JOHANNES KEPLER
UNIVERSITÄT LINZ
Netzwerk für Forschung, Lehre und Praxis

POND

Ein Agentensystem mit Fokus auf Sicherheit und Bezahlung für Leistungen unter Berücksichtigung rechtlicher Aspekte

DISSERTATION

zur Erlangung des akademischen Grades

DOKTOR DER TECHNISCHEN WISSENSCHAFTEN

in der Studienrichtung

INFORMATIK

Angefertigt am Institut für Informationsverarbeitung und Mikroprozessortechnik (FIM)

Betreuung:
o. Univ.-Prof. Dr. Jörg R. Mühlbacher

Von:
Dipl.-Ing. Michael Sonntag

Gutachter:
o. Univ.-Prof. Dr. Jörg R. Mühlbacher
o. Univ.-Prof. Dr. Roland Traunmüller

Linz, Jänner 2002

Johannes Kepler Universität Linz
A-4040 Linz • Altenberger Straße 69 • Internet: http://www.uni-linz.ac.at • DVR 0093696

My name is Pond, James Pond!

Danksagung

Zuallererst möchte ich meinen Eltern danken, welche durch ihre Unterstützung in jeder Hinsicht mein Studium ermöglicht haben, und immer ein offenes Ohr für mich hatten. Sie haben mir ein häusliches Umfeld geschaffen, das einem erfolgreichen Studienabschluß sehr förderlich war.

Zu Dank verpflichtet bin ich auch meinem Bruder, in dessen Firma ich die Leitung der EDV übernehmen konnte. Es ist immer wieder heilsam zu erleben, welche wissenschaftlich gesehen trivialen Probleme in der Praxis große Schwierigkeiten bereiten können. Es hilft, daran zu denken, daß ein wichtiges Augenmerk bei der Forschung auch die praktische Anwendung sein soll: Sowohl als Auslöser, wie auch als Ergebnis des Forschungsprozesses.

Allen Kolleginnen, Kollegen, Mitarbeiterinnen und Mitarbeitern des Institutes möchte ich für das gute Arbeitsklima danken.

Sehr hilfreich waren auch Univ.-Prof. Dr. Traunmüller und Univ.-Prof. Dr. Chroust, welche es mir ermöglichten, einzelne Aspekte bzw. Teile dieser Arbeit auf verschiedenen Konferenzen zu präsentieren bzw. an solchen teilzunehmen.

Herzlichst danken möchte ich auch Frau Univ.-Prof. Dr. Kappel und Herrn Univ.-Doz. Dr. Reich, welche durch Ihre Hinweise und Anregungen sehr positiv zu dieser Arbeit beigetragen haben.

Besonderer Dank gebührt Herrn. Univ.-Prof. Dr. Mühlbacher, welcher die laufende und kontinuierliche Betreuung übernahm und an dessen Institut ich Gelegenheit hatte, an mehreren Projekten mitzuarbeiten (gefördert vom Land Oberösterreich, Wi/Ge 201.515/1-2000/Wwin, und der Österreichischen Nationalbank, Projekt Nr. 7742), welche als Anstoß für diese Arbeit dienten. Sehr hilfreich waren auch seine vielen Anregungen und nützlichen Hinweise, welche dieser Arbeit zugute kamen. Prof. Mühlbachers Verhalten war und wird mir ein Leitbild für viele Lebensentscheidungen sein.

Vielen Dank!

Abstract

After a general presentation of intelligent agents, their areas of use, as well as their advantages and disadvantages, in this thesis three main aspects of mobile agents are explored:

1. Dynamic security system: Agents must be assessed according to their owner (configuration and goals) as well as the programmer of their code. An agent should also be granted permissions according to payment for them as well as any additional usage.
2. Payment for services by agents themselves: Immediate payment is desirable if agents draw on resources or utilize services. Data used for payment must be protected from illegal access and secured against loss or modifications.
3. Legal aspects: When using agents, legal questions like validity of their actions or liability for damages through undesirable conduct arise. In a practical view, method for securing evidence are required.

An agentsystem implemented in Java is introduced, which incorporates these three aspects. Finally, some agents as well as systems of agents are presented, which show the functionality of the system.

Kurzfassung

In dieser Arbeit werden neben einer allgemeinen Darstellung intelligenter Agenten, ihrer Anwendungsgebiete, sowie der damit verbundenen Vor- als auch Nachteile, drei Hauptaspekte von mobilen Agenten theoretisch untersucht:

1. Dynamische Sicherheitssysteme: Agenten sind sowohl nach ihrem Besitzer (Konfiguration und Beauftragung) als auch nach dem Hersteller ihres Programmcodes zu beurteilen. Die einem Agenten zu gewährenden Berechtigungen sollten weiters auf einer Bezahlung für zusätzliche Berechtigungen bzw. Nutzungen basieren.
2. Bezahlung für Leistungen direkt durch Agenten: Nehmen Agenten Dienste in Anspruch, so ist eine sofortige Bezahlung wünschenswert. Die hierfür verwendeten Daten sind vor unbefugtem Zugriff zu schützen und gegen Verlust oder Modifikationen zu sichern.
3. Rechtliche Aspekte: Bei der Verwendung von Agenten stellen sich verschiedene rechtliche Probleme, wie die Rechtsgültigkeit ihrer Handlungen oder die Haftung für Schäden durch unerwünschtes Verhalten. In praktischer Hinsicht sind Methoden zur Beweissicherung erforderlich.

Es wird ein in Java erstelltes System für mobile Agenten präsentiert, in welchem diese drei Punkte implementiert wurden. Als Abschluß werden verschiedene Agenten bzw. Systeme von Agenten kurz vorgestellt, anhand derer die Funktionalität des Agentensystems dargestellt wird.

Inhaltsverzeichnis

POND

Ein Agentensystem mit Fokus auf Sicherheit und Bezahlung für Leistungen unter Berücksichtigung rechtlicher Aspekte

1. Einleitung

Das Konzept von intelligenten Agenten ist nicht neu, doch ist sein Potential bisher noch nicht vollständig ausgeschöpft. Was ist überhaupt unter einem „intelligenten Agenten“ zu verstehen? Hier (zur genauen Definition siehe Abschnitt 2.1) soll als Erläuterung genügen, daß es sich um eigenständige Programme handelt, welche eine größere Autonomie als normale Programme besitzen, um Aufgaben selbständig, d. h. ohne dauernde Überwachung, durchzuführen. Aufgaben befinden sich z. B. im Bereich des E-Commerce (etwa im Hinblick auf kooperative Agenten [Hadad/Kraus 1999], zur Unterstützung der Verteilung und des Kundendienstes [Mühlbacher/Sonntag 1999]) oder als Hilfsmittel für Telearbeiter (z. B. [Sonntag 1998]).

1.1. Problemstellung

In der Autonomie von Agenten liegt gleichzeitig eines ihrer Problem: Die gesteigerte Selbständigkeit gegenüber einem Programm, welches lediglich auf Benutzereingabe hin tätig wird, bedingt auch ein erhöhte Fehleranfälligkeit. Es kommt erschwerend hinzu, daß die Kosten für Fehler aufgrund der Bedeutung der Aufgaben steigen. Ein Problem für die Einführung in der Praxis liegt daher darin, Vertrauen in Agenten zu schaffen. Hierzu gehören mehrere Aspekte:

? Vertrauen in die Software: Hier bieten klassische Methoden des Software-Engineering Möglichkeiten, Programmfehler zu minimieren und so die Zuverlässigkeit zu erhöhen. Diese Maßnahmen verhindern ein von Programmierern ungewolltes, zufällig entstehendes Verhalten. Auf diesen Punkt wird in dieser Arbeit nicht näher eingegangen.

? Vertrauen in die Umgebung von Agenten: Agenten sind auf eine Umgebung angewiesen, in welcher sie sich aufhalten, gewissermaßen einer zusätzlichen Betriebssystem-Schicht. Hierbei muß sichergestellt werden, daß Agenten gegen Angriffe von vielen Seiten geschützt sind, wodurch ungewolltes Verhalten verhindert wird, welches durch nicht vorgesehene Einflüsse von außen entsteht. Zur Zeit bestehen verschiedenste Ansätze für Sicherheitssysteme für Agenten, doch handelt es sich meist um punktuelle Elemente, welche nur für einzelne Probleme geeignet sind. Auch die notwendige Verbindung von Portabilität (z. B. Ja-

va) und einem erprobten dynamischen Sicherheitssystem, welches auch Aspekte zur Sicherung von Daten für eine spätere Nachvollziehbarkeit beinhaltet, fehlt zur Zeit.

? Vertrauen in andere Agenten: Agenten sind in vielen Fällen darauf angewiesen, mit anderen Agenten zu kooperieren, obwohl es sich hierbei um „fremde“ Agenten (im Sinne von einem anderen Besitzer) handelt. Es soll sichergestellt werden, daß notwendige Versprechungen (etwa, daß eine bestimmte Leistung erbracht werden wird) komplexere Aufgaben zu erfüllen, auch tatsächlich eingehalten werden. In diesem Fall ist eine Garantie schon nicht mehr möglich. Es muß daher auf einer niedrigeren Stufe angesetzt werden: Durch Informationsaustausch zwischen Agenten soll eine Einschätzung anderer Agenten ermöglicht werden. Dies verhindert ungewolltes Verhalten, welches durch mangelnde Qualität anderer Agenten bei vorgesehenen Tätigkeiten entsteht. Bei den bisherigen Agentensystemen handelt es sich großteils um geschlossene Systeme, bei denen dieses Problem nicht auftritt. Demgegenüber wird in offenen Systemen meist nicht darauf eingegangen: Diese Problematik ist durch den Anwender außerhalb des Systems zu lösen. Zu diesem Punkt gehört auch die Bezahlung von Leistungen: Bei einem Austausch ist naturgemäß eine Partei Vorleistungspflichtig, sofern nicht eine dritte Partei eingeschaltet wird, wobei jedoch beiderseitiges Vertrauen in diese Partei erforderlich ist.

? Rechtliche Aspekte von Agenten: Im Zusammenhang mit autonomen Agenten stellen sich viele Rechtsfragen, wie etwa in Verbindung mit Datenschutz, elektronischen Signaturen, oder der Rechtsgültigkeit von Handlungen von Agenten (sowohl im deliktischen wie auch im vertraglichen Bereich). Diese Fragen sind zur Zeit praktisch ungelöst. Hierdurch soll kein ungewolltes Handeln von Agenten verhindert, aber die Möglichkeit geschaffen werden, die hinter einem Agenten stehenden Personen zu verpflichten bzw. zu belangen. So ermöglicht z. B. erst die Durchsetzbarkeit von durch Agenten abgeschlossenen Verträgen deren Verbreitung im Bereich von E-Commerce.

Ein weiterer wichtiger Aspekt in Verbindung mit Agenten ist die Mobilität: Sie erlaubt es Agenten, sich für die Durchführung bestimmter Aufgaben auf einen anderen Server zu verlegen. Dies bringt verschiedene Vorteile mit sich, doch bedingt es auch zusätzliche Risiken. Eine derartige Mobilität ist z. B. in Verbindung mit PDAs von Bedeutung: Bei diesen Geräten kann nicht von einer dauernden Netzwerksverbindung und einer besonderen Rechenleistung auf mobilen Geräten ausgegangen werden, sodaß eine Verlagerung auf einen anderen Server für die Durchführung der Aufgabe besondere Bedeutung besitzt. Hier erfolgt nur die Konfiguration der Aufgabe und die Ablieferung der Ergebnisse auf dem mobilen Gerät.

1.2. Ziel dieser Arbeit

Es wird neben theoretischen Überlegungen hinsichtlich rechtlicher Aspekte ein komplettes Agentensystem vorgestellt, welches ein umfassendes, dynamisches Sicherheitssystem für autonome und mobile Agenten bietet und in der Programmiersprache Java erstellt wurde. Der Schwerpunkt liegt hier bei allen Aspekten sowohl auf der Sicherheit als auch auf der Beweissicherung, weniger jedoch auf der grundsätzlichen technischen Implementierung. So ist etwa Mobilität in technischer Hinsicht ausführlich untersucht, ebenso wie allgemeine Sicherheitsaspekte, doch eine Verbindung beider Elemente (sowohl theoretisch als auch praktisch), auch in Hinsicht auf die Folgen von Problemen (wie kann ein Schädiger überführt/zur Rechenschaft gezogen werden), fehlt und soll hier dargestellt werden.

Die Funktionalität des Agentensystems wird an Hand von einzelnen kleineren Agenten, über Systeme mehrerer Agenten für komplexere Aufgaben, bis hin zu einem großen System dargestellt. Bei letzterem handelt es sich um ein Webportal für Sport-Informationen, welches sowohl über Webseiten als auch über Agenten bedienbar ist. Die Bedienung über Agenten hat insbesondere den Vorteil, daß etwa fällige Mitgliedsbeiträge direkt von Agenten bezahlt werden können, was über das Web-Interface nicht oder nur schwer möglich ist. Hierbei fließen auch die Überlegungen für die direkte Bezahlung von Leistungen durch Agenten ein. Das Portal für Sport-Informationen und Sport-Clubs wird hier nicht näher beschrieben, da es Gegenstand einer gesonderten Arbeit ist.

Insgesamt soll ein Beitrag zur Förderung von Agentensystemen geliefert werden, besonders für den Bereich des E-Commerce, indem verschiedene Detailprobleme erläutert und Lösungen hierfür dargestellt werden. Diese Probleme umspannen einen größeren Bereich von organisatorischen über implementationstechnische bis hin zu rechtlichen Fragestellungen.

1.3. Aufbau

Die vorliegende Arbeit gliedert sich in folgende Abschnitte:

Im Kapitel 2 werden Agentensysteme im Überblick dargestellt. Allgemeine Fragen wie Definition von Agenten, Einteilung von Agentensystemen, sowie ein kurzer Überblick über Anwendungsgebiete sind enthalten. Anschließend werden mehrere existierende Agentensysteme kurz beleuchtet, insbesondere im Hinblick auf die Sicherheitselemente, welche sie integrieren.

Im Kapitel 3 wird ausführlich auf die Hauptelemente der Arbeit in theoretischer Hinsicht eingegangen: Sicherheit, Bezahlung durch Agenten und rechtliche Aspekte.

Das Kapitel 4 ist dem Agentensystem POND (Logo siehe Abbildung 1) gewidmet. Es wird sowohl die Sicherheitsarchitektur, als auch die Vorkehrungen für Bezahlung, sowie einzelne Elemente im Zusammenhang mit den im vorigen Kapitel erläuterten rechtlichen Aspekten dargestellt. Abschließend werden einige sonstige wichtige Elemente des Systems erläutert.

Abbildung 1: Logo des Agentensystems POND

Die Anwendungen für Agenten als persönliche Assistenten werden im Kapitel 5 dargestellt, wobei eine Gliederung in fünf Bereiche erfolgt und jeweils die Aufgaben, hierbei auftretende Schwierigkeiten, sowie passende Lösungsansätze, insbesondere solche, die mit dem Agentensystem POND möglich sind, dargestellt werden.

Im Kapitel 6 werden implementierte Agenten bzw. Systeme von Agenten kurz im Überblick dargestellt. Diese wurden unter Verwendung des Agentensystems POND programmiert und dienen zur Demonstration einzelner Aspekte bzw. der Gesamt-Funktionalität.

Einige Überlegungen, welche Probleme mit diesem System nicht gelöst werden können, sowie auf welchem Gebiet weitere Forschung bzw. Implementierung möglich oder wünschenswert ist, sind im Kapitel 7 enthalten. Hier erfolgt auch ein Vergleich des Agentensystems POND mit anderen existierenden Agentensystemen.

Im abschließenden Kapitel 8 wird die Arbeit kurz zusammengefaßt und ein Ausblick auf mögliche Entwicklungen bzw. weitere Arbeiten gegeben.

2. Agentensysteme im Überblick

In diesem Abschnitt wird ein allgemeiner Überblick über Agenten und Agentensysteme gegeben, wobei jedoch insoweit schon ein Augenmerk auf das implementierte System gelegt wird, als die hierfür wichtigen Elemente ausführlicher und detaillierter beschrieben werden.

Nach einer Definition, was im folgenden unter einem „Agent“ und anderen wichtigen Begriffen verstanden werden soll und einer Einteilung von Agentensystemen wird zunächst untersucht, warum eigentlich Bedarf danach besteht und welche Aufgaben Agenten übernehmen können, bzw. in Zukunft nicht nur in der Forschung sondern auch in der Praxis übernehmen sollen. Weiters werden aktuelle Entwicklungen sowie bestehende Standards erläutert. Anschließend wird der derzeitige Stand der Forschung an Hand von exemplarischen Agentensystemen dargestellt. Den Abschluß bildet ein kurzes Abriß über die Problemfelder, denen Agenten-Anwendungen ausgesetzt sind.

2.1. Definitionen

Am Anfang muß logischerweise eine Definition dessen stehen, was ein „Agent“ überhaupt ist. Dies ist nicht ohne weiteres möglich, da diese Bezeichnung für sehr viele verschiedene Dinge verwendet wird (und für viele Programme lediglich als Werbebezeichnung dient, welche nur wenige oder keine der grundlegenden Anforderungen erfüllen [Ma 1999]). Es existiert derzeit auch keine allgemein anerkannte Definition ([Greif 1994], [Franklin/Graesser 1996], [Janca/Gilbert 1998], [Foner 1993], [Doberkat 2000]).

2.1.1 Definitionsbeispiele und wichtige Elemente

In der Literatur werden Agenten etwa folgendermaßen definiert:

? „jeder im Auftrag und Interesse eines anderen Tätige“ [Duden 1984]. Diese allgemeine Definition für einen Agenten bezieht sich auf menschliche Agenten, doch kann eine Analogie verwendet werden: Auch Software- oder Hardware-Agenten handeln grundsätzlich im Auftrag und Interesse (da sie sonst nicht existieren würden) anderer Personen (oder rekursiv wiederum für Agenten). Schon aus dieser Definition lassen sich jedoch zwei wichtige Elemente herauslesen: Erstens benötigt ein Agent eine gewisse Autonomie. Er handelt zwar im Auftrag einer Person, jedoch nicht nach exakten Vorgaben und ist daher nicht bloß reines Erfüllungswerkzeug sondern besitzt eigene Kompetenz für Entscheidungen (in geringe-

rem oder größerem Umfang). Zweitens besitzt ein Agent Vorgaben oder Ziele die er erreichen soll. Dieser erstrebenswerte Endzustand kann einerseits fest in der Programmlogik oder der Hardware „verdrahtet“ sein, sodaß er sich implizit ergibt, doch besteht auch die Möglichkeit, daß lediglich eine grobe Spezifikation als Eingabe erfolgt und der Agent selbständig Sub-Ziele definieren und eine Strategie zu ihrer Erreichung bilden muß.

? „software system which *engage and help* all types of end users“ [Riecken 1994]: Hier wird das Augenmerk darauf gelegt, daß ein Agent eine Unterstützung für *verschiedenste* Menschen sein soll, insbesondere auch für Nichtspezialisten und Behinderte. Gleichzeitig wird der Begriff jedoch auf Softwareprogramme eingeschränkt. Hardware-Agenten fallen unter das Gebiet der Robotik und werden daher in dieser Arbeit nicht behandelt.

? Schwehm ([Schwehm 1998]) unterscheidet etwa zwischen einer deutschen (wonach ein Agent ein „Handelnder, also eine im Auftrag und im Interesse eines anderen tätige Person“ ist) und einer Englisch-Amerikanischen (wonach der Agentenbegriff „auch aktive Softwareobjekte und Komponenten mit bestimmten Eigenschaften umfaßt“) Auffassung. Letztere besitzt ein viel größeres Problem bei der Abgrenzung, welche Eigenschaften nun konkret notwendig sind, um einen Agenten zu begründen. Weiters führt sie sehr leicht zu einer „Inflation“ des Agentenbegriffes, wonach selbst einfachste Programmstücke als „Agent“ bezeichnet werden. Seine eigene Definition („Ein Prozeß, der aktiv von einem Rechner zum nächsten migrieren kann, und der sich aufgrund von vor Ort ausgeführten Berechnungen entschließen kann, von dort zu einem weiteren Rechner zu migrieren, wird als mobiler Agent bezeichnet.“) ist jedoch sehr eingeschränkt und betont, wie von ihm selbst ausgeführt, einen einzelnen Aspekt, die Mobilität, zu stark, ohne auf andere wichtige Elemente einzugehen.

? „entities that can communicate in an agent communication language“ (Michael Genesereth in [Milojicic 2000]). Hier handelt es sich um eine Definition, die auf dem Verständnis aufbaut, was eine Agenten-Kommunikationssprache ist (Zirkularität!). Dennoch ist ein wichtiges Element enthalten, da die meisten Agenten nicht alleine sind, sondern mit anderen Agenten zusammenarbeiten. Zu diesem Zweck ist eine gemeinsame Sprache (oder Protokoll) notwendig, welche eine genau definierte (=festgelegte Syntax) Kunstsprache und keine natürliche Sprache (=eindeutig festgelegte Semantik) ist. Ansonsten ist jeder Agent darauf angewiesen, seine Aufgaben komplett selbständig zu erfüllen (im Kontext von Agenten eher weniger sinnvoll), oder zumindest einen Teil einer natürlichen Sprache zu verstehen und sich ebenso darin auszudrücken (was viele Probleme bereitet).

? „computer systems in a complex environment that realize a set of tasks and goals they were designed for“ (Pattie Maes in [Milojicic 2000]). In dieser Definition werden Agenten als komplexe Werkzeuge gesehen (sowohl Soft- als auch Hardware), die für einen genau bestimmten Zweck entworfen werden und nur dafür eingesetzt werden können. Die explizite Erwähnung der Umgebung führt zu einem weiteren wichtigen Element für Agenten: Sie müssen in der Lage sein, ihre Umwelt wahrzunehmen (zumindest in einzelnen für sie wichtigen Teilen) und anschließend durch Handlungen auf diese einzuwirken, sodaß sich darin Veränderungen ergeben, welche (hoffentlich) den Agenten seinem Ziel näher bringen. Es handelt sich bei Agenten daher grundsätzlich um offene Systeme, welche nicht von ihrer Umwelt abgeschirmt werden können. Dies bringt viele Probleme und Schwierigkeiten mit sich, ist aber auch einer der besonderen Vorteile von Agenten.

? „systems capable of autonomous purposeful action in the real world“ (Jose Brustoloni in [Milojicic 2000]). Von dieser Definition sind wiederum sowohl Softwareprogramme als auch Roboter erfaßt. Auch hier wird besonderes Augenmerk auf die Einwirkung auf die Umwelt und die Zielorientierung gelegt. Auch das Element der Autonomie ist vertreten.

? „An agent is a computer program that acts autonomously on behalf of a person or organization. Each agent has its own thread of execution so tasks can be performed on its own initiative.“ [OMG/MASIF] Hier wird besonderes Augenmerk auf die Autonomie und die Proaktivität gelegt und die Definition eher Software-nahe angegeben (eigener Thread). Auffällt, daß nicht angeführt ist, daß ein Agent auch für einen anderen Agenten tätig werden kann. Dies ist zwar nicht ausgeschlossen (wird für einen anderen Agenten gearbeitet, so wird im Endeffekt für *dessen* Besitzer eine Tätigkeit durchgeführt), doch ist dies ein wichtiger Punkt und sollte daher explizit ausgeführt werden.

? „An agent is a computational process that implements the autonomous, communicating functionality of an application. Typically, agents communicate using an Agent Communication Language.“ [FIPA, FIPA abstract architecture specification, document number PC00001C] Auch bei dieser Definition ist der Hauptpunkt die Kommunikationsfähigkeit mit anderen Agenten, doch wird auch auf die Autonomie Wert gelegt. Nur Software-Agenten (Prozesse) werden hier betrachtet.

? „Title I, Sec. 106 (3): Electronic Agent. The term “electronic agent” means a computer program or an electronic or other automated means used independently to initiate an action or respond to electronic records or performances in whole or in part without review or action by an individual at the time of the action or response.“ [US Signature act] Was nun ein

"electronic agent" ist, wird hier näher spezifiziert. Es lassen sich einige Elemente herauslesen: Eine gewisse Autonomie ist vorauszusetzen, da sich sonst rechtlich gar kein Problem stellt. Ein interaktives Programm ist wie ein Werkzeug zu behandeln (z. B. Hammer), und daher immer demjenigen, der es einsetzt, zuzurechnen. Es kommt daher darauf an, daß im Augenblick der Handlung des Programmes zumindest in einem Teil keine direkte Aktion des, bzw. Überwachung durch, den Benutzer erfolgt (echte Autonomie). Es sind sowohl Software- wie auch Hardware-Agenten inkludiert.

2.1.2 Notwendige Eigenschaften von Agenten

Aus dieser (nicht erschöpfenden) Zahl unterschiedlicher Definitionen kristallisieren sich die wichtigsten Elemente heraus, welche jeder Agent in mehr oder minderem Ausmaß besitzen muß, und daher von einem Agentensystem unterstützt oder zumindest ermöglicht werden sollten (siehe Abbildung 2):

? Autonomie: Ein Agent muß in der Lage sein, selbständig zu agieren und selbst Entscheidungen zu treffen. Wie diese herbeigeführt werden, kommt auf die konkrete Implementierung an. Ihm müssen daher für die Erreichung der Ziele mehrere alternative Möglichkeiten offenstehen.

? Reaktivität: Ein Agent muß in der Lage sein, auf Veränderungen seiner Umwelt zu reagieren. Hierzu muß er „Sensoren" besitzen, welche ihm neue Umstände anzeigen. Eine dauernde Überwachung ist nicht erforderlich. Auch Rückmeldungen aus Anlaß von Aktionen genügen, wenn der Agent daraufhin seinen internen Zustand verändert.

? Aktivität: Ein Agent muß in der Lage sein, Umwelt zu beeinflussen. Dazu benötigt er Aktoren, welche verschiedenster Art sein können. Die resultierenden Aktionen müssen dazu führen, daß der von den Sensoren erfaßte Zustand verändert wird (Dies kann in Analogie zur Rückkoppelung einer Regelungsschleife gesehen werden). In dieser Untersuchung erfolgt eine Einschränkung darauf, daß die zu beeinflussende Umgebung rein aus Software besteht und reale Aktoren (Motoren, Anzeigen, etc.) nicht notwendig sind, bzw. durch separate Software (welche mit dem Agenten verbunden ist bzw. mit ihm kommuniziert) angesteuert werden.

? Zielorientierung: Ein Agent benötigt ein übergeordnetes Ziel, welches er durch seine Handlungen verfolgt. Dieses kann aus verschiedensten Elementen bestehen: Aufrechterhalten, Herbeiführen oder auch Verhindern eines bestimmten Zustandes der Umwelt. Über das Ent-

stehen dieser Ziele wird hier nichts vorgegeben. Sie können vom Benutzer explizit angegeben werden, in der Programmlogik enthalten sein, oder auch vom Agenten selbst gebildet werden (Subziele, aber ebenso „emergent behavior“).

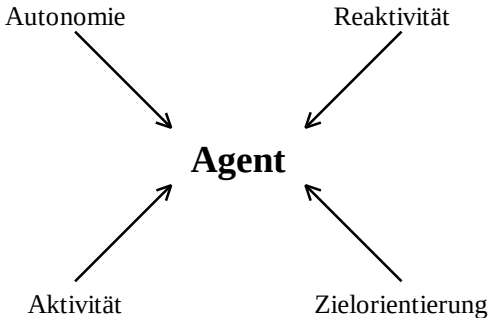

Abbildung 2: Notwendige Eigenschaften von Agenten

2.1.3 Optionale Eigenschaften von Agenten

Zusätzlich dazu existieren noch andere Eigenschaften welche Agenten besitzen können, jedoch nicht müssen (siehe auch Abbildung 4). Von manchen Autoren werden einzelne dieser Elemente noch in die Basisdefinition aufgenommen (z. B. [Jennings/Wooldridge 1996] setzen „Social ability“ voraus, welche hier in Kommunikativität und Interaktivität aufgeteilt ist).

? Proaktivität (Selbsttätige Aktivierung): Agenten können einen internen Zeitgeber besitzen, sodaß sie von sich aus Aktionen ergreifen, ohne daß dies durch eine Benutzereingabe oder eine Veränderung der Umgebung ausgelöst wurde. Dies wird bei den meisten Agentensystemen zumindest möglich sein. Viele Agenten kommen durchaus auch ohne dies aus und werden nur auf externe Anforderungen hin aktiv.

? Kommunikativität: Agenten sind ohne jegliche Kommunikativität denkbar, etwa wenn sie lediglich auf Umweltveränderungen reagieren, indem diese aufgenommen und umgekehrt wieder darauf eingewirkt wird (=Kommunikation über die Umwelt). In vielen Fällen werden Agenten jedoch in der Lage sein müssen, direkt mit anderen Agenten zu kommunizieren (Inter-Agenten-Kommunikation): Einzelne Agenten bringen keine besonderen Vorteile, erst die Kombination mehrerer zu einem System von Agenten läßt die meisten charakteristischen Vorteile entstehen.

? Interaktivität: Neben dieser internen Kommunikation wird es für viele Agenten notwendig sein, direkt mit Benutzern zu interagieren (Agenten-Benutzer-Kommunikation; externe Kommunikation). Insbesondere ist dabei zu beachten, daß dies in allgemein verständlicher Form geschehen sollte (z. B. über graphische Schnittstellen; siehe [Fricke et al. 2001] für

ein Beispiel zur graphischen Konfiguration von Agenten wie auch graphischen Darstellung von Marktplätzen), sodaß Benutzer ohne spezielle Schulung mit ihnen umgehen können (keine spezielle Mediation durch Agenten-Spezialisten oder andere Systeme nötig). Dies gilt sowohl für die Ausgabe von Ergebnissen oder Meldungen (relativ einfach) wie auch die Entgegennahme von Aufträgen (einfach bis sehr schwierig). Hierunter wird die direkte Kommunikation zwischen Benutzern und Agenten verstanden. Liegt dazwischen noch ein größeres System (z. B. Ausführung der vom Agenten generierten Aufträge), so wird für diese Agenten keine Interaktivität angenommen, da die Ausgabe nicht direkt dem Benutzer zugeführt wird (siehe Abbildung 3).

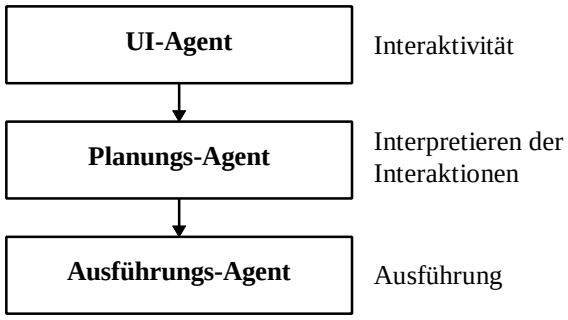

Abbildung 3: Benutzerinteraktions-Hierarchie von Agenten

? Robustheit: Dies kann auch als ein Unterpunkt der Autonomie betrachtet werden, doch ist in vielen Fällen Robustheit genau *der* oder zumindest *ein* wichtiger Hauptinhalt ([Schwehm 1998]) der Eigenständigkeit von Agenten: Kein komplexer Plan funktioniert so, wie er geplant war, sondern es treten immer wieder Fehler und Schwierigkeiten auf. Und genau diese Probleme zu überwinden oder zu umgehen ist vielfach die Hauptaufgabe von Agenten. Es ist einfach ein Softwareprogramm eine Folge von genau spezifizierten Aktionen (wenn auch komplexer Natur) durchführen zu lassen („Schönwetter-Code“). Jedoch alle möglichen Probleme zu erkennen und für sie vorauszuplanen ist sehr aufwendig. Sinn von Agenten ist es hierbei, vielleicht nicht die optimale Umgehungs- oder Problembehebungs-Strategie zu finden, jedoch eine Möglichkeit zur Zielerreichung zu identifizieren und schließlich durchzuführen. Der Verlust an Effizienz wird dadurch wettgemacht, daß eben nicht alle Probleme explizit im Voraus beachtet werden müssen (Abtausch von Entwicklungsaufwand gegen eine geringere Laufzeiteffizienz bei Eintritt von Ausnahmefällen). Es sollen jedoch keine genaue Umgehungen für einzelne Probleme programmiert werden, sondern Regeln, mit denen der Agent dies selbst erledigen kann, und welche auch auf ähnliche, aber eben nicht exakt gleiche, Schwierigkeiten anwendbar sind.

? Mobilität: Ein großer Anteil von Agenten fällt in die Klasse der mobilen Agenten [Kotz/Gray 1999]. Dies bedeutet daß Software-Agenten in der Lage sind, sich *selbsttätig* (durch eigene Entscheidung) von einem Computer auf einen anderen (von ihnen oder dem Benutzer ausgewählten) zu begeben, wenn ihre Aufgabe dies erfordert oder es sie erleichtert. Dies bedeutet immer den Transport des derzeitigen Programmzustandes. Doch in manchen Fällen werden auch das zugehörige Programm und andere Ressourcen mittransferiert, sodaß sich insbesondere die Verteilung von Programmen (bzw. das Einspielen neuerer Versionen) erübrigt. Man versucht dadurch eine größere Lokalität zu erreichen und so etwa die benötigte Bandbreite für Datentransporte zu verringern (was einerseits gelingt, siehe etwa [Theilmann/Rothermel 2000], aber nicht immer auch tatsächlich eintreten muß; siehe [Spalink et al. 2000] für ein Gegenbeispiel).

? Intelligenz: Dies ist ein wichtiger Punkt, da ohne ihn viele der anderen Elemente (insbesondere Autonomie, Robustheit, Zielorientierung) nicht oder nur in geringem Maße verwirklicht werden können. Aus diesem Grund wird daher meist nicht von „Agenten“ sondern von „Intelligenten Agenten“ (IA) gesprochen. Das Ausmaß der notwendigen Intelligenz ist jedoch nicht klar festzulegen: Von komplexeren Programmen über regelbasierte Systeme bis hin zu Expertensystemen ist alles möglich. Manchmal werden Agenten auch grundsätzlich als „Intelligente Agenten“ definiert (so etwa in [Jennings/Wooldridge 1998]), sodaß für einen „einfachen Agenten“ nur mehr wenig bis kein Raum verbleibt. Beispiele für einfache Agenten sind dann: OS-Dämons oder Prozess-Steuerungs-Systeme. Zu beachten ist auch, daß sich größere Intelligenz meist negativ auf viele andere Elemente auswirkt: Agenten mit integriertem Expertensystem sind kaum mehr mobil, da große Mengen an Daten und Programmen transferiert werden müßten. Auch leidet die Reaktionsfähigkeit, da Planung, Zielentwicklung und Suche nach Implementationsalternativen sehr viel Rechenzeit benötigt. Oft setzt man daher auf emergent behavior, um aus einer größeren Menge relativ unintelligenter Agenten durch deren Zusammenwirken und ihre Interaktionen neues Verhalten zu gewinnen, welches mit einem höheren Grad an Intelligenz beschrieben werden kann.

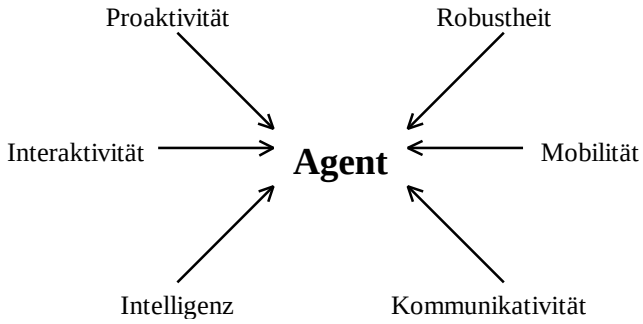

Abbildung 4: Optionale Eigenschaften von Agenten

2.1.4 Agentensysteme (=Basissystem, Framework)

Agenten benötigen in allen Fällen ein zugrundeliegendes System auf welches sie aufbauen. Dies kann im einfachsten Fall direkt das Betriebssystem eines Rechners sein. Fast immer jedoch wird zusätzliche Software benötigt werden, welche weitere und spezialisierte Dienste zur Verfügung stellt. Insbesondere bei mobilen Agenten ist dies zur Zeit unabdingbar.

Dieses Basissystem kann als eine neue Betriebssystem-Schicht gesehen werden und sollte daher auch nach entsprechenden Grundsätzen entworfen werden. Ebenso ist eine direkte Einbettung in ein solches möglich.

Im folgenden wird hierfür die Bezeichnung „*Agentensystem*“ oder „Agenten-Framework“ verwendet, was der englischen Diktion von „Agentsystems“ entspricht. Im Gegensatz dazu handelt es sich bei den Applikationen, welche aus mehreren Agenten bestehen, um „*Systeme von Agenten*“ bzw. um einzelne „*Agenten*“. Für eine Darstellung im Verhältnis zu Programmen und dem Betriebssystem siehe Abbildung 5.

