die nächste Nachricht gesendet werden kann. Die Nachrichten innerhalb der verschiedenen Warteschlangen werden nach einem fairen Scheduling-Verfahren, wie z. B. FIFO, abgearbeitet.

Diese Vorkehrungen machen Denial-of-service-Attacken durch Agenten gegen die Hosts schwieriger. Fällt ein Agent durch erhöhtes Senden von Nachrichten auf, kann der Host die Länge der Warteschlange auf Null setzen, was die Blockierung des Agenten bewirkt.

7.5.3.3 Schutz der Kommunikation und Migration

Ein bösartiger Agent könnte das System gefährden, indem es Nachrichten erspäht, die von anderen Agenten an einen Host gesendet wurden. Als Schutz vor Lauschangriffen kann jeder Agent den öffentlichen Schlüssel des Hosts, an den er die Nachricht sendet, auch direkt anfordern. Alle Nachrichten, die der Agent über den PoBoxAdder sendet, können dann mit diesem Schlüssel verschlüsselt werden. Auf dieselbe Art verschlüsselt der Host die Nachrichten mit dem öffentlichen Schlüssel des jeweiligen Agenten. Die Nachrichten lassen sich nur mit dem entsprechenden privaten Schlüssel des Empfängers entschlüsseln. Daher hat ein böswilliger Agent, der die Nachricht abfängt, nicht die Möglichkeit an die Informationen innerhalb der Message zu gelangen. Nachrichten, die lokal von einem Agenten an den anderen geschickt werden, verlassen die PoBox erst gar nicht, so dass ein Ausspähen hier gar nicht möglich ist. Da die Middleware die Nachrichten auch verschlüsselt, ist ein Abfangen der Nachricht auf dem Zwischenweg genauso nutzlos.

Neben den Hosts haben Agenten ihre eigenen privaten und öffentlichen Schlüssel. Um den Austausch von Nachrichten zu sichern, können Agenten die Nachrichten mit den öffentlichen Schlüsseln des Empfängeragenten oder Hosts verschlüsseln, der die Message bei Empfang mit seinem privaten Schlüssel entschlüsselt. Ferner speichern Agenten persönliche Daten des Benutzers immer in verschlüsselter Form. Es gibt keine Möglichkeit von Außen auf diese Daten heranzukommen, da der einzige Zugriff über Nachrichten realisiert ist.

Der Migration Manager ist die erste und die letzte Komponente, die den Lebenszyklus eines mobilen Agenten auf einem Agentensystem beeinflusst. Aus diesem Grund implementiert er auch einige grundlegende Sicherheitsmechanismen. Durch die sichere Authentifizierung der UbiMAS-Hosts ist garantiert, dass die Identität des Ziel-Hosts, auf den der Agent migrieren möchte, bestimmt ist. Zudem verhindert das Übertragungsprotokoll Umleitungen und das Duplizieren des mobilen Agenten. Grundsätzlich verbietet UbiMAS das Klonen von Agenten. Alle Agenten werden nur einmal geladen. Während des Migrationsprozesses achten

die Hosts darauf, dass die Agenten nach quittiertem Transfer vom vorhergehenden Host gelöscht werden. Es können zwar weiterhin Agenten vorhanden sein, die dieselbe Klasse besitzen und damit die selbe Funktionalität liefern, jedoch wird durch die Vergabe von eindeutigen IDs und unterschiedlichen Zertifikaten eine andere Identität festgelegt.

7.5.3.4 Schutz vor unautorisiertem Zugriff auf Ressourcen

In einigen Anwendungen können Agenten Zugriff auf externe Ressourcen gewährt werden. Externe Ressourcen können dabei persönliche Dokumente sein oder auch ein persönlicher Kontext, der in Form einer XML Datei in dem Context Database gespeichert wurde. Dieser Zugriff sollte gesichert werden. Diese Sicherung wird in erster Linie durch den Java Security Manager realisiert.

Für Zugriffe auf eine Systemressource benötigt die Anwendung eine *Permission*. In Java sind die Permissions in verschiedene Kategorien aufgegliedert:

- BasicPermission
- FilePermission
- PrivateCredentialPermission
- ServicePermission
- SocketPermission
- UnresolvedPermission

Diese Permissions werden intern wieder weiter aufgegliedert. Jeder Permission sind wiederum verschiedene *Actions* zugeordnet. Der FilePermission sind beispielsweise die Actions lesen, schreiben, löschen und ausführen zugeordnet. Führt eine Anwendung eine Aktion aus, die durch eine Permission geschützt ist, ruft Java die Methode *checkPermission* des Security Managers auf. Hat die Anwendung nicht die erforderliche Permission wird eine SecurityException geworfen. Benötigt ein Agent für die Ausführung seines Dienstes Systemressourcen, muss er sich vorher die Elaubnis beim Security Manager holen.

Hierbei tritt das Problem auf, dass der Security Manager das System nur gegen ganze Anwendungsprozesse schützt. Mobile Agenten sind jedoch keine Prozesse sondern sind als einzelne Threads implementiert. Wenn der Security Manager einem Agenten Zugriff gewährt, würden alle Agenten auf demselben Host den selben Zugriff bekommen. Dies ist sicherlich nicht akzeptabel. Was hier benötigt wird, ist eine Zugriffsvergabe für einzelne Threads.

Das Problem bei Threads ist wiederum, dass die Thread-ID nach der Migration wechselt, da jeder Host einen neuen Thread für den empfangenen Agenten startet. Um trotzdem zu garantieren, dass ein bestimmter Agent Zugriff auf eine gewählte Ressource bekommt, wird ein Zugriffsverfahren basierend auf einer eindeutigen ID unabhängig von der Thread-ID benötigt.

UbiMAS realisiert eine Erweiterung des Java Security Managers, so dass die Rechtevergabe für Systemressourcen auch auf Agententhreads angewendet werden kann. Der Security Manager benötigt mehrere Attribute für die Rechtezuordnung, wie Benutzername, Passwort, Bezeichnung der Erlaubnis (*permission name*, weitere Erlaubnisparameter und eine Erlaubnis-ID. Es muss sichergestellt sein, dass die Erlaubnis-ID eindeutig ist und sicher dem Agenten zuzuordnen ist, dem Zugriff gewährt werden soll. UbiMAS benutzt eine aus der Agenten-ID generierte Erlaubnis-ID. Es handelt sich dabei um einen Hashwert, berechnet über eine Hash-Funktion, die nur den Zugriff gewährenden Parteien bekannt ist.

8 Ubiquitäre Anwendungen auf UbiMAS und Evaluierung

In diesem Kapitel sollen ubiquitäre Anwendungen vorgestellt werden, die auf Basis des in den vorhergehenden Kapiteln vorgestellte ubiquitäre Agentensystem UbiMAS entwickelt wurden. Die Anwendungen sollen die Einsetzbarkeit von UbiMAS in ubiquitären Umgebungen evaluieren. Als reales Testumfeld wurde ein mit intelligenten Geräten angereichertes Bürogebäude gewählt. Es handelt sich dabei um die Einrichtungen des Lehrstuhls für Systemnahe Informatik und Kommunikationssysteme an der Universität Augsburg. Die Räumlichkeiten des Lehrstuhls wurden mit dem Ziel umgebaut, ein modernes flexibles Bürosystem zu schaffen. Eine innovative Idee bildeten elektronische Türschilder, die so genannten *Smart Doorplates* [TBPU03a] [TBPU03b]. Starre Türschilder wurden durch kleine LCD-Bildschirme ausgetauscht, die über eine berührungssensible Bildschirmoberfläche verfügen, so dass auch Eingaben von Seiten der Benutzer durch einfaches Tippen mit dem Finger getätigt werden können. In den folgenden Abschnitten sollen mögliche Szenarien vorgestellt werden, die mit Hilfe von Smart Doorplates in einer ubiquitären Umgebung realisiert werden können.

8.1 Vision des Smart Doorplate Projekts

Gewöhnliche Türschilder geben Informationen über den Raum wieder, der sich hinter der Tür verbirgt. In einem Bürogebäude sind es meist neben der Raumnummer die Namen der Bürobewohner und ihre Funktion in ihrem Tätigkeitsumfeld. Die Türschilder bleiben so lange an ihrem Ort hängen bis ein Mitarbeiter das Büro wechselt oder neue Mitarbeiter die Arbeit aufnehmen. Offensichtlich sind es seltene Ereignisse, die in einem weiten Zeitraum stattfinden. Die Idee des Smart Doorplate Projektes ist, dass Türschilder in Form von elektronischen Displays viel mehr Information über den Raum und die Bürobewohner liefern könnte und sich dem aktuellen Zustand anpassen könnte. Um ein solches Büroumfeld anschaulich verbildlichen zu können, sollen in diesem Abschnitt einige Szenarien vorgestellt werden, bei denen die Smart Doorplates einen echten Mehrwert im Büroalltag bieten würden. Für diese Szenarien soll zunächst vorausgesetzt werden, dass Personen im Gebäude identifiziert und vom System verfolgt werden können, so dass

der Aufenthaltsort jeder Person bekannt ist.

8.1.1 Smart Doorplates als Wegweiser

In diesem Szenario werden die elektronischen Türschilder als Wegweiser eingesetzt. Man stelle sich ein mehrstöckiges Bürogebäude vor mit hunderten von Räumen und Mitarbeitern. Für einen Besucher, dem das Gebäude unbekannt ist, stellt sich die Suche nach einem bestimmten Büro als höchst schwierig dar. Der Besucher irrt wie in einem Labyrinth und muss sich womöglich an jeder Abzweigung Hilfe einholen.

Diese Hilfe können Smart Doorplates dem Besucher an jeder Tür bieten. Bekanntlich hat jede Tür in so einem Bürogebäude ein Türschild. Würden diese durch Smart Doorplates ersetzt, könnten sie Richtungspfeile an jeder Tür, an dem der Besucher vorbeiläuft, anzeigen und ihn somit bis zum Zielbüro des gesuchten Mitarbeiters führen.

Wie im vorigen Abschnitt erwähnt, müssen die Personen identifiziert werden können. Daher muss sich der Besucher, der diesen Dienst in Anspruch nehmen möchte, im Eingangsbereich des Gebäudes zunächst vorstellen. Dies kann auch auf elektronischem Weg geschehen, indem die persönlichen Daten des Besuchers an einem Terminal selbständig eingegeben werden. Das Terminal würde dem Besucher daraufhin einen Besucherausweis ausstellen, der mit einem elektronischen Chip ausgestattet ist. Über diesen Ausweis kann das Location-Tracking-System den Besucher überall im Gebäude identifizieren und lokalisieren. Als nächsten Schritt nennt der Besucher den Namen des gesuchten Bürobewohners. Das Terminal kann ihn auch dabei unterstützen, indem er eine Suchfunktion über die Namen aller Bürobewohner anbietet.

Ist das Ziel eingegeben, kann das Smart Doorplate System den optimalen Pfad berechnen und dem Besucher eine erste Richtung am Terminal anzeigen. Sobald der Besucher in Sichtweite eines Türschildes kommt, präsentieren die Smart Doorplates ihm die Richtung, in die er weiterlaufen muss.

Wenn man annimmt, dass nur ein einzelner Besucher an den Türen vorbei läuft, ist ein Richtungspfeil allein ausreichend und angemessen. Sollten jedoch mehrere Besucher unterwegs sein und sich an einem Türschild kreuzen, müsste eine Synchronisation der konkurrierenden Informationen erfolgen, z.B. in Form einer kombinierten Anzeige mehrerer Pfeile in jeweils verschiedener Form und Farbe für jeden Besucher.

Auch wenn der Mitarbeiter sich nicht in seinem Büro aufhält, sondern in einem anderen Raum im Gebäude, können die Türschilder weiterhelfen. Da die Mitarbeiter auch vom Location-Tracking-System lokalisiert werden können, kann der

Besucher auf Wunsch zu dem aktuellen Raum geführt werden. Diese Funktion ist nicht nur für Besucher interessant. Auch die anderen Mitarbeiter würden immer den aktuellen Aufenthaltsort des jeweiligen Bürobewohners erfahren. Die Freischaltung dieser Information sollte in Händen des jeweiligen Mitarbeiters sein, so dass der Aufenthaltsort nur mit seinem Wunsch angezeigt werden würde.

8.1.2 Smart Doorplates als Raum-Zustandsanzeige

Dieses Szenario setzt voraus, dass der Besucher schon vor dem Zielbüro steht und der Mitarbeiter anwesend ist. In dieser Situation kann das Smart Doorplate wie eine halb-transparente Tür den aktuellen Zustand des Raumes nach außen durchschimmern lassen. Im Falle der Anwesenheit des Bürobewohners ergäben sich verschiedene Situationsmöglichkeiten:

- Der Bürobewohner ist am Telefon und möchte nicht gestört werden. Sicherlich war schon jeder von uns in so einer Situation. Man klopft an und riskiert einen Blick ins Büro. Der Bürobewohner am Telefon winkt ab und man schließt die Tür wieder. Da man nicht weiß, ob und wann das Telefongespräch beendet wird, findet man sich in einer unsicheren Situation. Bei öffentlichen Ämtern mit viel Besucherverkehr kommt es zudem oft vor, dass die Bürobewohner nach dem Telefonat den Besucher nicht hineinbitten, sondern warten bis dieser nochmal anklopft. In so einer Situation bietet das Smart Doorplate einen beruhigenden Ausweg. Das Türschild signalisiert das Telefonat, indem es ein Telefonsymbol anzeigt und den Besucher auffordert, nicht einzutreten. Wenn das Telefongespräch beendet ist, benachrichtigt das Türschild den Besucher und bittet ihn herein. So wird weder der Mitarbeiter während seinem Gespräch gestört noch fühlt sich der Besucher unsicher.

- Wenn das Büro von mehr als nur einem Mitarbeiter bewohnt wird, kann es vorkommen, dass der eine Mitarbeiter zwar beschäftigt ist, aber der andere den Besucher annehmen kann. Das Smart Doorplate zeigt in so einem Fall die Information getrennt für jeden Mitarbeiter an. Die jeweilige Situation wird hinter dem Namen des Mitarbeiters signalisiert. So kann der Besucher sehen, ob der Mitarbeiter, zu dem er möchte, gerade frei ist oder nicht.

- Der Bürobewohner ist in diesem Szenario in einem Meeting innerhalb seines Büros. Das Smart Doorplate zeigt dem Besucher an, dass der Mitarbeiter zwar anwesend ist, aber in einem Meeting ist und nicht gestört werden möchte. Der Bürobewohner könnte auch benachrichtigt werden, dass ein Besucher draußen wartet, indem am PC oder PDA des Mitarbeiters eine Nachricht mit dem Namen des Besuchers erscheint. Der Mitarbeiter kann daraufhin eine Nachricht an dem Türschild anzeigen lassen, z.B. mit der

Dauer des Meetings. So ist eine Kommunikation möglich ohne das laufende Meeting zu stören.

- Eine ähnliche Situation tritt auf, wenn der Raum, vor dem der Besucher steht, ein Besprechungszimmer ist. Hier kann das Türschild zusätzlich zu der Information, dass der Raum gerade benutzt wird, die Namen der Teilnehmer und die Dauer der Besprechung anzeigen. Diese Informationen könnten automatisch vom System gesammelt werden, wenn ein geeigneter elektronischer Terminkalender vorhanden ist.

8.1.3 Smart Doorplates als elektronisches Sekretariat

In dem Fall, dass der Bürobewohner beim Eintreffen des Besuchers sich nicht in seinem Büro befindet, kann das Smart Doorplate als elektronisches Sekretariat dienen. Das Türschild zeigt die Abwesenheit des Mitarbeiters an und gibt falls möglich den aktuellen Aufenthaltsort bekannt, wie im Gebäude, außerhalb des Gebäudes oder noch nähere Details, und vielleicht den Zeitpunkt, wann der Mitarbeiter zurück erwartet wird. Hat der Bürobewohner angegeben, dass Besucher zu einem anderen Büro weitergeleitet werden sollen, wird der Besucher mittels Pfeilen automatisch weiter geführt.

Alternativ hierzu könnte das System auch den Zeitpunkt der Rückkehr vorhersagen und den Besucher nahelegen zu warten. Möchte der Besucher eine Nachricht hinterlassen, kann das Smart Doorplate eine schriftliche oder gar gesprochene Nachricht aufnehmen. Bei Rückkehr des Mitarbeiters kann die Nachricht am Türschild oder an seinem PC angezeigt oder abgespielt werden (siehe nächsten Abschnitt). Dringende Nachrichten könnten sogar an den aktuellen Aufenthaltsort des Bürobewohners weitergeleitet werden, z.B. über das lokale Netz oder in Form einer SMS.

8.1.4 Smart Doorplates als Informationsanzeige

Wenn der Bürobewohner morgens oder nach einem Meeting in seinem Büro ankommt, wird er vom Smart Doorplate erkannt und es werden die Anzahl der Besucher, die hinterlassenen Nachrichten, verpasste Telefonanrufe und erhaltene E-Mails in übersichtlicher Form angezeigt. Wichtige Besuche, Anrufe und E-Mails werden dabei hervorgehoben. Die Anzeige kann beliebig nach persönlichen Wünschen und Interessen des Mitarbeiters konfiguriert werden.

Falls die Informationen persönliche Daten enthalten, werden diese nur angezeigt, wenn sich sonst keine Personen im Sichtbereich aufhalten. Die Smart Doorplates

geben dabei zunächst nur eine kurze Zusammenfassung der Informationen an. Die gesamte Nachricht kann der Mitarbeiter an seinem PC anschauen oder anhören.

8.2 Prototypische Implementierung der Smart Doorplates

Um die im vorigen Abschnitt beschriebenen Szenarien in einem realen Büroumfeld zu implementieren, wurden in dem Lehrstuhl für Systemnahe Informatik und Kommunikationssysteme an der Universität Augsburg zwei Prototypen der Smart Doorplates entwickelt, die hier näher erläutert werden sollen. Die Smart Doorplates dienen als Grundlage für verschiedene Forschungsfelder. So wird neben den ubiquitären mobilen Agenten auch im Bereich der Kontextvorhersage [PBTU03] [PBT+05] und Middleware für Organic und Autonomic Computing Systeme [TPBU04a] [TPBU04b] am Lehrstuhl intensiv geforscht.

Abbildung 8.1: Smart Doorplate mit RFID-Lesegerät

8.2.1 Erster Prototyp mit PDAs

In dem ersten Prototyp wurden PDAs als Smart Doorplates eingesetzt. Diese Geräte vom Typ Hewlett Packard iPAQ h5450 sind mit Bluetooth und WLAN ausgestattet, so dass eine kabellose Kommunikation gewährleistet ist. Zudem verfügt die Anzeige über eine berührungssensible Oberfläche (Touchscreen). Außerdem sind ein Mikrophon und ein eingebauter Lautsprecher vorhanden, so dass Spracheingaben und -ausgaben möglich sind.

Als Location-Tracking-System können verschiedene Ansätze gewählt werden. [HB01] gibt ein Überblick über Lokationssysteme für Ubiquitous Computing. Für den ersten Prototyp wurde die Transponder-Technologie gewählt, um Personen innerhalb des Gebäudes zu lokalisieren. Diese Technologie besteht aus Transpondern (RFID-Tags), die von den Personen getragen werden und entsprechenden RFID-Antennen, die diese Transponder auslesen können. Der Leseradius beträgt dabei bis zu fünf Metern.

Es gibt zwei mögliche Arten von RFID-Tags, aktive und passive. Aktive Transponder verfügen über eine Stromquelle in Form einer Batterie und können selbständig funken. Die passiven Transponder beziehen ihre Energie zum Funken aus einem Energieimpuls der Antenne. Daher ist die Reichweite der passiven RFID-Tags wesentlich geringer als die aktiven. Sie beträgt lediglich wenige Zentimeter. Dafür sind die Lesegeräte für passive Transponder kostengünstiger.