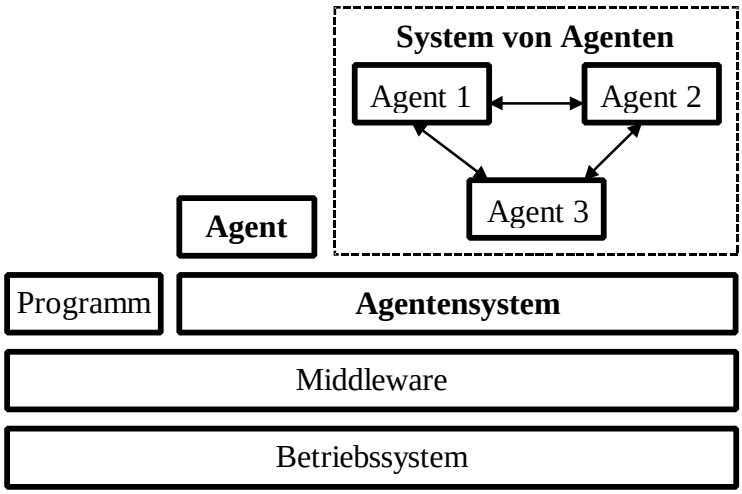

Abbildung 5: Agentensystem ? Agent ? System von Agenten

2.1.5 Definition im Rahmen dieser Arbeit

In dieser Arbeit sollen nicht alle Aspekte von Agenten behandelt werden, sondern nur ein wichtiger Ausschnitt. Für die folgenden Ausführungen wird daher unter einem „Agenten“ folgendes verstanden:

> *Ein Agent ist ein Softwareprogramm, welches aus seiner Umgebung Informationen aufnimmt und auf Grund dieser und seinem internen Zustand auf seine Umwelt einwirkt. Es erhält von einem Benutzer (direkt oder indirekt über andere Agenten) Ziele vorgegeben, welche es auf mehrere Arten umsetzen kann, wobei die Entscheidung für den konkreten Weg ihm überlassen ist. Zu diesem Zweck ist es in der Lage, mit Benutzern und anderen Agenten zu kommunizieren.*

Die Elemente (siehe auch Abbildung 6) dieser Definition sind:

1. Nur Software-Agenten (keine Roboter)
2. Reaktivität (Sensoren für die Umwelt)
3. Aktivität (Einwirkung auf die Umgebung)
4. Proaktivität (Auch Zustandsänderungen ohne äußeren Einfluß möglich)
5. Autonomie (Auswahlmöglichkeiten und selbständige Selektion des Durchführungsweges, bzw. das Finden von Teil- oder Zwischenzielen durch den Agenten selbst)
6. Zielorientierung (Keine festen Ziele sondern Konfiguration durch den Benutzer)
7. Kommunikativität (Agenten können untereinander kommunizieren; eine einzige gemeinsame Sprache für alle Agenten wird jedoch nicht festgelegt)
8. Interaktivität (Entgegennahme von Zielen und Präsentation der Ergebnisse direkt beim Benutzer)

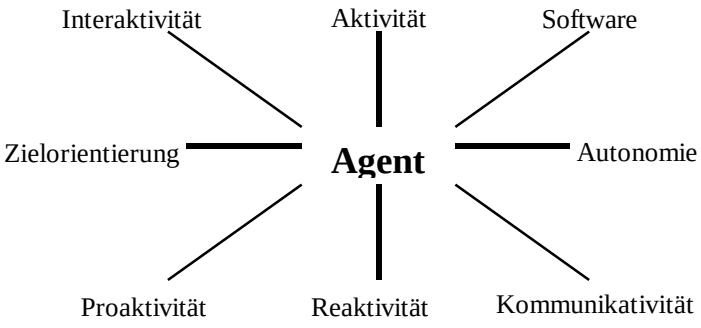

Abbildung 6: Definitionselemente eines Agenten

Aus dieser Definition läßt sich erkennen, daß die Grenze zwischen einem Agenten und einem Agentensystem nicht immer scharf zu ziehen ist: Nicht jeder Agent führt auch tatsächlich interne Zustandsänderungen durch und bleibt doch ein Agent, da es für ihn auf Grund des Basis-Systems, in welches er eingebettet ist, zumindest möglich wäre. Die Definition ist daher ein System beweglicher Elemente und schon die konkrete Möglichkeit (ohne tatsächliche Implementierung), ein einzelnes Element zu erfüllen genügt, wenn die anderen Punkte entsprechend stärker ausgeprägt sind. Es ist daher immer eine Gesamtschau maßgeblich.

Nicht explizit gefordert in dieser Definition ist die Mobilität. Sie spielt zwar eine große Rolle, doch sind viele Agentensysteme (auch komplexe und praktische Anwendungen) stationär und sollen nicht alleine deshalb ausgeschlossen werden. Handelt es sich um einen Agenten im Sinne obiger Definition und ist zusätzlich die Möglichkeit gegeben, daß der Agent von sich aus seine Tätigkeit auf einen anderen Rechner verlagern kann wobei sein interner Zustand erhalten bleibt, so sprechen wir von einem mobilen Agenten.

Die Robustheit wird hier als Unterpunkt der Autonomie angesehen. In manchen Anwendungen spielt sie keine so große Rolle und soll daher nicht als gesondertes Kriterium angesehen werden. Weiters ist eine gewisse Grund-Robustheit ein allgemeines Kriterium für Software und muß daher immer gegeben sein.

Bei der Zielorientierung ist zu beachten, daß es sich bei dieser Definition um eine (direkte oder indirekte) Einflußmöglichkeit des Benutzers auf die Aktionen des Agenten handelt. Hiervon zu unterscheiden ist die interne Zielfindung durch den Agenten, welche der Autonomie zuzuordnen ist. Agenten sind daher immer für verschiedene (wenn auch verwandte) Zwecke einsetzbar. Software, welche für ausschließlich einen einzigen engen Zweck verwendbar ist (d. h. nur Eingabe möglich, jedoch keine Konfiguration), wird für diese Arbeit nicht als Agent angesehen. Eine genaue Abgrenzung ist naturgemäß besonders schwierig, da jede Benutzereingabe als (von anderen Eingaben verschiedener) Verwendungszweck angesehen werden kann.

Ein Agent muß in der Lage sein, sowohl direkt aus seiner Umgebung (Reaktivität) als auch von Benutzern (Interaktivität) oder Agenten (Kommunikativität) Daten entgegenzunehmen und auch wieder an diese zu übermitteln bzw. Handlungen zu setzen (Aktivität).

Es werden keine besonderen Anforderungen an die Intelligenz gestellt. Ist ein Agent in der Lage, autonom gewisse Entscheidungen zu treffen, so soll dies schon ausreichen. Besondere Anforderungen, insbesondere in Hinsicht auf das Forschungsgebiet der künstlichen Intelligenz

(AI), werden nicht gestellt und die Agenten daher bei Erfüllung dieser Mindestanforderungen noch nicht als intelligente Agenten bezeichnet, sondern erst bei darüber hinausgehenden Fähigkeiten. Zu beachten ist jedoch, daß im Sinne der Definition in [Jennings/Wooldridge 1998] alle Agenten nach dieser Definition Intelligente Agenten sind („Intelligenz“ im Sinne der dortigen Kriterien wird durch folgende Punkte erfüllt: Reaktivität, Aktivität, Proaktivität, Kommunikativität).

2.1.6 Abgrenzungen

Zur Verdeutlichung der Definition von Agenten sind auch Abgrenzungen nützlich, da durch diese die einzelnen Elemente besser dargestellt und oftmalige Verwechslungen vermieden werden können.

Prozesse: Agenten werden zwar notwendigerweise durch Prozesse ausgeführt, doch erfüllen diese selbst viele der geforderten Kriterien nicht. Sie sind eines der Grundkonzepte, auf denen Agenten aufbauen. Insbesondere bei der Mobilität sind Agenten von Prozessmobilität zu unterscheiden ([Schwehm 1998]): Agenten entscheiden selbst, wann und wohin sie sich verlagern (und lösen dies explizit aus) während bei verteilten Betriebssystemen dieses selbst die Entscheidung übernimmt (und der Transport für den Prozeß selbst unbemerkt bleibt, was beim Agenten oft nicht der Fall ist; siehe auch 2.2.2).

Objekte: Von Objekten unterscheiden sich Agenten dadurch, daß bei Objekten Methoden einfach aufgerufen werden: Das Objekt hat hier keine Mitsprache. An Agenten hingegen werden Anforderungen gesandt, welche der Agent auch ablehnen kann: Keine Zeit, nicht autorisiert, nur gegen Bezahlung, etc. Agenten besitzen also die Möglichkeit, über sich selbst zu bestimmen, während Objekte rein passiv sind [Wooldridge/Ciancarini 2001].

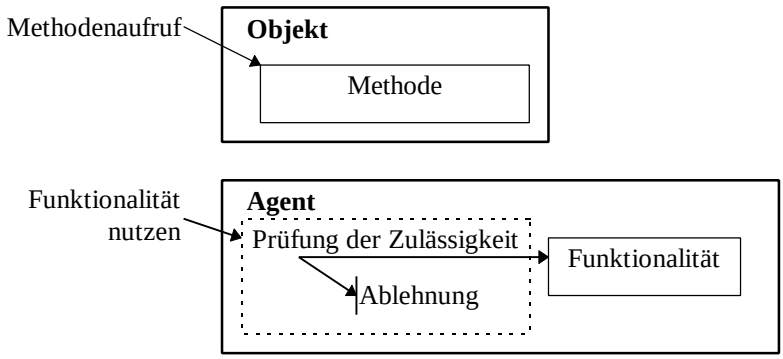

Abbildung 7: Unterschied zwischen Objekt und Agent

Applets (siehe [Applets]; gleiches gilt für Active-X Controls [Active-X]; beide wurden auch für andere Zwecke entworfen): Sie können zwar Agenten sein, unterscheiden sich aber meistens dadurch, daß sie nur geringe Autonomie besitzen (die Codemenge ist wegen der Bandbreite durch die für Benutzer gerade noch akzeptable Downloadzeit beschränkt) und daß die möglichen/erlaubten Einwirkungen auf die Umwelt (ebenso wie die Aufnahme von Informationen daraus) äußerst begrenzt sind (sofern es sich nicht um signierte und als vertrauenswürdig eingestufte Programme handelt). Ebenso läßt sich Kommunikation zwischen mehreren Applets nur schwer realisieren. Mobilität ist nicht gegeben, da lediglich Code transportiert wird und dies auch nur explizit auf Anforderung des Benutzers und nur in eine Richtung (Server ? Benutzerrechner) erfolgt und nicht auf Entscheidung des Applets bzw. Active-X Controls hin.

Expertensysteme: Diese bieten zwar eine im Gegensatz zu den meisten intelligenten Agenten überragende Intelligenz, doch fehlt meist jegliche direkte Verbindung zur Umwelt: Daten werden von Benutzern eingegeben und anschließend erfolgt eine Ausgabe. Je nach Implementierung können Autonomie und Kommunikativität sehr wohl vorhanden sein, im Gegensatz zur Mobilität, welche praktisch nie vorkommt. Es ist zu beachten, daß Expertensysteme oft ein wichtiger Teil oder überhaupt die Grundlage von Agenten oder Systemen von Agenten sind, diesen jedoch nicht gleichgesetzt werden dürfen.

Personal Digital Assistants (PDAs): Da es sich hier um Hardware handelt, können PDAs selbst keine Agenten sein. Auch bei Reaktivität und Aktivität würden sich Probleme ergeben, da Sensoren für die Umwelt nur sehr selten vorhanden sind bzw. nur einen stark eingeschränkten Bereich abdecken. Insbesondere ist während des Großteils der Zeit kein Netzwerksanschluß gegeben.

In einer zweiten Dimension besitzen diese Geräte jedoch eine besondere Bedeutung für intelligente Agenten: Sie sind ein wichtiger Anwendungsfall von Plattformen, auf denen Agenten erzeugt werden könnten. Diese erhalten ihre Aufgabe zugewiesen, wandern an einen anderen Ort und beginnen dort ihre Arbeit. In der Zwischenzeit kann die Netzverbindung des PDAs unterbrochen werden. Ist der Agent fertig, so hat er selbständig festzustellen, ob/wann das Gerät wieder verfügbar ist, wo es sich befindet (Ortswechsel möglich!) und sich zur Ergebnispräsentation wieder zurückzuverlagern. Hierfür sind sowohl Autonomie, Aktivität, Proaktivität, Reaktivität, als auch Mobilität notwendig.

Verschiedenes: Abschließend sollen noch einige Programme bzw. Systeme dargestellt werden, welche entweder offiziell oder im allgemeinen Sprachgebrauch als „Agent“ oder „Agenten-basiert“ bezeichnet werden, jedoch die Anforderungen der Definition nicht erfüllen.

? Microsoft Hilfe-Agenten (Office-Assistenten) [Office-Agents]: Es handelt sich hierbei um graphische „Personen“, welche als zusätzliches Hilfsmittel dem Benutzer zur Verfügung stehen. Sie dienen einerseits als Schnittstelle zur herkömmlichen Hilfe (Suchen, Anzeige der Ergebnisse), andererseits beobachten sie auch den Benutzer und geben Hinweise zu effizienterer Programmbenutzung oder wie (vermutlich gewünschte) Aktionen durchzuführen sind. Als Agenten im Sinne obiger Definition können sie deshalb nicht klassifiziert werden, da sie (bis auf geringfügige Ausnahmen) nicht auf ihre Umgebung einwirken können (mangelnde Aktivität). Insbesondere fehlen jedoch auch Zielorientierung und Autonomie: Die Agenten sind fest programmiert und bis auf zufallsgesteuerte Hinweise gibt es keine Alternativen. Auch ist eine genauere Festlegung der Tätigkeit nicht möglich, lediglich einige wenige Gruppen von Hinweisen können ausgeschaltet werden. Dennoch enthalten sie ein wichtiges Element von Agenten: Sie werden als eine eigene „Person“ dargestellt, welche Gefühle besitzt (Verwunderung, Langeweile, ...). Nur dann werden sie vom Benutzer auch als glaubhafte und vertrauenswürdige Beauftragte (=“Agenten“) akzeptiert ([Bates 1994]). Die graphische Darstellung ist auch deshalb von besonderer Wichtigkeit, da sie dann vom Benutzer leichter zu bedienen sind (siehe dazu [Hayes-Roth et al. 1999]) und ihr Zustand auch schneller und exakter festgestellt werden kann. Und zwar auf einen Blick: Der Mensch ist trainiert, Gefühle von Gesichtern sehr schnell abzulesen.

? Search-Agent (=Suchmaschine): Wie von Ma beobachtet ([Ma 1999]) und auch von Wooldridge und Jennings in [Wooldridge/Jennings 1998] und [Wooldridge/Jennings 1999] ausgeführt („7.1 You see agents everywhere“), ist eine der Gefahren im Hinblick auf Agenten, daß *alles* als „Agent“ bezeichnet wird. Als Beispiel hierfür können insbesondere Suchmaschinen im Internet dienen, welche oft auch als „Such-Agenten“ bezeichnet werden. Es fehlen jedoch Aktivität (reine Ausgabe des Ergebnisses), Proaktivität (Tätigkeit nur auf Anfrage hin; Automatische Indizierung der Webseiten ist eine Daueraktivität ohne Entscheidungskompetenz), Kommunikativität (außer es werden Daten mit anderen Suchmaschinen ausgetauscht oder es handelt sich um eine echte Meta-Suchmaschine) und ganz besonders fehlt die Autonomie: Die Suchmaschine soll exakt nach dem gefragten suchen und keine eigenen Entscheidungen treffen, was man vielleicht gerne als Ergebnis hätte.

2.2. Einteilungen von Agentensystemen

Agentensysteme bzw. einzelne Agenten und/oder Systeme von Agenten lassen sich auf viele verschiedene Arten einteilen. Dies ist einerseits aus den Elementen der Definition bedingt (z. B. Simpel - Intelligent, oder Stationär - Mobil), andererseits aus dem Entwurf des Agentensystems (Einzelagenten - Multi-Agentensystem; Gleichordnung - Hierarchie). Weitere Einteilungen sind nach der Art der Implementierung der „Intelligenz“ möglich, wie etwa deliberative, reaktive oder hybride Systeme. Schließlich kann auch eine Einteilung nach der Art der Zusammenarbeit zwischen den Agenten erfolgen.

Zu beachten ist, daß der Großteil der folgenden Punkte Eigenschaften von Agenten bzw. Systemen von Agenten sind. Im Hinblick auf Agentensysteme selbst kann daher eine Einordnung danach nur über die Unterstützung hierfür (oder bereits eingebaute Funktionen) erfolgen.

Hier erfolgt nur eine Übersicht über die wichtigsten Klassifikationsmöglichkeiten (Abbildung 8), welche regelmäßig verwendet werden (z. B. Simpel-Intelligent [Brenner et al. 1998], Stationär–Mobil [Hayzelden et al. 1999], Einzelagenten-Multi-Agenten-Systeme [Jennings/Wooldridge 1998], Reaktiv-Deliberativ-Hybrid [Klusch 1998], Kooperativ-Wettbewerb [Guttman/Maes 1998]).

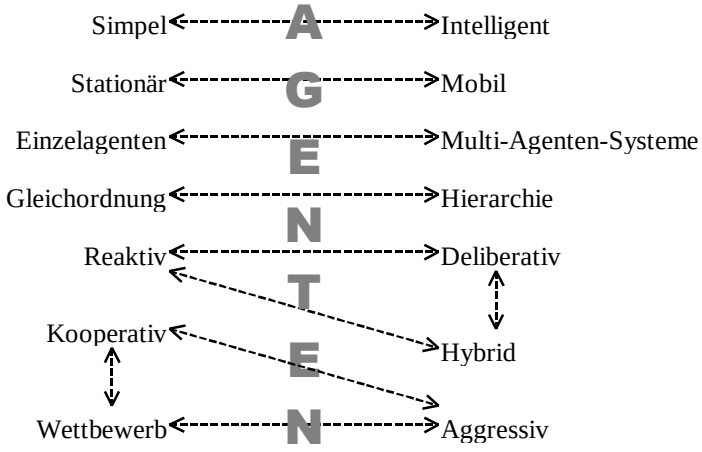

Abbildung 8: Klassifikationsmöglichkeiten

2.2.1 Simpel – Intelligent

Ein Klassifikation kann nach der Intelligenz von Agenten erfolgen, welche von simpel bis intelligent reichen kann ([Brenner et al. 1998]). Intelligenz drückt sich hauptsächlich in den folgenden Elementen der Definition aus, wobei der Schwerpunkt jedoch auf dem letzten Punkt, der Autonomie liegt:

? Reaktivität: Ein Agent muß auf Zustandsänderungen in seiner Umwelt reagieren. Dies muß einerseits sehr oft rasch erfolgen („Reflexe“), andererseits erfordert es manchmal genaue Planung, da bisherige Ziele unerreichbar werden und andere oder neue zu finden sind. Hier besteht ein großes Problem darin zu entscheiden, welche dieser beiden Reaktionen erfolgen soll (beides gleichzeitig durchzuführen ist meist nicht möglich). Werden Reflexe benötigt, so müssen fertige (und meist kurze) Programme ablaufen, welche die notwendigen Aktionen setzen. Diese auszuwählen ist relativ leicht und kann rasch erfolgen. Werden hingegen Plan-Entscheidungen verlangt, so sind komplexe Abläufe notwendig. Es müssen Alternativen erzeugt und bewertet werden, und eventuell ist auch Kommunikation mit anderen Agenten oder zusätzliche Informationssammlung notwendig. All dies kann nicht in kurzer Zeit geschehen und führt meist zu Ergebnissen, welche den ausgelösten Reflexen widersprechen. Ansonsten könnte ja mit diesen Reflexen alleine das Auslangen gefunden werden. Dieser Fall tritt jedoch eher selten auf und die Schwierigkeit ist zu entscheiden, wann dies der Fall ist und daher einstweilen nur vorläufige Aktionen zu setzen sind, welche das (vermutliche) Ergebnis nicht vereiteln werden.

? Proaktivität: Hier stellt sich das unter Reaktivität dargestellte Problem bei weitem nicht so stark, da schon beim Design darauf geachtet werden kann, den Entscheidungsprozessen genügend Zeit zu geben. Dies stellt daher kein Hindernis für größere Intelligenz dar.

? Autonomie: Hier ist der Hauptsitz der Intelligenz. Ein Agent besitzt mehrere Handlungsalternativen und muß eine davon auswählen. Hierzu muß eine Bewertung erfolgen, welche entweder fest vorgegeben sein kann oder durch Lernen aus früheren Aktionen gewonnen wird. Dies kann einstufig (Ziele werden durch einzelne Aktionen erreicht) oder mehrstufig erfolgen. Im letzteren Fall ist hohe Intelligenz nötig, da der Agent selbst ein oder mehrere Zwischenpunkte finden muß, welche durch (autonome) Aktionen erreicht werden können, um von diesen aus dann wieder neue Aktionen zu starten. Dies bedingt eine genaue Planung der Aktionen, resultiert dafür aber in einer größeren Flexibilität.

Zusammenfassend kann gesagt werden, daß viele Agenten bzw. Systeme von Agenten eher unintelligent sind, da das Ausmaß an Intelligenz proportional zum dafür notwendigen Aufwand ist und insbesondere höheren Laufzeit- und Speicheraufwand bedingt, größeren Code produziert, und aufwendigeres Design benötigt. Eine weitere Schwierigkeit ist auch, daß genaues und detailliertes Wissen über das konkrete Problem nötig ist, welches erfaßt und dem Computer zugänglich gemacht werden muß. Dies ist schon für sich ein hoher Aufwand und hat den

großen Nachteil, daß Agenten dadurch Spezialisten werden und eine Anpassung an andere Probleme schwieriger wird (geringere Universalität).

2.2.2 Stationär – Mobil

Agenten können auch danach unterschieden werden, ob sie stationär sind (auf den Rechner beschränkt, auf dem sie erzeugt wurden) oder mobil (eigenständige Verlagerung auf andere Rechner möglich). Bei mobilen Agenten wiederum kann unterschieden werden nach schwacher und starker Mobilität ([Vigna 1998]). Bei schwacher Mobilität werden Programmcode und Daten übertragen, der Agent ist jedoch selbst dafür zuständig, die Arbeit an der passenden Stelle wiederaufzunehmen. Im Gegensatz dazu ist bei starker Mobilität für die Verlagerung auf einen anderen Rechner ein normales Statement zuständig. Bei dessen Aufruf wird der komplette Agent transferiert und am Zielrechner die Abarbeitung automatisch beim folgenden Kommando fortgesetzt. Es müssen daher z. B. auch der Stack und lokale Variablen wiederhergestellt werden und nicht nur der Heap. Hierbei ist noch zu unterscheiden ([Cugola et al. 1996]), auf welche Weise Daten transferiert werden: Vollständig (replication strategy) oder nur als Referenz auf verbleibende (z. B. immobile) Daten (sharing strategy).

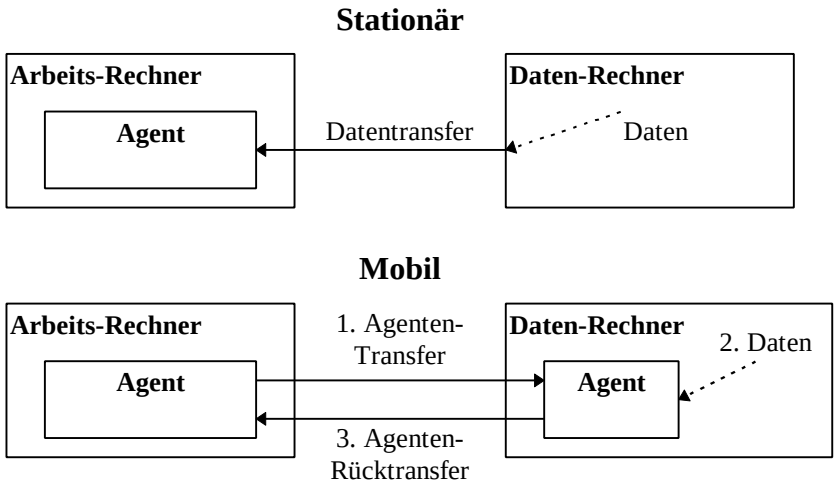

Abbildung 9: Stationäre vs. mobile Agenten

Von mobilen Agenten erwartet man sich insbesondere diese Vorteile [Lange/Oshima 1999]:

1. Verringerung der Netzwerkbelastung: Die Verarbeitung von Informationen wird möglichst nahe an die Informationsquelle herangebracht, um den Transfer der Daten über Netzwerke zu verringern. Da typischerweise das Ergebnis kleiner ist als die zu verarbeitenden Grunddaten, ergibt sich ein Netto-Gewinn.

2. Vermeidung von Netzwerks-Verzögerungen: In Realtime-Systemen kann die Transferzeit über Netzwerke (insbesondere bei hoher Belastung) ein Problem darstellen. Mittels mobilen Agenten kann dies vermieden werden und trotzdem eine zentrale Steuerung erfolgen.
3. Kapselung von Protokollen: In verteilten Systemen werden viele verschiedene Protokolle verwendet. Mittels Interface-Agenten kann auch auf Rechnern, welche ein bestimmtes Protokoll nicht verstehen, dieses verwendet werden. Weiters ist der Wechsel auf ein anderes Protokoll einfacher, da lediglich ein neuer Agent entsandt werden muß.
4. Asynchrone und autonome Ausführung: Besonders bei mobilen Geräten ist die Netzwerksanbindung schmalbandig und anfällig für Störungen bzw. Unterbrechungen. Mit mobilen Agenten kann die Konfiguration auf dem portablen Gerät erfolgen, während die eigentliche Ausführung dann auf einem stationären Rechner mit stabilem und breitbandigem Netzwerksanschluß durchgeführt wird.
5. Dynamische Anpassung: Mobile Agenten können sich an ihre Ausführungsumgebung anpassen. Bei einem System von mehreren Agenten (MAS, siehe 2.2.3) kann auch eine gleichmäßige Lastverteilung auf mehrere Rechner erfolgen.
6. Natürliche Heterogenität: Nachdem mobile Agenten einen eigenen Service-Layer benötigen, sind sie relativ unabhängig vom Träger-System (Plattform/Betriebssystem). Bei Verwendung einer portablen Sprache können sie auf jeder Hard- oder Software-Plattform eingesetzt werden.
7. Robustheit und Fehler-Toleranz: Treten auf einem Rechner Probleme auf (oder wird dieser abgeschaltet), so können mobile Agenten ihre Tätigkeit auf einen anderen Rechner verlagern, sodaß ihre Arbeit nur mit kurzer Unterbrechung (=Transferzeit) fortgesetzt wird.

Mobilität von Agenten bringt jedoch insbesondere *ein* großes Problem mit sich: Sicherheit. Bei stationären Agenten ist dies keine Schwierigkeit, da sie (wie normale Programme) von einem Benutzer lokal gestartet werden müssen und daher dessen Berechtigungen unterliegen. Weiters kann davon ausgegangen werden, daß der Rechner selbst (bzw. dessen Administrator) den Agenten nicht beschädigen wird. All dies kann bei mobilen Systemen nicht von vornherein ausgeschlossen werden, insbesondere nicht bei offenen Systemen. Die Problemgruppen sind folgende (siehe [Gray et al. 1998] für die ersten vier Probleme und Abhilfemöglichkeiten; siehe Abbildung 10 für eine graphische Darstellung. Die Zahlen entsprechen den erläuterten Punkten):

1. Schutz des Rechners vor Agenten: Agenten müssen in ihren Möglichkeiten eingeschränkt werden, sodaß sie weder den Rechner beschädigen können noch Ressource zu stark nutzen. Auch das Ausspähen von vertraulicher Information muß verhindert werden. Der Standard-Ansatz besteht in einer Authentikation des Besitzers des Agenten und anschließender Vergabe entsprechender Rechte. Andere Ansätze bestehen darin, den Code prüfbar zu machen, sodaß eine einfache Prüfung oder sogar Beweis der Unschädlichkeit möglich ist ([Java-Code], [Necula/Lee 1998]).

2. Schutz der Agenten vor dem Rechner: Ist ein Agent einmal auf einem Rechner angelangt, so hat dieser volle Kontrolle über ihn. Er kann ihn jederzeit beenden, die gesamte externe Kommunikation kontrollieren, und besitzt auch Zugriff auf die gesamten mitgeführten Daten. Lediglich Daten, welche verschlüsselt sind und bei denen der Schlüssel nicht vorhanden ist, sind sicher (doch dann auch für den Agenten nicht verwendbar!). Für ein System/Protokoll zur Entdeckung von Modifikationen des Programmablaufs siehe [Vigna 1998]. Hierbei bestehen jedoch zwei grundsätzliche Probleme: Erstens ist es nicht möglich eine Veränderung der Eingabewerte zu erkennen (Modifikation z. B. während des Empfangsvorgangs vom Netzwerk), und zweitens ist keine vereinfachte Prüfung möglich. Um festzustellen bzw. zu überprüfen ob eine illegale Veränderung erfolgte, muß der gesamte Ablauf des Agenten nachvollzogen werden (Zweit-Ablauf des vollständigen Programms), sodaß die Arbeit gleich lokal erfolgen hätte können. Ein anderer Ansatz ist die Verschleierung der Programmlogik und der Daten durch Umordnung von Speicherbelegung und Instruktionen. Dies ist jedoch keine Garantie und besitzt ein Ablaufdatum, ab dem mit einer Entschlüsselung gerechnet werden muß ([Hohl 1998]).

3. Schutz der Agenten vor einander: Da Agenten von verschiedenen Benutzern stammen können, müssen sie auch vor einander geschützt werden. Dies kann als Unterpunkt des Schutzes des Rechners vor Agenten gesehen werden: Ist im Agenten-Kommunikationssystem kein Fehler enthalten und kann der Agent das Sicherheitssystem des Rechners nicht umgehen, so sind auch andere Agenten vor Beeinflussung sicher.

4. Schutz einer Gruppe von Rechnern: Konsumiert ein Agent auf einem Rechner nur geringe Ressourcen, so kann er durch oftmaligen Wechsel einer ganzen Rechner-Gruppe dennoch Schwierigkeiten bereiten. Eine andere Möglichkeit ist die Erzeugung von sehr vielen Sub-Agenten, welche auch wieder als Einzelne nur wenige Ressourcen verbrauchen. Die Limitierung des Verbrauchs muß sich daher über mehrere Rechner und Agenten erstrecken.

Dies ist besonders schwierig und ist z. B. analog zur Sicherung des Internets vor Viren/Würmern (? Schneeballsystem) zu sehen.

5. Schutz von Rechnern untereinander: Die beteiligten Rechner müssen voreinander geschützt werden. Dies ist jedoch kein spezielles Problem von mobilen Agenten. Das Basissystem für Agenten ist lediglich ein *weiterer* möglicher Angriffspunkt und muß daher entsprechend gesichert und auf Fehler überprüft werden.

6. Schutz einer Gruppe von Agenten: Wie eine Gruppe von Rechnern geschützt werden muß, so sind auch Agenten-Gruppen zu schützen. So kann etwa durch eine unterschiedliche Behandlung von zusammengehörenden Agenten das Gesamt-Ergebnis beeinträchtigt werden. Eine andere Alternative ist die Erhöhung des Preises für Ressourcen um einen kleinen Betrag. Dies hat praktisch keine Auswirkung für einen einzelnen Agenten, kann jedoch bei mehreren zusammen ein Problem darstellen. Auch eine unterschiedliche Preisgestaltung (bestimmte Ressourcen verteuern und andere verbilligen) kann hier zu Diskriminierungen führen.

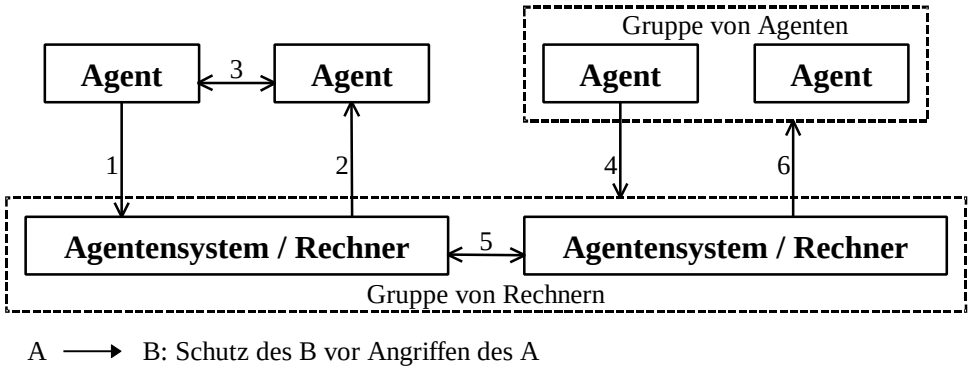

Abbildung 10: Notwendige Schutzbeziehungen

Das größte Problem ist der Schutz des Agenten vor dem Rechner, wofür es derzeit noch keine Lösung gibt, außer nur vertrauenswürdige Rechner zu besuchen (für einen Ansatz siehe [Sanders/Tschudin 1998]), was jedoch auch nicht ohne weiteres gewährleistet werden kann.

2.2.3 Einzelagenten – Systeme von Agenten (Multi-Agentensysteme, MAS)

Eine weitere Unterscheidungsmöglichkeit für Agenten ist, ob es sich um Einzelagenten handelt oder um Systeme von Agenten (MAS, Multi-Agentensysteme). Hier ist zu beachten, daß mehrere getrennte Agenten noch kein MAS erzeugen. Erst wenn zur Aufgabenerfüllung diese zusammenarbeiten oder zumindest miteinander kommunizieren, kann von einem MAS gesprochen werden.

Systeme von Agenten haben den Vorteil, daß eine Aufgabe auf mehrere Elemente aufgeteilt wird, wie es der top-down Entwurfsstrategie entspricht. So können Probleme auf eine Anzahl von Agenten aufgeteilt werden (was die Möglichkeiten für Parallelverarbeitung erhöht) und einzelne davon für verschiedene Systeme Verwendung finden. Auch ist ein Auswechseln von Elementen bzw. eine Rekonfiguration leichter, wenn es sich um mehrere Agenten handelt.

Bei der Kommunikation zwischen Agenten stellen sich zwei Hauptprobleme: Die technische Durchführung und die zu verwendende Sprache.

1. Technische Durchführung: Drei große Systeme stehen zur Verfügung: Message Passing (MP; Abbildung 11), Remote Procedure Calls (RPC; Abbildung 12) und Blackboards (Abbildung 13). RPCs erlauben sowohl eine einfachere Programmierung als auch (bei Verwendung von anerkannten Standards, z. B. DCE [DCE]) plattformübergreifende Kommunikation. Ihr großer Nachteil ist jedoch, daß Sicherheitsvorkehrungen viel schwerer (und großteils überhaupt unmöglich) sind und daß eine genaue Kenntnis der Methoden des aufgerufenen Agenten notwendig ist. Message Passing hingegen erlaubt eine sehr abstrakte Definition von Nachrichten, die von vielen verschiedenen Agenten verstanden werden können. Auch ist eine Sicherheitsprüfung leichter zu integrieren. Problematisch ist hingegen, daß eine Konvertierung von Parametern komplexer ist (meist keine Unterstützung durch OS oder Plattform) und die Programmierung aufwendiger (da selbst vorzunehmen). Jeder Agent muß regelmäßig überprüfen, ob er neue Nachrichten empfangen hat und der Sender muß selbst Vorkehrungen treffen, um bei Empfang einer Antwort an der richtigen Stelle weiterzuarbeiten (bei asynchronem Messaging). Dennoch empfiehlt sich für Agenten (zumindest konzeptuell) eher diese zweite Methode der Nachrichtenübermittlung, da sie der Autonomie von Agenten mehr entspricht: Es wird eine Nachricht übermittelt, worauf der Empfänger selbst entscheiden kann, welche Aktion gesetzt wird (siehe auch die Unterscheidung Agent ? Objekt, 2.1.6). Demgegenüber besitzt der Server-Agent bei RPCs meist keine Möglichkeit, diese zu unterbinden (außer alle global), wenn dies nicht explizit in jede auf diese Weise zugängliche Prozedur selbst eingebaut wird. Bei Verwendung eines Blackboards schreiben alle Agenten ihre Anforderungen bzw. Antworten in einen gemeinsamen Bereich, für den alle Agenten Zugriffsrechte besitzen. Dies ist naturgemäß nur für kleinere Agenten-Gruppen möglich. Ein weiteres Problem dieses Ansatzes ist, daß jegliche Kommunikation öffentlich erfolgt (zumindest im Hinblick auf die Existenz und die beteiligten Rechner; Möglichkeit von Verkehrsanalyse-Angriffen).

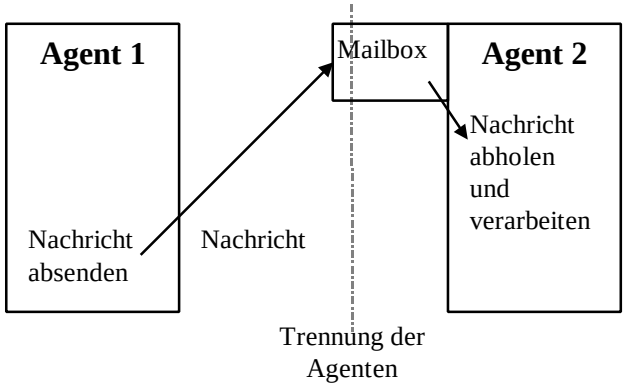

Abbildung 11: Kommunikation über Message-Passing

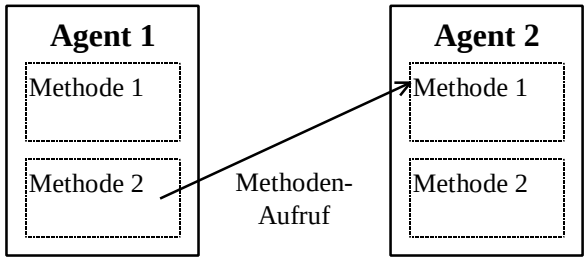

Abbildung 12: Kommunikation über Remote Procedure Calls (RPC)

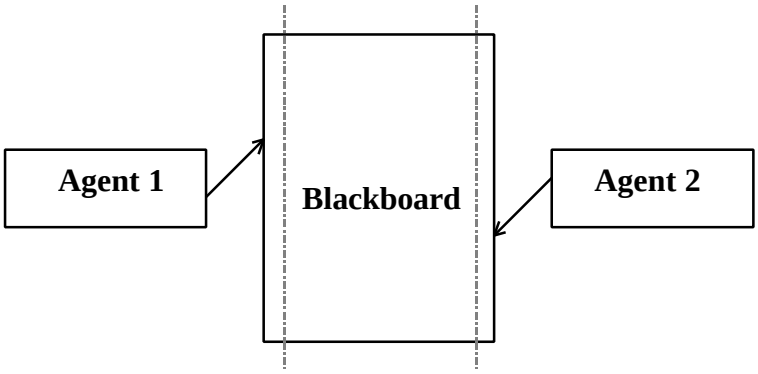

Abbildung 13: Kommunikation über Blackboards

2. Verwendete Sprache: Für die Kommunikation zwischen Agenten muß eine gemeinsame Sprache gefunden werden, wobei nicht nur auf die korrekte Umwandlung der Parameter (Marshalling) zu achten ist, sondern insbesondere auch auf deren Bedeutung. Dieser Aspekt ist so wichtig, daß Agenten-Definitionen existieren, welche dies als ausschließliches Kriterium verwenden ([Genesereth/Ketchpel 1994]: Ein Objekt ist dann und nur dann ein Software-Agent, wenn es korrekt in einer Agenten-Kommunikationssprache kommuniziert). Dies ist kein großes Problem, wenn es sich um ein geschlossenes System handelt: Es kann entweder selbst eine Sprache definiert werden oder eine in der Literatur beschriebene Verwendung finden. Sobald es sich jedoch um offene Systeme handelt, stellt sich ein fundamentales Problem: Wie können verschiedene Nachrichten bzw. RPCs voneinander unter-

schieden werden? Eine eindeutige Numerierung würde eine zentrale Registrierung *aller* Kommunikationsformen bedingen. Ist dies nicht der Fall, so muß immer damit gerechnet werden, daß ein Agent eines anderen Herstellers einen Auftrag verschickt, welcher den gleichen Namen (Nummer, ID, ...) besitzt wie eine bekannte Anfrage, jedoch einen unterschiedlichen Inhalt und eine abweichende Bedeutung hat. Dies bringt viele zusätzliche Probleme, da dies auch jederzeit im Laufe einer längeren Kommunikation (z. B. durch Versions-Unterschiede) erfolgen kann, sodaß ein Mißverständnis u. U. erst sehr spät entdeckt wird. Abhilfe kann die Verwendung von langen Zufallswerten zur (relativ) eindeutigen Kennzeichnung bringen, sodaß die Wahrscheinlichkeit solcher Kollisionen sehr gering wird (siehe dazu UUIDs [UUID]; ist im System eine Netzwerkkarte vorhanden so ist das Ergebnis tatsächlich einmalig, da die dort enthaltene Maschinenadresse (MAC) inkludiert wird, welche weltweit durch zentrale Vergabe an die Hersteller garantiert einmalig ist.