Wegen der höheren Reichweite kamen im Smart Doorplate System die aktiven Geräte zum Einsatz. Jede Tür musste mit einer Antenne ausgestattet werden und jede Person bekam einen RFID-Tag zur Identifikation. Abbildung 8.1 zeigt einen Versuchsaufbau mit dem Smart Doorplate oben und der RFID Antenne in der Mitte angebracht an einer Trägerwand.

Dieser erste Aufbau war zunächst nicht fest an der Wand installiert, um unter Laborbedingungen testen zu können. Zudem waren die Geräte in dieser Anordnung mobil und konnten leicht zu Vorführungszwecken transportiert werden. Abbildung 8.2 zeigt einen Benutzer, wie er mit dem RFID-Tag an dem Smart Doorplate vorbei läuft.

Für den ersten Prototyp wurden verschiedene graphische Oberflächen programmiert. Als Anwendung wurde das erste Szenario implementiert, in dem die Smart Doorplates als Wegweiser dienen sollten. Abbildung 8.3 stellt im linken Bild den PDA mit dem Hauptbildschirm dar.

Wie zu erkennen ist, zeigt der Hauptbildschirm die Informationen über den Raum und seine Benutzer wie sie auch auf einem gewöhnlichen Türschild zu sehen sind. In dem mittleren Bild wurde auf die Raumnummer getippt, so dass der Bildschirm alle anwesenden Personen im Büro auflistet. Das rechte Bild ist ein Screenshot der

Abbildung 8.2: Benutzer mit RFID-Tag

Wegweisungsanwendung, wo der Besucher aufgefordert wird, dem Richtungspfeil zu folgen.

Als Middleware wurde für diesen Prototyp eine erste Implementierung der ubiquitären Middleware und UbiMAS eingesetzt wie sie im Kapitel 6 beschrieben wurde. Das Location-Tracking-System bildete eine eigenständige Peergruppe. Jede Antenne realisierte eine Peer in dieser Gruppe und sendete eine Nachricht mit der ID des RFID-Tags, wenn eine Person im Empfangsbereich erkannt wurde. Die UbiMAS Peergruppe war Partner der Lokationsgruppe und bekam auch diese Nachrichten. Die eigentlichen Anwendungen basierend auf UbiMAS Agenten werden im Abschnitt 8.3 näher beschrieben.

8.2.2 Prototyp mit fest eingebauten LCD-Bildschirmen

Der erste Prototyp mit PDAs als Türschilder brachte wichtige Erfahrungen. Zunächst fiel auf, dass die Bildschirme der PDAs zu klein waren, um komfortable Benutzerführung realisieren zu können. Das Ziel des Projektes war auch, eine Festinstallation der Türschilder vorzunehmen, wofür die PDAs eine schlechte Alternative boten. Auch von Seiten der Rechenleistung gab es negative Erfahrungen. Die Kommunikation über Funk war meist entweder zu langsam oder die Verbindung brach ab. Im Abschnitt 8.17 werden einige Messergebnisse vorgestellt, die

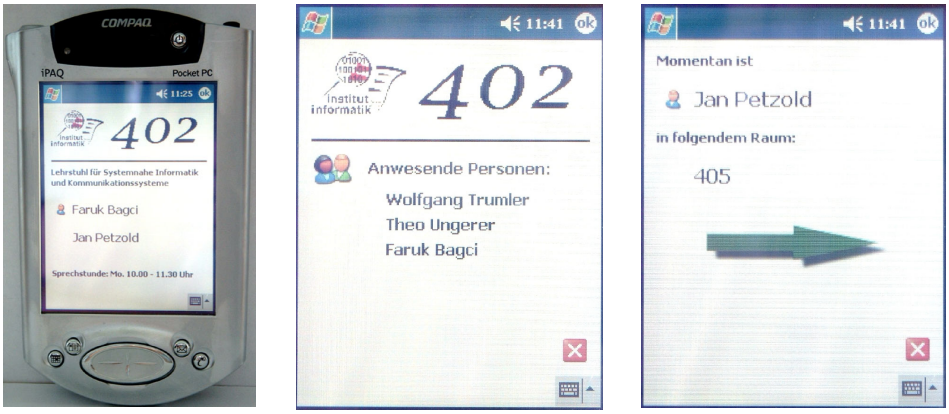

Abbildung 8.3: Smart Doorplate und verschiedene graphische Oberflächen

diese Nachteile bestätigen. Diese Erfahrungen resultierten in der Entscheidung PCs einzusetzen, an die LCD Bildschirme mit Touchscreen angeschlossen waren. Abbildung 8.4 zeigt ein solches Türschild eingebaut in der Wand vor einem Büro.

Die LCD Bildschirme sind genügend gross, um komfortabel alle Informationen anzuzeigen, die dann auch von größerer Entfernung erkennbar sind. Die Bildschirme sind durch die Wand mit einem leistungsstarken PC verbunden, der sich innerhalb des Büros befindet und an das lokale Netz angeschlossen ist. Wie man an Abbildung 8.5 erkennen kann, wurde jede Tür in dem Flur des Lehrstuhls mit Smart Doorplates ausgestattet.

Das Location-Tracking-System basierend auf Transponder Technologie ist zu diesem Zeitpunkt noch sehr teuer und noch nicht ausgereift. Eine Installation an jeder Tür des Lehrstuhls würde die Kosten des Projektes sprengen. Daher wurden mit dem neuen Prototypen alternative Techniken erforscht, die auf Infrarot und Funk als Kommunikationsmedien aufbauen. Zum Einsatz kamen Sensorboards, die sowohl einen Funkchip als auch Infrarot-Sender und -Empfänger besitzen.

Die Middleware in der aktuellen Version ist die komplette Implementierung der UbiMAS Middleware wie sie in Kapitel 6 vorgestellt wurde. Auch die aktuellste Realisierung der UbiMAS-Hosts kam auf dem neuen Prototyp zum Einsatz. Der nächste Abschnitt beschreibt Anwendungen, die mit reflektiven mobilen UbiMAS Agenten auf den beiden Prototypen implementiert wurden.

Abbildung 8.4: Smart Doorplate

8.3 Reflektive Agenten auf Smart Doorplates

Die Smart Doorplates sollten als Anwendungs- und Evaluationsbasis für die Idee der reflektiven mobilen Agenten dienen. Die Kombination von mehreren ubiquitären Geräten in einem Büroumfeld bot eine exzellente Möglichkeit, verschiedene ubiquitäre Anwendungen zu realisieren. Auch die Existenz von vielen verschiedenen Personen in dem Bürogebäude eignete sich zu der Idee der reflektiven Agenten, die ja jeweils eine Person repräsentieren sollten.

Abbildung 8.6 stellt die Architektur der Smart Doorplate Anwendung auf Basis von UbiMAS graphisch dar. Auf jedem Türschildrechner wurde die ubiquitäre Middleware installiert und als Dienst ein UbiMAS-Host gestartet. Die UbiMAS-Hosts wurden so konfiguriert, dass sie beim Start selbständig eine Peergruppe aufbauten. Zusätzlich zu dieser Peergruppe existierte eine Lokationsgruppe, die Nachrichten über den Aufenthaltsort der Personen im Gebäude versendete. Neben der RFID-Technologie wurden Lokationssysteme auf Basis von Funk und Infrarot eingesetzt. Die UbiMAS-Hosts konnten die Lokationsnachrichten als Partner auch empfangen. Über diese Schnittstelle hatten somit auch Agenten die Möglichkeit sich für die Lokationsnachrichten zu registrieren.

Für beide Smart Doorplate Prototypen wurden Benutzer-Agenten implementiert, die Informationen über die Person speicherten und jeweils auf den der Person am

Abbildung 8.5: Flur mit Smart Doorplates

nächsten Türschild migrierten. Die Smart Doorplates dienten zudem als Schnittstelle zum persönlichen Agenten, so dass der Benutzer seinen reflektiven Agenten an jedem Türschild beauftragen konnte, einen bestimmten Dienst für ihn auszuführen.

In der ersten Version des Smart Doorplates wurde ein Agent implementiert, der sich für die Lokationsnachrichten registrierte und auf das Türschild migrierte, an dem die Person aktuell erkannt wurde. Hatte die Person zuvor einen Wegweisungsdienst gestartet mit einem Bürobewohner als Ziel, berechnet der Agent die Laufrichtung und stellt einen Richtungspfeil auf dem Bildschirm dar. Als Daten über den Benutzer speichert der Agent den Namen, die ID des RFID-Tags, den der Benutzer trägt, und die Zielperson bzw. den Zielraum. Zusätzlich braucht der Agent den Raumplan, um die Richtung berechnen zu können. Dies wurde in der ersten Version durch eine Datenstruktur realisiert, die alle Räume mit ihren Nachbarn speichert.

Auf dem zweiten Smart Doorplate Prototyp wurde eine graphische Oberfläche programmiert, die als Schnittstelle zwischen Benutzer und reflektiver Agent dient. Über diese Schnittstelle kann der Benutzer das Türschild zur jeder Zeit dafür benutzen, seinem persönlichen Agenten Aufgaben zu erteilen. Diese Aufgaben liegen meist in Form von Diensten vor. Der persönliche Agent nimmt die nötigen Informationen zur Vollendung seiner Aufgaben vom Benutzer direkt an. Da der

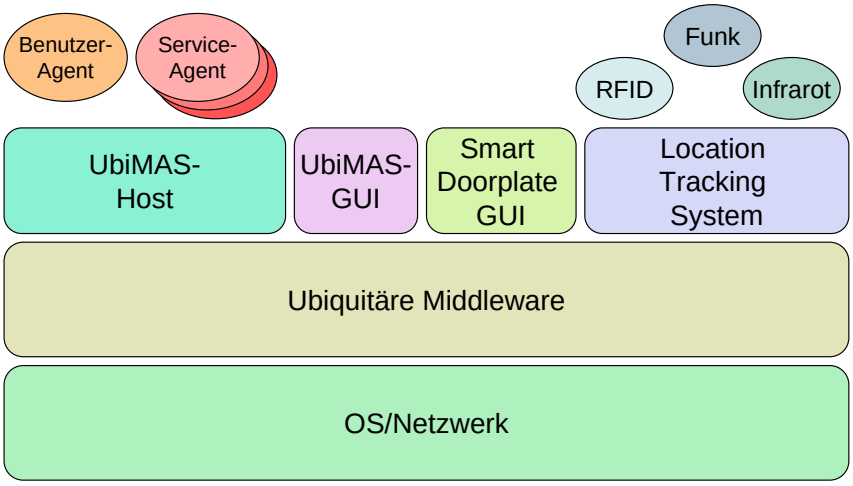

Abbildung 8.6: Architektur der Smart Doorplate Anwendung auf UbiMAS

reflektive Agent immer in der Nähe seines Benutzers bleiben soll, wurden Service-Agenten implementiert, die im Auftrag des Benutzer-Agenten einen bestimmten Dienst ausführen können. Die implementierten Dienste und die UbiMAS-GUI werden in [Sch03] und [Sod05] näher erläutert.

Die Service-Agenten teilen dem persönlichen Agenten mit, welche Informationen sie für die Abarbeitung des Dienstes brauchen. Der Benutzer-Agent kann diese Informationen, falls er sie nicht schon kennt, direkt vom Benutzer über das Türschild erfragen.

Die Oberfläche des Agentensystems stützt sich auf 5 Elemente. Neben dem Hauptfenster enthält die Oberfläche eine Tastatur, ein Dialogfenster, ein Zielauswahlfenster und ein Fenster für die Anzeige der Richtungspfeile. Im normalen Betriebszustand stellen die Türschilder klassische Informationen dar, wie Raumnummer, Bürobewohner und Sprechstunden, wie man in Abbildung 8.4 sehen kann. Steht der Benutzer vor dem Türschild kann die Oberfläche des Agentensystems aufgerufen werden, indem der Benutzer durch Tippen mit dem Finger zwischen den Fenstern umschaltet. Abbildung 8.7 zeigt einen Screenshot des UbiMAS Hauptfensters.

Der Aufenthaltsort des Benutzers kann nie auf Zentimeter genau berechnet werden. Die Lokationsnachrichten geben daher nur an, dass sich der Benutzer in der Nähe, d.h. auf einige Meter genau, eines Türschildes befindet. Es können sich also auch mehrere Benutzer im Umkreis von einem einzelnen Türschild aufhalten. Damit der Agent sicher sein kann, dass es sich um den richtigen Benutzer handelt, muss der Benutzer sich gegenüber dem Agenten authentifizieren. Hierzu ist es notwendig, einen Benutzernamen und ein Passwort einzugeben. Der Benutzer

Abbildung 8.7: UbiMAS Anmeldebildschirm

kann diese Eingabe über die UbiMAS-GUI komfortabel durchführen. Damit keine externe Tastatur für die Eingabe der Daten benötigt wird, bietet die Oberfläche eine Tastatur auf dem Bildschirm an. Durch tippen auf den Tastatur-Button wird das Tastaturfeld angezeigt, wie es in Abbildung 8.8 zu sehen ist.

Um so viele Zeichen wie möglich auf dem engen Raum benutzen zu können, wurde eine Methode implementiert, wie sie von Mobiltelefonen bekannt ist. Auf jedem Button der Tastatur sind mehrere Zeichen vorhanden, die durch schnelles Tippen hintereinander angewählt werden können. Die Tastatur lässt sich zudem erweitern, indem man auf Sonderzeichen tippt. Abbildung 8.9 stellt das Tastaturfeld mit aufgeklappten Sonderzeichen dar.

Die Benutzerdaten werden durch den UbiMAS-Host an den entsprechenden Benutzer-Agenten in Form einer verschlüsselten Nachricht übergeben. Der Agent überprüft den Benutzernamen und das Passwort und sendet dem UbiMAS-Host eine positive Bestätigung, falls diese stimmen. Daraufhin schaltet der UbiMAS-Host die Dienste frei, so dass der Benutzer ein Dienst auswählen und zum nächsten Fenster umschalten kann.

Als Dienste stehen dem Benutzer in der aktuellen Version ein Wegweisungsdienst, ein E-Mail-Dienst, ein Dienst zum Auslesen einer entfernten Datei und ein Dienst zum Holen einer entfernten Datei. Die Dienste werden jeweils von speziellen Service-Agenten ausgeführt, die der Benutzer-Agent beauftragt. Dazu sucht der

Abbildung 8.8: Tastaturfeld zur Eingabe der Benutzerdaten

Benutzer-Agent nach einem Service-Agenten. Ist ein passender Service-Agent im System schon gestartet und gerade frei, fordert der Benutzer-Agent diesen auf, zu dem Host, wo er sich befindet zu migrieren. Ist kein Service-Agent frei oder gestartet, kann der Benutzer-Agent den Host fragen, einen neuen Agenten zu erzeugen.

Wegweisungsdienst

Nach der Authentifizierung kann der Benutzer einen Dienst aus einer Liste auswählen. Wird der Wegweisungsdienst gewählt, muss der Benutzer angeben, zu welchem Ziel er geführt werden möchte. Hierzu wurde ein Zielauswahl-Fenster implementiert. Dieses Fenster zeigt einen Grundriss des Flurs mit all den vorhandenen Räumen und den Namen der Bürobewohner. Abbildung 8.10 zeigt einen Screenshot des Zielauswahl-Fensters. Die aktuelle Position des Benutzers ist grün unterlegt. Der Benutzer kann nun durch einfaches Tippen auf den Raum, zu dem er geführt werden möchte, dem Agenten den Zielraum mitteilen.

Der Agent übergibt die Zieldaten dem Service-Agenten, der den kürzesten Weg berechnet und einen entsprechenden Pfeil in die Richtung darstellt, in die der Benutzer gehen muss. Abbildung 8.11 zeigt das Fenster mit einem Richtungspfeil. Um zu gewährleisten, dass der Benutzer, während er durch den Flur läuft, immer einen Pfeil im Sichtfeld hat, werden auf den benachbarten Türschildern vor und

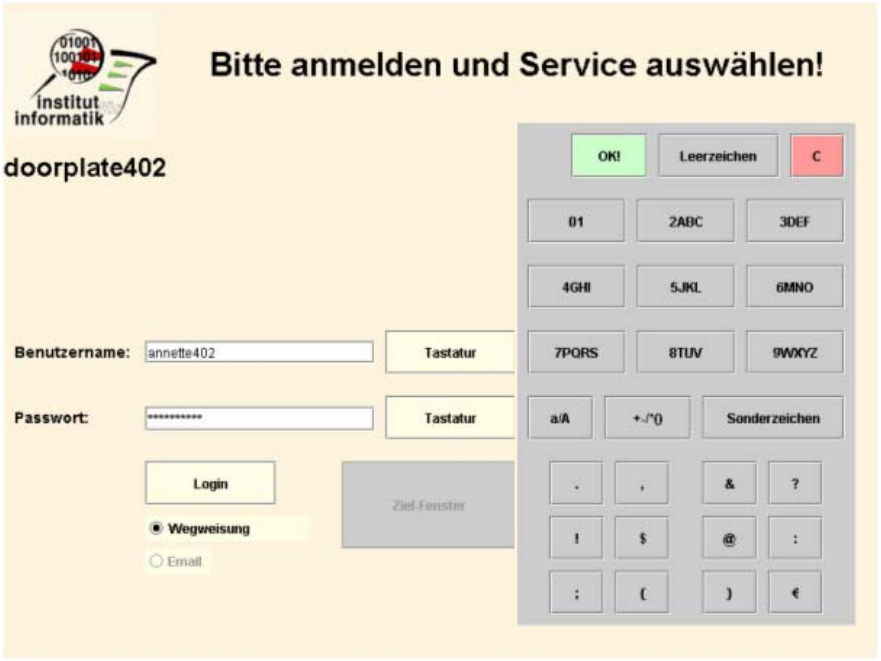

Abbildung 8.9: Tastaturfeld mit Sonderzeichen

hinter dem aktuellen Türschild die Richtungspfeile ebenfalls angezeigt. Dafür springt der Service-Agent auf die entsprechenden UbiMAS-Hosts und schaltet die richtigen Fenster ein. Der Benutzer wird also quasi durch eine Wolke von Pfeilen begleitet bis er das Zielbüro erreicht.

E-Mail-Dienst

Neben dem Wegweisungsdienst kann der Benutzer einen E-Mail-Dienst aufrufen, bei dem die eintreffenden E-Mails des Benutzers auf dem Bürorechner nach einem bestimmten Muster durchsucht werden können. So kann der Benutzer von jedem Türschild aus überprüfen, ob eine E-Mail von einem bestimmten Absender oder mit einer bestimmten Betreffzeile oder gar mit einem Schlüsselwort in der E-Mail angekommen ist. Diese E-Mail kann der Benutzer dann auf Wunsch auch direkt am aktuellen Türschild lesen.

Für diesen Dienst steht dem Benutzer ein spezieller Service-Agent zur Verfügung. Damit der Service-Agent auf die E-Mails des Benutzers zugreifen kann, müssen entsprechende Rechte auf dem Büro-PC gesetzt werden. In UbiMAS kennen nur die Benutzer-Agenten den Namen und das Passwort eines Benutzers. Aus diesem Grund muss der Service-Agent die Rechte über den Benutzer-Agenten beantragen. Der Service-Agent schickt die Spezifikation der benötigten Rechte an den Benutzer-Agenten. Dieser beantragt auf seinem Home-Host, welches der Büro-

Abbildung 8.10: Zielauswahl-Fenster

PC des Benutzers ist, die Rechte für den Service-Agenten und informiert ihn über die Erlaubnis. Die Erlaubnis wird nur diesem Service-Agenten gewährt, der über die eindeutige ID identifiziert werden kann. Die Sicherheitsfunktionalität dahinter wurde bereits im Abschnitt 7.5 detailliert beschrieben.