2.2.4 Gleichordnung - Hierarchie

Eine komplexe Klassifikation für Agenten ist die nach der Existenz und der Art der vorhandenen Hierarchien zwischen Agenten. Bei vielen Agenten-Systemen sind alle Agenten mehr oder weniger „gleich“ in dem Sinne, daß für alle potentiell die selben Bedingungen bestehen und keine Abhängigkeitsverhältnisse existieren. Unterschiede ergeben sich nur aus verschiedenen Berechtigungen und den zu erfüllenden Aufgaben. Im Gegensatz dazu existieren bei hierarchischer Organisation mehrere Schichten von Agenten. Dies kann einerseits dadurch erfolgen, daß ein Agent einen von ihm abhängigen Agenten erzeugt (Erzeugungshierarchie), andererseits auch aus der Berechtigung, Befehle zu erteilen, bestehen (Weisungshierarchie).

Eine andere Art von Hierarchie wird in [Vally/Courdier 1999] beschrieben. Agenten werden dort bestimmte Rollen zugeordnet und zu Gruppen und Gesellschaften geordnet. Es entsteht eine strenge Hierarchie die auch definiert welche Nachrichten ein Agent aussenden bzw. verstehen kann. Für einen Agenten besteht jedoch die Möglichkeit, in eine andere Ebene zu wechseln, sofern er dazu in der Lage ist.

Zusammenfassend kann gesagt werden, daß es sich hier weniger um eine Eigenschaft des Agentensystems selber handelt (da viele Systeme z. B. Agenten erlauben, andere Agenten zu erzeugen), als vielmehr um eine Eigenschaft eines konkreten Systems, welches mit Agenten implementiert wurde. Dennoch sollte darauf Wert gelegt werden, da es die Abbildung von realen Problemen auf MAS erleichtert.

2.2.5 Reaktiv - Deliberativ - Hybrid

Auch die Art der Erzeugung der Ausgangsbefehle aus den Daten der Umgebung und dem internen Zustand kann zur Klassifikation verwendet werden ([Klusch 1998], [Furbach et al. 2000]). Es handelt sich hier um die interne Konstruktion der Intelligenz des Agenten.

2.2.5.1 Reaktiv

Bei diesem Typ von Agenten werden über ein festgelegtes Schema die Eingangswerte direkt in Ausgangswerte umgewandelt (Abbildung 14), es handelt sich um ein reizorientiertes Verhalten. Sie enthalten kein Weltmodell ([Nwana/Ndumu 1998]). Aufgrund ihrer relativ einfachen internen Konstruktion erlauben sie rasche Reaktionen auf Veränderungen der Umwelt. Aus demselben Grund jedoch besitzen sie als Einzel-Agenten keine Intelligenz. Komplexe Systeme entstehen daher aus ganzen Gruppen von reaktiven Agenten durch auftauchendes Verhalten (emergent behavior). Bei der Implementierung eines konkreten Anwendungssystems ist daher besonderes Augenmerk auf die Dekomposition in einzelne Agenten und deren Interaktion zu legen, um komplexes Verhalten zu erzeugen.

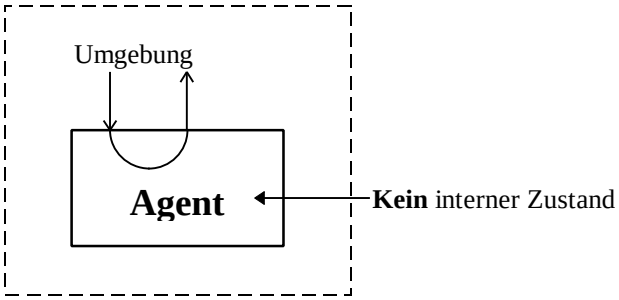

Abbildung 14: Reaktives Agentenmodell

2.2.5.2 Beratend (Deliberative)

Im Gegensatz zu reaktiven besitzen beratende (Englisch: „deliberative“; da dies auch bei einzelnen Agenten möglich ist, wird im Folgenden „Deliberativ“ verwendet, um Mißverständnisse zu vermeiden) Agenten ein eigenes Weltmodell, in welchem der interessierende Ausschnitt der Umgebung abgebildet wird. Sie zeigen reflektives Verhalten durch die Integration von Wissen über ihre eigenen Fähigkeiten in dieses Modell ([Klusch 1998]). Annahmen über andere Agenten sind dadurch möglich, ohne daß deren genaue interne Funktionsweise bekannt ist: Es wird vermutet, daß sie sich ähnlich wie man selbst verhalten werden ([Huhns/Singh 1998]; eine in manchen Fällen gefährliche Annahme). Alle Eingaben werden zuerst mitsamt dem internen Zustand im Weltmodell verarbeitet. Aus diesem Modell werden durch eine Planungs-Komponente der (oder die) auszuführenden Pläne erstellt, welche oft komplexer Natur sind.

Diese Pläne werden im Anschluß durch eine eigene Ausführungseinheit über die vorhandenen Aktoren realisiert. Da die Erstellung eines (für die Anwendung vollständigen) Weltmodells eine komplexe Aufgabe ist und auch in der Ausführung eine längere Zeit benötigt, ist dieser Agenten-Typ für dynamische Anwendungen weniger geeignet.

Deliberative Agenten werden oft auch als BDI-Agenten bezeichnet (Belief-Desire-Intention). Sie bestehen aus folgenden Elementen ([Klusch 1998], [Brenner et al. 1998]; siehe [Müller 1996] für eine Übersicht über Projekte, welche diese Architektur implementieren):

? Ansichten (Beliefs): Dies entspricht dem Weltmodell, wobei nicht sichergestellt ist, daß es auch immer dem tatsächlichen Zustand entspricht. Es enthält auch Erwartungen darüber, welche Veränderungen mit bestimmten Aktionen verbunden sind.

? Wünsche (Desires): Beschreibt den gewünschten Zustand der Umgebung. Inkonsistenzen und unerfüllbare Wünsche sind erlaubt und möglich.

? Ziele (Goals): Eine Untermenge der Wünsche, von denen der Agent glaubt, daß sie erreichbar sind und auch beabsichtigt, sie auszuführen. Hier sind keine Widersprüche erlaubt.

? Absichten (Intentions): Hierbei handelt es sich um eine Untermenge der Ziele. Da ein Agent meistens nicht alle Ziele gleichzeitig verfolgen kann, muß er eine Auswahl treffen.

? Pläne (Plans): Die Kombination von Absichten zu konsistenten Einheiten. Alle Pläne zusammen entsprechen den Absichten, während einzelne Absichten Subpläne eines übergeordneten Plans sind[1].

[1] Entgegen dem üblichen Sprachgebrauch handelt es sich bei Plänen *nicht* um Vorgehensmodelle (wie bestimmte Absichten oder Gruppen von Absichten erreicht bzw. durchgeführt werden können), sondern ausschließlich um eine konsistente Untermenge aller Absichten.

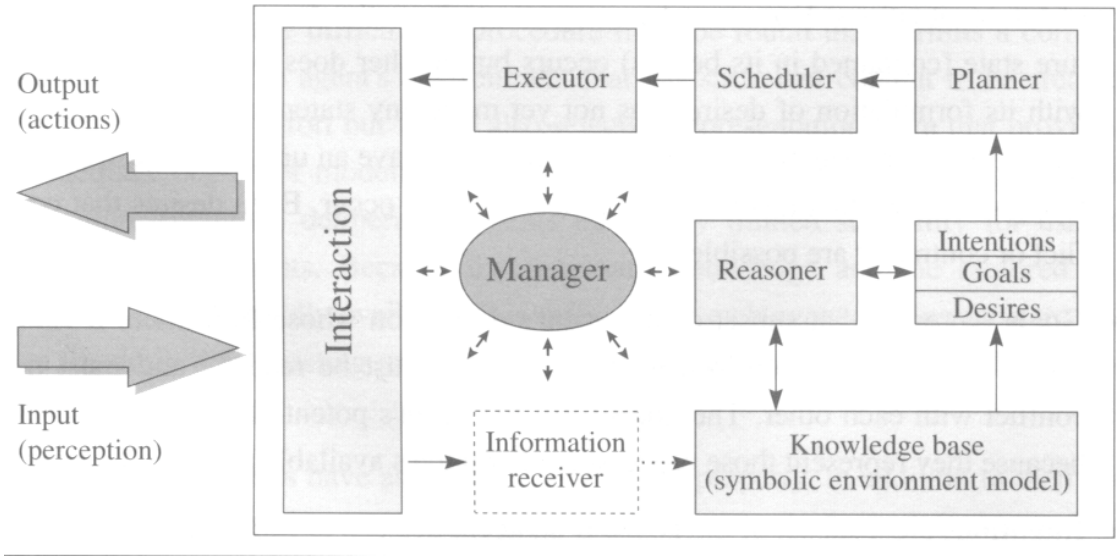

Abbildung 15: Deliberatives Agentenmodell [Brenner et al. 1998]

2.2.5.3 Hybrid

Hybride Agentensysteme beinhalten verschiedene Elemente, wobei die ersten beiden meistens vertreten sind. Sie besitzen typischerweise ein Schicht-Architektur, wie etwa die von Müller in [Müller 1996] beschriebene InteRRaP Architektur:

1. Weltmodell und Verhaltensschicht: Auf dieser Ebene ist das Wissen um die Umgebung sehr konkret. Es wird auf Veränderungen der Umgebung direkt reagiert (Reaktive Schicht).
2. Mentales Modell und lokale Planungsschicht: Das Wissen um die Umgebung ist abstrakter gehalten und um Pläne und Wissen darüber erweitert. Aufgrund von Informationen aus der Verhaltensschicht und dem mentalen Modell werden Pläne erstellt und ausgewählt. Die tatsächliche Durchführung erfolgt durch Anweisungen an die Verhaltensschicht.
3. Soziales Modell und kooperative Planungsschicht: Es ist kein Wissen über die reale Umwelt mehr vorhanden, sondern nur mehr über die anderen Agenten im System, ihre Absichten und ihre Pläne (soweit bekannt). Hier findet die Abstimmung mit anderen Agenten zu gemeinsamer Tätigkeit statt. Informationen werden nur aus der lokalen Planungsschicht gewonnen (z. B. lokale Pläne, welche zur Durchführung andere Agenten benötigen), aber nicht aus dem Weltmodell. Zur Durchführung von Aktionen wird die lokale Planungsschicht verwendet.

Durch diese Schichtung ist sowohl eine schnelle Reaktion möglich (direkt in der Verhaltensschicht) als auch die Planung komplexer Vorgänge (über die lokale als auch die kooperative Planungsschicht). Kommunikation und Anweisungen erfolgen jeweils nur mit den angrenzenden Schichten, sodaß auch der Entwurf überschaubar bleibt.

Ein anderes Beispiel ist in [Sloman/Logan 1999] beschrieben, wo zwischen reaktiven, deliberative und reflektiven Schichten unterschieden wird. Letztere Ebene soll es auch ermöglichen, die Grenzen der Schichten aufzuweichen und Verschiebungen dazwischen durchzuführen, etwa die Verlagerung eines bestimmten Verhaltens auf eine niedrigere Ebene, um schnellere Antworten zu erreichen.

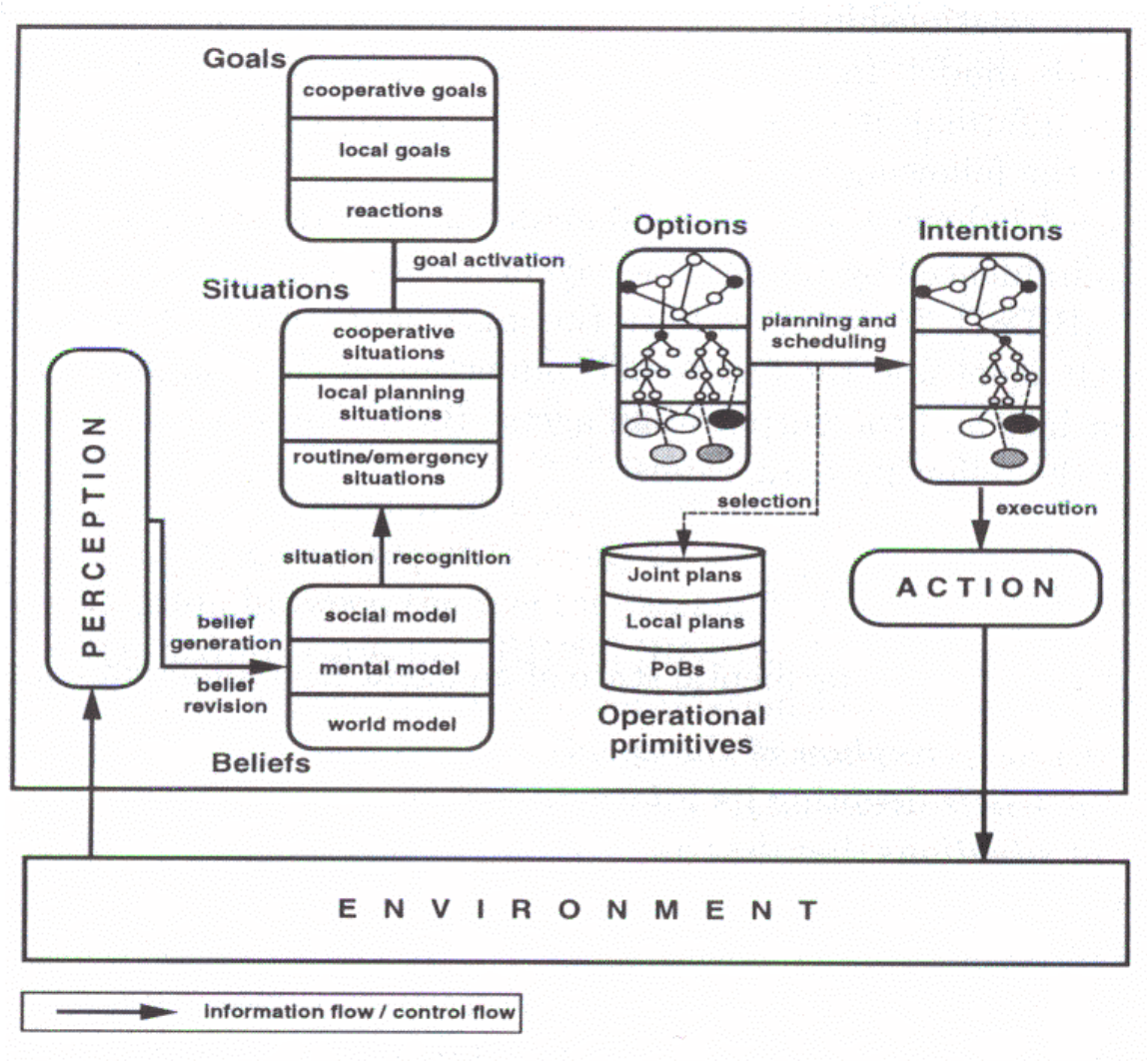

Abbildung 16: Hybrides Agentenmodell [Müller 1996]

2.2.6 Kooperativ – Wettbewerb – Aggressiv

Bei MAS kann eine Einteilung auch danach erfolgen, wie die einzelnen Agenten zusammenarbeiten. Dies hat insbesondere für das Sicherheitssystem große Konsequenzen. Die einzelnen Alternativen sind:

? Kooperativ: Jeder Agent existiert nur, um anderen Agenten zu dienen. Es gibt keinerlei Konflikte zwischen den Agenten und alle arbeiten auf das gemeinsame Ergebnis hin und versuchen, nach Kräften hierzu beizutragen. Der Vorteil ist, daß sich die Interaktion genau

vorausplanen läßt und die resultierende Qualität am Höchsten sein wird. Nachteilig ist, daß dies nur bei geschlossenen Systemen möglich ist: Beliebige (fremde) Agenten können nicht zugelassen werden.

? Wettbewerb: Agenten stehen im Wettbewerb miteinander und versuchen, ihre Ressourcen möglichst nutzbringend einzusetzen. Im Gegensatz zu kooperativen Systemen handelt es sich um eigennützige Agenten, welche versuchen ihren eigenen Gewinn zu maximieren. Sie werden jedoch niemals versuchen, einem anderen Agenten zu schaden, ihn zu behindern oder das Gesamtergebnis zu verringern. Auch hier muß es sich um ein geschlossenes System handeln. Dennoch ist es leichter, fremde Agenten zu integrieren, da lediglich sicherzustellen ist, daß sie keinen Schaden anrichten können; die Interaktion muß jedoch nicht genau vorausgeplant werden. Systeme dieser Art werden meist dann eingesetzt, wenn eine analytische Zerlegung des Systems nicht mehr möglich ist (und damit wird ein kooperatives System mehr sinnvoll) oder ein System zu modellieren ist, in welchem bereits Wettbewerbselemente enthalten sind. Werden immer alle Agenten befragt und der am besten geeignete ausgewählt, so ist das Ergebnis gleich einem kooperativen System. Ein höherer Aufwand ergibt sich durch den Auswahlprozeß, wodurch die Qualität des Endergebnisses auch etwas geringer ausfallen kann.

? Aggressiv: Bei dieser Art von Kooperation wird davon ausgegangen, daß jeder Agent nur für seinen eigenen Vorteil arbeitet. Dies inkludiert im Prinzip auch die Schädigung anderer Agenten, sowohl zum eigenen Vorteil (keine Bedachtnahme auf Konkurrenten) wie zum eigenen Nachteil (wohl kaum als Design-Element). Dieses Modell kann auch bei einem offenen System angewandt werden, da beliebige Agenten Zutritt erlangen können. Es ist jedoch auf die Sicherheit besonderer Wert zu legen, sodaß gewisse Angriffe ausgeschlossen sind (z. B. Modifikation der Daten anderer Agenten), oder zumindest Konsequenzen herbeiführen. Das Resultat solcher Systeme wird in der Regel unter dem von kooperativen oder Wettbewerbs-Systemen liegen. Um es funktional zu erhalten muß sichergestellt werden, daß abweichendes Verhalten unter einer bestimmten (noch akzeptablen) Schwelle bleibt. Dies kann dadurch erreicht werden, daß Mißverhalten von Agenten auch Konsequenzen bei dessen Besitzer nach sich zieht und nicht ausschließlich bei den Agenten selbst. Die Festsetzung der Höhe der Schwelle kann schwierig sein.

Welches Kooperationsmodell gewählt wird, verursacht daher große Auswirkungen sowohl auf das Agentensystem selbst, als auch auf die einzelnen Agenten und Systeme von Agenten. Kooperative Systeme bedürfen genauer Vorausplanung, während Wettbewerbs-Systeme mehr auf

auftauchendes Verhalten setzen (aber auch genau geplant werden müssen). Bei beiden Typen werden an die Agenten keine besonderen Anforderungen gestellt und es kann auch ohne ein Sicherheits-System das Auslangen gefunden werden. Sollen jedoch beliebige Agenten eingebunden werden, so sind extensive Vorkehrungen zu treffen, um Mißbrauch zu verhindern. Der große Vorteil ist jedoch, daß alle Benutzer ihre Agenten auf beliebigen Rechnern verwenden können, was insbesondere in Verbindung mit mobilen Agenten oft erforderlich ist.

2.3. Warum Intelligente Agenten?

In diesem Abschnitt wird ein kurzer Überblick über die Gründe für den Einsatz von Agenten bzw. Systemen von Agenten gegeben. Da es keine „Killer-Applikation“ gibt (siehe auch Abschnitt 2.3.5), ist die Begründung zwar schwieriger, aber dennoch problemlos möglich. Es wird kurz das Lokalitätprinzip erläutert, welches ein wichtiges Element besonders für mobile Agenten ist, sowie die tatsächlichen und erwarteten Vorteile durch die Integration von Intelligenz in Agenten hervorgehoben. Als Abschluß wird erläutert, welche Synergieeffekte durch die Kombination von mobilen Agenten mit PDAs entstehen können.

2.3.1 Warum überhaupt Agenten verwenden: Programmierer-Sicht?

Agenten können als Fortsetzung der objektorientierten Programmierung gesehen werden ([Kendall 2000]). Es werden ganze Aufgaben zu Einheiten mit dem Ziel zusammengefaßt, diese wiederverwendbar und modular zu gestalten. Auch bei Agenten liegt ein großes Augenmerk auf der Gestaltung der Schnittstelle. Hierbei allerdings weniger in der Methodendefinition als vielmehr der Definition der Protokolle und Sprachen. Dies erlaubt es, wie in der OOP, einen Agent gegen einen anderen auszutauschen und so die Implementierung zu ändern, ohne daß das Gesamtsystem erneuert werden muß, solange die Schnittstelle und das Datenformat bzw. die Bedeutung von Parametern gleich bleiben. Unterschiede zur OOP bestehen jedoch darin, daß der interne Zustand eines Agenten viel komplexer ist und z. B. auch Pläne und Ziele enthält (BDI-Agenten; siehe 2.2.5). Agenten erlauben es daher, auf Methoden für die Entwicklung von OOP aufzubauen und eine weitere Abstraktionsschicht einzuziehen. Dies bringt viele Vorteile bei komplexen Systemen, insbesondere wenn auch örtliche Verteilung geplant ist. Der Grund hierfür ist, daß die Kommunikation über Nachrichten bei Agenten besser geeignet ist, als ein Methodenaufruf bei Objekten.

Hand in Hand mit der Erweiterung entstehen auch neue Methodologien für den Entwurf von Systemen von Agenten ([Zambonelli et al. 2000]). So ist eine Dekomposition eines Gesamtsy-

stems nach der Funktionalität: Abgeschlossene Teilaufgaben, Zusammenwirken mehrerer selbständiger Elemente zur Erreichung eines gemeinsamen Zieles, … einfacher zu verstehen als eine Zergliederung in Daten oder Objekte ([Jennings 1999]). Auch entspricht die Aufteilung in mehrere relativ autonome Agenten stärker realen Systemen: Bei diesen existiert oft keine gemeinsame „Spitze“ im Sinne eines allen übergeordneten Kontrollpunktes (Kontrolle und Rückkopplung finden auf verschiedensten Ebenen statt). Jede Subaufgabe sollte daher selbständig Entscheidungen treffen können: Dies entspricht der Autonomie von Agenten ([Jennings 2001]).

Anhand der in [Wenger/Probst 1998] beschriebenen Analogie kann die Entwicklung hin zu Agenten wie folgt erläutert werden: Daten und der Fortschritt der Informationstechnologie (IT) können mit Gütern und der Industrialisierung verglichen werden. Ursprünglich waren der Transport und die Lagerung von Gütern sehr wichtig, da die Bearbeitung nur an wenigen einzelnen Stellen (=Maschinen; Antrieb durch Mühlen oder Dampfmaschinen) möglich war. Die Produktion selbst war durch den Einsatz von menschlicher Arbeitskraft geprägt. Durch die Einführung der Elektrizität veränderte sich der Herstellungsprozeß: Maschinen konnten nun in beliebiger Menge und an beliebigen Orten verwendet werden, sodaß Transport und Lagerung zu untergeordneten Tätigkeiten wurden. Der wichtigste Punkt in der Produktion von Gütern ist seither die Bearbeitung, da dort auch die Wertschöpfung entsteht. Analog dazu mußten in der Anfangszeit der IT die Daten zu einzelnen zentralen Computern gebracht werden und Dateneinein- und -ausgabe war ein essentieller Teil. Datenübermittlungen waren komplex und teuer (Transport von Magnetbändern oder Lochkartenstapeln). Wie in der Anfangszeit der Industrie konnten nur zentrale Tätigkeiten durch Automatisierung unterstützt werden (z. B. Buchhaltung). Mit der Verbreitung leistungsfähiger Rechner trat auch hier ein Wandel ein: Der Transport von Daten ist heute kein Problem mehr und besitzt nur untergeordnete Funktion. Applikationen unterstützen mehr und mehr auch verteilte Aufgaben. Diese Analogie kann jedoch noch weitergeführt werden. Wie Maschinen immer komplexer und selbständiger wurden (bis hin zu vollautomatischen Produktionsstraßen), werden auch Programme immer komplexer. Erhalten sie auch die Möglichkeit, eigene Entscheidungen zu treffen, so gelangt man zu Agenten. Ebenso wie Werkzeuge ursprünglich passiv (Hammer) waren und immer aktiver wurden (CNC-Maschinen), werden auch Programme immer aktiver: Von Texteditoren, welche ausschließlich durch den Benutzer bedient werden, hin zu Agenten, die selbsttätig auf ihre Umwelt einwirken.

Ein weiterer Grund für die Anwendung von Agenten kann darin gesehen werden, daß dies den Designer zwingt, sich Gedanken darüber zu machen, welche Funktionalität genau bereitgestellt werden muß und wie diese von außen genutzt werden kann. Für einem anderen, eventuell von einem unterschiedlichen Hersteller stammenden, Agenten muß eine genau spezifizierte Schnittstelle bereitgestellt werden, was für einen menschlichen Benutzer nicht notwendig ist. Anstatt eines monolithischen Systems (eventuell OO implementiert), entsteht ein System vieler Teile, was auch den Wettbewerb und damit die Qualität hebt. Dies könnte auch dazu führen, daß Programme oder Produkte besser bedienbar werden. Auch erlaubt dies, die eigenen Produkte offen zu gestalten, sodaß fremde Programme mit diesen zusammenarbeiten können. Ein Beispiel für den Vorteil solch offener Systeme ist das WWW, welches nur durch (allerdings sehr genau spezifizierte) Protokolle „zusammengehalten" wird und gerade davon enorm profitiert hat. Ein Zukunfts-Szenario für ein solches offenes System wird in [Huhns 1999] dargestellt: Alle Küchengeräte besitzen eingebettete Agenten und können so miteinander kommunizieren und ihre Arbeit aufeinander abstimmen, um Arbeiten unter Restriktionen (Energieverbrauch, Geräuschpegel, ...). durchzuführen. Da alle Geräte von verschiedenen Herstellern stammen können und die potentiellen Konfigurationsmöglichkeiten enorm sind, ist eine exakte Planung unmöglich. Jedes Gerät muß daher mit gewisser Autonomie und mit Kommunikationsmöglichkeiten ausgestattet werden, was am besten durch Agenten realisiert werden kann.

2.3.2 Das Lokalitätsprinzip

Ein wichtiges Element für das Konzept von Agenten ist das Lokalitätsprinzip, welches für Daten (bei [Seitz 2000] als „Reduktion der Kommunikationskosten" bezeichnet) aber auch ebenso für Programme existiert.

Daten sollten dort verarbeitet werden wo sie anfallen, entstehen oder gelagert sind. Erst nach erfolgter Verarbeitung werden (hoffentlich komprimierte) Zwischenergebnisse an einen anderen Ort transferiert, wo sie weiterverarbeitet werden (was nur bei Zusammenfassung mit anderen Daten Sinn hat, da die Verarbeitung sonst vor dem Transfer hätte stattfinden sollen). Dieses System ist jedoch nur dann sinnvoll, wenn die Ergebnisse „wertvoller" sind als die Anfangsdaten. Dies kann sich einerseits in geringerem Umfang ausdrücken (Datenauswahl, wie gerade eben beschrieben), aber sogar bei größerem Datenvolumen sinnvoll sein: Lokalitätsprinzip für Programme (siehe unten).

Ähnlich wie bei den Daten sollte auch die Verarbeitung als solche möglichst nahe an der Quelle stattfinden, sodaß etwaige Rückfragen, Überprüfungen, weitere notwendige Daten, aber auch

Reaktionen möglichst rasch, einfach und zuverlässig (weniger Kommunikationsverbindungen welche möglicherweise ausfallen) erfolgen können. Gegenüber der Daten-Lokalität existieren jedoch einige gute Gründe, Ausnahmen von diesem Grundsatz zu machen. So ist etwa die Verarbeitung auf einem spezialisierten Rechner schneller als auf einem normalen (der z. B. mit einer Datenbank „belastet“ ist). Weiters sind insbesondere Steuerungen oft nicht dafür geeignet, komplexe Berechnungen auf ihnen durchzuführen, sodaß die Programmlokalität zwangsweise gelockert werden muß. Ein weiteres Problem kann sein, daß Endgeräte eher ausfallen (oder ausgeschaltet werden) als zentrale und gesicherte Server. In Bezug auf Agenten ist daher auch eines der Ziele, Agenten von PDAs zur Datenverarbeitung auf andere Rechner zu transferieren und erst anschließend wieder zurückzukehren. Gegenüber der Daten-Lokalität, welche praktisch immer nur Vorteile mit sich bringt, ist daher die Lokalität von Programmen nicht *immer* und nicht immer *ausschließlich* günstig.

Bei der Implementierung von Systemen ist zu beachten, daß jeder Transfer auch zusätzliche Kosten verursacht. Wird daher ein Agent ausgesandt, um Daten lokal auf einem anderen Rechner zu verarbeiten und kehrt er anschließend mit den Ergebnissen zurück, so ist der Aufwand für den zweifachen Agenten-Transport mitzuberücksichtigen. Eine lokale Verarbeitung ist daher nur dann sinnvoll, wenn es sich entweder um größere Datenmengen (statischer Aspekt; Daten-Lokalität) oder aufwendigere Berechnungen (dynamischer Aspekt; Programm-Lokalität) handelt.

Abgesehen von Effizienzgründen kann das Lokalitätsprinzip jedoch auch für die Modellierung von Systemen von Agenten wichtig sein: Es erlaubt die Verbindung von bestimmten Daten und Verarbeitungsvorgängen mit Orten, was (bei guter Modellierung dieser Einzelteile) eine Erleichterung durch die bereits erfolgte Vorstrukturierung mit sich bringen kann. Auf den verschiedenen Stationen im Laufe einer Verarbeitung werden so jeweils bestimmte Ergebnisse erzeugt, welche dann in einen weiteren Schritt zusammengefügt werden.

2.3.3 Erwartete Vorteile durch intelligente Agenten

Von der Integration von mehr Intelligenz in Agenten werden unter anderem folgende Verbesserungen für den Benutzer erwartet:

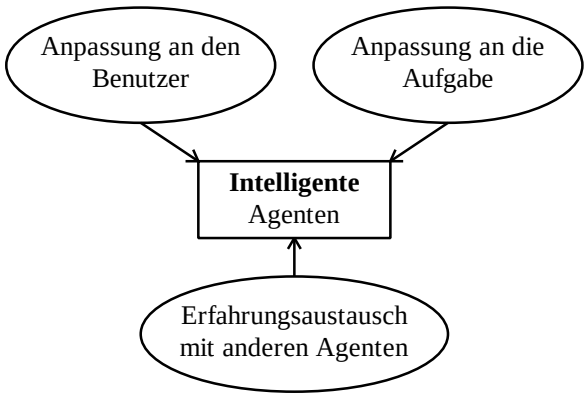

Abbildung 17: Erwartete Vorteile durch *intelligente* Agenten

? Selbständiges Lernen der Wünsche des Benutzers (Automatische Personalisierung). Nach wiederholten ähnlichen Aufgaben sollte ein intelligenter Agent Schlüsse daraus ziehen und z. B. immer gleichartig beantwortete Rückfragen nicht mehr stellen. Noch besser ist, den Grund für die gleichartigen Antworten versuchen herauszufinden und entsprechend diesem zu handeln oder ihn vorzuschlagen. Dies soll auch eine Verbesserung bei der Konfiguration von Programmen bringen: Anstatt in sehr großen Dialogen alle Einzelheiten selbst einzustellen, sollen für den Benutzer automatisch die Einstellungen vorgenommen werden (Dies hat natürlich mit Vorsicht zu erfolgen: Veränderungen in der Benutzerumgebung können äußerst verwirrend sein). In diese Kategorie gehört auch die automatische Anpassung an den Kompetenzgrad des Benutzers (Anfänger, normaler Benutzer, Experte).

? Analog zur Anpassung an den Benutzer soll auch eine Anpassung an die konkrete Aufgabe erfolgen: Bisher versuchte erfolglose Lösungswege abschneiden und erfolgreiche mit einem höheren Prioritätswert versehen. Diese Strategie erreicht dann automatisch das (allerdings eventuell nur lokale, nicht globale) Optimum, sofern Änderungen nur langsam erfolgen (daher u. U. nur beschränkt für die Auswahl von Kommunikationsverbindungen verwendbar, da sich hier die Bedingungen sehr rasch und stark verändern können). Ein gutes Beispiel ist auch in [Fuchs 2000] angeführt: Ein Agent zur Überwachung des freien Festplattenplatzes sollte seine Regeln selbständig anpassen, wenn eine größere Festplatte eingesetzt wird. Das 90% Limit bei einer 1 GB Platte ist bei einer 20 GB Platte unsinnig (2 GB: das Doppelte der anfänglichen Gesamtgröße!).

? Zur Verbesserung der allgemeinen Leistungsfähigkeit sollen Erfahrungen zwischen Agenten ausgetauscht werden: Nicht nur auf Daten, sondern auch auf der Informations- und, durch die Intelligenz, auch auf der Wissens-Ebene. Dies hat zweierlei Vorteile: Erstens erfahren Agenten von neuen Lösungswegen, was zu einer Verbesserung führen kann, andererseits

profitieren sie von den Erfahrungen derer, die bereits viele Aufgaben durchgeführt haben. Nachteilig bei diesem Ansatz ist, daß alle Erfahrungen ausgetauscht werden, auch falsche oder absichtlich irreführende.

Zusätzlich zu diesen sachlichen Verbesserungen kann auch die Implementationsseite davon profitieren:

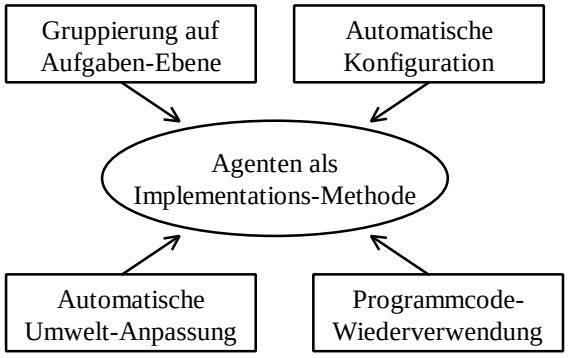

Abbildung 18: Erwartete Vorteile durch Agenten als *Implementationsmethode*

? Agenten sollen sich selbst an ihre Umwelt anpassen. Versteht beispielsweise ein Agent ein konkretes Protokoll nicht, kennt aber dessen einzelne Bestandteile (=Sub-Protokolle; kurze Teilstücke), so soll er sich ein passendes Protokoll „maßschneidern“, indem einzelne Elemente zusammengefügt werden. Dies kann sowohl auf syntaktischer (Adapter für verschiedene Kommunikationsarten mit anderen Agenten/Objekten: OLE, RPCs, Messages, …) als auch auf semantischer (Meta-Beschreibung der Bedeutung einzelner Methoden/Nachrichten wird ausgewertet und entsprechende Befehle weitergegeben oder andere Agenten gesucht) Ebene erfolgen. Siehe hierzu als Basis 4.5.1.5.

? Mehrere Agenten sollen sich automatisch zu einem System konfigurieren: Durch einfaches hinzufügen oder entfernen von Agenten soll das Gesamtsystem verändert werden können, ohne daß noch weitere Arbeiten notwendig sind. Dazu müssen die Agenten intensiv miteinander kommunizieren, um wieder einen konsistenten und sinnvollen Zustand zu erreichen. Dieses Ziel ist natürlich nur eingeschränkt zu erreichen; eine gewisse Konfiguration (=Vorgabe der Gesamtziele) wird immer notwendig sein. Als Beispiel siehe das System zur Verarbeitung von Nachrichten (Abschnitt 6.4).

? Die von der Objektorientierung erwartete Code-Wiederverwendung soll erreicht oder zumindest verbessert werden. Im Bereich von low-level-Software (Benutzerinterface, Algorithmen, Datenstrukturen) hat sich Objektorientierung als sehr erfolgreich herausgestellt und zu vielfacher Wiederverwendung geführt. Auf einer höheren Ebene hingegen konnte dieser

Vorteil nur in geringem Maße realisiert werden. Ein Grund dafür ist etwa, daß größere und komplexe Objekte meist sehr spezifisch für eine konkrete Implementation in einem einzelnen Anwendungsfall sind, sodaß eine Wiederverwendung nicht oder nur schwer möglich ist. Auch ein Problem hierfür ist die Semantik des Objektes: Was passiert genau, wenn eine bestimmte Methode aufgerufen wird? Dies ist nur in der Dokumentation enthalten und meist zusätzlich schwer textuell vollständig erfassbar. Im Gegensatz dazu sind Agenten weniger implementations- als anwendungsspezifisch aufgebaut ([Farhoodi/Fingar 1997]). Schon vom Entwurf an liegt daher das Hauptaugenmerk nicht auf der Implementation in einer konkreten (objektorientierten) Sprache mit allen Besonderheiten und Einschränkungen, sondern dem gewünschten Verhalten. Insbesondere von intelligenten Agenten wird erwartet, daß sie dieses Verhalten, zumindest in gewissem Umfang, auch selbst beschreiben können (zur Vereinfachung der Zusammensetzung mehrerer Agenten).

? Gruppierung auf höherer Ordnung. Um so größer eine Aufgabe ist, desto mehr einzelne Objekte sind zu ihrer Erfüllung notwendig. Dies bedeutet jedoch zumindest überlineare (bis maximal exponentielle) Kommunikation zwischen den einzelnen Objekten. Durch intelligente Agenten kann eine Gruppierung auf höherer Ebene erfolgen, die dennoch in sich abgeschlossen ist. In diesem Sinne könnte man eine Aufbauhierarchie eines Projektes als Kommando – Prozedur – Objekt – Agent – Programm darstellen (siehe Abbildung 19). Auf diese Weise kann die Gesamt-Kommunikation wieder auf ein erträgliches (beim Design verständliches und während der Laufzeit verarbeitbares) Ausmaß reduziert werden.