Hat der Service-Agent die Erlaubnis und die Filterinformationen vom Benutzer-Agenten bekommen, migriert er zu dem Büro-PC und durchsucht die E-mails nach dem angegebenen Filter. In der Zwischenzeit muss der Benutzer nicht an dem aktuellen Türschild warten, sondern kann einfach weitergehen. Der Service-Agent registriert sich auch für die Lokationsnachrichten, so dass er immer weiß, wo sich der Benutzer und somit auch der Benutzer-Agent befindet. Nach erfolgreicher Suche springt der Service-Agent auf das aktuelle Türschild und übergibt das Ergebnis dem Benutzer-Agenten. Dieser kann den Benutzer dann über das Smart Doorplate visuell oder per Ton auf die E-Mails aufmerksam machen.

Der Benutzer bekommt zunächst eine Zusammenfassung der gefundenen E-Mail mit Absender und Betreffzeile. Aus dieser Liste kann er dann die E-Mails auswählen, die er direkt vor Ort lesen möchte.

Datei-Dienste

Die Datei-Dienste basieren auf einer Idee, die es dem Benutzer erlaubt im Büroumfeld ohne technische Geräte, die er bei sich trägt, trotzdem auf entfernte

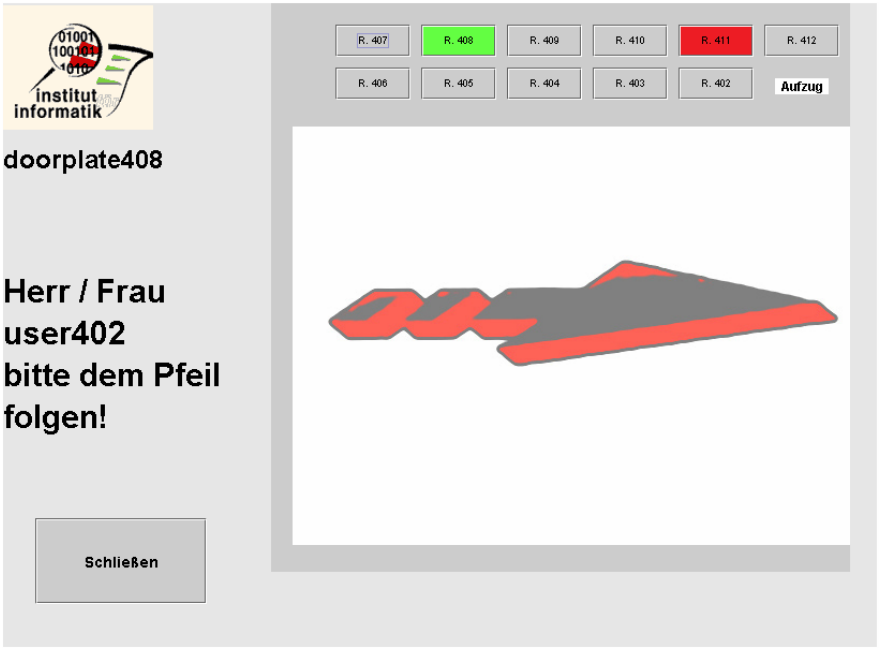

Abbildung 8.11: Fenster mit Richtungspfeil

Dateien zuzugreifen. Man stelle sich die Situation vor, dass eine Person sich in einer Besprechung befindet und keinen Laptop oder PDA bei sich hat. Fällt dieser Person ein, dass er eigentlich ein bestimmtes Dokument gern den anderen Teilnehmern zeigen möchte, kann er seinen Benutzer-Agenten beauftragen die Datei für ihn zu holen.

Die Dienste zum Auslesen oder Holen einer entfernten Datei sind ähnlich aufgebaut, wie der E-Mail-Dienst. Für beide Datei-Dienste gibt es entsprechende Service-Agenten. Der Benutzer kann den gewünschten Dienst aufrufen und nennt den Namen der Datei auf dem Büro-Rechner, die er gern gelesen oder geholt haben möchte. Der Benutzer-Agent beauftragt daraufhin den entsprechenden Service-Agenten, beantragt die entsprechenden Rechte beim Büro-PC und übergibt den Dateinamen an den Service-Agenten.

Der Service-Agent migriert zum Büro-PC und liest die angegebene Datei vom lokalen Dateisystem. Danach kehrt der Agent zu dem Türschild zurück, wo sich der Benutzer befindet und legt die Datei auf einem öffentlichen Ordner des Türschild-Rechners ab oder visualisiert den Inhalt auf Wunsch an dem Türschild.

8.4 Leistungsmessungen

Bei den Smart Doorplate Anwendungen ist es notwendig, dass der mobile Agent mindestens genau so schnell von einem Türschild zum Anderen springt, wie der Benutzer von einem Empfangsradius einer Tür zum anderen läuft. Um die möglichen Konstellationen der Geräte und Kommunikationsmedien auf Geeignetheit für die Smart Doorplate Szenarien zu untersuchen, wurden verschiedene Messungen durchgeführt. Es ging dabei um die Migrationszeit des mobilen Agenten zu erfassen, während die Größe des im Speicher des Agenten befindlichen Daten variiert wurde.

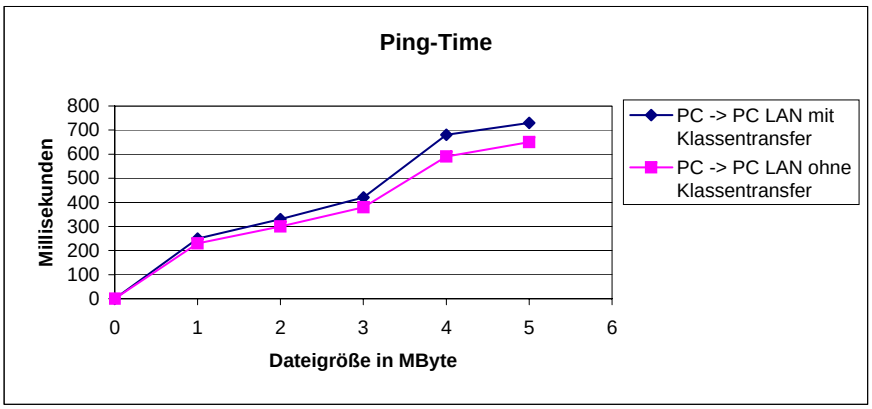

Abbildung 8.12: Agenten Ping Time bei PCs über LAN

Die ersten Messungen wurden auf dem ersten Smart Doorplate Prototyp durchgeführt. Um möglichst viele Konstellationen zu evaluieren, wurden verschiedene Testszenarien aufgebaut. Im ersten Testdurchlauf wurde das UbiMAS-Agentensystem auf rechenstarken PCs gestartet, die über Ethernet miteinander verbunden waren. Ein einfacher Agent wurde implementiert, der zu einem Knoten und wieder zurück springen musste. Diese Migrationszeit, die auch Ping-Time genannt wird, wurde gemessen und aufgezeichnet. Es wurden jeweils 50 Sprünge ausgeführt und die mittlere Ping-Time ermittelt. Dieser Durchlauf wurde mit verschiedenen Größen der Agentendaten wiederholt. UbiMAS fordert die Klasse eines unbekannten Agenten vom Sender-Host an. Um den Zusammenhang zwischen Agentenmigration mit Klassentransfer und ohne zu evaluieren, wurde die oben beschriebene Testreihe einmal mit Klassentransfer und einmal ohne Klassentransfer durchgeführt. Abbildung 8.12 stellt die Ergebnisse der ersten Messung graphisch dar.

Es ist zu erkennen, dass die Migrationszeit bei Ethernet-verbundenen PCs im Millisekundenbereich liegen. Sogar bei einer Agentengröße von mehr als 5 MByte

braucht der Agent weniger als eine Sekunde von einem Host zum anderen zu migrieren.

In dem nächsten Testdurchlauf wurde die Kommunikation über WLAN evaluiert. Dabei wurde UbiMAS statt auf zwei PCs, auf einem PC und einem PDA gestartet. Die Ping-Time wurde wiederum bei variierender Agentengröße gemessen. Abbildung 8.13 zeigt die Ergebnisse dieser Messung. Die Kommunikation über WLAN und der Einsatz eines PDAs als UbiMAS-Host spiegeln sich in höheren Migrationszeiten wider. Schon bei kleinen Größen dauert die Migration des Agenten mehr als 2 Sekunden. Bei 5 MByte Agentengröße muss der Benutzer mehr als 8 Sekunden auf den Agenten warten.

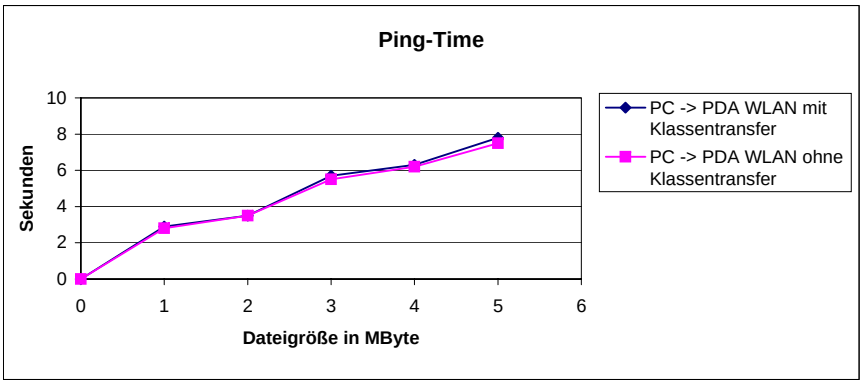

Abbildung 8.13: Agenten Ping Time bei PC und PDA über WLAN

Diese Messungen ließen auch für die folgenden Testdurchläufe nichts Gutes erhoffen. Die Messungen wurden mit einem PC und einem PDA, die diesmal über Bluetooth kommunizierten wiederholt. Abbildung 8.14 stellt die Ergebnisse dieser Messung dar. Bei den darauf folgenden Tests wurden zwei PDAs, die einmal über WLAN und einmal über Bluetooth verbunden waren, eingesetzt. Die Ergebnisse sind in Abbildung 8.15 und Abbildung 8.16 zusammengefasst.

Aus den vorliegenden Ergebnissen wurde deutlich, dass der Einsatz von UbiMAS auf mobilen Geräten und drahtlosen Netzen keine zufriedenstellenden Werte liefert. Die Migration des Agenten sollte unter der Zeit liegen, die der Benutzer zum Gehen von einer Tür zur nächsten braucht. Hinzu kommt, dass das Location-Tracking-System auch einen Offset besitzt, der in den Messungen noch gar nicht betrachtet wurde.

Aus diesen Gründen wurde der zweite Prototyp der Smart Doorplates auf rechenstarken PCs, die über Ethernet verbunden sind, realisiert. In den oben vorgestellten Tests wurde die Sicherheit völlig außer acht gelassen, da Java 2 ME bisher noch kein Security Paket bietet, d.h. auf den PDAs keine Verschlüsselung möglich

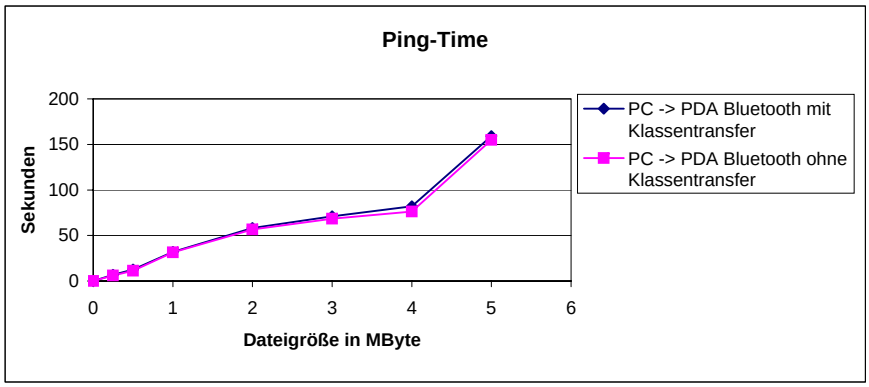

Abbildung 8.14: Agenten Ping Time bei PC und PDA über Bluetooth

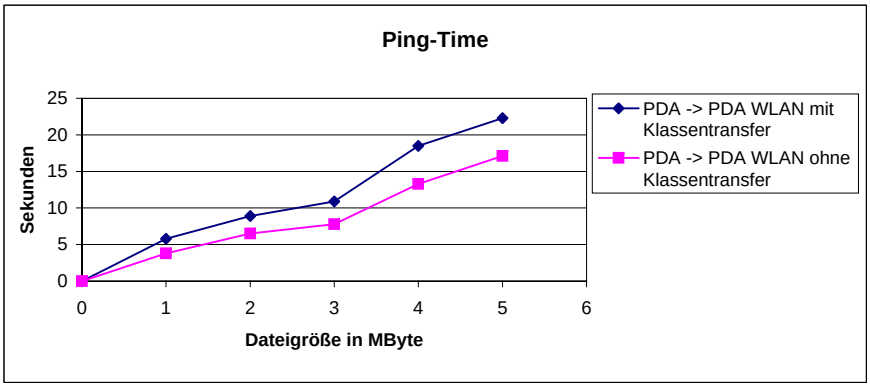

Abbildung 8.15: Agenten Ping Time bei zwei PDAs über WLAN

war. Auch wenn es zukünftig möglich sein sollte, würde dies die Migrationszeiten noch verschlechtern, da bekanntlich Verschlüsselung sehr rechenintensiv ist.

In der aktuellen Version des Smart Doorplates wurde die Migrationszeit eines Agenten auch unter Berücksichtigung von Sicherheitsfunktionen gemessen. Abbildung 8.17 stellt die Migrationszeiten mit und ohne Verschlüsselung bei der Übertragung des Agenten dar.

Es ist deutlich zu erkennen, dass die Verschlüsselung mehr Zeit in Anspruch nimmt. Diese Zeit nimmt bei wachsender Größe des Agenten zu, da damit mehr Daten zu verschlüsseln sind. Jedoch liegen die Migrationszeiten auch bei 4 MByte Agentengröße und Verschlüsselung gerade bei 1,5 Sekunden. Es kann also gefolgert werden, dass der Einsatz von UbiMAS auf PCs mit allen Sicherheitsfunktionen akzeptabel und zufriedenstellend ist.

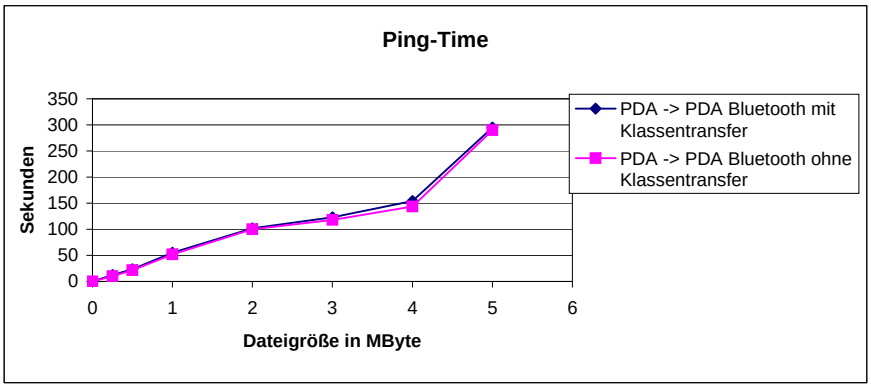

Abbildung 8.16: Agenten Ping Time bei zwei PDAs über Bluetooth

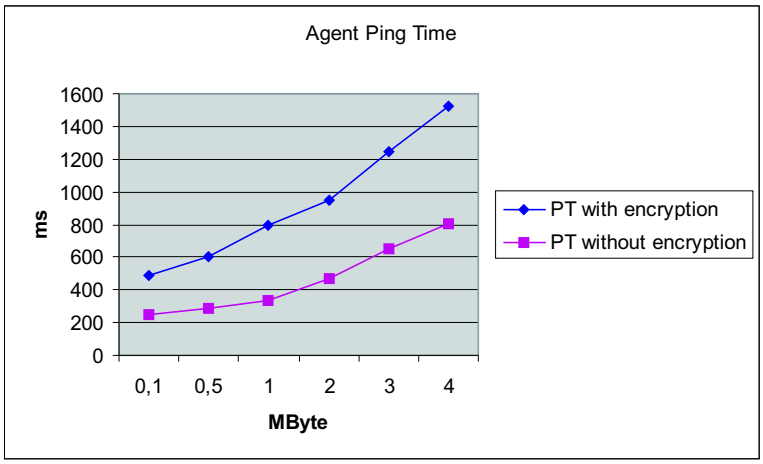

Abbildung 8.17: Agenten Ping Time mit und ohne Verschlüsselung

9 Zusammenfassung und Ausblick

Das letzte Kapitel dieser Arbeit soll die Ideen, Umsetzungen und Ergebnisse, wie sie in den vorhergehenden Kapiteln beschrieben wurden, zusammenfassen und einen Ausblick auf mögliche zukünftige Arbeiten präsentieren.

9.1 Zusammenfassung

Ubiquitäre Systeme sind eine Erweiterung der „eingebetteten Systeme“. Die letzteren sind Rechnersysteme zur Steuerung von technischen Abläufen, also z.B. zur Fertigungssteuerung in einer Fabrik, als Fahrkartenautomat oder als ABS im Auto. Als ubiquitäre (allgegenwärtige) Systeme bezeichnet man eingebettete Rechnersysteme, die selbstständig auf ihre Umwelt reagieren. Bei einem ubiquitären System kommt zusätzlich zu einem eingebetteten System noch Umgebungswissen hinzu, das es diesem System erlaubt, sich in hohem Maße auf den Menschen einzustellen. Die Benutzer sollen nicht in eine virtuelle Welt gezogen werden, sondern die gewohnte Umgebung soll mit Computerleistung angereichert werden, so dass neue Dienste entstehen, die den Menschen entlasten und ihn von Routineaufgaben befreien.

Die Einbeziehung von Informationen aus der natürlichen Umgebung der Geräte stellt ein wesentliches Kennzeichen ubiquitärer Systeme dar. Die Berücksichtigung der Umgebung, des Kontexts, geschieht über die Erfassung, Interpretation, Speicherung und Verbindung von Sensordaten. Zusätzlich zu den Daten, die direkt aus der Umgebung gewonnen werden können, spielen persönliche Informationen über den Menschen, wie Profile, Präferenzen, Vorlieben und Gewohnheiten eine große Rolle.

Ubiquitäre Systeme sind dezentral aufgebaut. Ein zentralisierter Ansatz ist wegen der Vielzahl der verschiedenen Geräte und Informationen fast nicht möglich. Ein dezentrales System hat auch den Vorteil, weniger fehleranfällig zu sein, da bei einem Ausfall eines Knoten, die anderen Knoten ihre Funktionalitäten aufrecht erhalten können.

Wechselt der Benutzer in einem ubiquitären System den Aufenthaltsort, stellt sich die Frage, wie die persönlichen Informationen an die aktuelle Position gelangen können. Der Benutzer könnte die Daten mit sich tragen und bei Bedarf automatisch an das ubiquitäre System in der Umgebung weiter geben. Dazu müsste er ein elektronisches Gerät, wie Laptop, Tablet PC, PDA oder Wearables immer mit sich führen. Solche Geräte sind gewöhnlich an eine einzige Person gebunden, so dass die Sicherheit und Geheimhaltung in der Verantwortung dieser Person liegen. Der Nachteil dieser Methode ist, dass Menschen oft aus Unbequemlichkeit die Geräte vergessen mitzunehmen oder achtlos herumliegen lassen. Eine bessere Methode ist hier, die Verantwortung für die Speicherung und Übertragung der persönlichen Information an das ubiquitäre System weiterzugeben, wobei das System für die Sicherheit der Daten vor unbefugten Zugriff Sorge tragen muss.

Das Paradigma der mobilen Agenten passt geradezu ideal auf den dezentralen Aufbau und die Idee des virtuellen Abbildes. Betrachtet man mobile Agenten im ubiquitären Umfeld, so stellt der Agent ein virtuelles Abbild des Benutzers dar, der persönliche Informationen mit sich trägt. Auf Grundlage dieser Daten kann der reflektive Agent selbständig im Namen seines Benutzers verschiedene Dienste ausführen. Zudem kann der mobile Agent über geeignete Schnittstellen Informationen über die Umgebung erhalten, die von dem lokalen ubiquitären System angeboten werden.