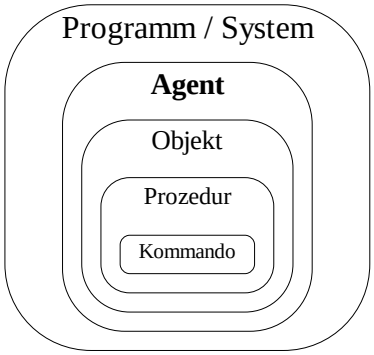

Abbildung 19: Gruppierung auf verschiedenen Ebenen

2.3.4 Agenten als Ergänzung von PDAs

Wie weiter oben unter 2.1.6 ausgeführt handelt es sich bei PDAs nicht um Agenten, jedoch um Geräte welche sehr gut als Plattform für diese geeignet sind. Bei dieser Verbindung von (meistens mobilen) Agenten und mobilen Geräten sollen Agenten insbesondere folgende Aufgaben erfüllen:

? Beschränkte Kommunikationsbandbreite. Selbst bei Fortschritt der Technik ist davon auszugehen, daß die Bandbreite für externe Kommunikationsverbindungen immer geringer als die von fest installierten Geräten mit fixem Kommunikationszugang bleibt. Dies trifft in besonderem Maße für die Kommunikation vom Gerät zum Netz hin zu (upstream). Agenten können in diesem Zusammenhang Verbesserungen bringen: Daten werden nicht mehr lokal verarbeitet und anschließend weitergeleitet (was bedeutet, daß die gesamten Daten über das Kommunikationssystem empfangen werden müssen), sondern der Benutzer nimmt nur die Konfiguration auf dem Gerät vor. Der Agent wandert anschließend auf einen anderen (festen) Rechner (auf dem keine Bandbreiteneinschränkung besteht), von wo aus er seine Aufgaben erfüllt, d. h. die Daten lädt und verarbeitet. Während dieser Zeit steht ihm eine sichere und breite Netzwerksanbindung zur Verfügung. Ist das Ergebnis komplett, so begibt er sich zurück zum mobilen Gerät (mit den ausgewählten Enddaten in komprimierter Form, also mit zu den Eingabedaten vergleichsweise geringem Bandbreitenbedarf) und präsentiert dort die Ergebnisse.

? Agenten folgen den Benutzern. Wird eine länger dauernde Aufgabe gestartet und ist der Benutzer bei ihrem Abschluß nicht mehr an dem selben (ortsfesten) System angemeldet, so müßte ein Agent warten bis der Benutzer sich dort wieder einloggt (was u. U. auch nie erfolgt, z. B. auf Reisen). Da die meisten Agenten bzw. Systeme von Agenten portabel gestaltet sind (d. h. durch mobile Agenten implementiert und nicht ortsgebunden), ist es jedoch auch möglich, die Ergebnisse dem Benutzer nachzutragen: Die Agenten folgen mit ihren Meldungen automatisch dorthin wo sich ihr Besitzer gerade aufhält. Auf dieselbe Weise kann natürlich auch der Benutzer selbst seine Agenten jederzeit kontaktieren und neue Aufgaben erteilen oder alte modifizieren oder beenden. Sollte ein Agent auf ein schwieriges Problem stoßen und eine Rückfrage an den Besitzer haben, so kann auch diese kurzfristig gestellt werden.

? Agenten als Darstellungs-Hilfsmittel. Agenten können dazu verwendet werden, das Problem der kleinen Bildschirme von PDAs zu verringern. Einerseits können große Einstellungsmöglichkeiten der Darstellung angepaßt werden, andererseits können intelligente Agenten weniger wichtige Elemente auch selbst festlegen. Dies kann nach einprogrammierten Regeln erfolgen, oder auf den früheren Eingaben des Benutzers basieren. Weiters kann eine Konsistenzkontrolle vorgenommen werden: Sind die Eingaben komplett und zusammenpassend bzw. welche sonst nicht angezeigten Elemente sind in Sonderfällen doch zu zeigen? Je nach dem verwendeten System können Agenten ihre Dienste oder andere Systeme an Geräte-

Besonderheiten anpassen. So kann ein Agent zur Bedienung komplexer Programme verwendet werden, welche real auf einem anderen Rechner ablaufen.

? Agenten-Repository. Da auf PDAs meist nur wenig Speicherplatz (sowohl Festplatte, falls vorhanden, als auch Hauptspeicher) zur Verfügung steht, ist für das Funktionieren dieses Systems ein weiterer Rechner notwendig, auf welchem die Agenten gespeichert werden während sie inaktiv sind. Wird aus einer größeren Auswahl fertiger Agenten ein bestimmter benötigt, so ergeht eine Anforderung und der Agent verlagert sich auf den PDA zur Ziel-Konfiguration. Diese Konstruktion hat noch den weiteren Vorteil, daß Agenten nicht immer neu erzeugt werden und so über längere Zeit Wissen und Erfahrung ansammeln. In Verbindung mit dem vorigen Punkt kann ein Agent aus zwei Teilen bestehen: Einem Hauptteil, der die eigentliche Tätigkeit durchführt, und einem kleinen „lightweight" GUI-Agenten, welcher auf den PDA verlagert wird, dort Eingaben entgegennimmt, überprüft, vervollständigt, und schließlich, nach erfolgter Konfiguration, zurückkehrt.

2.3.5 Zusammenfassung

Das größte Problem für die Einführung von Agenten ist das Fehlen einer *offensichtlichen* Zielvorstellung; es erscheint aus folgenden Gründen nicht klar, warum man diesen (teilweise zusätzlichen) Aufwand auf sich nehmen soll:

1. Die Aufgaben lassen sich auch mit „normalen" Programmen lösen.
2. Der den Versprechungen nicht gerecht werdende Erfolg der objektorientierten Programmierung macht vorsichtig.
3. Vorteile stellen sich nur bei großen Systemen ein.
4. Verteilte und asynchrone Systeme sind sehr komplex.
5. Programme werden gerne als statische Produkte betrachtet, obwohl sie regelmäßig weiterentwickelt werden. Agenten hingegen bringen besondere Vorteile bei Programmen mit raschem Wechsel (neue Komponenten hinzu, alte entfernen, Abänderungen, etc.).

Es fehlt also eine sogenannte „Killer-Applikation" ([Guilfoyle 1998]), so wie das WWW im Internet. Dies gilt insbesondere auch für den Bereich des E-Commerce, in dem Agenten eine besondere Rolle spielen sollen ([Wong et al. 1999]). Wer Webseiten über die eigene Firma anbieten möchte oder solche besuchen will, muß in der Praxis ins Internet, unabhängig davon ob er dies möchte oder nicht (und ohne Berücksichtigung von dessen Nachteilen und Gefahren). Gleichzeitig wird dies von einem Großteil aller beteiligten Personen gewünscht.

Dennoch bringen Agenten Vorteile, welche analog zu den obigen Punkten dargestellt werden:

1. Programme bestehend aus Agenten lassen sich leichter warten, da automatisch eine weitere Untergliederung in abgeschlossene Teile (=Agenten) notwendig ist.
2. Die Erwartungen an Agenten sind realistischer: Es wird eine Verbesserung geben, aber sie sind kein Allheilmittel und zur Zeit auch nicht für alle Anwendungen geeignet (z. B. lokale Programme, welche selten geändert werden und an die besondere Ansprüche hinsichtlich Korrektheit und Zuverlässigkeit gestellt werden).
3. Systeme werden immer komplexer und sowohl Umfang, Integration alter Programme, als auch Komplexität werden in Zukunft vermutlich auch weiter steigen.
4. Ein Ziel von Agenten ist es, die verteilte und asynchrone Programmierung zu vereinfachen, indem die Abstraktion „Agent“ verwendet wird. Dies kann zwar nicht alle Probleme lösen doch bedeutet es zumindest eine Vereinfachung (etwa die örtliche Transparenz des Kommunikationspartners bei mobilen Agenten).
5. Die meisten Programme sind inhärent dynamisch: Sie werden ausgebaut, korrigiert, angepaßt usw. Ein besonderes Problem ist hier die Integration von neuen Elementen. Umso mehr Elemente bisher existieren, desto riskanter wird es, neue einzubauen, da die Anzahl der möglichen Konflikte steigt. Intelligente Agenten sind in der Lage einen Großteil dieser Probleme selbständig zu lösen oder zumindest vorzubereiten, sodaß die Integration von neuen Elementen (aber umgekehrt auch das Entfernen alter) viel geringere Schwierigkeiten bereitet.

Insgesamt kann daher gesagt werden, daß das Konzept von intelligenten Agenten sehr wohl Vorteile bringt, diese sich jedoch erst bei größeren Systemen und über einen längeren Zeitraum hin auswirken.

2.4. Anwendungsgebiete

Agenten können (als Software-Programme gesehen) grundsätzlich für alle Aufgaben eingesetzt werden, doch nicht überall macht ihr Einsatz auch Sinn. Daher soll hier dargestellt werden, auf welchen Gebieten der Einsatz von Agenten und Systeme von Agenten besonderen Sinn macht, warum dies, und welche Vorteile dadurch konkret zu erwarten sind.

Um auch tatsächlich praktikabel zu sein muß ein Agent folgende zwei Eigenschaften besitzen ([Kearney 1998]), ansonsten würde es sich nur um ein Forschungsprojekt bzw. einen Prototypen handeln:

? Eine nützliche Aufgabe ist zu erfüllen, welche eine *autonome* Tätigkeit durch den Agenten erfordert (eigene maßgebliche Entscheidungen treffen).

? Der Agent muß diese Aufgabe zuverlässig unter allen äußeren Umständen erfüllen, und zwar ohne den Benutzer durch übermäßige Rückfragen oder Bestätigungen zu belasten.

Die meisten existierenden Systeme besitzen Probleme mit dem zweiten Punkt: Rückfragen oder Bestätigungen sind regelmäßig erforderlich. Dies hat insbesondere den Grund, daß die Entscheidungen oft wichtig sind (sonst wäre keine Autonomie notwendig), der Programmierer oder Besitzer des Agenten jedoch meist nur ein relativ geringes Vertrauen in dessen Fähigkeiten besitzt (was auf der unvollständigen Intelligenz beruht).

Informations-geprägt	? System-geprägt
?? Informationssammlung	?? E-Commerce (Allgemein)
?? Informationsfilterung	?? E-Commerce (Auktionen)
?? Beratung	?? Steuerungssysteme
?? Groupware	?? Entertainment

Abbildung 20: Anwendungsgebiete: Prägung durch Informations- bzw. System-Aspekte

2.4.1 Informationssammlung

Insbesondere im Internet stellt sich oft das Problem eine gewünschte Information zu finden, wobei diese unter Umständen aus mehreren verstreuten Teilen zusammenzusetzen ist. Dieser Suchprozess ist normalerweise kontinuierlich durch den Benutzer selbst voranzutreiben, wobei Entscheidungen über die weitere Vorgangsweise zu treffen sind. Dies ist, wie Jennings und Wooldridge betonen ([Jennings/Wooldridge 1998]), mit ein großer Teil des sogenannten „information overload“. Der Suchvorgang selbst ist nur ein Weg zum gewünschten Ziel, der auch von Agenten übernommen werden kann.

Agenten können auf diesem Gebiet auf folgenden Stufen tätig werden (für ein Beispiel siehe den Agenten zur Kaufunterstützung, 6.5):

? Initiale Suche nach Informationen aus vielen verschiedenen Quellen, wie etwa WWW und Newsgruppen, aber insbesondere auch aus Spezial-Datenbanken. Diese Quellen können sowohl lokal als auch über Netzwerke erreichbar sein. Bei ersteren kann die Arbeit von

Agenten an konventionelle Suchmaschinen delegiert werden, doch bei letzteren ist eine Auswahl der für diese Datenbank relevanten Elemente der Gesamtabfrage und die anschließende Formulierung in der für diese Datenbank maßgeblichen Abfragesprache nötig.

? Ausfiltern unerheblicher, veralteter, nicht mehr vorhandener (z. B. tote Links) oder doppelter (schon gefundener) Informationen, wobei komplexere Kriterien angewendet werden, z. B. Ähnlichkeitsmaße wie sie schon in [Salton 1983] beschrieben werden.

? Zusammenstellung der gefundenen Informationen für den Benutzer, wobei eine Reihung nach Relevanz und Wichtigkeit oder ähnlichen Informationen erfolgt, aber auch etwa nach früheren Suchvorgängen, welche von diesem Agenten durchgeführt wurden (Personalisierung; siehe auch „Personal Web IR Agent“ in [Hsiang/Hsieh-Chang 1999]; und [O'Meara/Patel 2001]). Auch kann der Agent die Daten selbst gleich mitliefern bzw. eventuell bei größeren Dokumenten auch nur Teile davon präsentieren.

Der Vorteil, den intelligente Agenten auf diesem Gebiet ermöglichen, ist in der selbständigen Durchführung zu sehen. Anstatt beim Suchvorgang dauernd neue Entscheidungen treffen zu müssen, zu warten bis die Information abgerufen ist, die relevanten Stellen und Verweise zu suchen, und wieder von vorne zu beginnen, kann ein Großteil der Aufgaben von Agenten durchgeführt werden. Diesen macht das Warten nichts aus und sie können längere Suchen auch Nachts oder bei geringerer Netzwerks- oder CPU-Belastung durchführen.

Vielfach wird hier auch eine Aufteilung auf mehrere Agenten sinnvoll sein, sodaß einerseits eine Partitionierung nach Lokalitäten (mehrere gleichartige Agenten durchsuchen unterschiedliche Datenbestände) als auch nach Aufgaben erfolgt (Agenten zur Suche, Agenten zur Integration der Ergebnisse, Agenten zur Kommunikation mit dem Benutzer). Dies kann etwa nach der Interaktion der einzelnen Agenten erfolgen (Interaktions-orientierte Programmierung; IOP [Singh/Huhns 1999]).

2.4.2 Informationsfilterung

Agenten können insbesondere auch dazu verwendet werden aus größeren Datensammlungen oder -mengen diejenigen Teile herauszufinden, welche für den konkreten Benutzer von Interesse sind und somit als eine Art „persönlicher Assistent“ ([Maes 1994]) dienen. In Frage kommen insbesondere folgende Aufgaben:

? Mail Bearbeitung (Vorschläge für Weiterleitung, Löschen, Dringlichkeit)

? Organisation von Treffen (Terminvereinbarung) (siehe dazu [Mitchell et al. 1994])

? Buch/Musik - Auswahl (je nach persönlichen Vorlieben)

Dieser Aspekt wird oft auch als „Personalisierung“ bezeichnet und besitzt in Verbindung mit Datenbanken im Internet oder Internet-Portalen große praktische Bedeutung. Allerdings auch nicht immer in Zusammenhang mit oder Implementierung durch Agenten.

Das Augenmerk sollte eher auf eine Zusammenarbeit *mit* dem Benutzer gerichtet sein, als auf eine Arbeit *für* den Benutzer: Beim heutigen Stand der Technik werden dem Agenten mit Sicherheit noch Fehler unterlaufen, welche dann unter Umständen nicht mehr korrigiert werden könnten. Auch ist es bei einer Zusammenarbeit für den Agenten möglich, durch Beobachtung der Aktionen des Benutzers und Vergleich mit den eigenen Vermutungen darüber für die Zukunft zu lernen (siehe etwa das Projekt Letizia, bei dem ein intelligenter Agent beim Surfen des Benutzers durch das WWW zusieht und ähnliche Seiten empfiehlt [Liebermann 1999]). So ist es möglich Agenten sinnvoll einzusetzen ohne daß eine vorherige „Programmierung“ notwendig wäre. Hierbei stellen sich jedoch mehrere Probleme: Am Anfang besitzt der Agent kein Wissen und ist daher praktisch nutzlos. Erst durch längere Beobachtung und explizites Training erwirbt er Wissen. Es dauert daher längere Zeit bis der Agent eine tatsächliche Arbeitsersparnis herbeiführt. Besonders bei Agenten zur Mail-Bearbeitung entsteht das Problem, daß es sich vielfach um Einzel-Aufgaben oder nur kurze Serien handelt, sodaß die gelernten Verhaltensweisen bereits wieder obsolet sind sobald der Agent sie endlich beherrscht. Dies beobachtet auch schon [Mitchell et al. 1994]: Der Zeitraum zum Lernen zuverlässiger Regeln muß kürzer sein als der Zeitraum, in welchem die zu bearbeitenden Regelmäßigkeiten konstant bleiben. Auch bringt eine bloße Beratung keine besonderen Vorteile für Benutzer: Erst die Ersatz-Durchführung resultiert in echter Arbeitsersparnis. Um Fehler zu vermeiden muß jedoch das Limit für die Richtigkeit seiner Vermutungen, ab dem der Agent eine Aufgabe auch tatsächlich durchführt, relativ hoch angesetzt werden. Mit steigender Intelligenz von Agenten (und falls er sich in der Praxis als zuverlässig erweist), ist jedoch eine Herabsetzung, eventuell nur für bestimmte Bereiche, möglich.

Vorteile von Agenten in diesem Zusammenhang sind die Arbeitserspranis durch die Vorsortierung/Kategorisierung von Nachrichten bzw. der selbständigen Durchführung davon (etwa bei der Termin-Abstimmung für Konferenzen). Durch die Kommunikation zwischen Agenten ist auch ein Transfer von Wissen oder zumindest Information von Agenten anderer Personen möglich, sodaß sich eine echte Verbesserung gegenüber einem Ein-Personen-System ergibt.

2.4.3 Beratung

Es handelt sich in diesem Falle um eine Mischung aus Informationssammlung und Informationsfilterung, wobei ein größerer Einsatz von „Intelligenz“ in Form von Expertensystemen oder mehr Rechenzeit erfolgt. Ihrer Größe und ihrem Grunddaten-Bedarf entsprechend handelt es sich meist um stationäre Agenten. Wichtiges Element solcher Systeme ist, daß sie in der Lage sind, ihre Ratschläge auch zu begründen, sodaß eine Kontrolle der Korrektheit bzw. Plausibilität durch den Benutzer möglich ist.

Ein Beispiel für Agenten diesen Typs sind etwa Systeme zur Unterstützung beim Portfolio Management [Sycara et al. 1998]. Im Gegensatz zu den beiden vorhergehenden Typen, wo es sich meist um einzelne Agenten handelt, findet hier ein System von Agenten Anwendung. Der Grund ist, daß es sich um große Mengen sehr verschiedener Daten handelt, für die auch unterschiedliche Strategien Verwendung finden. Es ist daher nicht günstig einen einzelnen Gesamt-Agenten zu konstruieren, welcher eine enorme Komplexität besitzen müßte und auch Laufzeitprobleme hätte, sollte er alle Aufgaben rechtzeitig erledigen können. Vielmehr findet eine Aufgabenteilung durch Spezialisierung statt, bei der für jede Spezialaufgabe ein eigener Agent zuständig ist. Hierdurch wird das System auch zuverlässiger, da der Ausfall eines Elementes das Gesamtsystem zwar beeinträchtigt, aber nicht komplett zum Stillstand bringt (kein „single point of failure“ [Sycara et al. 1998]). Der Nachteil eines solchen Systems ist, daß sich fast zwangsläufig Konflikte oder Widersprüche zwischen Agenten ergeben. Ein Agent ist etwa der Meinung, eine bestimmte Aktie sollte gekauft werden, ein anderer würde sie verkaufen. Alle diese Schwierigkeiten müssen gelöst werden. Für diese Aufgaben werden Techniken der verteilten künstlichen Intelligenz (DAI) verwendet, welche Konfliktauflösung, Verhandlung, als auch Argumentation beinhalten. Die Aufteilung der Aufgaben erfolgt in folgender Weise: Informationsagenten sind jeweils für bestimmte Typen von Informationsquellen zuständig (Aktienkurs-Ticker, Marktbeobachtung, Firmen-Mitteilungen, Aktienkurs-Historie, etc.), während Aufgaben-Agenten („Task Agents“) jeweils bestimmte Aufgaben durchführen und verantwortlich für Qualität, Filterung und Prioritäts-Zuordnung sind und zu diesem Zweck auf verschiedene Informationsagenten und eventuell andere Subaufgaben-Agenten zugreifen. Eine Zusammenfassung und endgültige Entscheidung erfolgt dann durch einen Portfolio Management Agenten, welcher dem Benutzer Empfehlungen für durchzuführende Transaktionen gibt. Ein besonderer Vorteil des von Sycara et al. vorgeschlagenen Systems liegt darin, daß auch bei Ausfall eines Sub-Agenten die komplette Aufgabe weiter durchgeführt werden kann: Fällt etwa ein Agent zur Bilanzanalyse aus, so kann seine Aufgabe von einem anderen Bilanzanalyse-

Agenten übernommen werden. Dieser ist zwar auf einen anderen (Wirtschafts-) Sektor spezialisiert, daher ist das Ergebnis von geringerer Qualität, aber es entsteht kein Totalausfall.

Besonderes Augenmerk liegt daher neben der parallelen Ausführung von Aufgaben auf der Robustheit des Gesamtsystems. Ein weiteres wichtiges Gestaltungselement ist die Spezialisierung: Für eigenständige Aufgaben werden gesonderte Agenten zur Kapselung der Komplexität in kleineren Portionen zur Ermöglichung der Wiederverwendung, und zu übersichtlicher Gestaltung des Systems verwendet. Wichtig ist jedoch der Gegensatz zur Objektorientierung ([Jennings/Wooldridge 1998]): Agenten kapseln auch Verhalten, d. h. Agenten führen nicht Methoden eines anderen Agenten aus, sondern bitten andere Agenten, bestimmte Aufgaben für sie zu erledigen. Dies kann dann entweder geschehen oder auch nicht und liegt in der autonomen Entscheidung des ersuchten Agenten.

2.4.4 Groupware

Auch im Bereich der Groupware können Agenten nutzbringend eingesetzt werden ([Greif 1994]). Beispiele für ihre Aufgaben sind die konstante oder regelmäßige Überprüfung von gemeinsamen Daten, um etwa bei Änderungen (jeglicher Art oder nur bei bestimmten oder solchen in größerem Ausmaß) den Benutzer sofort oder beim nächsten Einstieg in das System davon zu benachrichtigen und sogleich an die relevante Stelle zu verweisen. Ansätze in diese Richtung bietet etwa das System BSCW [BSCW], auch wenn es sich dort nur um relativ einfache Hinweise handelt (Neu, Änderungsdatum, Notizen, Sperrvermerke) und die Implementierung nicht durch Agenten erfolgt. Insbesondere bei asynchroner Arbeitsweise können Agenten bei diesen Aufgaben sehr hilfreich sein. Auch bieten sie die Möglichkeit, bestehende Applikationen einzubeziehen, indem für Groupware notwendige Elemente durch sie erfüllt werden, indem sie die Programme entsprechend bedienen (z. B. über Skript-Spachen). Ein weiteres Anwendungsgebiet sind Agenten für den Workflow bei Gruppenarbeit, um Dokumente je nach Inhalt oder Zustand an andere Personen weiterzuleiten bzw. zu verteilen, wobei die Regeln dafür auch durch Benutzerbeobachtung ergänzt werden können.

Eines der großen Probleme bei Groupware ist das der „Awareness“: Was machen die anderen Mitarbeiter jetzt an welcher Stelle? Hier können Agenten wie bei asynchroner Bearbeitung eingesetzt werden (siehe oben), aber auch zusätzliche Informationen bieten, wie etwa vermutliche Gründe für die Änderung (gewonnen aus Beobachtung oder durch Befragung von Agenten des anderen Benutzers) oder Darstellung des Änderungsvorgangs, um dem Benutzer den Nachvollzug zu erleichtern.

Obwohl sich sogenannte „Socialware“ [Hattori et al. 1999] von traditioneller Groupware dadurch unterscheidet, daß die Teilnehmer keiner externen Organisation (Firma; gemeinsame und vorgegebene Aufgabe) unterliegen, kann sie bei einem etwas breiteren Verständnis auch unter Groupware eingeordnet werden (oder auch unter Entertainment, siehe unten) und soll daher hier kurz besprochen werden. Es sind dies Systeme, welche einer losen Gruppe von Benutzern bei sozialen Aktivitäten Hilfestellung leisten soll. Typische Aufgaben sind die Strukturierung größerer Teilnehmermengen in kleinere Gruppen mit (temporär) gleichen Interessen, Unterstützung von komplexer Kommunikation (z. B. Suche von Freiwilligen für die Durchführung von Diensten) sowie das Finden von Beziehungen zwischen Personen, z. B. Auskunftspersonen zu bestimmten Themen oder bestimmte Rollen, die manche Personen einnehmen (Initiator, Vermittler etc.) sollen. Agenten dienen hierbei insbesondere als „Stellvertreter“ von Personen und speichern Interessen, Vorlieben und Fähigkeiten, wobei eine wichtige Aufgabe die Personalisierung von Informationen und der Sicht auf die Umgebung ist. Zusätzlich existieren „Gemeinschafts-Agenten“, welche Treffpunkte darstellen und für die Einzelagenten Informationen bereithalten und Verbindungen zu anderen Agenten herstellen (Directory-Dienste). Vorteile bringen Agenten hier insbesondere in der Darstellung für den Benutzer und der leichten Konfiguration: Weitere Teilnehmer oder Plätze können problemlos eingefügt oder zu anderen Lokalitäten verlegt werden. Durch die Kenntnis der Agenten über ihren Besitzer und die interne Kommunikation ist es den Agenten viel einfacher möglich, passende Gesprächspartner, Plätze oder Ereignisse vorzuschlagen.

2.4.5 E-Commerce (Allgemein)

Auch im Bereich des E-Commerce können Agenten Hilfestellung insbesondere für den Käufer leisten. Einige Phasen des Kaufprozesses können beschleunigt oder automatisiert werden ([Maes et al. 1999]), vor allem im Business-to-Business Bereich und für Güter, bei denen die Anforderungen leicht quantisiert werden können. Beispiele hierfür sind standardisierte Produkte und einfache Massenartikel (Beispiel: Schrauben, Schreibpapier, etc.). Bei entsprechender Unterstützung der Datensammlung ist sogar eine komplette Verlagerung möglich, sodaß nur mehr geringfügige menschliche Kontrolle notwendig ist. Der Agent übernimmt die Beobachtung der Verbrauchsdaten, Sammlung von Angeboten, Verhandlung, Bestellung und schließlich die Meldung an die Buchhaltung.

Problematisch, insbesondere im Bereich von mobilen Agenten, ist jedoch die Bezahlung durch Agenten: Zu diesem Zweck muß „wertvolle Information“ (Kreditkartennummer, PIN-Code,

private Schlüssel, ...) mitgeführt werden, welche naturgemäß ein besonderer Angriffspunkt ist und daher (bei mobilen Agenten auch vor dem Host) geschützt werden muß. Auch die Transaktion selbst ist ohne einen vertrauenswürdigen Dritten nicht sicher durchführbar (siehe [Vogler et al. 1998] für ein Protokoll zur Bezahlung durch mobile Agenten in Verbindung mit First Virtual als Bank-Service, welches auf eine trust-service provider aufbaut). Siehe hierzu auch Abschnitt 3.2.4: Ein Protokoll zum sicheren Austausch von Daten.

Ein weiterer Problembereich besteht darin, daß insbesondere große Anbieter kein Interesse daran haben, Agenten ihre Informationen abfragen zu lassen ([Guttman et al. 1999]) und dies mit allen Mitteln zu verhindern suchen. Dies hat den Grund, daß dann meist ein Vergleich ausschließlich über den Preis stattfindet und zusätzliche Services, die angeboten werden, vom Benutzer nicht bemerkt und daher auch nicht in seine Kaufentscheidung miteinbezogen werden. Abhilfe ist eventuell nur dann möglich, wenn Agenten weiterentwickelt werden, sodaß auch andere Elemente einfließen und eine umfassende Darstellung erfolgt. Ansätze hierfür bieten etwa kooperative Versteigerungen (siehe 2.4.6).

Agenten bieten im E-Commerce den Vorteil der Arbeitsersparnis und der Hilfe bei der Auswahl des geeigneten Anbieters. Es werden sowohl Elemente der Informationssammlung (Preisvergleich mehrerer Anbieter) als auch der Beratung (Spezifikation zusätzlicher Eigenschaften, Angabe von Lieferzeit, etc.) eingebaut. Ein weiterer Vorteil kann sein, daß die Beschaffung in mehr rationaler Weise durchgeführt wird: persönliche Vorlieben, Gewohnheiten oder unlautere Praktiken werden umgangen und die Kaufentscheidung auf vorher determinierte Elemente (allein die exakte Definition dieser kann bereits förderlich sein) reduziert.

2.4.6 E-Commerce (Versteigerungen)

Ein besonders stark vertretener Bereich von Agenten im E-Commerce sind Auktionen ([Maes et al. 1999]). Als Beispiele siehe etwa [E-Bay], [Atrada] oder [Ricardo], wo Agenten als Zusatz ermöglicht werden, aber auch [Kasbah] oder [Auction-Bot], welche als Systeme von Agenten konzipiert wurden und auch Verhandlungen zwischen Agenten beinhalten. Ein vergleich von Auktions-Agenten ist in [Greenwald/Stone 2001] enthalten (Ein Beispiels-Algorithmus für Auktions-Agenten: [Anthony et al. 2001]; dieser ist besonders interessant, da über *mehrere simultane* Auktionen entschieden wird: Bei welcher Auktion mitbieten und wieviel. Ein gutes Beispiel für die Autonomie von Agenten; diese Strategien wären händisch kaum mehr durchführbar.). Agenten übernehmen hier die Rolle sowohl von Verkäufern als auch Käufern. Dies ist insbesondere deshalb wichtig, da sich Versteigerungen im Internet oft über

längere Zeiträume hinziehen (etwa mehrere Tage bis Wochen). Problematisch bei diesen Auktionen ist jedoch, daß sich durch die große Anonymität vielfältige Manipulationsmöglichkeiten ergeben, welche bei Wettbewerbs-Auktionen (Verhandlung über ein Element: Preis; der Gewinn des Einen ist der Verlust des Anderen) unvermeidlich sind (siehe [Guttman/Maes 1998]). Auch hier können Agenten eingreifen und die (komplexeren) kooperativen Auktionen (Verhandlung über mehrere Elemente: Preis, Qualität, Garantie, Lieferzeitpunkt; Gewinn für alle Parteien möglich) unterstützen. Eine weitere Schwierigkeit ist, daß es bei Agenten viel wahrscheinlicher ist, daß mit einem Deadlock vergleichbare Zustände auftreten (wie etwa von [Noriega/Sierra 1998] beschrieben): Verwenden zwei Agenten dieselbe Strategie, so kann sehr leicht eine endlose Schleife entstehen, welche aufzulösen schwierig oder ungerecht sein kann (z. B. Zufallsauswahl). Im Gegensatz dazu ist bei menschlichen Partnern dies eher unwahrscheinlich und meist leicht zu lösen.

Trotz dieser Nachteile sind Agenten oder Agenten-ähnliche Tools in diesem Bereich weit verbreitet. Der Grund ist, daß hierfür keine große „Intelligenz“ notwendig ist. Bereits mit relativ einfachen Algorithmen oder regelbasierten Systemen lassen sich Vorteile für die Benutzer wie automatisches Bieten trotz Abwesenheit, Verhandlung über den Preis oder andere Elemente nach genau vorgegebenen Prioritäten erzielen.

2.4.7 (Produktions-) Steuerungssysteme

In der produzierenden Industrie existieren drei Hauptbereiche, in denen Agenten eingesetzt werden ([Parunak 1998]):

? Design-Koordination von Teilelementen: Mehrere verschiedene Firmen und Einheiten müssen zusammenarbeiten, um ein neues Produkt zu entwerfen. Hierbei existieren viele Abhängigkeiten (Wenn Teil A1 statt A2 verwendet wird, muß B7 oder B8 verwendet werden und nicht mehr B9) und übergreifende Vorgaben (z. B. Gesamtgewicht des Produktes). Werden Einzelteile oder Subsysteme durch Agenten modelliert, so können Markt-Mechanismen verwendet werden, um Ressourcen zu verteilen (Agenten „kaufen“ etwa Gewicht von anderen Elementen, müssen dafür aber eventuell Stromverbrauch verkaufen). Es ist daher zu jedem Augenblick möglich festzustellen, ob die Gesamtvorgaben eingehalten werden. Ebenso kann das Zusammenspiel der Teile überprüft werden.

? Simulation komplexer Systeme: Einzelelemente werden hier wiederum durch Agenten repräsentiert, wodurch auch eine Integration von physischen Modellen möglich wird. Dies er-

laubt es, alle Eigenschaften eines Teiles im Agenten zu modellieren und für verschiedene Simulationen (Stromverbrauchssimulation, Bewegungs-Simulation, Materialbelastungs-Simulationen, etc.), welche in verschiedenen Programmen durchgeführt werden, die jeweils benötigten Eigenschaften bereitzustellen. Die Modellierung als Agent vereinfacht auch die zeitliche Synchronisation: Durch Kommunikation zwischen den Agenten kann eine passende Geschwindigkeit oder auch eine temporäre Verzögerung vereinbart werden.

? Kontrolle von Produktions-Systemen: Hier finden Agenten eine besonders vielfältige Anwendung, da sowohl Maschinen (Ressourcen), Teile, als auch Prozesse jeweils als eigener Agent modelliert werden. Damit ist es sehr leicht möglich verschiedene Strategien (und diese teilweise auch gleichzeitig) zu verfolgen: Vorausplanung, Pull-Prinzip (Einspeisen eines Auftrags am Ende, jedes Element fordert die notwendigen Teile von vorhergehenden an; Kanban) oder auch lokale Planung nach den jeweiligen Einschränkungen einer Maschine. Auch ist dadurch eine leichtere Anpassung an veränderte Aufträge oder Probleme möglich (etwa bei Ausfall einer Maschine oder Verzögerung einer Zulieferung).

Ein weiterer Bereich in dem Agenten Bedeutung besitzen ist die Fracht-Logistik ([Burmeister et al. 1998]). Sowohl Transportmittel als auch Transportgüter wie Speditionen werden als Agenten modelliert, wobei jeder Typ besondere Fähigkeiten besitzt. So ist etwa nur der Transportmittel-Agent in der Lage, Routen zu planen (und daher Transportzeiten zu berechnen). Speditions-Agenten wiederum kooperieren mit anderen Speditions-Agenten, um bei verbundenen Unternehmen eine bessere Auslastung zu erreichen. Dies ermöglicht es den verschiedenen Firmen zu kooperieren, aber trotzdem jeweils nur die unbedingt notwendigen Informationen weiterzugeben. Der große Vorteil von Agenten in diesem Zusammenhang ist, daß keine globale Planung erfolgen muß, was insbesondere bei vielen zeitlich versetzt eintreffenden Aufträgen (Realität bei Speditionen) höchst kompliziert wäre und ständige Umplanungen bedingen würde. Beispielhafte Systeme verwenden etwas das contract-net protocol (Siehe auch 2.5.3), wonach jeder Transportagent sich um einen Auftrag bewirbt und lokal die für ihn dadurch entstehenden Kosten berechnet. Der Agent mit den geringsten Kosten erhält anschließend den Zuschlag und damit den Auftrag.

Auch im Bereich der Telekommunikation finden Agenten Anwendung (für einen Überblick über Projekte und Anwendungsbereiche siehe [Weihmayer/Velthuijsen 1998]). Einsatz finden Agenten auf Grund der natürlichen Verteilung der Elemente: Leitungen, Vermittlungsstellen sowie Inhaber und der komplexen Struktur: verschiedene Bandbreiten, Systeme, Protokolle, etc. Hauptaufgabe ist einerseits das Finden und Aufbauen einer passenden Verbindung bei

komplexen Anforderungen sowie die Behebung von Fehlern durch Rekonfiguration. Hauptaspekte der Agenten sind daher Robustheit und besondere Autonomie, weshalb auch Expertensysteme zum Einsatz kommen. Aufgrund der Verteilung ist auch die Kommunikation ein wichtiges Element, jedoch auch ein Problem, da sie meist über dieselben Kanäle erfolgt, welche verwaltet werden (und bei deren Ausfall der Fehler umgangen werden soll).

2.4.8 Entertainment

In diesem Bereich lassen sich verschiedene Elemente einordnen. Entgegen [Brenner et al. 1998], die darunter Hilfe bei der Informationssuche im Internet zu Unterhaltungsthemen verstehen, wie etwa Film-, Musik, Kinoinformationen (welche hier unter Informationssammlung bzw. -Filterung eingereiht wurden), soll hier mehr auf den eigentlichen Inhalt und nicht das Auffinden abgestellt werden. Anwendungsbereiche sind demnach die Simulation von Gegnern oder Partnern in Computerspielen, wobei als Umgebung hier nicht die reale Welt, sondern die Umwelt innerhalb des Spieles verstanden wird (z. B. MUD, Chat), Beratung bei der Auswahl von Freizeitaktivitäten oder eventuell auch sogenannte „Socialware“ ([Hattori et al. 1999], siehe auch oben unter Groupware).

2.5. Standards für Kommunikation, Kooperation und Mobilität

In diesem Abschnitt werden bestehende Standards für zwei wichtige Punkte von Agenten beschrieben: Kommunikation und Mobilität. Wichtig für ein offenes System ist auch die Zusammenarbeit mit anderen Implementierung, welche auf diese Weise zu erreichen erhofft wird.

2.5.1 Standard für Mobilität von Agenten

Um es unterschiedlichen Agentensystemen zu ermöglichen, Agenten untereinander auszutauschen, ist ein gemeinsamer Standard für die Übertragung notwendig. Dies bezieht sich einerseits auf die technische Ausführung, andererseits aber auch auf den Aufbau der Agenten, sodaß diese dann lokal wieder gestartet werden können. Bisher definieren fast alle Agentensysteme ihre eigenen Schnittstellen, sodaß eine Kooperation nur mit gleichartigen Systemen möglich ist. Mit anderen Implementierungen, selbst bei großer Ähnlichkeit (z. B. beide in Java, Agenten werden als serialisierte Objekte verschickt), ist keinerlei Integration möglich.

Ein Ansatz hierfür ist die Spezifikation der MASIF (Mobile Agent System Interoperability Facilities Specification; [MASIF-Spec]), welche sowohl das Management, die Verfolgung und

den Transfer von Agenten, jedoch nicht die Kommunikation zwischen diesen behandelt. Diesem Standard liegt das in Abbildung 21 dargestellte Aufbaumodell zugrunde.