Da es um sensible Daten über Personen geht, darf die Sicherheit bei den reflektiven Agenten nicht zu kurz kommen. Es müssen sowohl Agenten-Host, als auch die einzelnen mobilen Agenten gegen potentielle Bedrohungen geschützt werden. Diese Arbeit stellte eine umfassende Bedrohungsanalyse vor, auf deren Grundlage ein neues ubiquitäres Agentensystem, das so genannte Ubiquitous Mobile Agent System (UbiMAS), entwickelt wurde. Das UbiMAS Framework beinhaltet eine ubiquitäre Middleware, die aus Komponenten für Kommunikation, Gruppenmanagement, Sicherheit und Kontextmanagement besteht. Agenten-Hosts laufen parallel neben anderen Diensten oberhalb der ubiquitären Middleware.

Das Sicherheitsmodell des UbiMAS Frameworks verlagert Sicherheitsaufgaben, die die knotenübergreifende Kommunikation betreffen, auf die Middleware. Die Dienste oberhalb der Middleware implementieren weitere Sicherheitsmethoden nach den jeweiligen Anforderungen des Dienstes. Der UbiMAS-Host kann sich also als Dienst auf eine sichere Kommunikation verlassen und realisiert nach den Anforderungen eines ubiquitären Agentensystems eine umfassende Sicherheitsarchitektur.

UbiMAS wurde für verschiedene Anwendungen in einem realen ubiquitären Testumfeld eingesetzt. Im Rahmen des Smart Doorplate Projektes an dem Lehrstuhl für Systemnahe Informatik und Kommunikationssysteme wurden gewöhnliche Türschilder durch elektronische Displays ersetzt. Es wurden reflektive Agenten

vorgestellt, für die die Smart Doorplates als Schnittstelle zum Benutzer dienten, und die verschiedene ubiquitäre Anwendungen, wie der Wegweisungsdienst oder E-Mail-Dienst, implementierten.

Auf den Prototypen der Smart Doorplates wurden Leistungsmessungen von UbiMAS durchgeführt. Für diese Evaluierungen wurde UbiMAS sowohl auf rechenstarken PCs als auch auf mobilen PDAs, die drahtlos per WLAN oder Bluetooth kommunizieren konnten, installiert und getestet. Die Migrationszeiten von Agenten waren für die Messungen ausschlaggebend. Die Ergebnisse zeigten, dass der Einsatz von UbiMAS auch auf mobilen Geräten möglich ist, jedoch die Migrationszeiten über drahtlose Netze bei mobilen Geräten keine zufriedenstellenden Werte liefern. Für sicherheitsrelevante und zeitkritische Anwendungen wurde der Einsatz auf PCs, die über Ethernet verbunden sind, empfohlen, da auf aktuellen mobilen Geräten keine Sicherheitspakete angeboten werden und die Migrationszeiten der Agenten über Funk länger dauern. UbiMAS wurde auf verschiedenen Veranstaltungen und Präsentationen, wie die *Nacht der Wissenschaft*, erfolgreich vorgestellt. Der tägliche praktische Einsatz ist zur Zeit wegen des Fehlens eines weiträumigen Location-Tracking-Systems leider nicht möglich, steht aber zukünftig in Planung.

9.2 Ausblick

Diese Arbeit verdeutlichte, dass das Potential der mobilen Agenten noch nicht ausgeschöpft ist. Mit neuen Technologien entstehen neue Anwendungsfelder für mobile Agenten. Das Hauptproblem der Agentensysteme ist weiterhin die Sicherheit. Mangelnder Schutz vor potentiellen Gefahren resultiert in der Ablehnung des Agentensystems. Zukünftige Entwicklungen sollten die Sicherheit nie vernachlässigen. Wie diese Arbeit auch zeigte, kann durch gezielte Maßnahmen die Mehrheit der Bedrohungen abgewehrt werden.

Der mögliche Mehrwert, den Anwendungen auf Basis mobiler Agenten bieten, sollte auch betrachtet werden. So könnten mobile Agenten zukünftig administrative Aufgaben in ubiquitären Umgebungen übernehmen. Man stelle sich nur vor, wie aufwendig und zeitraubend es sein wird, bei der Vielzahl an verschiedenen Geräten, die Software (Treiber, Middleware, Anwendungen) auf den Rechnern auf den aktuellen Stand zu bringen. Mobile Agenten könnten selbständig Updates je nach Gerät und Funktion gezielt herunter laden und installieren.

Auch der Einsatz der mobilen Agenten im Bereich des Organic und Autonomic Computing könnte untersucht werden. Autonomic und Organic Computing beschäftigen sich mit Systemen, die selbst-organisierend, selbst-konfigurierend, selbst-heilend, selbst-schützend, selbst-erklärend, kontextbewusst und voraus-

schauend sind. Die mobilen Agenten könnten das Netz-Management unterstützen, indem sie im Sinne der Selbst-Optimierung Informationen von entfernten Knoten sammeln und verteilen, und im Sinne des Selbst-Schutzes Anomalien erkennen und Alarm schlagen oder Gegenmaßnahmen einleiten.

Der Einsatz von mobilen Agentensystemen auf tragbaren Geräten ist zur Zeit zwar immer noch rechenaufwendig, doch mit der rasanten Entwicklung der Mikrochips dürfte dies in den kommenden Jahren keine Probleme mehr bereiten. Die Nachfolger der PDAs, die in dieser Arbeit eingesetzt wurden, haben jetzt schon einen 50 % schnelleren Prozessor und die ersten Geräte mit 1 GHz Taktfrequenz und mehreren Hundert MByte Speicher stehen bereits in den Startlöchern.

Literaturverzeichnis

[ACFT97] Michel Abdalla, Walfredo Cirne, Leslie Franklin, and Abdallah Tabbara. Security issues in agent based computing. In *15th Brazilian Symposium on Computer Networks*. Sao Carlos, Brasilien, 1997.

[AEH+02] Gregory D. Abowd, Maria Ebling, Guerney Hund, Hui Lei, and Hans-Werner Gellersen. Context-Aware Computing. In *IEEE Pervasive Computing*, Juli 2002.

[ANI00] ANIMA Project. http://anima.ibermatica.com/anima, 2000.

[Bei02] Michael Beigl. Context aware computing. In *Tutorial at ARCS 2002*, Karlsruhe, April 2002.

[BHR+97a] Joachim Baumann, Fritz Hohl, Nikolaos Radouniklis, Kurt Rothermel, and Markus Straßer. Communication Concepts for Mobile Agent Systems. In *Proc. First International Workshop on Mobile Agents (MA'97)*. Vol. 1219 Lecture Notes in Computer Science, Springer Verlag, Berlin, 1997.

[BHR97b] Joachim Baumann, Fritz Hohl, and Kurt Rothermel. Mole - concepts of a mobile agent system. Technical Report TR-1997-15, 1997.

[BPR99] Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa. JADE - A FIPA-compliant agent framework. In *CSELT internal technical report*, 1999.

[BPTU03a] Faruk Bagci, Jan Petzold, Wolfgang Trumler, and Theo Ungerer. Einsatz mobiler Agenten in verteilten ubiquitären Systemen. In *19. PARS-Workshop*, Basel, Switzerland, March 2003.

[BPTU03b] Faruk Bagci, Jan Petzold, Wolfgang Trumler, and Theo Ungerer. Einsatz von XML zur Kontextspeicherung in einem agentenbasierten ubiquitären System. In *XMIDX-Workshop*, Berlin, Germany, February 2003.

[BPTU03c] Faruk Bagci, Jan Petzold, Wolfgang Trumler, and Theo Ungerer. Ubiquitous Mobile Agent System in a P2P-Network. In *UbiSys-Workshop at the Fifth Annual Conference on Ubiquitous Computing*, Seattle, USA, October 2003.

[BR02] Walter Binder and Volker Roth. Secure mobile agent systems using java: where are we heading? In *Proceedings of the 2002 ACM symposium on Applied computing*, pages 115 – 119. ACM Press, Madrid, Spain, 2002.

[BR05] Peter Braun and Wilhelm R. Rossak. *Mobile Agents*. Copublished by Morgan Kaufmann Publishers and dpunkt.verlag, 2005.

[Bro98] Gary Deward Brown. *System 390 Job Control Language*. John Wiley & Sons, New York, 1998.

[BSP+05a] Faruk Bagci, Holger Schick, Jan Petzold, Wolfgang Trumler, and Theo Ungerer. Communication and Security Extensions for a Ubiquitous Mobile Agent System (UbiMAS). In *Proceedings of the 2nd conference on Computing frontiers*, pages 246–251, Ischia, Italy, May 2005.

[BSP+05b] Faruk Bagci, Holger Schick, Jan Petzold, Wolfgang Trumler, and Theo Ungerer. Support of Reflective Mobile Agents in a Smart Office Environment. In *18th International Conference on Architecture of Computing Systems*, pages 79–92, Hall in Tirol/Innsbruck, Austria, March 2005.

[BWZ98] Walter Brenner, Hartmut Wittig, and Rudiger Zarnekow. *Intelligent Software Agents: Foundations and Applications*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1998.

[Che98] David M. Chess. Security issues in mobile code systems. In *Mobile Agents and Security*, pages 1–14, London, UK, 1998. Springer-Verlag.

[CHK95] David M. Chess, Colin Harrison, and Aaron Kershenbaum. Mobile Agents: Are They a Good Idea? In *IBM Research Report RC 19887*, 1995.

[Con02] The World Wide Web Consortium. W3C. http://www.w3.org, 2002.

[Con03] FIPA Consortium. Foundation for Intelligent Physical Agents. http://www.fipa.org, 2003.

[CROO00] Rem W. Collier, Colm F.B. Rooney, Ruadhan P.S. O'Donoghue, and Gregory M.P. O'Hare. Mobile BDI Agents. In *Proceedings of the 11th*

Irish Conference (AICS 2000) on Artificial Intelligence & Cognitive Science. Galway, Ireland, 2000.

[DIC00] DICEMAN Project. Distributed Internet Content Exchange with MPEG-7 & Agent Negotiations. http://www.teltec.dcu.ie/diceman/, 2000.

[DRD+99] Alan Dix, Tom Rodden, Nigel Davies, Jonathan Trevor, Adrian Friday, and Kevin Palfreyman. Exploiting space and location as a design framework for interactive mobile systems. In *ACM Transaction on Computer-Human Interaction (TOCHI)*, pages 285–321, 1999.

[FAC00] FACTS Project. http://www.labs.bt.com/profsoc/facts, 2000.

[FG97] Stan Franklin and Art Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In *ECAI '96: Proceedings of the Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages*, pages 21–35, London, UK, 1997. Springer-Verlag.

[GAA+95] Don Gilbert, Manny Aparicio, Betty Atkinson, Steve Brady, Joe Ciccarino, Benjamin Grosof, Pat O'Connor, Damian Osisek, Steve Pritko, Rick Spagna, and Les Wilson. IBM Intelligent Agent Strategy. In *IBM Corporation*, 1995.

[GBH98] Michael S. Greenberg, Jennifer C. Byington, and David G. Harper. Mobile agents and security. In *IEEE Communications Magazine*, pages 76 – 85. 36(7), 1998.

[Gra96] Robert S. Gray. Agent tcl: A flexible and secure mobile-agent system. In *Proceedings of the 1996 Tcl/Tk Workshop*, pages 9 – 23. USENIX Association, 1996.

[Gro98] Object Management Group. Mobile agent system interoperability facilities specification. In *OMG TC Document orbos/98-03-09*, März 1998.

[GSB02] Hans-W. Gellersen, Albrecht Schmidt, and Michael Beigl. Multi-Sensor Context-Awareness in Mobile Devices and Smart Artifacts. In *Mobile Networks and Applications 7*, pages 341–351, 2002.

[GV97] Carlo Ghezzi and Giovanni Vigna. Mobile Code Paradigms and Technologies: A Case Study. In *Proc. First International Workshop on Mobile Agents (MA'97), Berlin*, Apr. 1997.

[GW89] James R. Groff and Paul N. Weinberg. *SQL: The Complete Reference*. Addison-Wesley, 1989.

[HB01] Jeffrey Hightower and Gaetano Boriello. Location Systems for Ubiquitous Computing. *IEEE Computer*, pages 57–66, August 2001.

[Hew77] C. Hewitt. Viewing Control Structures as Patterns of Passing Messages. In *Artifial Intelligence, Vol. 8, Nr. 3.* North-Holland Publishing Company, Amsterdam, 1977.

[HNS01] Uwe Hansmann, Martin S. Nicklous, and Thomas Stober. *Pervasive computing handbook.* Springer-Verlag New York, Inc., New York, NY, USA, 2001.

[Hoh01] Fritz Hohl. Sicherheit in Mobile-Agenten-Systemen. In *Dissertation.* Universität Stuttgart, 2001.

[HR95] Barbara Hayes-Roth. An architecture for adaptive intelligent systems. *Artificial Intelligence*, 72(1-2):329–365, 1995.

[IET97] IETF RFC2141. http://www.faqs.org/rfcs/rfc2141.html, May 1997.

[Inc99] Adobe Systems Incorporated. *PostScript(R) Language Reference.* Addison-Wesley, 1999.

[jdo02] JDOM. http://www.jdom.org, 2002.

[JW98] Nicholas R. Jennings and Michael J. Wooldridge. *Applications of intelligent agents.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1998.

[Kay84] Alan Kay. Computer Software. In *Scientific American 251(3)*, pages 53–59, 1984.

[KLO97] Günter Karjoth, Danny B. Lange, and Mitsuru Oshima. A security model for aglets. *IEEE Internet Computing*, 1(4):68–77, 1997.

[KO03] Stephen Keegan and Gregory M.P. O'Hare. Easishop: Enabling uCommerce through IntelligentMobile Agent Technologies. In *Proceedings of 5th International Workshop on Mobile Agents for Telecommunication Applications.* Springer-Verlag LNCS, Marrakesh, Morocco, 2003.

[Lei03] Frank Joachim Leitner. Architektur eines sicheren Mobile-Agenten-Systems für das Netzmanagement. In *Dissertation.* Technische Universität München, 2003.

[LMKQ02] Samuel J. Leffler, Marshall K. McKusick, Michael J. Karels, and John S. Quaterman. *The Design and Implementation of the 4.3 BSD UNIX Operating System.* McGraw-Hill Osborne Media, 2002.

[LOO02] Terry D. Lowen, Peter T. O'Hare, and Gregory M.P. O'Hare. Mobile Agents point the WAY: Context Sensitive Service Delivery through Mobile Lightweight Agents. In *Proc. of Autonomous Agents and Multi-Agent Systems (AAMAS-2002)*. Bologna, 2002.

[LY99] Tim Lindholm and Frank Yellin. *The Java(TM) Virtual Machine Specification (2nd Edition)*. Addison-Wesley, 1999.

[MAR99] MARINER Project. http://www.teltec.dcu.ie/mariner, 1999.

[MB01] Patrik Mihailescu and Walter Binder. A Mobile Agent Framework for M-Commerce. In *Agents in E-Business (AgEB-2001)*. Workshop of the Informatik 2001, Vienna, Austria, 2001.

[Mon00] Monads Project. Adaptation Agents for Nomadic Users. http://www.cs.helsinki.fi/research/monads/, 2000.

[Nwa96] Hyacinth S. Nwana. Software Agents: An Overview. In *Knowledge Engineering Review*, pages 205–244. 11(3), 1996.

[OO03] Gregory M.P. O'Hare and Michael J. O'Grady. Gulliver's Genie: A Multi-Agent System for Ubiquitous and Intelligent Content Delivery. In *Press, Computer Communications*, pages 1177 – 1187. Vol. 26, Issue 11, Elsevier Press, 2003.

[PBT+05] Jan Petzold, Faruk Bagci, Wolfgang Trumler, Theo Ungerer, and Lucian Vintan. Next Location Prediction Within a Smart Office Building. In *1st International Workshop on Exploiting Context Histories in Smart Environments (ECHISE'05) at the 3rd International Conference on Pervasive Computing*, Munich, Germany, May 2005.

[PBTU03] Jan Petzold, Faruk Bagci, Wolfgang Trumler, and Theo Ungerer. Global and Local Context Prediction. In *Artificial Intelligence in Mobile Systems 2003 (AIMS 2003)*, Seattle, WA, USA, October 2003.

[PG03] Holger Pals and Claus Grewe. Dynamisch-adaptive Lastverwaltung für Mobile Agenten. In *Proc. 19. PARS Workshop*, pages 97–108. Mitteilungen - Gesellschaft für Informatik e.V. Parallele Algorithmen und Rechnerstrukturen, Nr. 20, Basel, 2003.

[Pro02] Project JXTA. http://www.jxta.org, November 2002.

[SBG99] Albrecht Schmidt, Michael Beigl, and Hans-W. Gellersen. There is more to Context than Location. In *Computer & Graphics Journal*, volume 23, pages 893–902, Dezember 1999.

[Sch03] Holger Schick. Entwicklung und Implementierung von Kommunikations- und Sicherheitskomponenten eines ubiquitären mobilen Agentensystems. In *Diplomarbeit*. Universität Augsburg, 2003.

[Sei05] Christian Seitz. Ein Framework für die Profilbasierte Gruppenbildung in ad hoc Umgebungen. In *Dissertation*. Universität Augsburg, 2005.

[Sod05] Aboua Sodonougbo. Entwicklung und Implementierung einer grafischen Oberfläche für die Benutzer-Agenten-Kommunikation und eines Wegweisungsdienstes. In *Bachelorarbeit*. Universität Augsburg, 2005.

[TBPU03a] Wolfgang Trumler, Faruk Bagci, Jan Petzold, and Theo Ungerer. Smart Doorplate. In *First International Conference on Appliance Design (1AD)*, Bristol, GB, May 2003. Reprinted in Pers Ubiquit Comput (2003) 7: 221-226.

[TBPU03b] Wolfgang Trumler, Faruk Bagci, Jan Petzold, and Theo Ungerer. Smart Doorplate - Toward an Autonomic Computing System. In *The Fifth Annual International Workshop on Active Middleware Services (AMS2003)*, Seattle, USA, June 25 2003.

[TPBU04a] Wolfgang Trumler, Jan Petzold, Faruk Bagci, and Theo Ungerer. AMUN - Autonomic Middleware for Ubiquitious eNvironments Applied to the Smart Doorplate Project. In *International Conference on Autonomic Computing (ICAC-04)*, New York, USA, May 17-18 2004.

[TPBU04b] Wolfgang Trumler, Jan Petzold, Faruk Bagci, and Theo Ungerer. Towards an Organic Middleware for the Smart Doorplate Project. In *GI Workshop on Organic Computing - in connection with 34. Jahrestagung der Gesellschaft für Informatik*, Ulm, Germany, September 2004.

[Tur50] A. M. Turing. Computing machinery and intelligence. In *Mind, 59*, pages 433–460, 1950.

[Vig98] Giovanni Vigna. Mobile Code Technologies, Paradigms, and Applications. In *PhD Thesis, Politecnico di Milano/Italien*, Febr. 1998.

[VOO95] Sankar Virdhagriswaran, Damian Osisek, and Pat O'Connor. Standardizing agent technology. *StandardView*, 3(3):96–101, 1995.

[Wei91] Mark Weiser. The Computer for the 21st Century. In *Scientific American*, pages 94–104, September 1991.

[Wei93] Mark Weiser. Hot topics: Ubiquitous computing. In *IEEE Computer*, Okt. 1993.

[Whi94] James O. E. White. Telescript Technology: Mobile Agents. In *General Magic White Paper*, 1994.

[Wil02] Brendon J. Wilson. *JXTA*. New Riders Publishing, 2002.

Abbildungsverzeichnis

Tabellenverzeichnis