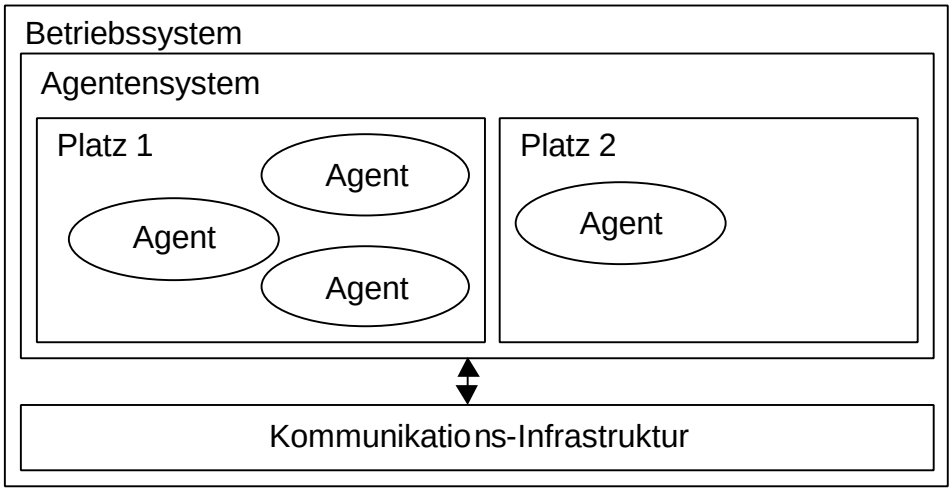

Abbildung 21: MASIF Aufbau-Modell [MASIF-Spec]

Zum Auffinden von Agenten können diese bei einer zentralen Stelle registriert werden. Für jede Region, welche mehrere Rechner bzw. Agentensysteme enthalten kann, wird ein MAF-Finder (Mobile Agent Facility) erzeugt. Bei dieser Stelle können sowohl Agenten, Agentensysteme als auch Plätze registriert und gesucht werden (jedoch nicht Systeme von Agenten).

Als Schnittstelle zu den eigentlichen Agentensystemen wird ein relativ schmales Interface definiert, welches zu implementieren ist. Es stellt Funktionen zum Erzeugen von Agenten (auf Anforderung eines anderen Rechners), Laden von benötigten Klassen, Auflistung von Agenten und Plätzen (zusätzlich zum MAF-Finder), Empfangen eines Agenten und Managen des Lebenszyklus von Agenten (suspend, resume, terminate; sowie diesen Status abfragen) zur Verfügung. Da nicht jedes Basissystem, auch wenn es dieses Interface implementiert, alle Agenten starten bzw. empfangen kann, sind bei den entsprechenden Anfragen zusätzliche Parameter anzugeben, wie etwa die verwendete Programmiersprache, die Art der Serialisierung und der Codierungsmechanismus. Aufgrund dieser Informationen kann das System dann entscheiden die Anfrage durchzuführen oder die Ausführung zu verweigern.

Zu beachten ist, daß keine Sicherheitsmaßnahmen integriert sind. Agenten können nur nach ihrem Besitzer (welcher als short-Wert gespeichert wird!) beurteilt werden. Auch bezüglich der Kommandos erfolgt keine Überprüfung; diese hat außerhalb stattzufinden.

2.5.2 Standard für die Kommunikation zwischen Agenten (Sprache):

Analog zur Interaktion verschiedener Agentensysteme durch den Austausch von Agenten, jedoch von noch größerer Bedeutung, ist die Kommunikation zwischen Agenten. Durch starke Formalisierung ergeben sich hier viel geringere Probleme bei unterschiedlichen Plattformen, sodaß auch sehr verschiedene Implementierungen problemlos miteinander kommunizieren können. Das Hauptproblem ist hier, wie Nachrichten aufgebaut sein sollen und wie die Bedeutung einer Nachricht spezifiziert wird. Der genaue Ablauf eines Protokolls (welche Nachrichten auf welche hin gesendet werden und welchen Inhalt diese enthalten müssen bzw. dürfen) wird jedoch nicht festgelegt da er applikationsspezifisch ist.

Protokolle können grundsätzlich auf zwei Arten spezifiziert werden: Durch eine Zustandsmaschine (wann welche Nachrichten gesendet werden müssen/dürfen, wann ein Abbruch erlaubt ist, etc.) oder durch die Definition von Rollen ([Singh 1998]) und damit verbundenen Verpflichtungen (etwa Preisauskünfte über einen bestimmten Zeitraum zu garantieren oder ein Versprechen auch zu halten). Der letztere Ansatz ist meiner Meinung nach jedoch nicht für eine Spezifikation ausreichend, sollte jedoch zu den normalen Spezifikationen hinzutreten, sodaß die Bedeutung der Nachrichten genauer festgelegt werden kann (Semantik-Integration).

Wichtig für die Kommunikation ist die verwendete Ontologie. Hierunter versteht man ein Basisvokabular sowie eine genau definierte Bedeutung für diese Elemente ([ONTOLOGY.ORG]). Es sind auch Regeln enthalten, welche Bedeutung bestimmte Konstruktionen (Verbindungen von Wörtern) besitzen. Hierdurch wird der Austausch von Wissen zwischen Personen, Computern bzw. zwischeneinander erleichtert, da Unabhängigkeit von konkreten Systemen oder Anwendungen erreicht wird.

Zwei konkrete Beispiele sollen hier kurz erläutert werden: Der FIPA-Standard und KQML (Knowledge Query and Manipulation Language). Beide beruhen, wie auch die meisten anderen Ansätze, auf der Sprech-Akt-Theorie.

2.5.2.1 FIPA-Standard

Eine Nachricht ist aus folgenden Teilen aufgebaut ([FIPA-Msg-Structure]), wobei lediglich der erste Teil (Performative) verpflichtend ist (aber wohl zumindest sender, receiver und content praktisch immer enthalten sein werden):

1. Performative: Der generelle Typ der Nachricht (Näheres siehe unten)

2. Sender: Der Absender der Nachricht
3. Receiver: Der Empfänger der Nachricht
4. Reply-to: Antworten auf diese Nachricht sind an einen bestimmten Agenten zu senden
5. Content: Der Inhalt der Nachricht
6. Language: Die für den Inhalt verwendete formale Sprache
7. Encoding: Die Codierung des Inhalts
8. Ontology: Die Ontologie für die Interpretation des Inhalts
9. Protocol: Das Protokoll innerhalb dessen diese Nachricht verschickt wird
10. Conversation-ID: Die laufende Nummer der konkreten Instanz dieses Protokolls
11. Reply-with: Eine Bezeichnung, um Subteile innerhalb eines Protokolls zu identifizieren (keine inhärente Bedeutung, nur zur Programmvereinfachung)
12. In-reply-to: Der Wert des Feldes Reply-with aus der vorigen Nachricht
13. Reply-by: Ein Zeit- oder Datumswert bis zu welchem der Sender die Antwort erwartet

Der Typ der Nachricht kann aus einem Katalog ([FIPA-Comm-Act]) ausgewählt werden der u. A. die folgenden Elemente enthält: Accept, Agree, Cancel, Call for proposal, Confirm, Disconfirm, Failure, Inform, Not understood, Propose, Query if, Refuse, Reject proposal, Request, Subscribe. Mit diesen Nachrichtentypen lassen sich fast alle Protokolle realisieren, doch können auch private Nachrichtentypen definiert werden. Verpflichtend verständlich für alle Agenten ist lediglich „Not understood“, welches immer dann zu senden ist, wenn die empfangene Nachricht aus irgendeinem Grund nicht ausgewertet werden konnte (Sprache oder Ontologie unbekannt, Syntax falsch, etc.).

2.5.2.2 KQML

Es handelt sich hierbei sowohl um eine Kommunikationssprache wie auch ein Protokoll zum Austausch von Informationen und Wissen (Ausführliche Informationen unter [KQML], Spezifikation: [Finin et al. 1993], [Labrou/Finin 1997a]). Wie der FIPA-Standard (Ein interessanter Kommentar zum FIPA Standard, insbesondere im Hinblick auf die mangelnde Trennung zwischen Agenten-Modell und Kommunikationssprache sowie die formale Spezifikation ist in [Labrou/Finin 1997b] enthalten) basieren Nachrichten auf „performatives“, welche die erlaubten Einflußmöglichkeiten auf andere Agenten darstellen. Als Unterstützung der Zusammenarbeit von mehreren Agenten werden „Facilitators“ ([Finin et al. 1994]) eingesetzt, welche so-

wohl ein Verzeichnis von Agenten darstellen, Nachrichten weiterleiten, aber auch das Finden von geeigneten Agenten für bestimmte Aufgaben übernehmen.

Eine KQML Nachricht wird als String dargestellt und ist in einer an LISP angelehnten Notation codiert. Die einzelnen festen Parameter sind ähnlich denen des später entstandenen FIPA-Standards. Es besteht eine lange Liste an „Kommandos“ (Englisch: performatives), doch ist diese optional. Wird allerdings eines der dort angeführten implementiert, so hat es der Spezifikation zu entsprechen. Im Gegensatz zum FIPA-Standard werden Performative auch in Gruppen eingeteilt (Discourse, Intervention and Mechanics, und Facilitation and Networking) sowie definiert, welche Arten von Folgenachrichten erlaubt sind (Antwort erforderlich, nur Antwort, keine Antwort).

Ein Beispiel für eine KQML-Nachricht mit der ein Agent A einen Agenten B ersucht („achieve“; kein Befehl!), den Parameter „torque“ des Motors „motor1“auf 5 zu verändern, ist:

```
(achieve :sender A
:receiver B
:in-reply-to id1
:reply-with id2
:language Prolog
:ontology motors
:content "torque(motor1,5)" )
```

Es existiert auch ein Vorschlag zur Erweiterung ([Thirunavukkarasu et al. 1995]) von KQML, sodaß auch verschlüsselte bzw. signierte Nachrichten ausgetauscht werden können.

2.5.3 Kooperation zwischen mehreren Agenten:

Kooperation zwischen mehreren Agenten kann auf verschiedenen Ursachen beruhen: Zufällige Verbesserung des Gesamtergebnisses, weil die Einzelaktionen gut zusammenpassen bis hin zu einer geplanten Zusammenarbeit, bei welcher jeder Agent auch die Ansichten und Wünsche der anderen beteiligten Agenten kennt und berücksichtigt. Kooperation kann daher auf die in Abbildung 22 dargestellte Weise eingeteilt werden ([Doran et al. 1997]).

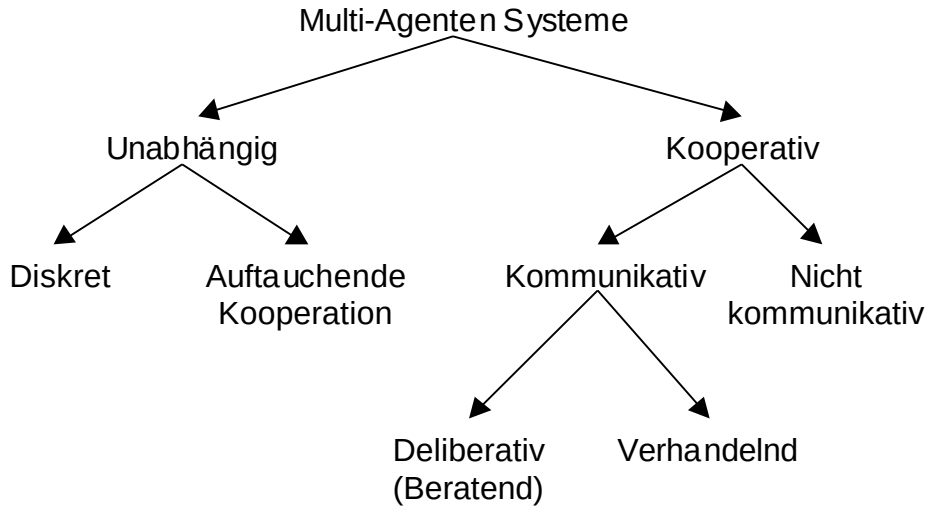

Abbildung 22: Einteilung von Kooperation [Doran et al. 1997]

? Diskret: Jeder Agent arbeitet für sich alleine. Kooperation existiert nur zufällig (ohne Wissen und Willen der Teilnehmer).

? Auftauchende Kooperation: Jeder Agent arbeitet für sich alleine. Die Gesamtheit der Agenten (das System) wurde jedoch so entworfen, daß sich die einzelnen Ziele und Tätigkeiten ergänzen. Nach außen hin ergibt sich der Anschein von Kooperation, während die Einzelagenten dies nicht beabsichtigen. Nach diesem Prinzip arbeitet beispielsweise das System Swarm ([Swarm]). Lauter einzelne Agenten erzeugen zusammen ein besonderes Verhalten (emergent behaviour).

? Nicht-kommunikativ: Agenten arbeiten zwar zusammen, doch erfolgt keine Kommunikation zwischen ihnen. Die Abstimmung erfolgt rein durch die Beobachtung der Umgebung und der Veränderungen darin, welche durch die Aktionen der anderen Agenten ausgelöst werden. Dies ist nur sehr selten sinnvoll, da explizite und direkte Kommunikation zwischen Agenten viel einfacher zu realisieren ist, als der Rückschluß von sichtbaren Ergebnissen auf die dahinterstehenden Intentionen. Der Vorteil ist hier, daß Agenten nicht „lügen“ können: Ihre Aktionen sprechen für sich und fehlerhafte oder absichtlich irreführende Kommunikation existiert nicht. Dies ist insbesondere relevent für inkompatible Systeme, welche nicht in der Lage sind, direkte Kommunikationsbeziehungen aufzubauen.

? Deliberativ: Alle Agenten planen gemeinsam den Ablauf für alle Agenten und müssen sich darüber einig werden. Diese Methode wird bei geschlossenen Systemen angewendet, da alle Agenten tatsächlich auf das gemeinsame Ziel hinarbeiten müssen. Einzelne Agenten können durch den gemeinsamen Plan auch benachteiligt werden und nehmen dies jederzeit

hin. Ein Ansatz nach diesem Kooperationsprinzip ist „Partial Global Planning“ ([Decker/Lesser 1992]).

? Verhandelnd: Zusätzlich zum deliberativen Ansatz kommt noch eine Konkurrenz zwischen den einzelnen Agenten hinzu. In der Regel wird ein einzelner Agent daher nur dann eine Tätigkeit durchführen, wenn sie für ihn vorteilhaft ist (ansonsten verweigert er sie). Dieses Kooperationsprinzip ist hauptsächlich bei offenen Systemen zu finden, oder wo Marktähnliche Strategien verfolgt werden. Im Gegensatz zu deliberativen Systemen, bei welchen bei optimaler Planung immer das bestmögliche Ergebnis erreicht wird, können sich hier auch suboptimale Resultate ergeben, obwohl insgesamt gesehen bessere möglich wären. Ein Beispiel hierfür ist das „Contract Net Protocol“ ([Sandholm 1993]).

2.6. Existierende Agentensysteme für mobile Agenten

Im folgenden werden einige Systeme von mobilen Agenten beschrieben, wobei auch schon ein Augenmerk auf den Vergleich mit dem im Kapitel 5 dargestellten System gelegt wird. Es handelt sich hierbei nicht um eine Aufzählung *aller* derzeit existenten Systeme, sondern nur um eine Auswahl aus denen, die entweder eine kommerzielle Bedeutung besitzen, oder in der Literatur größere Beachtung gefunden haben. (Für einen Überblick über Firmen, welche sich mit Agenten befassen siehe [Guilfoyle 1998], ein Vergleich von mehreren Systemen ist auch unter [Pazandak 1998] zu finden).

Am Ende werden die hier erläuterten Agentensysteme im Hinblick auf die oben dargestellten Kriterien verglichen soweit dies möglich ist (manche Kriterien betreffen Systeme von Agenten und nicht Agentensysteme; diese sind daher hier nicht anwendbar).

2.6.1 Aglets [Aglets], [Aglets.org], [Tai/Kosaka 1999]

Es handelt sich hierbei um ein Framework für mobile Agenten (inzwischen Open Source), welches komplett in Java (JDK 1.x; inkompatibel mit Java2) geschrieben ist. Agenten können frei aufeinander zugreifen und Methoden aufrufen. Um dadurch die Sicherheit nicht zu gefährden wird jedes Aglet von einem Proxy umgeben welcher Zugriffsprüfungen durchführt. Dieses Proxy-Objekt ist meist stationär und auch dafür zuständig Nachrichten weiterzuleiten wenn ein Aglet auf einen anderen Rechner verlagert wurde. Kommunikation kann zusätzlich auch über den Versand von Nachrichten erfolgen, welche immer nur zwischen den Proxys (siehe Abbildung 23) versendet werden (keine direkte Interaktion möglich).

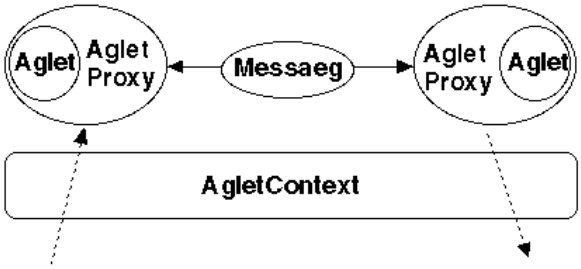

Abbildung 23: Aglet Komponenten ([Aglet] Aglet Specification 1.1)

Es existieren eigene Konstrukte zur Synchronisation der Kommunikation zwischen mehreren Aglets (Monitor-Konzept).

Die Übertragung von Aglets erfolgt mittels des OMG/MASIF Standards (siehe 2.5.1), wobei zwei Implementierungen vorhanden sind: Ein eigenes Protokoll ATP (Aglet Transfer Protocol) und RMI. Siehe hierzu Abbildung 24.

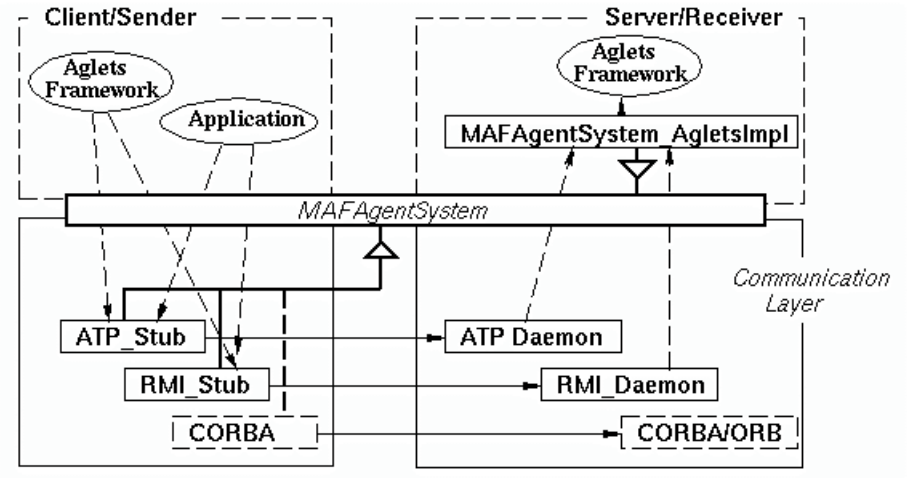

Abbildung 24: Aglet-Transfersystem ([Aglet] Aglet Specification 1.1)

Das Sicherheitssystem bei Aglets beruht auf dem Standard-Sicherheitssystem für Java, wobei Berechtigungen nach dem Besitzer und der Code-Herkunft vergeben werden (physikalisch = Pfad von dem die Dateien geladen wurden). Eine Signierung des Programmcodes ist nicht vorgesehen. Mehrere Server können zu einer Domäne zusammengeschlossen werden, indem für sie ein gemeinsamer geheimer Schlüssel festgelegt wird. Jede Kommunikation zwischen diesen Servern erfolgt anschließend unverschlüsselt, doch zumindest mit einer Integritätsprüfung. Ob ein Benutzer vertrauenswürdig ist wird festgestellt, indem er sich bei einem System anmelden muß. Erst anschließend können Agenten erzeugt werden. Diese Agenten können sich dann innerhalb einer Domäne frei bewegen, doch sollte er einmal diese verlassen haben und zurückkehren (oder es handelt sich um einen extern erzeugten Agenten), so werden dem Agenten weniger Berechtigungen zugeteilt oder der Zutritt überhaupt verwehrt. Die Sicherheitsüber-

prüfung findet daher effektiv nur am Beginn der Agententätigkeit statt. Er kann einen festen Bereich zusammengehöriger Rechner mit gemeinsamer Organisation nicht verlassen.

2.6.2 Grasshopper [Grasshopper]

Dieses kommerzielle System ist auf Java aufgebaut. Ein Agent ist ein unabhängiges Programm, welches autonom Aufgaben durchführen kann. Es wird streng zwischen stationären und mobilen Agenten unterschieden, wobei die Entscheidung schon bei der Programmierung erfolgen muß. Mehrere Agenten-Server („Agency“) werden zu größeren Einheiten zusammengefaßt („Region“). Innerhalb einer Region können Server durch ihren Namen adressiert werden. Es besteht ein automatisch aktualisiertes Agenten-Verzeichnis. Ein Agent kann jederzeit in eine andere Region wechseln, doch muß er dazu die genaue Adresse (Hostname, Portnummer, Protokoll) der Empfangs-Agency kennen.

Es wird sowohl die OMG/MASIF (Interoperabilität) als auch die FIPA-Spezifikation (ACL) unterstützt. Ist ein Agent von einer entsprechenden Klasse abgeleitet, so kann jederzeit (insbesondere periodisch) ein Snapshot erzeugt und auf die Festplatte gespeichert werden, um so gegen Ausfälle und Fehler gesichert zu werden. Eine weitere Einstellung ist, Agenten, welche längere Zeit nicht mit anderen kommuniziert haben, auf einen Hintergrundspeicher zu sichern (Persistenz) und anschließend zu beenden. Werden sie zu einem späteren Zeitpunkt kontaktiert, so erfolgt ein Neustart mit den gespeicherten Daten.

Der Programmcode eines Agenten kann entweder vom lokalen Filesystem oder einem Webserver (über das HTTP Protokoll) geladen werden. Es wird versucht, Programmcode zuerst lokal zu laden, anschließend vom vorigen Aufenthaltsort, weiters von anderen (im Agenten spezifizierten) Orten, insbesondere Webservern, und schließlich von der Home-Agency.

Die Kommunikation zwischen Agenten erfolgt über Methodenaufrufe. Zu diesem Zweck hat ein Agent ein Interface aller öffentlich zugänglichen Methoden zu spezifizieren. Zu diesem Interface wird ein lokaler Proxy erzeugt, welcher anschließend die Kommunikation zu dem anderen Agenten weiterleitet, egal wo sich dieser befindet (solange er in der selben Region bleibt). Sowohl synchrone wie auch asynchrone Kommunikation ist möglich. Es können auch Multicasts verwendet werden. Dies erfolgt jedoch nicht technisch, sondern lediglich logisch über Multicasts. Vielmehr handelt es sich um einen gemeinsamen Proxy für mehrere Agenten, welche bei jedem Methodenaufruf sequentiell hintereinander einzeln verständigt werden.

Zum Schutz von Agenten während der Übertragung wird SSL unterstützt. Die Server können durch ein dem Java2-Sicherheitsmodell entsprechendes System geschützt werden. Die Zuteilung von Rechten erfolgt hierbei nach dem Zertifikat des Besitzers, welches ein Agent mit sich führen muß. Schlüssel und Zertifikate werden über den Java-Keystore verwaltet.

2.6.3 Voyager [Voyager]

Hierbei handelt es sich um einen kommerziellen ORB (d. h. eine Erweiterung von CORBA), welcher mit Agenten-Erweiterungen ausgestattet ist und in Java programmiert wurde. Agenten können verteilt erzeugt werden und sich selbständig auf andere Rechner verlagern. Ein universelles Verzeichnissystem ist mit Hilfe des CORBA-Frameworks (CORBA Naming Service und JNDI=Java Naming and Directory Interface; [JNDI]) integriert. Die Kommunikation erfolgt transparent mit lokalen wie auch entfernten Objekten durch Methodenaufruf, wobei sowohl Broad- als auch Multicasts möglich sind. Zu Vereinfachung der Arbeit ist sowohl verteilte Garbage-Collection wie auch Transaktionsunterstützung integriert.

Der große Vorteil von Voyager ist darin zu sehen, daß es auf einem verbreiteten Standard (CORBA) aufbaut und transparente Kommunikation bietet: Methodenaufrufe zwischen Objekten sind unabhängig vom tatsächlich verwendeten Protokoll (CORBA, RMI, VNM, DCOM; siehe Abbildung 25).

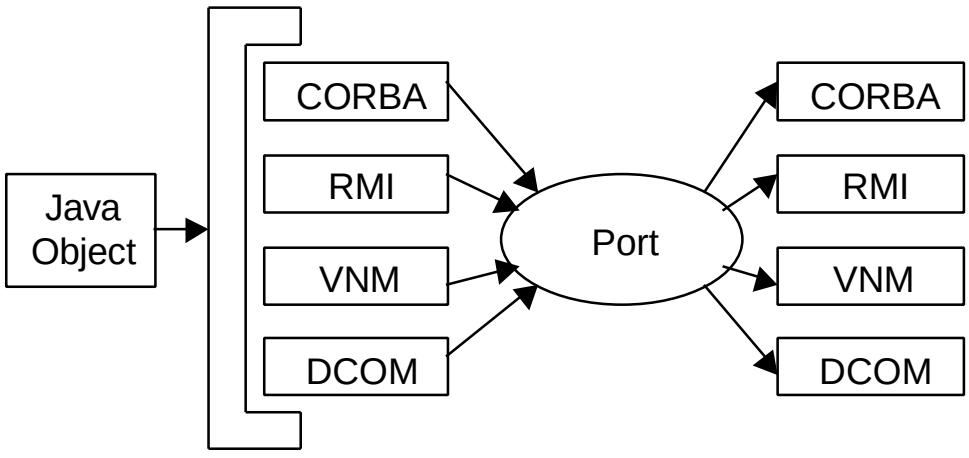

Abbildung 25: Kommunikatiosnmodelle von Voyager [Voyager]

Im Hinblick auf Sicherheit wird die Verwendung von SSL und HTTP Tunneling (Kommunikation durch Firewalls hindurch) für die Kommunikation unterstützt.

2.6.4 Hive [Hive]

Dieses Projekt ist ein Basissystem für mobile Agenten, wobei auch einige (mehr oder weniger) nutzbringende Anwendungen (interne Verwendung im MIT Media Lab) existieren. Ein Netzwerk wird hierbei in einzelne Elemente zerlegt auf denen sich Agenten befinden können ("Cell"; entspricht einem Server mit dem entsprechenden Programm, um Agenten aufnehmen zu können).

Bei Hive wurde noch eine zusätzliche Abstraktion eingeführt: Lokale Dienste (jeglicher Art außerhalb des Agentensystems: Lokale Spezialprogramme, angeschlossene Hardware, …) werden durch sogenannte „Shadows" im System repräsentiert. Dies sind immobile lokale Objekte die Zugriffsmethoden auf diese Dienste bereitstellen. Sie unterscheiden sich von Agenten insbesondere dadurch, daß sie keinen dynamischen Aspekt besitzen (kein Thread), sondern völlig passiv sind. Hiermit soll insbesondere auch eine Unterscheidung zwischen lokalem (und damit vertrauenswürdigem) und entferntem Code möglich sein. Diese Elemente sind zusammen mit Agenten in Abbildung 26 dargestellt.

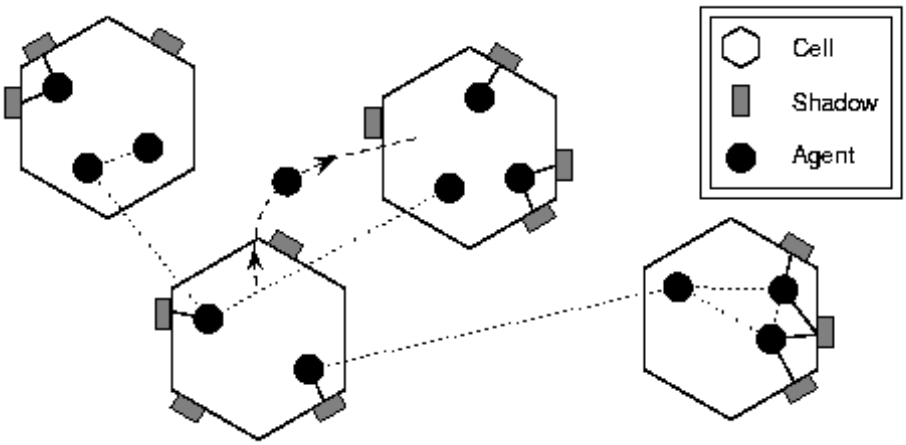

Abbildung 26: Hive Architektur [Hive]

Es ist zur Zeit kein Sicherheitssystem implementiert. Es bestehen jedoch Planungen für ein auf Java2 basierendes System an Berechtigungen, da dieses als notwendig erkannt wurde [Minar et al. 2000]. Da es sich um ein geschlossenes System für eher kleinere Gruppen handelt, ist dieser Aspekt von nicht so großer Bedeutung.

Als Besonderheit ist in Hive ein Versuch integriert Agenten bzw. Shadows auch in ihrer Semantik zu beschreiben („self-describing"), um anderen Agenten die Auswahl bzw. Verwendung zu erleichtern. Hierzu ist eine Beschreibung in XML [Sturm 2000], [Holzner 2001], [XML-Spec] integriert, welche der Agent während der Laufzeit verändern kann. So kann z. B. der

momentane Aufenthaltsort des Agenten (in Bezug auf physikalische Räume, nicht Cells!), oder die Bedeutung der zur Verfügung gestellten Meßwerte angegeben werden. Es wird auch ein Matching-Service zur Verfügung gestellt, um anhand einer Beschreibung einen entsprechenden Agenten zu finden.

2.6.5 Jini [Jini]

Hierbei handelt es sich nicht direkt um ein Agentensystem sondern vielmehr um eine Erweiterung (Klassenbibliothek) für verteilte Programmierung für die Programmiersprache Java. Der Mittelpunkt dieses Frameworks sind „services“, welche sich in einem Netzwerk befinden können. Hierbei kann es sich sowohl um Programme (eventuell Agenten), als auch um Hardware handeln (ebenso möglich: Kommunikationskanal, anderer Benutzer, etc.). Diese können sehr einfach zu einem Netzwerk hinzugefügt oder entfernt werden. Es werden auch Hilfsmittel zur Verfügung gestellt, um passende Services aufzufinden. Die Kommunikation zwischen einzelnen Diensten erfolgt mittels RMI.

Die Dienste selbst sind nicht mobil, jedoch können benötigte Teile für die Inanspruchnahme von speziellen Diensten von anderen Objekten über das Netzwerk geladen werden. Ein Beispiel hierfür ist der Download von speziellen Dialogen, um dem Benutzer die Konfiguration eines zu verwendenden Dienstes zu ermöglichen oder erleichtern.

Die Sicherheit basiert auf Access Control Lists (ACL’s), wobei jeder Dienst angeben muß, für wen er die aktuelle Leistung benötigt (principal). Aufgrund dieser Information wird die Liste überprüft und der Zugriff genehmigt oder verweigert. Dies ist natürlich nicht für offene oder sehr große Systeme geeignet, doch ist die Zielgruppe explizit als „Workgroup“, d. h. eine kleinere Einheit (Büro, Haushalt, …) spezifiziert.

Von Bedeutung ist noch, daß ein extensives System von Events besteht und Unterstützung für Transaktionen (alle beteiligten Objekte müssen dem Abschluß zustimmen, bevor dieser allen anderen Objekten übermittelt wird) integriert ist.

2.6.6 Mole [Mole], [Straßer et al. 1997]

Das Mole-System ist auf Java aufgebaut und bietet Unterstützung für mobile Agenten, welche sich auf (logischen) „Plätzen“ aufhalten. Es wird streng zwischen stationären (Anbieter von lokalen Diensten) und mobilen (Benutzer von diesen Diensten) Agenten unterschieden. Das

System ist nach außen hin abgeschlossen, wobei die eindeutige Identifikation von Agenten durch das System sichergestellt wird.

Die Kommunikation erfolgt sowohl über RPCs als auch über MP. Ersteres ist hauptsächlich für die Kommunikation mit stationären Agenten gedacht, während Nachrichten (Messages) für die Kommunikation zwischen mehreren mobilen Agenten intendiert sind. Um die Identifikation von Agenten zu erleichtern können diese öffentliche „Badges“ tragen. Dies dient dazu, Agenten zu identifizieren, welche an einer bestimmten Aufgabe oder in einer Gruppe zusammenarbeiten. Agenten können ihre badges selbst verändern. Um die Koordination zu vereinfachen erfolgt Kommunikation in „Sessions“, wobei jeder Agent andere auffordern kann, an einer solchen teilzunehmen, aber jeder Agent dies ebenso ablehnen kann. Über Ereignisse (events) ist auch anonyme Kommunikation möglich.

Das Sicherheitssystem ist nur sehr rudimentär und baut auf der Unterscheidung zwischen stationären und mobilen Agenten auf: Stationäre Agenten haben Zugriff auf die Hardware und unterliegen keinen Einschränkungen, können jedoch auch nur lokal erzeugt werden. Mobile (=Benutzer-)Agenten haben demgegenüber nur sehr eingeschränkte Rechte und können lediglich mit andern Agenten kommunizieren. Zusätzlich kann auf jedem System entschieden werden, welche Agenten Zutritt haben.

2.6.7 D’Agents [DAgents]

Dieses Projekt unterscheidet sich von anderen hier besprochenen dadurch, daß die Programmiersprache TCL ist (auch Scheme und Java; Modifikation der VM für starke Mobilität erforderlich) und starke Mobilität unterstützt wird (transparente Weiterverarbeitung nach einem Transfer auf einen anderen Rechner). Agenten werden nicht global eindeutig identifiziert, sondern erhalten auf jedem Rechner eine lokale Nummer. Über ein globales Verzeichnis (eigene Applikation) ist das Suchen von Agenten möglich, sofern sich diese registriert haben.

Die Kommunikation erfolgt über binäre Streams („direct conection“) oder asynchrone Nachrichten („events“), wobei keinerlei Syntax festgelegt ist. Während der Übertragung von Agenten, oder einer Kommunikation zwischen Agenten auf verschiedenen Rechnern, werden diese Daten durch Verschlüsselung gesichert.

Wird ein Agent erzeugt, so muß der Besitzer diese Anforderung mit seinem privaten Schlüssel signieren. Nur wenn die Prüfung erfolgreich verläuft, wird der Agent erstellt und kann arbeiten. Wandert dieser Agent auf einen anderen Rechner so ist dies nur möglich, wenn der Zielrechner

dem Quellrechner vertraut, daß dieser die Überprüfung durchgeführt hat, oder er den Besitzer selbst kennt. Es besteht daher eine transitive Vertrauensbeziehung nach welcher ein Agent Rechte erhält. Gelingt es einem Agenten in dieses System einzudringen, so ist er ab diesem Zeitpunkt ungefährdet da keine weiteren Überprüfungen durchgeführt werden. Berechtigungen werden Besitzern über Listen zugeordnet, wobei extensive Vorkehrungen bestehen, da nicht nur gefährliche Aktionen sondern auch Ressourcen geschützt werden. So wird etwa zusätzliche Rechenzeit genauso wie der Zugang zu potentiell gefährlichen Kommandos nur über den Sicherheits- bzw. Ressourcen-Manager gewährt.

2.6.8 Vergleich der Agentensysteme

Oben unter 2.2 wurden die folgenden Kriterien für Agentensysteme bzw. Systeme von Agenten angeführt, ebenso wie unter 2.1.5 die notwendigen Eigenschaften von Agenten. Nicht alle Kategorien sind jedoch zur Einteilung von Agentensystemen geeignet (oder zumindest nicht unmittelbar). Für einen Überblick siehe Tabelle 1.

? Simpel-Intelligent: Hier handelt es sich um eine Eigenschaft von einzelnen Agenten. In besonderen Fällen kann sich Intelligenz auch erst aus dem Zusammenwirken einzelner unintelligenter Elemente („Agenten“) ergeben (emergent behavior). Es wird daher nur angegeben, ob im Agentensystem besondere Vorkehrungen getroffen wurden (z. B. Integration eines Expertensystems, Methoden zur Auswertung von Regeln, etc.). Ebenso Eigenschaften von Agenten und nicht von Agentensystemen sind etwa Autonomie und Zielorientierung.

? Stationär – Mobil: Auch hier handelt es sich um eine Agenten-Eigenschaft, doch sind im Agentensystem größere Vorkehrungen erforderlich, um mobile Agenten zuzulassen. Es wird daher die Unterstützung hierfür in der Tabelle angeführt (wobei bei Agentensystemen für mobilen Agenten natürlich auch immobile möglich sind).

? Einzelagenten – Systeme von Agenten/MAS: Diese Kategorie kann hier grundsätzlich keine Erwähnung finden, da ausschließlich Agentensysteme betrachtet werden und keine Applikationen (für welche die Einteilung Sinn besitzt). Weiters ist jedes Agentensystem darauf ausgelegt, mehrere Agenten zu beherbergen. Eine Kooperation dieser ist auch in jedem Fall möglich.

? Gleichordnung – Hierarchie: Dies ist analog zur Mobilität zu sehen. Auch hier wird nur angegeben, ob eine Hierarchie von Agenten (Erzeugung durch andere Agenten mit Herr-

schafts- oder Abhängigkeitsverhältnis; Vererbung von Sicherheitsmerkmalen; etc.) möglich ist oder nicht. Daß Agenten dynamisch Über-/Unterordnungs-Beziehungen bilden, um eine Aufgabe zu lösen (z. B. Koordinator und Aktions-Agenten), wird nicht als Hierarchie in diesem Sinne verstanden.

? Reaktiv – Deliberativ – Hybrid: Dies ist eine Einteilung der Intelligenz des Agenten: Wie wird sie realisiert. Es ist keine Abbildung auf Agentensysteme möglich, welche über die Unterstützung hinausgeht. Da kein untersuchtes System irgendeine Unterstützung hierfür bietet, wurde auf die Kategorie verzichtet.

? Kooperativ – Wettbewerb – Aggressiv: Hierbei handelt es sich um eine Einteilung von Systemen von Agenten. Unterstützung durch das Agentensystem ist nur als „Schiedsrichter" bzw. als Garant für die Einhaltung von Mindeststandards (z. B. sichere Identifizierung von Agenten, Trennung der Ausführung, Sicherheitsprüfungen, etc.) möglich.

Ähnliches gilt für die Eigenschaften von Agenten: Auch hierbei kann nur bedingt vom Agentensystem auf die damit implementierten Agenten (oder eben nicht Agenten) geschlossen werden.

? Proaktivität: Es wird angegeben, ob das Agentensystem selbst dies unterstützt, z. B. indem es Timer zur Verfügung stellt. Eine mögliche Funktion ist auch die, einen Agenten komplett zu stoppen (kein laufender Thread/Prozeß mehr), um Ressourcen zu sparen. Dieser würde dann zu einem festgelegten Zeitpunkt oder auf bestimmte Ereignisse hin aufgeweckt werden. Proaktivität als solche ist praktisch immer durch die Programmiersprache möglich (im schlimmsten Fall Busy-wait oder regelmäßige Zeitabfrage).

? Hardware-Schnittstelle: Es kann durch das Agentensystem eine Schnittstelle zur Hardware bereitgestellt werden, sodaß alle Anforderungen über das Agentensystem erfolgen müssen. Dies wäre insbesondere vom Sicherheitsstandpunkt aus gesehen günstig. Problematisch ist, daß dann eine Beschränkung auf bestimmte Hardware oder einzelne Funktionen erforderlich wäre. Dies kann aber ebenso, wie bei der Proaktivität, der Programmiersprache bzw. dem Betriebssystem überlassen bleiben (wobei hier jedoch manche Sprachen einen direkten Zugriff verbieten).

? Interaktivität: Bei allen untersuchten Agentensystemen handelt es sich um Systeme, welche auch interaktiv zu bedienen sind. Dennoch ist hier eine Unterscheidung möglich: Wird das Benutzerinterface vom Agentensystem zur Verfügung gestellt (Repräsentation, Modifikation der Attribute, Konfiguration von Beziehungen zwischen Agenten, etc.), oder ist dies

vom Autor des Agenten zu erstellen und stellt das Agentensystem nur ein „Rahmenfenster“ zur Verfügung?

? Verzeichnisdienste: Wird ein Verzeichnis aller Agenten (oder zumindest derer, welche sich eingetragen haben) zur Verfügung gestellt und wie umfassend ist es? Das Verzeichnis kann sich ausschließlich auf den lokalen Rechner beziehen, oder Server-übergreifend sein (Organisation oder weltweit). Weiters ist noch eine Implementierung anstatt durch das Agentensystem selbst durch spezielle Agenten möglich, welche als Aufgabe eben die Führung eines Verzeichnisses besitzen.

? Kommunikation: Können Agenten nur mit lokalen Agenten kommunizieren oder auch mit solchen auf anderen Rechnern? Hierbei wird vorausgesetzt, daß der Name/Identifikator bzw. die genaue Position (z. B. IP-Adresse) des Agenten bekannt ist.

? Snapshots: Hierbei werden Hilfsmittel für die Robustheit angesprochen. Es können verschiedene Ausprägungen existieren. Snapshots dienen dazu, den Zustand eines Agenten festzuhalten, sodaß er bei einem Fehler wiederhergestellt werden kann (oder z. B. als Beweismittel). Persistente Snapshots sind eine Erweiterung, sodaß auch bei einem Fehler des Agentensystems (und darauffolgendem Absturz) das System an Agenten wiederhergestellt werden kann. Im Gegensatz dazu könnten Wiederaufsetzpunkte auch durch den Agenten selbst reaktiviert werden. Dies ist jedoch ein besonderes Problem in rechtlicher Hinsicht, da hierdurch z. B. ausgegebenes Geld plötzlich wieder zurücktransferiert werden müßte, Vereinbarungen durch einseitige Aktionen wegfallen würden, etc. Eine Limitierung bzw. Begrenzung bei externen Aktionen wäre erforderlich.

Kriterium	**Aglets**	**Grasshopper**	**Voyager**	**Hive**	**Jini**	**Mole**	**D' Agents**
Unterstützung für Intelligenz von Agenten	✍	✍	✍	✍	✍	✍	✍
Stationär (S) – Mobil (M)	M	M	M	M	S	M	M
Gleichordnung (G) – Hierarchie von Agenten (H)	G	G	H	G	G	H	G
Proaktivität durch Agentensystem	✍	✍	✍	✍	✍	✍	✍
Schnittstelle zur Hardware (Re-/Aktivität) Programmiersprache (P) – Agentensystem (A)	P	P	P	P	P	P	(P)[2]
GUI für Agenten durch System (Interaktivität)	✍	✍	✍	✍	✍	✍	✍
Verzeichnisdienste: Keine/durch Agenten (K) - Lokal (L) – Global (G)	L	G	G	L	G	G	K
Kommunikation: Lokal (L) – Global (G) (Kommunikativität)	G	G	G	G	G	G	G
Snapshots (S) - Persistente Snapshots (P) - Wiederaufsetzpunkte (W) (Robustheit)	S	P	P	✍	S	✍	✍

Tabelle 1: Vergleich der vorgestellten Agentensysteme

2.7. Schwierigkeiten bei Agentensystemen

Bei Agentensystemen bzw. Systemen von Agenten ergeben sich viele Fragestellungen, welche über reine Informatik-Aspekte hinausgehen. Vier ausgewählte Punkte davon sollen hier kurz erläutert werden:

? Schwierigkeiten technischer Natur; die „natürliche" Domäne des Informatik-Bereichs

? Psychologische Probleme: Ängste und Wünsche der Benutzer sind wichtige Elemente.

? Rechtliche Fragen: Realer Einsatz bringt auch Fehler mit sich, welche u. U. juristische Konsequenzen nach sich ziehen.

? Ethische Gesichtspunkte: Manche Gestaltungen sind zwar technisch möglich, doch sind diese auch erwünscht?

[2] Je nach der verwendeten Programmiersprache (z. B. nicht bei TCL ohne Modifikation des Interpreters)

2.7.1 Technischer Natur

Bei intelligenten Agenten stellen sich auf der technischen Seite (neben mehreren vergleichsweise kleineren; Abbildung 27) zwei Hauptprobleme: die mangelnde Intelligenz und Sicherheitsfragen.

1. Mangelnde Intelligenz: Die Verbindung von künstlicher Intelligenz und Agenten steht vor besonderen Problemen, sodaß die ansonsten bereits gut entwickelten Expertensysteme hier nicht richtig zum Tragen kommen. Künstliche Intelligenz baut meist auf einer großen Wissensbasis in einem stark eingeschränkten Gebiet auf („Spezialist“), während für Agenten oft ein anderer Ansatz notwendig wäre: Er muß über sehr viele Dinge Allgemeines kennen und verarbeiten („Generalist“). Auch verlangen KI-Systeme oft Eingaben in bereits speziell vorverarbeiteter Struktur wie [Brenner et al. 1998] ausführen. Für einen Agenten ist dies jedoch gerade eine zentrale Aufgabe, welche durch Intelligenz unterstützt werden muß: Die Realität, i. e. seine Umwelt, in eine symbolische Repräsentation umzusetzen. Von Bedeutung ist hier auch, daß Agenten (aufgrund der erwünschten Portabilität und der deswegen verwendeten Programmiersprachen) oft nur langsam ausgeführt werden können. Dies stellt in Verbindung mit KI eine Schwierigkeit dar, da für die Berechnung von Ergebnissen große Rechenleistung benötigt wird, der Agent Antworten jedoch vielfach zumindest annähernd in Echtzeit benötigt. Letztlich stellt sich bei mobilen Agenten noch das weitere Problem, daß die Wissensbasis große Datenmengen umfaßt, was einer schnellen und einfachen Übertragung auf ein anderes System entgegensteht.

2. Sicherheitsfragen (Siehe [Chess 1998] für eine Einführung): Einen Rechner vor darauf befindlichen Agenten zu schützen ist nicht einfach, jedoch möglich, auch wenn die Effizienz darunter stark leidet: Fast jede Aktion ist zuvor auf ihre Erlaubtheit zu prüfen und darf erst anschließend durchgeführt werden. Auch andere Aspekte, der Schutz von Agenten während der Übertragung bzw. der Kommunikation zwischen Agenten, ist lösbar. Um Agenten in einem offenen System eindeutig identifizieren zu können, ist schon ein erheblich größerer Aufwand erforderlich. Eine Public-Key Infrastruktur (PKI, siehe 3.1.5) ist zu schaffen, in der jedem Agenten ein eigenes Zertifikat zugeteilt wird. Für das Problem des Schutzes von Agenten vor dem Rechner, auf dem sie sich befinden, existieren bisher nur unvollkommene Ansätze (Einzige echte Lösung: Einbau sicherer Hardware, d. h. von Dritten geprüft und versiegelt, sodaß Veränderungen unmöglich sind). Zu diesem Thema siehe insbesondere die Abschnitte 3.1 und 4.2.

3. Komplexe Bedienung: Viele Sicherheitsfragen sind zwar gelöst (siehe voriger Punkt), doch bedeutet dies oft eine sehr komplexe Bedienung. Für einen Einsatz in größerem Umfang ist eine Vereinfachung unbedingt erforderlich. Einem Benutzer kann nicht zugemutet werden, für jeden Agenten bei jedem Start ein neues Zertifikat zu erzeugen. Ebensowenig dem Besitzer eines Servers, die Zertifikate aller Agenten selbst zu überprüfen und ihre Einstufung festzulegen. Siehe hierzu auch 4.2.10: Standard-Sicherheitseinstellungen.

4. Vergessen: Wenn Agenten längere Zeit hindurch aktiv sind, so sammeln sie eine große Menge an Daten und Erfahrungen an. Ein Teil davon kann unmittelbar gelöscht werden (Spezialdaten zu einem konkreten Auftrag), aber viele sollten erhalten bleiben um für die Zukunft daraus lernen oder auf sie zurückgreifen zu können. Analog dazu muß ein Agent auch vergessen können, um unnötigen Ballast loszuwerden und die zu übertragenden Daten wieder zu reduzieren. Informationen welche erwiesenermaßen falsch sind, können bedenkenlos gelöscht werden. Jedoch wird dies oft nicht einfach zu beurteilen sein. Selten verwendete Daten können trotz allem äußerst wichtig sein, während häufig benutzte Daten ev. trivial sind. Es ist daher bei der Strategie des Vergessens nicht nur auf die Häufigkeit der Verwendung, sondern auch auf die (präsumptive) Bedeutung Rücksicht zu nehmen.

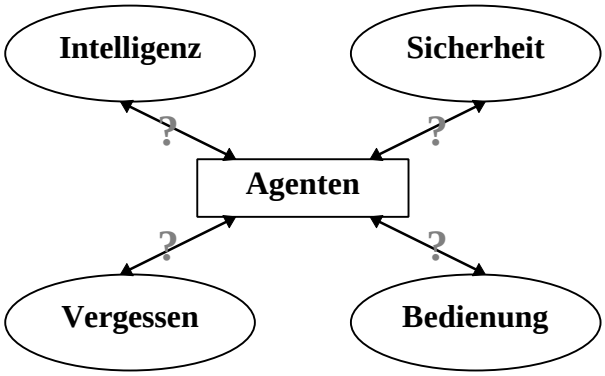

Abbildung 27: Technische Problembereiche

2.7.2 Psychologischer Natur

Im Zusammenhang mit den Problemen rechtlicher Natur (siehe nächster Abschnitt) stellt sich insbesondere auch ein psychologisches Problem (Abbildung 28): Warum sollte ein Benutzer einem Agenten soweit vertrauen, daß er ihm Geld zur eigenen Verwendung übergibt (etwa zur Bezahlung von Leistungen auf Rechnern, z. B. Entgelt für Rechenzeit oder Erlaubnis zur Transferierung auf diesen Rechner)? Dies kann analog zur Verwendung von Kreditkarten zum Kauf im Internet gesehen werden: Die Wahrscheinlichkeit für eine betrügerische Manipulation ist viel geringer als im normalen Leben (z. B. Übergeben der Kreditkarte an den Kellner zur

Bezahlung in einem Restaurant bedeutet ebenso eine Preisgabe von Nummer und Ablaufdatum) und dennoch bestehen große Vorbehalte dagegen. Ein möglicher Grund hierfür ist, daß man den beispielhaften Kellner zumindest sieht, während die Gefahr im Internet von einer großen, gesichtslosen Masse ausgeht. Es ist daher sehr schwer, die tatsächliche Gefahr abzuschätzen. Im Gegensatz hierzu wäre jedoch bei Agenten eine Beschränkung möglich: Agenten könnten so ausgestattet werden, daß sie nur auf eine beschränkte Anzahl von Rechnern transferiert werden können, was eine Überschaubarkeit des Risikos und höhere Akzeptanz zur Folge haben könnte.

Ein weiterer Punkt ist auch die fehlende Praxis: Agenten mit Geld auszustatten wird nur dann erfolgen, wenn der Benutzer bereits mit einfachen Agenten Erfahrung gesammelt hat. Er ist nur dann in der Lage abzuschätzen, welche Qualität die Ergebnisse dieses Agenten besitzen, wie zuverlässig er ist, und welchen Betrag/welches Risiko er daher einzugehen gewillt ist.

Für einen Verwender ist es auch sehr schwierig zu durchschauen, was ein Agent eigentlich tut: wenn dieser seine Arbeit verrichtet, ist er nicht vorhanden (bei mobilen Agenten) oder oft nicht ansprechbar. Dem Benutzer bleiben Einzelheiten verborgen und er kann häufig auch während der Verarbeitung keine Änderungen mehr vornehmen. Dies ist das genaue Gegenteil des Ziels, welches man bei normaler Software erreichen möchte (keine Zustände in der Software): Der Benutzer soll jederzeit beliebige Aktionen durchführen können, ohne erst in einen anderen Zustand (z. B. Herbeiholen des Agenten, Abbrechen der Verarbeitung, …) wechseln zu müssen ([Blaschek 2000]). Da Agenten jedoch autonom sein sollen, ist dies nicht vermeidbar: Der Benutzer soll eben gerade *nicht* jederzeit eingreifen müssen (und daher oft auch nicht können). Während der Konfiguration des Agenten sollte der Grundsatz der Zustandslosigkeit jedoch sehr wohl beachtet werden. Besonderes Augenmerk ist daher auf die Dokumentation zu legen, sodaß ein Verwender zumindest daraus entnehmen kann, welche Aktionen der Agent wann setzt. Eine andere Verbesserungsmöglichkeit in diesem Zusammenhang wäre, die Agenten mit der Fähigkeit auszustatten ihre Aktionen zu erläutern und zu begründen, wie es beispielsweise in AI-Systemen zum Teil erfolgt.

Das größte Problem für die Verbreitung von Agenten ist jedoch der Mangel an einer „Killer-Applikation“ (siehe 2.3.5). Durch die Verwendung von Agenten kann vieles besser erfolgen, doch nichts sensationelles Neues. Auch handelt es sich um eine Technik, welche nur für große und komplexe Systeme sinnvoll ist. Andernfalls erscheint der Zusatzaufwand im Vergleich zu den Nutzen zu hoch.

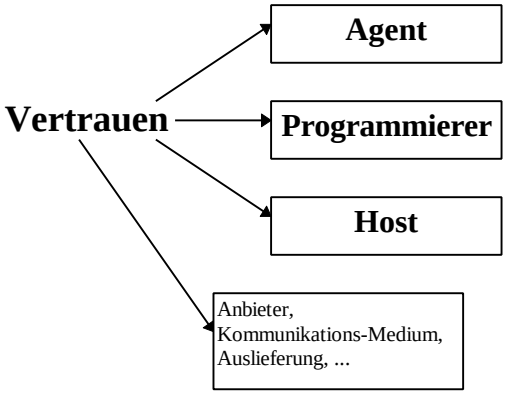

Abbildung 28: Psychologische Probleme

2.7.3 Rechtlicher Natur

In den Grundsätzen sind die rechtlichen Regelungen für Software-Agenten sehr einfach: Es handelt sich um ein Werkzeug des Besitzers, welcher daher für alle Aktivitäten verantwortlich ist bzw. daraus berechtigt wird. In der Praxis stellen sich jedoch beträchtliche Probleme analog denen im Internet: Welches Recht ist anwendbar, wo ist es einklagbar, ist es auch tatsächlich durchsetzbar (Analog dem Unterschied: Recht haben – Recht bekommen)? Wie erfolgt die Aufteilung des Risikos zwischen dem Hersteller des Agenten und dem Verwender (Produkthaftung)? Neben diesen Rechtsproblemen entstehen jedoch auch besondere beweistechnische Schwierigkeiten: Wie kann festgestellt werden, daß eine Manipulation erfolgte und wer dafür verantwortlich ist? Nur dann kann in einem etwaigen Verfahren, sei es vor einem Gericht oder einem Schiedsgericht, auch Beweis geführt werden. Zu diesem Zweck können „trusted third parties“ verwendet werden, also außenstehende Dritte, denen beide an einer Transaktion beteiligten Parteien so weit vertrauen, daß sie keine Manipulationen zu Gunsten eines Teils vornehmen werden und welche die übermittelten Daten eine bestimmte Zeit lang verfügbar halten. Hier besteht ein jedoch leider unlösbares Problem, daß dieser Dritte dann jederzeit auch tatsächlich erreichbar sein muß, was nicht garantiert werden kann. Ist nicht unbedingt ein voller Beweis erforderlich, wie in vielen Fällen (z. B. bei niedrigem Transaktionswert), so kann auch mit ausführlichen Aufzeichnungen (Logs) das Auslangen gefunden werden. Analog zur Entwicklung im E-Commerce, wo der Trend hin zur Schiedsgerichtsbarkeit und weg von einzelstaatlichen Gerichten geht, ist es auch bei Agenten vorstellbar, daß bei Streitigkeiten ein unparteiischer dritter Agent aufgesucht wird. Hier stellen sich jedoch noch große Probleme, insbesondere bei der Beweiswürdigung, sodaß außer in extrem eingeschränkten Fällen dies zur Zeit nicht praktikabel ist.

Eine erste explizite rechtliche Regelung für die Rechtsfolgen des Einsatzes von Agenten findet sich im US Electronic signature act [US signature act], worin eine Nicht-Diskriminierungs-Klausel eingeführt wird: Handelt jemand durch einen Agenten für den er verantwortlich ist, so sind dessen Aktionen rechtserheblich. Dies ist allerdings auf das Aushandeln, den Abschluß und den Transport von Verträgen beschränkt. Auch wird nichts darüber ausgesagt, *wann* die Handlungen eines Agenten einer natürlichen oder juristischen Person zurechenbar sind.

In diesem Zusammenhang interessant zu bemerken ist auch noch, daß im US Bundesstaat Illinois Agenten auch rechtsgültige elektronische Signaturen erzeugen können, denn auch sie können Zertifikatsinhaber sein: Ein Zertifikat wird u. A definiert als „names or otherwise identifies its subscriber or a device or electronic agent under the control of the subscriber“ ([Illinois EC Security Act]).

2.7.4 Ethischer Natur

Insbesondere im Internet ist es für Agenten möglich, in einem eingeschränkten Bereich wie ein Mensch aufzutreten und eventuell den Kommunikationspartner darüber zu täuschen. Dies kann erwünscht und akzeptiert sein (z. B. virtueller Museumsführer) oder auch nicht (z. B. automatische Anfragebeantwortung), ist immer jedoch auch ein ethisches Problem (siehe Abbildung 29). Ein Beispiel hierfür ist, daß ein Agent ohne weiteres die Webseiten von Online-Shops abfragen kann, dort gewisse Informationen extrahiert, und diese anschließend für einen Vergleich darstellt. Erfolgt dies über die normale HTTP-Schnittstelle, so ist es weder für den Webserver noch für einen menschlichen Benutzer, der dies nachzuvollziehen versucht, erkennbar, ob ein Mensch mit einem Webbrowser oder ein Agent diese Seite abgerufen hat. Erste Systeme zum Preisvergleich waren noch auf einen speziellen Zugang zu der hinter den Webseiten befindlichen Datenbank angewiesen, doch ist dies (bei einigem Aufwand für die Untersuchung der Webseiten) heute nicht mehr notwendig.

Analog dazu stellt sich in der Zukunft sicher auch die umgekehrte Frage: Ist die Beratung, die ich über das Internet erhalte, von einem Agenten oder einem Menschen? Dies kann insbesondere auch für rechtliche Probleme (z. B. bei falschen oder unvollständigen Auskünften) Bedeutung erlangen.

Es sollte daher überlegt werden, ob Agenten nicht grundsätzlich als solche zu kennzeichnen sind: Wer sie nicht wünscht, kann sie aussperren oder nicht verwenden. Doch sollte auch der ökonomische Druck beachtet werden: Wird ein Service *nur* von einem Agenten angeboten, so

besteht keine große Wahl! In jedem Fall besteht dann für den Benutzer zumindest die Möglichkeit, die Aktionen bzw. Antworten des Agenten entsprechend zu beurteilen (z. B. Verläßlichkeit bei Anfragen außerhalb des Spezialgebietes des Agenten). In diesem Zusammenhang ist erwähnenswert, daß etwa die meisten Indizierungs-Agenten von Suchmaschinen (siehe dazu auch 2.4.1) eine gewisse Kennzeichnung durchführen: Im Feld User-Agent (das normalerweise den verwendeten Webbrowser angibt) wird ein Hinweis auf die Suchmaschine angegeben, sodaß über eine Auswertung der Anfragen nach diesem Feld ein Ausfiltern möglich ist.

Abbildung 29: Ethische Fragestellungen

3. Ausgewählte Problemstellungen

In diesem Kapitel werden die drei Hauptgebiete der Arbeit in theoretischer Hinsicht vorgestellt:

- Sicherheitsaspekte mobiler Agenten
- Selbständige Bezahlung von Leistungen durch Agenten
- Ausgewählte Rechtsfragen im Zusammenhang mit Agenten

Zuerst erfolgt eine Darstellung von Sicherheitsaspekten mobiler Agenten, wobei auf verschiedene Angriffe sowie Erkennungs- und Abwehrmaßnahmen eingegangen wird. Mobilität an sich wird hier nicht näher betrachtet, da es sich um ein ausführlich erforschtes Gebiet handelt (Exemplarisch: Vor- und Nachteile von Mobilität [Gehmeyr et al. 1998], Koordination und Design Pattterns für Mobilität [Tolksdorf 1999], Überblick über Mobile Agentensysteme und Algorithmen zur Wegplanung [Brewington et al. 1999], Least-cost-routing mittels einfachster mobiler Agenten als Anwendung [Schoonderwoerd/Holland 1999]). Wegen der Bedeutung für die im Agentensystem POND durchgeführte Implementierung und die Identifizierung von Agenten, insbesondere auch im Hinblick auf die Sicherheitsprobleme, welche bei Mobilität und dem Umgang mit Geld naturgemäß besonders wichtig sind, wird in einem Exkurs ein kurzer Überblick über eine Public Key Infrastruktur gegeben. Darin ist auch eine Darstellung des Gegensatzes zwischen Identifizierung und Anonymität von Agenten bzw. Besitzern, einem kontroversiellen Thema zwischen Anbietern und Konsumenten, zu finden. Anschließend wird genauer auf die selbständige Bezahlung von Leistungen direkt durch Agenten selbst eingegangen. Hierbei werden zwei Detailprobleme (sicherer Datenaustausch mit Übergabenachweisen für beide Beteiligten und der Nachweis der Übergabe von Informationen an den Agenten, ohne daß eine dritte Partei beteiligt werden muß) näher erläutert und Lösungen dargestellt. Den Abschluß bildet ein Überblick über Rechtsfragen, welche im Zusammenhang mit Agenten besondere Bedeutung besitzen, wie etwa die Zurechnung von Handlungen von Agenten zu Personen und die Bedeutung des Datenschutzes für und mit Agenten.

3.1. Sicherheitsaspekte mobiler Agenten

Mit größerer Bedeutung (insbesondere auch im E-Commerce) werden Agenten auch immer mehr Angriffen ausgesetzt sein. Diese können von verschiedenen Elementen des Gesamtsy-

stems ausgehen (andere Agenten, Server, Dritte, etc.) und gegen unterschiedliche Elemente gerichtet sein. Wichtig ist hierbei insbesondere die Unterscheidung, ob ein Angriff zwar erkannt, aber nicht verhindert werden kann, oder ob eine tatsächliche Sicherung dagegen möglich ist, sodaß ein Schaden erst gar nicht entsteht. Erstrebenswert ist insbesondere Letzteres, doch vielfach nur sehr schwer oder gar nicht zu erreichen. Bei Software existiert noch eine Zwischenstufe: Eine Verhinderung ist nicht möglich, doch wird der Angriff erkannt, bevor noch ein relevanter Schaden entsteht. Die Implementierung von Lösungen für derartige Sicherheitsprobleme erfolgt u. A. meist mittels asymmetrischer Kryptographie, was eine Infrastruktur zur Verteilung von Zertifikaten voraussetzt; diese wird daher kurz erläutert.

3.1.1 Klassifikation von Angriffen

Angriffe auf Agenten lassen sich in eine Vielzahl von Arten einteilen, welche meist zueinander orthogonal sind. Eine neue Sicht, welche über rein technische Probleme hinausgeht, sind die folgenden Klassen, welche ihre Basis auch in rechtlichen Grundgedanken haben.

3.1.1.1 Bereicherung/Schädigung

Ist es das Ziel des Angreifers sich zu bereichern (Nutzung fremder Ressourcen), oder lediglich möglichst viel Schaden anzurichten (Blockieren fremder Ressourcen)? Angriffe der ersten Art sind viel leichter abzuwehren, da „nur“ der Aufwand für die Störung über den Wert der zu erlangenden Informationen hinaus gesteigert werden muß. Auch kann unter Umständen mit der Erkennung das Auslangen gefunden werden, da über andere Mittel (z. B. Gerichte) anschließend der „richtige“ Zustand wiederhergestellt werden kann. Ist hingegen die Schädigung selbst das Ziel, so kann mit dem vorigen Ansatz nichts erreicht werden (und es ist über Gerichte meist mangels Vermögen des Schädigers selbst bei Identifizierung und Erfolg kein Vermögensausgleich möglich). Hier muß eine echte Vorbeugung erfolgen (Erkennung alleine ist nicht ausreichend), oder die Kosten für einen (erfolgreichen) Angriff so hochgeschraubt werden, daß der Angreifer nicht mehr in der Lage ist diesen durchzuführen. Dies entspricht auch der Unterteilung in rationale und irrationale Angreifer (Abbildung 30).

Bereicherung: Rückabwicklung ev. möglich
Kosten des Angriffs wichtig
Erkennung oft ausreichend
? **„Rationale" Angreifer**

Schädigung: Rückabwicklung unmöglich
Angriffskosten großteils egal
Vorbeugung erforderlich
? **„Irrationale" Angreifer**

Abbildung 30: Aspekte von Bereicherung bzw. Schädigung

3.1.1.2 Externe/Interne Angriffe

Bei einem internen Angriff erfolgt dieser von einer Seite, welcher vom Agenten bzw. dessen Besitzer vertraut wird (z. B. Mitarbeiter; mit diesen besteht eine Vertragsbeziehung, sodaß Verhaltensvorschriften und Konsequenzen festgelegt werden können: Haftung aus Vertrag). Es wird hier meist ausgenützt, daß gegen solche Attacken nur wenige oder gar keine Sicherheitsvorkehrungen bestehen (Abbildung 31). In Bezug auf Agenten bedeutet dies insbesondere Angriffe durch Server, welchen Vertrauen entgegengebracht wird. Dieser Angriff muß nicht unbedingt auf dessen Inhaber zurückzuführen sein. So etwa beim Ausnützen von Programmfehlern durch andere Agenten oder Infektion mit einem Virus. Externe Angriffe hingegen entspringen einer Quelle, welche mit Mißtrauen beobachtet wird (Dritte Personen; Haftung nur aus Delikt oder über Bereicherung). Diese Differenzierung ist für Agenten von geringerer Bedeutung als für Menschen, da es ihnen wenig ausmacht, jedesmal wieder (und auch auf seit langem als zuverlässig bekannten Systemen) komplexe und zeitintensive Sicherheitsprüfungen oder Sicherheitsvorkehrungen durchzuführen, sofern kein besonderer Zeitdruck besteht oder Zusatzkosten hiermit verbunden sind. So könnte etwa einem Menschen nicht zugemutet werden, bei jedem Zugriff auf eine Webseite eine Identifizierung über ein 30-stelliges Paßwort durchzuführen oder bei jeder Verwendung einer Software zuerst die Checksumme aller betroffenen Programmdateien zu überprüfen. Beides ist jedoch für einen Agenten kein Problem und kann unabhängig vom Kommunikationspartner, Server, etc. durchgeführt werden.

Diese Unterteilung hat sowohl praktische Auswirkungen (Mitarbeitern sind Manipulationen leichter nachzuweisen), als auch rechtliche: Vertragshaftung geht viel weiter und umfaßt oft größeren Ersatz. So z. B. bei leichter Fahrlässigkeit: Bei Vertragshaftung sind auch bloße Vermögensschäden (kein Schaden an absolut geschützten Rechtsgütern wie etwa dem Eigentum) und der entgangener Gewinn ersatzfähig, während bei Haftung aus Delikt lediglich der positive Schaden (d. h. kein Ersatz des entgangenen Gewinns) ersetzt wird.

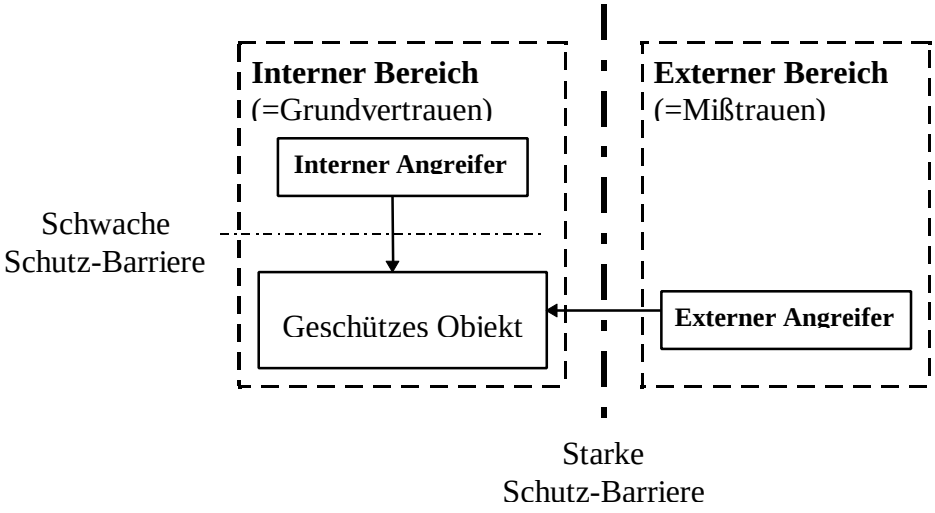

Abbildung 31: Vergleichende Darstellung interner und externer Angriffe

3.1.1.3 Mit/Ohne Datenveränderung

Angriffe können nicht nur durch die Veränderung von Daten erfolgen sondern auch auf andere Weise. Werden die Daten oder der Agent selbst verändert, so ist dies wiederum in zwei Gruppen zu sehen: Der Programmcode und Initialisierungsdaten können sehr einfach durch (kryptographische) Prüfsummen gesichert werden, da sie ohnehin keiner (normalen) Veränderung unterliegen sollen (Selbstmodifizierende Programme sind selten und unnötig). Die angesammelten Daten bzw. der interne Zustand des Agenten hingegen kann nur während der Übertragung auf einen anderen Rechner gegen Veränderungen gesichert werden und ist daher viel stärker gefährdet. Wirkliche Probleme existieren nur bei Übertragung auf Rechner, denen nicht vertraut wird. Diese können durch ein extensives Sicherheitssystem gelöst werden: Der Programmcode und die Initialisierungsdaten werden vom Besitzer des Agenten signiert, während der derzeitige Zustand von dem Rechner signiert wird, welcher den Agenten auf einen anderen Server hin transferiert. Durch dieses „Hand-off“ ist gesichert, daß jede Manipulation von außen durch Dritte unmöglich ist und, wenn sie entdeckt wird, auch eine Manipulation durch einen Server auf diesen zurückgeführt werden kann (auf welchem Server in einer Kette die Manipulation erfolgte; siehe [Hohl 2000]). Angriffe ohne Datenveränderung sind demgegenüber viel schwerer zu entdecken und noch schwerer zu verhindern. Sie beruhen darauf, den Agenten in seiner Ablaufgeschwindigkeit, Bewegungsfreiheit, Informationsgewinnung oder durch Kopien seiner selbst zu behindern. Es handelt sich daher sehr oft um ein Beweisproblem: Eine Datenveränderung ist (relativ) einfach nachzuweisen, während dies bei anderen Manipulationen fast unmöglich ist. Für letzteren Fall muß daher ein größerer Wert auf Vorbeugung gelegt werden, sowie Versuche zur Sicherung von Beweisen unternommen werden.

3.1.1.4 Langfristige/Kurzfristige Angriffe

Bei langfristigen Angriffen versucht der Angreifer in der Regel zuerst das Vertrauen des Opfers zu erreichen, auf daß bestimmte Sicherheitsvorkehrungen als unnötiger Aufwand wegfallen. Dies beinhaltet unter Umständen auch ehrliche Geschäfte, um später bei einer bestimmten einzelnen Transaktion zuzuschlagen. Eine mögliche Abhilfe hierfür ist (jedoch nur über einen längeren Zeitraum hinweg) die Verteilung von Erfahrungen von Agenten mit anderen Agenten an die Gemeinschaft. Ein einfacher Ansatz hierfür wurde im Agentensystem POND implementiert, ist jedoch in dieser Arbeit nicht beschrieben. Derartige Angreifer versuchen auch besonders ihre Spuren zu verwischen, um eventuell noch weitere Angriffe durchführen zu können. Kurzfristige Angriffe hingegen sind darauf angewiesen jede Sicherheitsvorkehrung möglichst schnell zu überwinden, bevor der Angriff bemerkt und in der Folge abgewehrt wird. Diese Klassifizierung kann auch als heimlich/offen oder in rechtlicher Hinsicht grob als Betrug/Gewalt gekennzeichnet werden.

3.1.1.5 Einzelangriffe/Gruppenangriffe

Angriffe können sowohl durch einzelne Parteien als auch durch Gruppen durchgeführt werden. Diese Differenzierung baut jedoch nicht auf physischen Personen auf, sodaß etwa auch mehrere Agenten eines einzelnen Besitzers einen Gruppenangriff durchführen können. Die große Gefahr eines koordinierten Angriffs besteht darin, daß etwa die „Fluchtmöglichkeiten“ verringert werden (z. B. mehrere Server), oder auch Vergleiche nicht mehr möglich sind (z. B. fast alle andere Agenten auf diesem Rechner verhalten sich ähnlich, wenn auch falsch) und daher die Erkennung stark erschwert wird. Eine Erleichterung kann sich jedoch dadurch ergeben, daß ein gehäuftes Fehlverhalten viel leichter auffällt als ein Einzelnes (welches u. U. in der Masse der Tätigkeiten untergeht). In rechtlicher Hinsicht besteht der Vorteil für den Geschädigten bei einem Gruppenangriff darin, daß in vielen Fällen eine Solidarhaftung für die Schäden eintreten wird. Dies bedeutet, daß von jedem beliebigen der Täter die volle Summe verlangt werden kann (insgesamt jedoch nur einmal).

3.1.2 Ausgewählte Angriffe

Aus der Vielzahl der möglichen Angriffe[3] wurden die folgenden für eine nähere Betrachtung ausgewählt:

- Abhören des Agenten-Transfers
- Kopieren von Agenten
- Behinderung / Denial of Service
- Ein-/Ausgabe-Modifikationen

Sowohl der Angriff selbst, als auch kurz welche Erkennungs- bzw. Verhinderungsmaßnahmen möglich sind, werden erläutert. Die Auswahl erfolgte besonders im Hinblick auf mobile Agenten welche unmittelbar wertvolle Daten mit sich führen (z. B. eine E-Cash-Variante). Angriffe, welche auf die Veränderung des Agenten oder seiner Daten abzielen, werden hier ausgespart.

3.1.2.1 Abhören des Agenten-Transfers

Eine Möglichkeit an Informationen des Agenten zu gelangen ist, die Daten während der Übertragung auf einen anderen Rechner zu kopieren (Abbildung 32). Da der Agent zu diesem Zeitpunkt inaktiv sein muß, hat er keine Möglichkeit irgendwelche Abwehrmaßnahmen zu setzen, sondern ist auf die beiden beteiligten Agentensysteme angewiesen. Für einen Angreifer besteht nach erfolgreichem Abhören nur mehr die Schwierigkeit festzustellen, an welcher Stelle sich die ihn interessierenden Daten befinden. Da er den Agenten genauso wie der eigentliche Empfänger rekonstruieren kann, ist es nicht schwer dies herauszufinden.

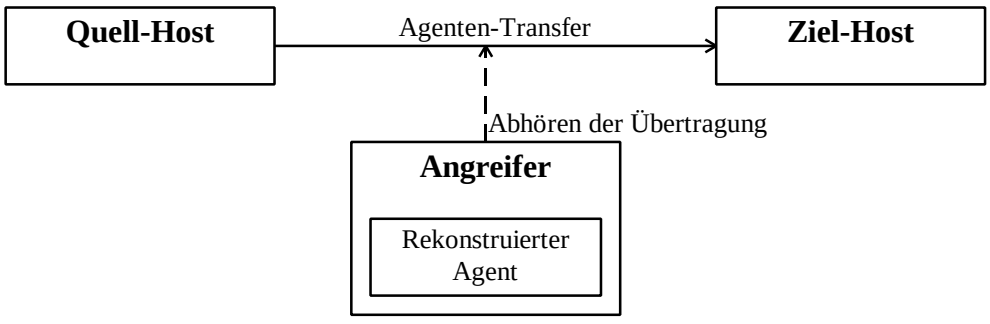

Abbildung 32: Abhören des Transfers eines Agenten

[3] Siehe auch [Chess 1998], wo auch grundlegende Annahmen über Programme (und deren erforderliche Modifikationen für mobile Agenten) erläutert werden.

3.1.2.2 Kopieren von Agenten

Analog zum vorigen Angriff, jedoch nicht auf das Abhören beschränkt, ist die Vervielfältigung von Agenten. Eine Kopie kann entweder durch Abhören der Übertragung (Abbildung 33) oder direkt auf einem Rechner (Abbildung 34) hergestellt werden. Für einen Agenten ist es unmöglich (auf sich alleine gestellt) herauszufinden, ob dies erfolgt bzw. ob er das „Original“ oder eine Kopie darstellt.

Selbst wenn hiermit keine Veränderung verbunden ist, können sich für den Besitzer des Agenten nachteilige Folgen ergeben. So ist es einerseits möglich, die etwa vom Agenten bestellten Produkte mehrfach bestellen zu lassen (erhöhte Kosten; z. B. bei Zahlung per Kreditkarte) oder den Ruf zu schädigen, indem Zahlung mehrfach versucht, jedoch vom Partner abgelehnt werden (z. B. bei der Verwendung von E-Cash: das Original zahlt erfolgreich, die Zahlungsversuche der Kopien werden jedoch von der Bank abgelehnt).

Dies ist auch eine der Voraussetzungen für den „Diebstahl“ von Agenten (Abbildung 35): Eine Umleitung ist meist nur sehr schwer möglich, doch eine Kopie zu erstellen und anschließend das Original zu zerstören (oder die Übertragung zu behindern) ist einfacher.

Gegen ein Kopieren von Agenten auf dem Übertragungsweg hilft einerseits die Verschlüsselung derselben, andererseits auch das Verhindern von Replay-Attacken, wodurch ebenfalls Kopien erstellt werden könnten.

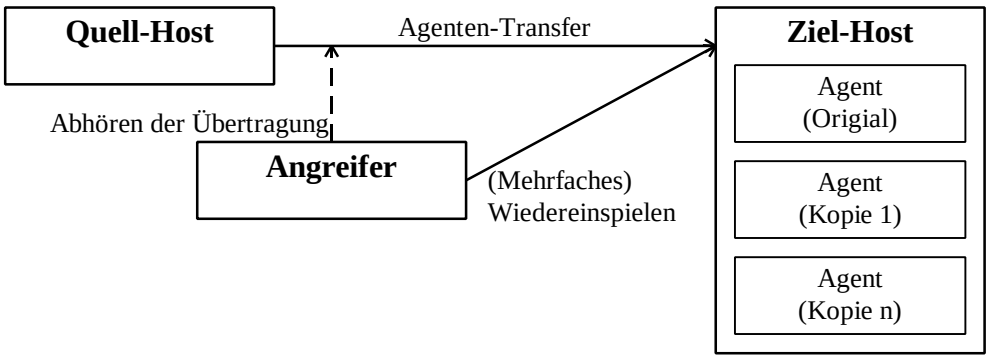

Abbildung 33: Kopieren durch Abhören und Wiedereinspielen

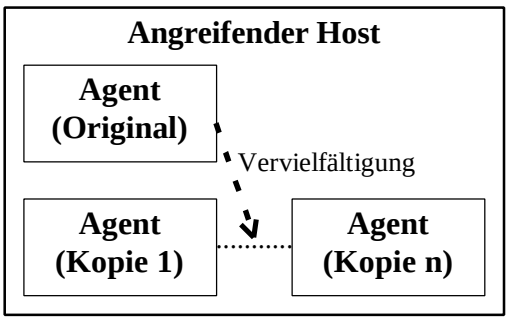

Abbildung 34: Kopieren eines Agenten durch den Host

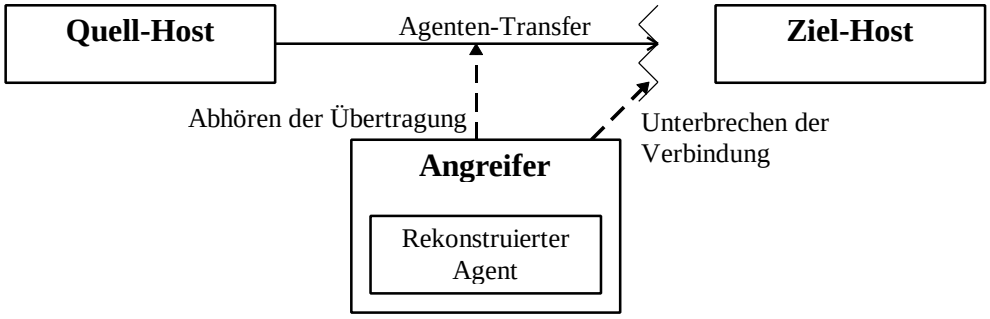

Abbildung 35: "Diebstahl" eines Agenten

3.1.2.3 Behinderung/Denial of Service

Sehr gefährlich, da fast nicht erkennbar und meist auch nicht verhinderbar, sind reine Behinderungen bzw. Denial-of-Service (DoS) Angriffe. Hier wird ein Agent in seinem Ablaufverhalten gestört, sodaß ein anderes Ergebnis entsteht. Dies kann einerseits dadurch erfolgen, daß das Laufzeitverhalten verändert wird (z. B. geringe Zuteilungen an Rechenzeit, niedrige Priorität, etc.), andererseits durch die Verhinderung von Mobilität (z. B. Blockieren der Verbindungen zu anderen Systemen) oder der Ein-/Ausgabe (siehe unten).

Insbesondere bei konkurrierenden Agenten ist die zeitliche Komponente sehr wichtig: Erfahren zwei Agenten zum selben Zeitpunkt von einem besonderen Angebot, so gewinnt der Schnellere bzw. ein zufälliger Agent (und im Lauf der Zeit damit alle annähernd gleich oft). Soll jedoch ein bestimmter Agent gewinnen, so reicht schon eine winzige Verzögerung aus, um den anderen Ersten werden zu lassen. Für einen Agenten ist dies nicht erkennbar, da auch etwaige Zeitinformationen (lokale Zeit; Agent mißt etwa Differenz) gefälscht und über einen längeren Zeitraum hin ausgeglichen werden können. Die einzige Abhilfe gegen diesen Angriff sind signierte Zeitangaben einer sicheren Stelle. Diese sind jedoch nur bedingt verläßlich, da sie einerseits transportiert werden müssen (und so eine nicht genau zu bestimmende Zeit lang verzögert werden), über den Server abzufragen sind (und daher ebenso verzögert werden können) sowie nur mit beschränkter Genauigkeit verfügbar sind (durch die Laufzeit haben Mikrosekunden-Angaben keinen Sinn, dies reicht aber bereits für bedeutende Manipulationen aus). Siehe hierzu

[Tanenbaum 1995] (3.1.2 Physikalische Uhren); logische Uhren können keine Anwendung finden, da es auf die exakte Zeit und nicht nur eine eindeutige Reihung ankommt. Eine andere Möglichkeit ist die Simulation von Verbindungsproblemen, sodaß ein Agent (oder auch die Agenten auf einem bestimmten Server) Informationen nur langsam erhalten.

Agenten können hier sogar in gewissem Ausmaß eine Verringerung der Bedrohung erreichen: Durch Mobilität ist es in manchen Fällen, sofern nicht das Agentensystem selbst Angriffsobjekt ist, möglich, dem Angriff zu entgehen indem auf einem anderen Server weitergearbeitet wird. Ebenso können Agenten aufgrund ihrer Flexibilität auf einen anderen Anbieter umsteigen, falls ein Einzelner durch einen derartigen Angriff blockiert sein sollte.

Werden Agenten „eingesperrt“, so kommt dies ebenfalls dem Abschneiden von Informationen oder Möglichkeiten gleich. Der „Vorteil“ dieses Angriffs besteht jedoch in der leichteren Verschleierung („allgemeine Netzwerksprobleme“) und schweren Erkennung (zu diesem Zeitpunkt ist der Agent eventuell nicht mehr aktiv, da bereits zum Transport vorbereitet). Im Gegensatz zu zeitlichen Veränderungen ist hier jedoch eine Erkennung zumindest im Nachhinein teilweise möglich. Ein Vergleich mit anderen Agenten erlaubt die Feststellung, ob tatsächlich Verbindungsprobleme auftraten (welche alle Agenten gleichermaßen hätten betreffen müssen) oder nicht. Auch sind hier die Zeitdifferenzen ausreichend groß, um mit der Verwendung von gesicherten Zeitangaben Verzögerungen beim Versand erkennen zu können. Der Vorteil ist, daß bereits einfache Zeitangaben genügen, sofern nicht Sende- und Empfangsrechner den Angriff gemeinsam durchführen (Gruppen-Angriff).

3.1.2.4 Modifikation von In-/Output

Durch die Veränderung der von externen Quellen empfangenen bzw. an solche gesendeten Daten (z. B. Modifikation des Preises bei einer Anfrage oder deren Beantwortung) kann ebenfalls das Ergebnis der Verarbeitung verändert werden. Besonders gefährdet ist ein Agent hier auch wieder gegenüber dem Server (Agentensystem: Abbildung 36 bzw. Betriebssystem: Abbildung 37), auf welchem er sich befindet. Dieser hat Zugriff sowohl auf die Daten als auch deren zeitliches Eintreffen bzw. Versand. Externe Angreifer haben lediglich während der Übertragung Zugriff, was unabhängig von Agenten und nicht sehr einfach ist.

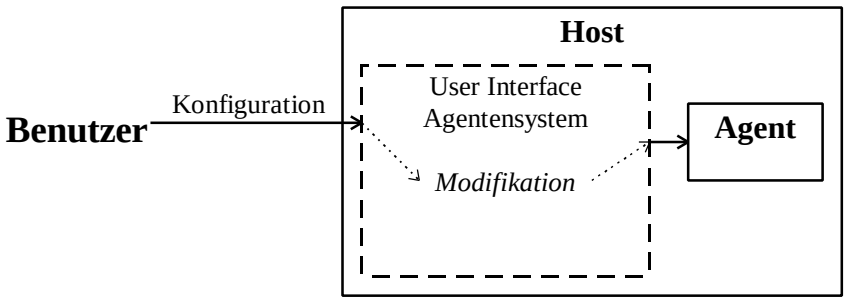

Abbildung 36: Verändern der Eigangsdaten durch das Agentensystem

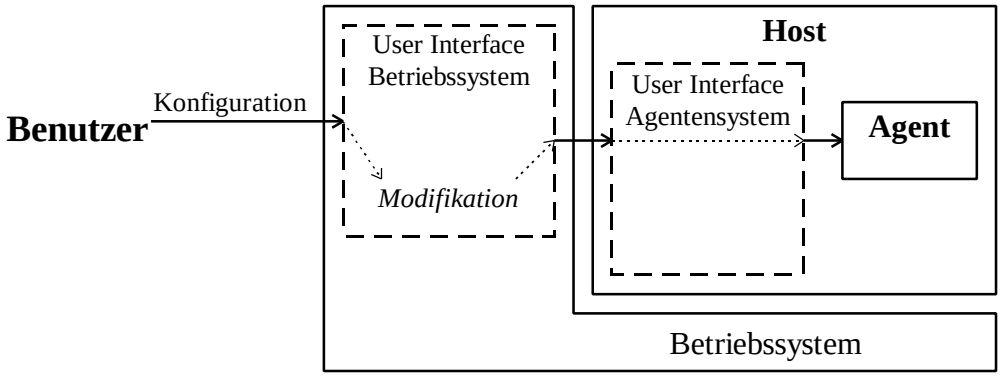

Abbildung 37: Verändern der Eingangsdaten durch das Betriebssystem

Wird die Eingabe vorher über andere Agenten geführt (z. B. Vor-Filterung), so ist auch durch diese eine Modifizierung möglich (Abbildung 38). Es ist daher darauf zu achten, daß Agenten sicher identifiziert werden, bevor mit ihnen zusammengearbeitet wird. Dies insbesondere auch dann, wenn es sich um Agenten desselben Besitzers handelt, welche immer gemeinsam auf einen anderen Rechner transferiert werden (beim Transfer könnte einer ausgetauscht werden; z. B. durch Ersetzen der Daten oder indem bereits ein anderer Agent mit derselben ID existiert, und daher der neue abgelehnt wird; dies kann aber auch zufällig vorkommen und muß auch für diese Fälle erkannt werden).

Abbildung 38: Veränderung der Eingangsdaten durch einen angreifenden Agenten

Analog zu den normalen Problemen mit Software muß bei Agenten *besonders* darauf geachtet werden, daß die übernommenen Parameter auf ihre Gültigkeit und Sinnhaftigkeit geprüft werden, sodaß nicht durch deren Manipulation unerwünschtes Verhalten erreicht werden kann. Aufgrund der inhärenten Autonomie von Agenten ist eine extensivere Interpretation erforderlich, was zu größeren Fehlern des Endergebnisses bei Fehlern oder Mißverständnissen hinsichtlich der Eingabewerte führt.

3.1.3 Erkennungs-/Abwehrmaßnahmen

Analog zu einigen im vorigen Abschnitt besprochenen Angriffen sollen hier Konzepte zur Erkennung bzw. Sicherung gegen Angriffe erläutert werden, wobei auch hier besonderes Augenmerk darauf gelegt wird, wie mobile Agenten gesichert werden können.

3.1.3.1 Sichere Hardware

Eine Möglichkeit zur Sicherung von Agenten, insbesondere die zur Zeit einzige wirklich verläßliche gegen Angriffe des Servers selbst, ist die, sichere Hardware zu verwenden. Dies setzt jedoch voraus, daß in jeden Rechner eine „tamper-proof-box“ eingebaut wird (Abbildung 39), welche den Großteil des Rechners umfaßt. Ein Äquivalent hierzu wäre auch die Überprüfung des Rechners und anschließende Versiegelung (eine Sicherung gegen Installation von Software über das Netzwerk müßte integriert und ohne Brechen der Versiegelung nicht außer Kraft setzbar sein). In diesem Fall kann garantiert werden (je nachdem wofür der Hersteller/Überprüfer geradesteht!), daß Agenten nur auf die vorgesehene Art behandelt werden und keine Manipulationen stattfinden.

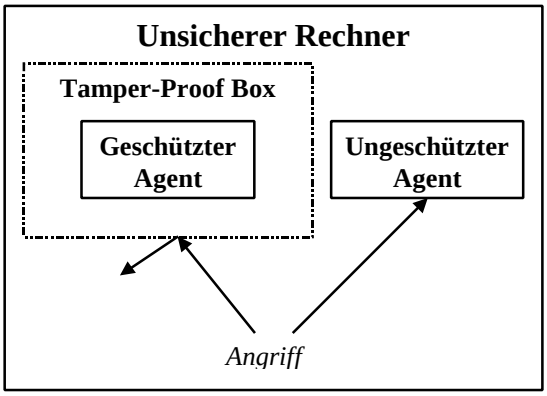

Abbildung 39: Sicherung durch geprüfte und abgesicherte Hardware (Tamper-proof box)

Problematisch ist hierbei jedoch die Verfügbarkeit und die Kosten: Solche Hardware wird (zumindest zur Zeit) nicht serienmäßig hergestellt und ist auf jeden Fall äußerst kostspielig (da der Besitzer ja beliebig viel Zeit hätte, um sie vorsichtig zu entfernen). Auch ist eine solche Versiegelung im Fall von Defekten oder notwendigen Updates sehr hinderlich. Dieser Ansatz ist daher zumindest für eine breite Verwendung ausgeschlossen.

Ein im Ergebnis gleicher Ansatz ist, als Besitzer seine Agenten nur auf solche Server sich hinverlagern zu lassen, bei denen deren Inhaber bekanntermaßen vertrauenswürdig ist. Dies kann auf vielerlei Arten sichergestellt werden wie z. B. persönliche Kenntnis, im Besitz einer bekannten Firma oder auch die Erteilung eines Server-Zertifikates durch eine Zertifizierungsstelle. Bei den derzeit ausgestellten Zertifikaten wird allerdings ausschließlich die Identität des

Inhabers garantiert, keinerlei Eigenschaften der Server oder der Software. Diese Methode der Sicherung ist auch für eine Massenanwendung denkbar und sinnvoll.

Das Agentensystem POND beruht ausschließlich auf Software, daher ist dieses Schutzkonzept nicht enthalten.

3.1.3.2 Übertragungs-Verschlüsselung

Hauptsächlich gegen externe Angreifer sichert die Verschlüsselung während der Übertragung (Abbildung 40). Dies verhindert einerseits das Abhören, läßt aber auch Modifikationen während des Transfers ebenso wie Übertragungsfehler erkennen.

Der Agent sollte angeben können, welche die Mindestanforderungen sind, die er für seine Übertragung als notwendig erachtet. Da er seinen eigenen Transfer in der Regel nicht überwachen kann (für die Übertragung muß er serialisiert werden) ist er darauf angewiesen, daß das Agentensystem die Verschlüsselung für ihn übernimmt und keine schwächere verwendet. Dabei liegt das Risiko beim Agenten. Für eine noch weitgehendere Kontrolle des Agenten besteht dennoch eine Möglichkeit: Da nur der Empfangsteil beim Transfer eines Agenten zwangsweise im Agentensystem enthalten sein muß wäre es denkbar, das Absenden durch jeden Agenten selbst durchführen zu lassen. Hier stellen sich jedoch sehr schnell Kompatibilitätsprobleme ein. Der Agent kann zwar dann selber festlegen, auf welche Weise und in welcher Stärke seine Übertragung gesichert werden soll, doch bleibt er gegen Angriffe des Servers selbst weiterhin anfällig (Ein-/Ausgabe-Modifikationen). Es ist daher meist sinnvoller, da dem Rechner ohnehin vertraut wird/werden muß, diesem die Aufgabe zu überlassen und lediglich die Voraussetzungen festzulegen.

Bei diesem Ansatz ist festzulegen was passieren soll, wenn die gewünschte Verschlüsselungsstärke nicht erreichbar ist. Etwa wenn der andere Server dazu nicht in der Lage ist. Dies kann effektiv dazu führen, daß sich der Agent nicht mehr auf einen anderen Rechner begeben kann, um dort weiterzuarbeiten. Als Lösung kommt hier nur in Frage entweder auf den Heimatrechner zurückzukehren (dieser sollte die Anforderungen des Agenten erfüllen können, da sie schon beim ersten Transfer, hin zum derzeitigen Server, bestanden), zu terminieren, oder wichtige Daten zu löschen und anschließend mit geringerer Sicherheit weiterzuarbeiten oder zurückzukehren. Ein letzte Alternative wäre noch zu warten, ob nicht ein anderer Server zu einem späteren Zeitpunkt erreichbar bzw. zur gewünschten Sicherheitsstufe in der Lage ist.

Im Agentensystem POND wurde eine einfache Version dieses Schutzes implementiert: Die Daten werden bei der Übertragung grundsätzlich immer in voller Stärke verschlüsselt sowie signiert. Der Agent hat hierauf keinen Einfluß.

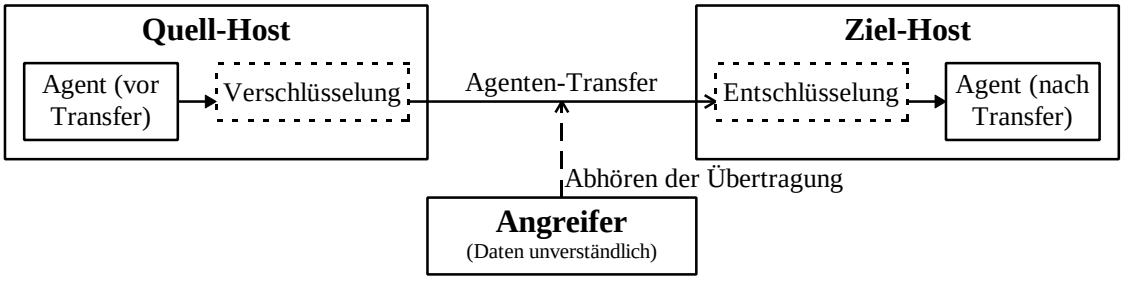

Abbildung 40: Sicherung der Agenten-Übertragung dirch Verschlüsselung

3.1.3.3 Partner-Identifikation

Eine weitere Sicherheitsvorkehrung ist die Identifikation von Transaktionspartnern. Nur wenn diese verläßlich erfolgt, ist auch eine genaue Beurteilung ihrer Eigenschaften möglich. Es handelt sich daher hier meistens nicht um eine eigenständige Sicherungsmaßnahme, als vielmehr um eine Voraussetzung für die Entscheidung, welche dann zu setzen sind.

Die Identifikation des Transaktionspartners kann auf mehreren Ebenen stattfinden, wobei die wichtigste die gegenseitige Identifizierung und Authentifizierung von Rechnern (bei mobilen Agenten), sowie von Agenten (mit welchem Agenten wird kommuniziert) ist. In beiden Fällen dient diese Identifikation auch dazu, „man-in-the-middle“ Angriffe abzuwehren, bei denen sich ein Dritter in die Mitte der Kommunikationsverbindung einzuschalten versucht (siehe Abbildung 41).

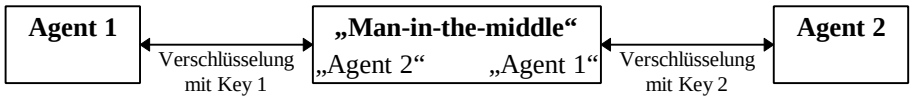

Abbildung 41: "Man-in-the-middle" Angriff

Die einfachste Methode hierfür ist die Verwendung von Zertifikaten und eines Challenge-Response-Protokolls. Jeder Partner schickt dem anderen von ihm ausgewählte Daten, welche mit dem öffentlichen Schlüssel des anderen Partners verschlüsselt sind (kann dem Zertifikat entnommen werden). Dieser hat die Daten mit seinem privaten Schlüssel zu entschlüsseln und im Klartext wieder zurückzusenden. Da (hoffentlich) nur der wahre Inhaber des Zertifikates den zugehörigen privaten Schlüssel kennt, ist damit die Identität festgestellt. Probleme ergeben sich, falls das Kryptographiesystem nicht in der Lage ist Verschlüsselung durchzuführen, sondern nur Signaturen erlaubt (z. B. DSA). Hier kann die Prüfung dadurch erfolgen, daß eine

Nachricht des anderen Partners signiert wird. Die Gefahr liegt nun darin, daß die zu signierenden Daten auch eine andere Bedeutung besitzen könnten (z. B. der Hashwert zu einem Vertrag, der somit ungewollt unterschrieben würde). Um dem abzuhelfen werden die empfangenen Daten (Challenge) auf eine für beide bekannte Art vor der Signierung weiterverarbeitet. Anwendung finden Hashverfahren, weil hierdurch sichergestellt wird, daß der andere Teil keine Manipulationen durchführen kann. Hierzu müßte die inverse Funktion zur Hashfunktion berechnen werden können, was für Hashfunktionen per Definition äußerst schwer sein muß. Eine Alternative ist die Signierung der Identitäten der beiden Partner und Einsatz einer fortlaufenden Nummer um Wiedereinspielen zu verhindern.

Im Agentensystem POND werden Server bei der Verlagerung von Agenten gegenseitig über Zertifikate identifiziert. Eine wechselseitige Identifizierung von Agenten erfolgt nur beim Sport-Portal. Hierfür wurde ein eigenes Protokoll entworfen, welches verschiedenste Identifizierungs-Mechanismen enthalten kann (z. B. Name-Paßwort, Signaturen, ...).

3.1.3.4 Software-Prüfsummen/Code-Signierung

Insbesondere bei mobilen Agenten stellt sich die Frage, ob der Code, der ausgeführt werden soll, auch tatsächlich den Vorstellungen des Programmierers bzw. des Besitzers entspricht. Dies einerseits deshalb, um Angriffe durch Veränderung desselben zu verhindern, andererseits auch einfach aus potentiellen Versionskonflikten: Auf anderen Rechnern sind u. U. noch alte Versionen installiert, welche anderes Verhalten besitzen oder bestimmte Aktionen nicht unterstützen. Eine der möglichen Lösungen ist die Verwendung von Prüfsummen für den Code (Veränderungen und Versionskonflikte erkennen) oder die Signierung desselben (Veränderungen erkennen und Hersteller feststellen).

Schwierigkeiten bereitet hier das Henne-Ei-Problem. Um festzustellen ob der Code den Vorstellungen entspricht, muß zumindest ein Teil ausgeführt werden, um die Überprüfung durchzuführen. Dies ist daher etwa geeignet festzustellen, ob Programmbibliotheken, welche auf dem Rechner vorausgesetzt werden, unverändert vorliegen, jedoch nicht zur Überprüfung des Agenten-Codes durch den Agenten selbst (wurde er verändert, so höchstwahrscheinlich auch bzw. gerade die Prüfungsroutine).

Im Agentensystem POND gehen Code-Signaturen in die Berechtigungen ein, welche einem Agenten zur Verfügung stehen (sowohl grundsätzlich, als auch optional gegen Bezahlung). Ein Agent kann jederzeit abfragen, welche Code-Signaturen überprüft und akzeptiert wurden.

3.1.3.5 Authoritative Kopie

Eine besondere Möglichkeit zur Sicherung stellt eine „authoritative Kopie“ dar, obwohl die Bezeichnung nicht ganz korrekt ist. Es existiert der Agent mehrfach, wobei ein Exemplar auf einem sicheren Server verbleibt und der andere Teil mobil ist. Es erfolgt zusätzlich eine Aufteilung der Informationen auf beide Versionen, z. B. indem Geldwerte-Informationen wie die Kreditkartennummer auf dem sicheren Rechner verbleiben (daher keine exakten Kopien). Dies erfordert spezielle Kommunikation zwischen den beiden Geschwister-Agenten, sodaß die passenden Informationen zum richtigen Zeitpunkt an den richtigen Partner weitergegeben werden (was selbst nicht problemlos ist: Diese Kommunikation könnte von einem anderen Agenten/dem Server gefälscht werden, da die hierfür notwendigen Informationen vom Agenten mitgeführt werden *müssen*).

Jedoch insbesonders gegen zerstörende Angriffe ist dies ein probates Sicherungsmittel, da eine weitere (sichere) Kopie existiert. Zu diesem Zweck ist auch ein kontinuierliches Update durch den mobilen Agenten möglich. Ein Verlust des Agenten bei der Übertragung, durch einen Rechner-Absturz oder einen Angriff bedeutet daher nur einen Verlust von wenigen Daten und (bei Inanspruchnahme kostenpflichtiger Leistungen) relativ geringen Wertes. Dies ist insbesondere bei der Verwendung von E-Cash sinnvoll, da ansonsten der gesamte Wert vernichtet würde, wenn keine lokale Kopie mehr existiert (problematisch ist jedoch herauszufinden, welche „Banknoten“ von der Kopie vor deren Verschwinden noch ausgegeben wurden).

Zu beachten ist hier jedenfalls, daß allein durch die Nicht-Existenz der mobilen Kopie noch kein Schluß auf den Grund dieses Fehlens gezogen werden kann: Es kann ein Angriff vorliegen, doch genauso gut ist ein zufälliger Fehler des Servers (z. B. Verbindungsprobleme oder auch Absturz) oder ein Programmfehler (Absturz des Agenten und folgende Terminierung/Beseitigung der Überreste) möglich.

Im Agentensystem POND wird dies nicht besonders unterstützt, doch bedienen sich einige der implementierten Agenten dieser Technik (wie etwa zur Anzeige einer Nachricht in einem anderen Agentensystem).

3.1.3.6 Signierte Nachrichten

Um Veränderungen in Nachrichten zu verhindern, können diese signiert werden. Dies hat auch gleichzeitig den Zweck, daß der Absender später nicht mehr abstreiten kann, sie abgeschickt zu haben (Ursprungsbeweis). Der Empfänger einer derartigen Nachricht kann später zuverlässig

nachweisen, was er empfangen hat (Inhalt und Absender). Die einzige Möglichkeit für den Absender dies abzustreiten wäre, ein Veränderung durch das Agentensystem zu beweisen (bzw. die Durchführung der Signatur durch dieses, welche im normalen Programmablauf nicht stattgefunden hätte). In Verbindung mit E-Commerce ist es daher für beide Seiten hilfreich, sich auf einem „neutralen“ Server zu treffen, da ansonsten für einen Teil ein Vorteil besteht (der Besitzer des Agentensystems ist mit dem Besitzer eines Agenten identisch). Für den Aufbau einer derartigen Nachricht siehe Abbildung 42.

Im Agentensystem POND ist die Möglichkeit zur Versendung digital signierter Nachrichten vorgesehen, wird jedoch aus Effizienzgründen (lange Dauer des Signiervorgangs bzw. des Überprüfens) bei den implementierten Agenten nur vereinzelt durchgeführt.

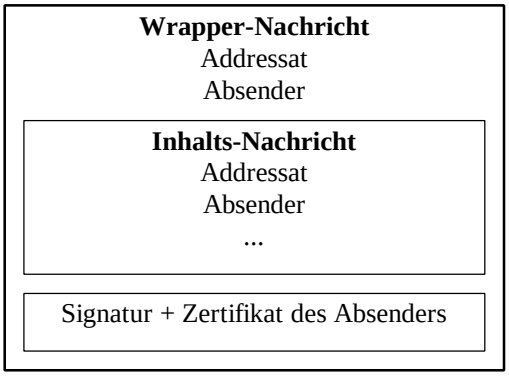

Abbildung 42: Struktur einer signierten Nachricht

3.1.4 Identifizierung und Anonymität

Ein wichtiger Wunsch von Kunden im Geschäftsleben ist es, anonym zu bleiben. Dies gilt insbesondere für Anfragen nach Standard-Produkten, während bei individuellen Angeboten und Firmenkunden dies weniger Bedeutung besitzt. Werden dann Produkte gekauft, so sind die meisten Käufer bereit ihre Daten preiszugeben, was aufgrund der Zahlungsart oder der physischen Lieferung ohnehin vielfach notwendig ist. Doch insbesondere im Falle von Agenten, welche für einzelne kleinere Dienstleistungen oder Daten Gebühren bezahlen sollen, wird auch in diesem Fall die Identifizierung unerwünscht sein. Im Gegensatz hierzu haben Anbieter widersprüchliche Ansätze: Einerseits sollen Daten vermieden werden, um nicht in Konflikt mit Datenschutzgesetzen zu gelangen und die notwendige Archivierung und Auswertung zu vermeiden, andererseits ist es von vitalem Interesse möglichst viel über die Kunden bzw. Interessenten zu erfahren. Da letzteres überwiegt, wird meist versucht, möglichst viele Daten zu erlangen und diese auch auszuwerten und zu verwenden (Personalisierung; Wissen über den Kunden ist eine Voraussetzung für die meisten Personalisierungsmethoden).

Anonymität betrifft aber nicht nur den Agenten bzw. dessen Eigentümer, sondern auch den Hersteller des Programmcodes (sofern dieser signiert ist). Für diesen besteht jedoch kein besonderes Schutzbedürfnis, sodaß es nicht notwendig ist hier vorzusorgen, damit der Programmierer anonym bleiben kann. Ganz im Gegenteil ist eine möglichst weite Verbreitung auf Grund ihrer Werbewirkung eher erwünscht und nicht (oder unter Pseudonym) signierter Code sollte nur mit großer Vorsicht zugelassen werden.

3.1.4.1 Identifizierung

Bei der Identifizierung ist zu unterscheiden, ob der Agent identifiziert wird oder dessen Besitzer. Der erste Punkt ist meist für beide Seiten (Besitzer und Server) von geringerer Bedeutung, doch läßt sich sehr oft daraus auf den Besitzer schließen (was der eigentliche Punkt für beide ist). Wird lediglich der Agent identifiziert, so kann bereits eine Anpassung des Angebots erfolgen, da er wiedererkannt werden kann, wenn er zu einem späteren Zeitpunkt zurückkehrt. Die Daten können hierfür aber nur auf diesen einen Agenten zugeschnitten werden. Demgegenüber erlaubt die Identifizierung des Besitzers eine Anpassung für *alle* Agenten desselben, wobei auch auf Aspekte eingegangen werden kann, die nicht direkt mit einem konkreten Agenten verbunden sind.

Einen Identifizierung kann durch ein Zertifikat erfolgen, wobei bei Agenten darauf zu achten ist, von wem es ausgestellt wurde (dem Benutzer oder einer Zertifizierungsstelle), und daß die gesamten Zertifikate für den Pfad bis zur Wurzel-Zertifizierungsstelle verfügbar sind. Hiermit können beliebige Besitzer bzw. Agenten verläßlich identifiziert werden, ohne daß zusätzliche Arbeit erforderlich ist. Zu beachten ist, daß die Verwendung von Zertifikaten eine Verkehrsanalyse stark vereinfacht: Damit Zertifikate plattformunabhängig sind, ist ihre Struktur genau definiert. Es ist daher ohne großen Aufwand möglich, diese aus einem Datenstrom herauszufiltern. Zur Verhinderung ist es erforderlich, jede Kommunikation, die eventuell abgehört werden könnte (z. B. Transfer eines Agenten zu einem anderen Server), zu verschlüsseln.

Eine Alternative hierzu ist, auf (ungesicherte, da nicht in einem Zertifikat enthaltene) Angaben des Agenten zu vertrauen. Für manche Anwendungen mag dies ausreichen, z. B. wenn er ohnehin nur geringe Berechtigungen erhält und ansonsten mit elektronischem Geld (welches einen sofortigen Übergang des Wertes ohne Risiko bedeutet) bezahlt. In diesem Fall könnte auch von einer Identifizierung des Besitzers vollkommen abgesehen werden.

3.1.4.2 Anonymität

In Bezug auf die Erkennung/Identifizierung von Benutzern bzw. deren Agenten durch Agentensysteme existiert Anonymität in mehreren Stufen:

1. Vollständige Anonymität: Dies bedeutet, daß ein Agent während eines einzigen Aufenthalts nicht als der selbe Agent wiedererkannt wird. Diese Art von Anonymität kann nicht erreicht werden, da der Server einen Agenten immer als eine bestimmte Einheit betrachtet und betrachten muß, z. B. um Nachrichten an ihn zustellen zu können.

2. Anonymität im Hinblick auf mehrere Besuche: Besitzt ein Agent mehrere Zertifikate (oder gar keines), so kann er sich bei jedem Besuch als ein anderer Agent ausgeben. Eine Identifizierung wäre nur durch einen vollständigen Vergleich der Daten möglich. Sollte der Agent daher zwischen den Besuchen andere Tätigkeiten durchgeführt oder einen neuen Auftrag erhalten haben, so ist dies nicht mehr möglich. Diese stärkste mögliche Form von Anonymität wird für die meisten Benutzer nicht erforderlich sein, insbesondere da sie auch mit Aufwand verbunden ist (Vielzahl von Zertifikaten, Entfernen anderer kennzeichnender Elemente wie öffentlicher Schlüssel, etc.).

3. Pseudonym: Gibt der Agent die Identität des Besitzer zwar bekannt, doch handelt es sich hierbei um ein Pseudonym (=Zertifikat unter Decknamen oder keine Besitzer-Angabe), so kann er bei jedem Besuch wiedererkannt werden und dennoch bleibt die dahinter stehende Person unbekannt. Hierbei wird es sich um den häufigsten Fall handeln, da er auch für den Server-Betreiber akzeptabel ist. Im Falle von besonderen Gründen (z. B. Schädigung durch den Agenten) kann ein Pseudonym-Zertifikat in den wahren Namen aufgelöst werden. Dies ist jedoch meist nur über ein Gericht möglich. Im Normalfall ist daher die Anonymität gewährleistet, während bei überwiegenden Interessen anderer Personen die Identität dennoch festgestellt werden kann. Der Agent kann seine Anonymität auch freiwillig aufgeben, indem er ein anderes, nicht mit Pseudonym versehenes, Zertifikat präsentiert.

4. Identifizierung: Wird der vollständige und richtige Name angegeben und nachgewiesen, so sind der Agent bzw. dessen Besitzer nicht mehr anonym.

Probleme hinsichtlich der Anonymität stellen sich insbesondere dann, wenn bestimmte Eigenschaften des Agenten oder seines Besitzers bestätigt werden müssen. Ein Beispiel hierfür ist etwa die Volljährigkeit, um an Glücksspielen teilnehmen zu können. Eigenschaften wie Berufsberechtigungen oder Vertretungsmacht besitzen in Verbindung mit Anonymität hingegen kei-

nen Sinn. Deren Bescheinigung kann z. B. durch Attributzertifikate (siehe unten) erfolgen. Diese beziehen sich immer entweder auf eine Person (keine Pseudonyme, daher auch keine Anonymität mehr) oder auf ein Zertifikat. Es ist zwar denkbar, für ein Pseudonym-Zertifikat ein Attributzertifikat zu erhalten, doch erhöht dies die Gesamtanzahl der Zertifikate stark, da für jedes Basis-Zertifikate jedes Attributzertifikat gesondert benötigt würde. Es sollte daher genau überlegt werden, für welche Fälle Anonymität gewünscht wird und hierfür gesonderte Zertifikate (mit den entsprechenden Attributen eventuell direkt enthalten) beantragt werden.

3.1.5 Exkurs: Public Key Infrastructure (PKI)

In diesem Abschnitt werden kurz die technischen und organisatorischen Grundlagen einer Public-Key-Infrastructure ([BSI-A1], [BSI-A2]) erläutert, wie sie z. B. für die Signierung von Agenten durch deren Besitzer, die Signierung des Codes durch dessen Hersteller, oder für die Verschlüsselung bei der Übertragung zwischen zwei einander unbekannten Rechnern erforderlich ist. Im speziellen wird hier nur auf Zertifikate nach dem X.509 Standard [X.509] eingegangen, welcher weltweite Verbreitung erlangt hat.

3.1.5.1 Zertifikate nach X.509v3

Zertifikate sind eine Art “digitaler Ausweis”. Sie dienen dazu, eine eindeutige Verbindung zwischen einer Person und einem öffentlichen Schlüssel (und damit auch zu einem privaten Schlüssel) herzustellen. Zertifikate werden von einer bestimmten Instanz ausgestellt, einer “Zertifizierungsstelle” (Certificate Authority, CA). Ein Zertifikat hat jedoch nur dann eine Bedeutung, wenn der Empfänger des Zertifikats, der dessen Echtheit überprüfen möchte, Vertrauen zu dieser Ausgabestelle hat. Eine wichtige Voraussetzung dafür ist, daß die Zertifizierungsstelle exakt bekanntgibt, welche Angaben vor Ausstellung eines Zertifikates überprüft werden: Nur auf diese kann man sich verlassen, wenn eine Überprüfung erfolgreich verläuft.

Um ein Zertifikat öffentlich übertragen zu können und die Unverändertheit der bestätigten Informationen zu garantieren werden die Daten bei der Zertifizierungsstelle eingereicht, diese überprüft sie und versieht sie anschließend mit einer elektronischen Signatur (wozu ihr eigener privater Schlüssel verwendet wird). Um die Gültigkeit zu überprüfen, ist daher die Kenntnis des öffentlichen Schlüssels der Zertifizierungsstelle notwendig (welcher wieder über ein Zertifikat verteilt wird).

Zertifikats-Bestandteile nach X.509v3

Zertifikate bestehen aus folgenden Elementen:

1. Der X.509 Standard hat mehrere Versionen hinter sich. Daher muß bei jedem Zertifikat angegeben werden, welcher Version es entspricht.
2. Um ein Zertifikat eindeutig identifizieren zu können (eine Person kann mehrere Zertifikate mit demselben Schlüssel von derselben Zertifizierungsstelle besitzen; z. B. für verschiedene Verwendungszwecke in verschiedenen Qualitäten), muß jede CA eine eindeutige Seriennummer vergeben.
3. Da ein Zertifikat immer nur in Zusammenhang mit einer Zertifizierungsstelle und ihren Prüfungs-Richtlinien („Certification Policy“) eine Bedeutung hat, muß diese im Zertifikat eindeutig gekennzeichnet werden (idealerweise Klartext und URL).
4. Ein Zertifikat ist immer nur während einer beschränkten Zeit gültig. Dies soll verhindern, daß ein einmal gebrochenes Zertifikat (einem Angreifer gelang es an den privaten Schlüssel heranzukommen) zu große Folgen nach sich zieht: Nach seiner Ablaufzeit ist es jedenfalls ungültig. Dies erfolgt durch die Angabe von einem Beginn- und einem Ende-Datum.
5. Die eine Hälfte des Hauptteils ist das “Subjekt”: Für wen das Zertifikat ausgestellt wird, also wem der enthaltene öffentliche Schlüssel zugeordnet wird.
6. Die andere Hälfte ist der öffentliche Schlüssel, welcher dem Subjekt zugeordnet ist. Er ist in einer Algorithmus-spezifischen Form codiert.
7. Bei einem Zertifikat der Version 3 können noch Erweiterungen angebracht werden: Ein Beispiel dafür ist, wozu das zertifizierte Schlüsselpaar verwendet werden darf: z. B. nur zur Signatur oder auch zur Verschlüsselung von Nutzdaten oder zur Verschlüsselung von anderen Schlüsseln.

Um ein Zertifikat auf allen Plattformen verwenden zu können, erfolgt eine eindeutige Repräsentation. Diese legt bis auf das letzte Bit genau fest, wie die einzelnen Element gespeichert werden. Hierzu wird die ASN.1 - Notation (Abstract Syntax Notation One, [X.680])[4] mit DER-Kodierung (siehe [Kaliski] für eine Einführung) verwendet. Ein X.509v3 Zertifikat ist als ASN.1 Sequenz definiert:

```
Certificate ::= SEQUENCE
{
   tbsCertificate       TBSCertificate,
```

[4] Definiert in dem ITU-T X.208 Standard. (International Telecommunications Union; früher CCITT). X.509 Zertifikate werden nach den Distinguished Encoding Rules (DER) kodiert.

```
   signatureAlgorithm   AlgorithmIdentifier,
   signature            BIT STRING
}
```

Es enthält das eigentliche Zertifikat, eine Kennzeichnung des zur Signierung verwendeten Algorithmus und die Signaturdaten, an Hand derer das Zertifikat überprüft werden kann. Der konkrete Zertifikats-Inhalt ist definiert als:

```
TBSCertificate ::= SEQUENCE
{
   version         [0]  EXPLICIT Version DEFAULT v1,
   serialNumber         CertificateSerialNumber,
   signature            AlgorithmIdentifier,
   issuer               Name,
   validity             Validity,
   subject              Name,
   subjectPublicKeyInfo SubjectPublicKeyInfo,
   issuerUniqueID  [1]  IMPLICIT UniqueIdentifier OPTIONAL,
                        -- If present, version must be v2 or v3
   subjectUniqueID [2]  IMPLICIT UniqueIdentifier OPTIONAL,
                         -- If present, version must be v2 or v3
   extensions      [3]  EXPLICIT Extensions OPTIONAL
                        -- If present, version must be v3
}
```

Zertifikats-Hierarchien

Zertifizierungsstellen sind üblicherweise in einer baumartigen Struktur angeordnet: Das Zertifikat einer CA, mit deren privatem Schlüssel die Zertifikate von Kunden signiert werden, wird von der Zertifizierungsstelle höherer Instanz signiert. Die oberste Zertifizierungsstelle wird "Oberste Zertifizierungsstelle" oder „Wurzel-Zertifizierungsstelle" (Root-CA) genannt. Ihr Schlüssel muß besonders sicher sein und sehr gut geschützt werden: Wird ihr privater Schlüssel bekannt, sind *alle* mit diesem Schlüssel ausgestellten Zertifikate (und ebenso alle über diese ausgestellten Zertifikate, usw.) wertlos. Ihr Zertifikat wird mit dem eigenen Schlüssel signiert, daher ist es mit dem Schlüssel, der darin enthalten ist, zu überprüfen (self-signed certificate). Ob ein solches Zertifikat gültig ist, kann innerhalb des Systems nicht festgestellt werden. Dies hat extern zu erfolgen, z. B. mit der Veröffentlichung einer Prüfsumme in Zeitungen (Österreich: Das Zertifikat der Aufsichtsstelle ist im Amtsblatt zur Wiener Zeitung zu veröffentlichen) oder einer Verteilung über sichere Kanäle (z. B. bei Behörden abzuholen).

Widerrufs-Listen

Da Zertifikate auch kompromittiert werden können oder auch der Grund ihrer Ausstellung nicht mehr existiert (Beispielsweise eine darin zertifizierte Kreditkartennummer ist wegen mangelnder Bonität nicht mehr gültig), muß es eine Möglichkeit geben, Zertifikate vor ihrem Ablaufdatum für ungültig zu erklären. Dazu werden Widerrufs-Listen verwendet, in denen die

ungültigen Zertifikate aufgelistet sind. Diese Listen müßten entweder von jeder Zertifizierungsstelle für ihre Zertifikate oder von einer Root-CA für alle Zertifikate zur Verfügung gestellt werden. Auch bei ihnen ist darauf zu achten, daß sie während der Übertragung nicht verändert werden können. Daher sind diese Listen ebenfalls zu signieren, unter anderem auch, um ein späteres Abstreiten durch die Zertifizierungsstelle zu verhindern.

Beispiel eines Zertifikates

```
 [ [  Version: V1
  Subject: CN="Server Certificate, RSA (self-signed)", OU=FIM, O=Uni-Linz,
C=AT
  Signature Algorithm: MD5withRSA, OID = 1.2.840.113549.1.1.4
  Key:  public exponent: 3
modulus:
8ecbd113a02932ed0b04199eecd84f12ea076378ae7c24ba9a60c02adbf29a5bacb1bc39aaf
9475e24e7131781c49b7a900bb9bda7ef09037a7c357ad62d20dbce969c1017cb5549007820
cf8b8e1fa916dd49b61d583c25876289b3e2f8bc3337ddc9714
7e50f2030bf563486db8371bb1f00b8d62b3b06a87de776cf79a729
  Validity: [From: Wed Oct 20 23:11:37 GMT+02:00 1999,
              To: Sat Nov 20 23:11:37 GMT+01:00 1999]
  Issuer: CN=FIM Test CA, OU=FIM, O=Uni-Linz, C=AT
  SerialNumber: [    01]
]
  Algorithm: [MD5withRSA]
  Signature:
0000: 7E C4 DD C5 05 D0 54 40   09 D8 AF 3A 1B 29 F1 7F  ......T@...:.)..
0010: 74 36 05 B5 2D D3 9C D7   63 F3 64 D9 71 D1 86 F4  t6..-...c.d.q...
0020: 9A 5C BE 9E EE 01 B5 0F   AB 1E 1F DC FE CC 68 84  .\............h.
0030: DF 6D 7F 8F 03 28 7E 4D   F4 E8 12 42 C5 1C 40 54  .m...(.M...B..@T
0040: 71 4A 01 B0 68 66 6A FE   FC B6 D3 D6 07 CE 5C A4  qJ..hfj.......\.
0050: 7E DF 4B 13 3F 84 C8 4F   15 9A 04 F3 98 21 AF 49  ..K.?..O.....!.I
0060: 0A 90 4A 2D D2 3E 2D 3B   6F A2 40 19 21 B6 7A D6  ..J-.>-;o.@.!.z.
0070: 45 34 8C B1 5A 07 74 3D   16 18 97 BD 52 3D 2C 6E  E4..Z.t=....R=,n
]
```

Das selbe Zertifikat in Base64-Codierung zur Übertragung:

```
-----BEGIN CERTIFICATE-----
MIICDzCCAXgCAQEwDQYJKoZIhvcNAQEEBQAwRDELMAkGA1UEBhMCQVQxETAPBgNV
BAoTCFVuaS1MaW56MQwwCgYDVQQLEwNGSU0xFDASBgNVBAMTC0ZJTSBUZXN0IENB
MB4XDTk5MTAyMDIxMTEzN1oXDTk5MTEyMDIyMTEzN1owXjELMAkGA1UEBhMCQVQx
ETAPBgNVBAoTCFVuaS1MaW56MQwwCgYDVQQLEwNGSU0xLjAsBgNVBAMTJVNlcnZl
ciBDZXJ0aWZpY2F0ZSwgUlNBIChzZWxmLXNpZ25lZCkwgZ0wDQYJKoZIhvcNAQEB
BQADgYsAMIGHAoGBAI7L0ROgKTLtCwQZnuzYTxLqB2N4rnwkuppgwCrb8ppbrLG8
Oar5R14k5xMXgcSbepALub2n7wkDenw1etYtINvOlpwQF8tVSQB4IM+Ljh+pFt1J
th1YPCWHYomz4vi8MzfdyXFH5Q8gML9WNIbbg3G7HwC41is7Bqh953bPeacpAgED
MA0GCSqGSIb3DQEBBAUAA4GBAH7E3cUF0FRACdivOhsp8X90NgW1LdOc12PzZNlx
0Yb0mly+nu4BtQ+rHh/c/sxohN9tf48DKH5N9OgSQsUcQFRxSgGwaGZq/vy209YH
zlykft9LEz+EyE8VmgTzmCGvSQqQSi3SPi07b6JAGSG2etZFNIyxWgd0PRYYl71S
PSxu
-----END CERTIFICATE-----
```

3.1.5.2 Certificate Authorities

Zertifizierungsstellen stellen eine Verbindung zwischen einem Zertifikat und einer bestimmten Person her. Da Zertifikate nur für öffentliche Schlüssel sinnvoll sind, kann dieses Verfahren nur bei asymmetrischen Systemen eingesetzt werden. Da grundsätzlich jede Person eine solche

Stelle einnehmen kann, indem sie Zertifikate von anderen Personen signiert, ist bei der Prüfung eines Zertifikates immer genau zu untersuchen, wer dieses bestätigt hat.

Ein Zertifikat selbst hat grundsätzlich keinerlei Bedeutung für die Bestätigung einer Identität. Erst die Verbindung des Zertifikates mit dem privaten Schlüssel einer Person, von der man annimmt, daß sie ihn nur zu bestimmten Zwecken verwendet und geheimhält, bedeutet eine Erhöhung der Sicherheit. Ein besonders wichtiger Punkt ist daher das Vertrauen der Benutzer in die Zertifizierungsstelle: Ein Zertifikat hat höchstens das gleiche Vertrauen wie die Zertifizierungsstelle (Mangelndes Vertrauen in die Zertifizierungsstelle weil diese z. B. unbekannt ist bedeutet, daß deren Zertifikate wertlos sind!).

Für eine Zertifizierungsstelle ist es daher besonders wichtig, daß ihre Richtlinien (Policies = wann sie wem ein Zertifikat ausstellt) möglichst weit bekannt sind und daß die Endbenutzer ein hohes Vertrauen haben, daß diese Richtlinien auch tatsächlich eingehalten werden. Um dieses Vertrauen zu erhöhen existiert oft auch eine staatliche Aufsicht, welche die Einhaltung der selbst aufgestellten bzw. vorgeschriebenen Richtlinien überprüft (so etwa gemäß der EU-Richtlinie in allen Mitgliedsstaaten). Dies insbesondere auch deshalb, da der Staat im elektronischen Rechtsverkehr (z. B. Einreichungen bei Ämtern) selbst auf die Akzeptierung von Zertifikaten fremder Anbieter angewiesen ist, will er nicht selbst eine Zertifizierungsstelle einrichten. Um dies zu ermöglichen muß es zumindest eine Zertifikatskategorie geben, die staatlicherseits akzeptiert wird und daher staatlichen Mindestbedingungen zu entsprechen hat.

Darstellung einer Zertifizierungshierarchie

Im folgenden ist eine Zertifizierungshierarchie schematisch dargestellt. Außer auf der obersten Ebene (dort unabhängig voneinander, also ohne Überschneidung der Sub-Bäume; graphentheoretisch gesehen ein Wald) können natürlich noch viele weitere Instanzen vorhanden sein, egal ob dies andere Zertifizierungsstellen oder Benutzer sind (Diese können auch gemischt auftreten: Eine CA kann sowohl andere Zertifizierungsstellen als auch Endbenutzer zertifizieren).

In der Praxis ist die Hierarchie jedoch ziemlich flach (wie in Abbildung 43 dargestellt, oder ohne Zwischen-CA‘s, d. h. nur Root-CA - Benutzer) und es erfolgt keine Mischung von Zertifizierungsobjekten: Es werden entweder CA’s *oder* Benutzer, aber nicht beide von einer Stelle zertifiziert.

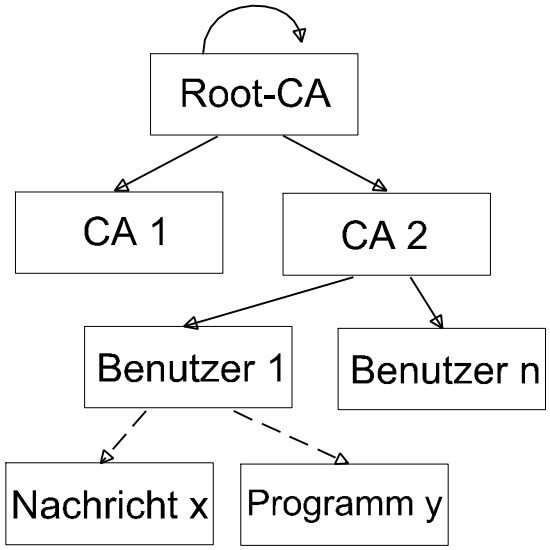

Abbildung 43: Zertifizierungsstellen-Hierarchie

Root-CA: Die oberste Zertifizierungsstelle muß sich selbst zertifizieren, da es keine übergeordnete Instanz mehr geben kann. Ihr Zertifikat ist daher mit dem privaten Schlüssel signiert, der zu dem im Zertifikat enthaltenen öffentlichen Schlüssel paßt ("self-signed certificate").

CA x: Die Zertifikate der einzelnen Zertifizierungsstellen sind mit dem privaten Schlüssel der obersten Zertifizierungsstelle signiert und enthalten den eigenen öffentlichen Schlüssel. Es können auch mehrere solcher CA's untereinander existieren, die Kette kann also dergestalt aussehen: Root-CA ? CA Stufe 1 ? CA Stufe 2 ? CA Stufe 3 ? Benutzer.

Benutzer x: Die Zertifikate der einzelnen Benutzer stellen das untere Ende der Zertifizierungshierarchie da. Natürlich können auch sie wiederum selbst Zertifikate ausstellen, doch wird diesen vermutlich nur ein geringer Wert beigemessen werden[5].

Nachricht x: Einzelne Nachrichten werden von den Benutzern signiert. Hier ist jedoch ein signifikanter Unterschied zu den oberen Verhältnissen zu beachten: Der Benutzer bestätigt hier nicht mehr die Identität der Nachricht (wie es CAs mit Benutzern tun), sondern stellt nur mehr eine besondere Verbindung mit sich selbst her: Die Nachricht ist keine eigene "Person", sondern wird lediglich dem Signator zugerechnet.

[5] Siehe dazu aber PGP [PGP]: Dieses System baut darauf auf, daß jede Person die Schlüssel von anderen Personen bestätigt (also ihnen ein Zertifikat ausstellt). Vertraut man nun einer bestimmten Person, so wird man auch den von ihr zertifizierten Schlüsseln vertrauen. Dies ist natürlich für kleine oder geschlossene Gruppen gut geeignet, jedoch für offene Systeme wie etwa bei E-Commerce ungeeignet: Wie sollte ein Händler je die Schlüssel aller möglicher Kunden oder Agenten erhalten und ihnen auch vertrauen können?

Programm x: Ähnliches ist bei Programmen der Fall: Auch sie erhalten keine eigentliche Persönlichkeit, sondern sind nur "Erweiterungen" dessen, der ihre Signatur erstellt hat. Dies ist insbesondere deshalb wichtig zu beachten, da CA's bestimmte Eigenschaften der von ihnen zertifizierten Personen garantieren (z. B. den Namen), während Endbenutzer üblicherweise *keine* Garantie abgeben. Aus der Tatsache der Zertifizierung läßt sich der Urheber zurückverfolgen, etwaige Garantien können jedoch in jedem einzelnen Fall differieren, oder auch gar nicht existieren.

Liste von anerkannten CA's

Die unten angeführten Zertifizierungsstellen sind allgemein anerkannt. Insbesondere von denjenigen, die international tätig sind, werden die Root-Zertifikate in den am meisten verwendeten Browsern (Netscape Navigator und Microsoft Internet Explorer) standardmäßig als vertrauenswürdig inkludiert. Es handelt sich keineswegs um eine vollständige Liste, sondern nur um einige exemplarische Beispiele.

Bei praktisch allen CA's gibt es verschiedene Klassen von Zertifikaten: Einerseits solche für Server, andererseits Zertifikate für Einzelpersonen. Es werden auch meist mehrere Stufen angeboten: Von einfachsten Zertifikaten, welche lediglich das Vorhandensein einer funktionierenden E-Mail Adresse bestätigen (zum Zeitpunkt der Ausstellung!) bis hin zu sehr vertrauenswürdigen Zertifikaten, die ausschließlich mit notariell beglaubigten Urkunden als Nachweis erhältlich sind. Entsprechend der Sicherheit bemißt sich auch die einmalige und die regelmäßig wiederkehrende Gebühr.

International:

1. Verisign: http://www.verisign.com/
2. Thawte: http://www.thawte.com/
3. GTE: http://www.gte.com/cybertrust/index.html

Österreich:

Telekom Control Kommission: http:/www.tkc.at/
Bestätigungsstelle A-SIT: http://www.a-sit.at/

1. a-sign (Datakom): http://a-sign.datakom.at/ (Elektronische Zertifikate und Smartcards)
2. e-sign (A-Trust): http://www.e-sign.at/ (Smartcards)

3. A-Cert (Arge Daten): https://a-cert.argedaten.at , http://www.a-cert.at/ (Zertifiziert nur PGP-Schlüssel; führt vorher eine Identitätsüberprüfung durch)
4. net.surance Security (EA Generali): http://www.generali.co.at/security (Elektronische Zertifikate; nur in Verbindung mit einer Versicherung für dessen Verwendung)

3.1.5.3 Attribute Authorities (AA)

Bei einer Attribute Authority handelt es sich um eine Stelle welche analog zu einer CA Zertifikate ausstellt, es sich jedoch um Attributzertifikate handelt. Diese enthalten keinen öffentlichen Schlüssel sondern (neben anderen Informationen) einen Verweis auf ein (normales) Basis-Zertifikat und bestimmte zusätzliche Eigenschaften des Inhabers des Basis-Zertifikats. Der Vorteil hierbei ist, daß nicht die CA alle Angaben überprüfen muß, sondern die hierfür jeweils zuständige Stelle dies übernehmen (und dann das Attributzertifikat ausstellen) kann.

Ein Beispiel wäre etwa die Integration der Zulassung als Arzt in ein Zertifikat oder die Modellierung von Unterschriftsbefugnissen in der öffentlichen Verwaltung (siehe [Sonntag 2001]). Für ersteres könnte etwa von der Ärztekammer ein Attributzertifikat ausgestellt werden, in welchem die genaue Berechtigung (Allgemeiner Arzt, Facharzt + Spezialgebiet, etc.) angegeben wird. Eine andere Möglichkeit wäre etwa, daß ein Provider für Agenten seiner Kunden Attributzertifikate ausstellt in welchen der Ursprung (und daher eine „Rücknahme“-Garantie) und eventuell bis zu einem bestimmten Betrag (bzw. Bonität) garantiert wird.

3.2. Selbständige Bezahlung von Leistungen durch Agenten

Insbesondere in Verbindung mit E-Commerce sind Agenten von besonderem Interesse: Einerseits um als Vermittler für Benutzer auftzutreten, andererseits um selbst direkt Leistungen anzubieten oder in Anspruch zu nehmen. Eine wichtige Voraussetzung hierfür ist aber auch, daß Agenten anschließend direkt die Bezahlung bzw. das Einkassieren übernehmen können. Da es sich um reales Geld bzw. Werte handelt, sind die Sicherheitsanforderungen sehr hoch.

Als Alternative hierzu, zwar mit geringerer Sicherheitsanforderung aber vom System her ähnlich, ist die Verwendung von Tokens, z. B. als Lastausgleich. Jeder Agent erhält eine gewisse Menge an „Münzen“ (=Tokens), mit denen er seine Aufgabe zu erfüllen hat. Für die Inanspruchnahme gewisser Dienste sind bestimmte Mengen zu entrichten, während durch die Weitergabe von Informationen oder die Erbringung von Leistungen wieder Tokens gesammelt werden können. Diese Systeme basieren darauf, daß eine funktionierende „Wirtschaft“ zwi-

schen den einzelnen Elementen (hier Agenten) aufgebaut wird (siehe etwa [Bredin et al. 1998] oder [Yemini et al. 1998]). Es ist daher ein genauer Entwurf erforderlich, da es sonst leicht zu Problemen kommt (Token sammeln sich an einer Stelle, Token fehlen an anderen Stellen, Stockungen, …). Bei derartigen Systemen bestehen jedoch meist nur geringe bis gar keine Sicherheitsanforderungen in Hinsicht auf die Übertragung dieser Tokens auf andere Agenten.

3.2.1 Vertrauen in Agenten

Eine der Grundvoraussetzungen für die direkte Inanspruchnahme von Leistungen durch Agenten, unabhängig davon ob er diese selbst bezahlt oder dies später vom Besitzer erledigt wird, ist, daß dem Agenten von seinem Auftraggeber genügend Vertrauen entgegengebracht wird. Muß der Besitzer befürchten, daß trotz Planung von geringen Ausgaben tatsächlich sehr hohe anfallen (oder falsche Ergebnisse geliefert werden), so wird ein solcher Einsatz nicht erfolgen. Dies setzt voraus daß Agenten robust sind (Fehler führen u. U. zu Verlust!) und „verstehen“ welche Konsequenzen ihr Tun hat. Es ist daher von intelligenten Agenten oder zumindest exakt definierbaren Aufgaben auszugehen, da nur in diesen Fällen den Anforderungen entsprochen werden kann.

Zur Erfüllung der ersten Anforderung ist die Verwendung von elektronischem Geld (siehe unten) besonders geeignet, da der Agent nicht mehr ausgeben kann, als ihm der Besitzer mitgegegeben hat. Doch auch hier bleibt der Wunsch bestehen, daß dieses Geld im Sinne des Besitzers verwendet wird. Als Alternative kommen sonst nur Post-Paid-Verfahren (Nachnahme, Überweisung im Nachhinein, …) in Frage.

3.2.2 Arten von elektronischer Bezahlung durch Agenten

Für die Bezahlung von Leistungen oder Waren durch Agenten stehen großteils die selben Möglichkeiten (siehe Abbildung 44) offen wie beim „normalen“ Einkauf im Internet. Nicht möglich ist etwa die Bezahlung mit Chipkarte (z. B. „Quick“ [Europay-Quick]), da ein Agent, insbesondere ein mobiler, natürlich keine physikalische Karte mitführen kann, was hierfür Voraussetzung ist.

1. Nachnahme
2. Überweisung (Zahlung / Ausführung)
3. Bankeinzug
4. Elektronisches Geld
5. Gutscheine
6. Daten
7. Kreditkarte (Nummer, Ablaufdatum)
8. Kreditkarte (SET)

Abbildung 44: Zahlungsarten für direkte Durchführung durch Agenten

3.2.2.1 Nachnahme

Diese Zahlungsart ist nur dann möglich, wenn die Ware physikalisch über die Post ausgeliefert wird. Es ergeben sich keine Besonderheiten in Bezug auf Agenten, da diese lediglich die Wahl von Nachnahme als Bezahlungsweise sowie den Namen und die Anschrift des Empfängers bekanntgeben müssen (was für eine physische Lieferung ohnehin notwendig ist).

3.2.2.2 Überweisung (Zahlung bzw. Ausführung)

Auch hier ergeben sich bei der Zahlung keine besonderen Aspekte: Der Agent muß die entsprechenden Daten mit zu seinem Besitzer nehmen, der anschließend die Überweisung selbst durchführt, da eine sofortige Bezahlung nicht erforderlich ist. Im Gegensatz zur Nachnahme kann jedoch dieser Vorgang bei der Durchführung automatisiert werden, indem der Agent die Überweisung selbst vornimmt (unabhängig davon, ob es sich um eine Vorausüberweisung handelt oder diese im Nachhinein erfolgt). Der Vorteil hierbei ist, daß die benötigten Daten (Kontonummer, PIN, TANs) nicht mit transportiert werden müssen, sondern auf einem sicheren Rechner verbleiben können. Kehrt der Agent zurück, so kann er anschließend mit diesen Informationen eine elektronische Überweisung durchführen, ohne daß das Agentensystem diese beeinflussen könnte. Allerdings müssen hierfür die komplexen Protokolle von Electronic-Banking-Systemen verstanden werden. Sonst ist zumindest eine Vorbereitung möglich, wie etwa das Ausdrucken vorausgefüllter Erlagscheine.

3.2.2.3 Bankeinzug

Beim Bankeinzug stellt sich zum ersten Mal das Problem, daß Daten vor dem Anbieter geheim gehalten werden müssen. Die Kontonummer darf erst dann herausgegeben werden, wenn der Agent auch tatsächlich beabsichtigt die Transaktion durchzuführen. Hier stellt sich noch kein großes Sicherheitsproblem, da der Inhaber des Kontos jederzeit eine Rückbuchung veranlassen kann und es dann dem Anbieter obliegt, die Berechtigung zur Abbuchung nachzuweisen. Hierfür kommen etwa signierte Nachrichten eines Agenten in Verbindung mit einem Überga-

benachweis an diesen, sowie einem kryptographischen Hand-off an den nachfolgenden Server in Frage (wie im Agentensystem POND implementiert). Auch ist mit einer Kontonummer ein Mißbrauch nicht so leicht möglich, da die Transaktion immer nachvollziehbar bleibt (die Abbuchung erfolgt nicht in Bar sondern auf ein anderes Konto).

3.2.2.4 Elektronisches Geld

Unter „Electronic Cash“ wird ein elektronischen Äquivalent zu Bargeld verstanden, welches den Wert „in sich“ trägt. Das Hauptproblem hierbei ist, daß eine Kopie herzustellen trivial ist und daher ein doppeltes Einlösen („double spending“, [Rott 1998]) zu verhindern ist. Der große Vorteil ist in der absoluten Anonymität zu sehen, sowie der Eignung auch für die Bezahlung kleinster Beträge. Letzterer Punkt ist insbesondere für Agenten von Interesse, da hiermit Gebühren für Rechte, Dienstleistungen oder Daten eingehoben werden könnten.

Ein Beispiel ist eCash ([eCash]), wobei die Anonymität des Kunden gesichert ist. Doppelte Verwendung wird dadurch hintangehalten, indem die „Münzen“ vom Händler sofort bei der Bank eingereicht und dort auf ihre Gültigkeit überprüft werden.

Bei der Verwendung durch Agenten ist darauf zu achten, daß andere Agenten oder das Agentensystem keinen Zugriff auf diese Daten erlangen, da ansonsten der Agent bei der Bezahlung (als zweiter Einreicher) abgelehnt würde. Da die Münzen problemlos kopiert werden können ist es möglich, daß der Agent eine Kopie zu Hause läßt, sodaß selbst im Falle eines Fehlers kein Verlust eintritt.

Soll ein Agent solches Geld akzeptieren (was im Gegensatz zu Kreditkartennummern viel wahrscheinlicher ist, da hierfür z. B. ein Vertrag mit dem Kreditkartenunternehmen nicht erforderlich ist), so muß darauf geachtet werden daß die Münzen sofort eingereicht werden, und zwar unter dem richtigen Namen des Agenten (hier ist keine Anonymität möglich, außer es erfolgt ein Umtausch auf andere elektronische Münzen). Befindet sich der Agent noch auf dem Rechner, von welchem er die Daten erhalten hat und soll dennoch die Münzen einreichen, so kann dies schwierig sein (Gefahr des Abfangens/Veränderns der externen Kommunikation). Es bleibt die Möglichkeit übrig, die Einsendung zu signieren, da hiermit das Problem auf die Geheimhaltung eines Schlüssels zurückgeführt wird. Jedoch besteht noch eine weitere Angriffsmöglichkeit: Der Rechner kann den Agenten zurückhalten oder terminieren, wenn er sich plötzlich doch noch dazu entschließen möchte die Transaktion abzubrechen (nachträgliche einseitige Vernichtbarkeit). Da dies nur bis zum Verlassen des Servers durch den Agenten oder

der Einreichung bei der Bank möglich ist (welches der frühere Zeitpunkt ist), kann der Server keinen großen Vorteil daraus ziehen (nach der Einreichung gar keinen mehr, bei vorherigem Verlassen geht der Agent jedoch ein Risiko ein, da er seine Leistung bereits erbracht hat, ohne der Leistung des Servers sicher sein zu können, falls er selbst E-Cash erhalten hat).

3.2.2.5 Gutscheine

Bei Gutscheinen handelt es sich um eine Zahlungsweise welche nicht mehr direkt Geld zum Inhalt hat, aber immer noch nach diesem bewertet wird. Hier wird davon ausgegangen, daß es sich nicht um ein Schecksystem handelt, sondern nur Gutscheine akzeptiert werden, welche ein Agent bzw. ein Agentensystem selbst ausgegeben hat. Es ist zwischen zwei Arten von Gutscheinen zu unterscheiden: Solchen, die namentlich ausgestellt wurden (also nur für einen einzigen Agenten bzw. Besitzer gültig sind), und anonymen (bei welchen die Innehabung alleine ausreicht und die durch jeden eingelöst werden können).

Vorteilhaft in Beziehung auf Agenten ist hierbei, daß bei Bezahlung mit einem Gutschein schon vorher eine Geschäftsverbindung bestanden haben muß (für den Erwerb des Gutscheins) und daher schon Erfahrungen mit dem Partner vorliegen. Dieser Erwerb kann ebenso auf andere Arten erfolgen, z. B. durch den Besitzer selbst und nicht durch einen Agenten, sowie mittels einer unterschiedlichen Zahlungsart (entspricht dann dem Ersatz einer Zahlungsart durch eine Andere). Auch ist bei namentlichen Gutscheinen ein Diebstahl und eine Verwendung durch andere Agenten nicht zu befürchten. Problematisch ist hingegen, daß es sich naturgemäß nicht um ein universelles Zahlungsmittel handelt: Es kann mit einem bestimmten Typ von Gutschein nur bei einer einzigen Stelle (oder einer bestimmten Anzahl davon) bezahlt werden. Ebenso ist dieser vorher zu erwerben, was einen zusätzlichen Schritt bedeutet.

Im Agentensystem POND werden Gutscheine unterstützt, doch muß es sich hierbei um Zertifikate (Subklassen von java.security.cert.Certificate) handeln. Dies hat den Vorteil, daß Gutscheine plattformunabhängig und kryptographisch gesichert sind.

3.2.2.6 Daten

Eine Bezahlung kann auch generell durch Daten erfolgen, welche für den Empfänger einen Wert darstellen. Genaugenommen handelt es sich hierbei dann jedoch um einen Tausch und nicht einen Kauf. Der Ablauf ist fast identisch, es muß jedoch nicht die Leistung und der Preis vereinbart werden, sondern die wechselseitigen Leistungen (wobei oft Bewertungsfragen eine

Rolle spielen werden). Auch hier ist ein sofortiger Übergang des Wertes möglich indem die Daten an den Partner gesendet werden.

Bei dieser Zahlungsweise ist die Geheimhaltung vor dem anderen Agenten bzw. dem Agentensystem vital, da die Informationen ansonsten wertlos sind. Vorteilhaft ist, daß eine Kopie der Daten auf einem sicheren Server deponiert werden kann, sodaß ein Verlust nicht zu befürchten ist. Ebenso ist eine Wiederholung der Bezahlung bei Fehlern möglich.

Als generelle Zahlungsweise sind Daten jedoch nicht geeignet, da vor Abschluß des Vertrages der Partner über den Inhalt genau informiert werden muß, sodaß eine Bewertung möglich ist. Dies bedeutet jedoch, daß ein gemeinsames Vokabular mit einheitlicher Bedeutung für eine Beschreibung zur Verfügung steht. Dies kann bei Agenten jedoch nicht generell vorausgesetzt werden, da diese meist nur für *eine* Richtung (Beschreibung der zu kaufenden/verkaufenden Ware) derart ausgestattet sind.

Im Agentensystem POND ist diese Zahlungsweise als „Ausweichmethode“ vorgesehen. Bei Akzeptierung dieser Zahlungsart wird vorausgesetzt, daß eine Vereinbarung über den Inhalt bereits vorher erzielt wurde (die Inhaltsbestimmung ist nicht Teil des Zahlungsvorganges).

3.2.2.7 Kreditkarte (Nummer, Ablaufdatum)

Soll per Kreditkarte (direkt, ungesichert, oder über eine verschlüsselte Verbindung; sonst siehe nächster Punkt: SET) bezahlt werden, so müssen Kartennummer, Ablaufdatum und der Name auf der Karte an den Verkäufer weitergegeben werden. Die Kreditkartennummer im Agenten zu speichern bereitet besondere Probleme, da sie in ihrem Aufbau genau definiert ist. So wäre es ohne weiteres möglich den gesamten Datenbestand des Agenten zu durchsuchen, und potentielle Nummern anhand ihres Aufbaus (Stellen, Formatierung, Prüfsumme) zu erkennen. Es ist daher auch eine interne Verschlüsselung notwendig. Dies ist zwar keine garantierte Sicherung (daher reicht auch ein sehr einfacher Algorithmus aus), bedeutet jedoch zumindest einen hohen Aufwand zur Erlangung der Kreditkartennummer (insbesondere wohl nicht mehr automatisch möglich). Dies gilt sowohl für den Besitzer der Kreditkartennummer als auch den Empfänger. Außerdem muß der Agent auch eine Empfangsbestätigung erhalten, ohne daß er abstreiten kann, auf diese auch tatsächlichen Zugriff zu haben (siehe dazu 4.4.1). Soll der Einkauf anonym erfolgen, so kann diese Methode nicht verwendet werden, da der Händler notwendigerweise den Namen für die Verrechnung kennen muß (anders bei SET).

In rechtlicher Hinsicht ist zu bedenken, daß eine ungesicherte Übertragung der Kartendaten im Internet nach den AGBs der Karten-Anbieter eine Sorgfaltsverletzung darstellt (z. B. Punkt 10.1 bei Mastercard [Mastercard-AGB]), und daher keine Haftung für nachfolgenden Mißbrauch mehr besteht. Um dies auszuschließen muß daher der Agent auf seiner gesamten Reise verschlüsselt, und ein Zugriff von anderen Agenten aus unmöglich sein. Werden die Daten *vom ordnungsgemäßen Empfänger* (d. h. einem anderen Agenten oder dem Agentensystem) nicht ordentlich gesichert oder mißbraucht, so besteht jedoch Schutz (siehe hierzu § 31a Konsumentenschutzgesetz [KSchG]).

Um ein Ausspähen der Daten auf dem Transport zu verhindern, sollte die Übertragung der Daten nur verschlüsselt erfolgen. Hier kann ein Agentensystem sogar von Vorteil sein, da dies nicht mehr notwendig ist, wenn ein Agent auf dem Server des Händlers mit dem Agentensystem (oder einem Agenten) ebendieses Händlers eine Transaktion durchführen möchte: Ein Abhören ist nur dem direkten Vertragspartner möglich, welcher die Daten ohnehin erhält. Für einen Agenten ist es daher nicht erforderlich, *diese* Kommunikation zu verschlüsseln.

3.2.2.8 Kreditkarte (SET)

Eine sichere Art der Bezahlung mittels Kreditkarte über das Internet ist SET (Secure Electronic Transaction). Hierbei erfolgt eine Verschlüsselung der Daten und die Identifizierung der Teilnehmer. Weiters werden Informationen nur soweit weitergegeben als es unbedingt notwendig ist. Hierzu ist es erforderlich, daß sowohl der Käufer, der Händler, als auch das Payment-Gateway ein Zertifikat besitzen (Siehe [SET], [Mastercard-SET]). Der Ablauf einer Transaktion bei Durchführung mittels eines Agenten wäre folgender:

1. Der Agent wählt die Zahlungsart SET aus. Anschließend überprüft er die Zertifikate des Händlers und des Payment-Gateways. Dadurch ist sichergestellt, mit wem der Agent „spricht“. Der erste Punkt könnte bei Agenten entfallen, falls sich nicht nur der Agent mit einem Zertifikat bei der Ankunft ausweisen muß, sondern auch der Server und mit diesem das Geschäft abgeschlossen wird. Befindet sich der Agent auf dem Server, so besteht das übliche Problem der Möglichkeit der Beeinflussung. In dieser Phase stellen sich ansonsten keine Sicherheitsprobleme, da nur eine Überprüfung von Zertifikaten anderer Teilnehmer notwendig ist.
2. Der Agent erstellt eine Zahlungsaufforderung, welche eine Prüfsumme über die Rechnung (hiermit wird der Inhalt der Transaktion festgelegt, ohne daß das Kreditkarteninstitut oder die Bank diesen erfährt) sowie die Währung und den Betrag enthält. Diese wird vom

Agenten signiert und zusammen mit seinem Zertifikat an den Händler verschickt. Hier stellt sich das Problem, daß der Agent hierfür den privaten Schlüssel zum SET-Zertifikat besitzen muß. Dieser Schlüssel ist jedoch strengstens geheim zu halten, um Mißbräuche zu verhindern und die Haftung nicht zu verlieren (siehe oben).

3. Der Händler überprüft zuerst die Unterschrift um sicherzugehen, daß es sich tatsächlich um den berechtigten Zertifikatsinhaber handelt. Da die Kreditkartennummer (und der zugehörige Name) nicht im Zertifikat enthalten ist, erhält er keinen Zugriff darauf. Anschließend schickt er eine Autorisierungsanfrage an das Payment-Gateway, welche die soeben erhaltenen Daten des Kunden sowie eine gleichartige Meldung, welche mit seinem privaten Schlüssel unterschrieben ist, enthält.
4. Das Payment-Gateway überprüft beide Signaturen (Händler und Käufer) sowie ob die darin enthaltenen Daten identisch sind (gleicher Betrag in gleicher Währung und gleiche Rechnungs-Prüfsumme). Sind alle Prüfungen erfolgreich wird Betrag und Währung sowie die Kartennummer (welche das Gateway an Hand des Zertifikates aus seiner eigenen Datenbank feststellt) an die Autorisierungsstelle weitergeleitet, welche die Bonität, nicht erfolgte Sperre der Karte, Einhaltung des Limits, etc. überprüft.
5. Die Antwort des Autorisierungsrechners wird vom Gateway an den Händler zurückgeschickt, welcher mit dieser Antwort die Bestätigung für die erfolgreiche Buchung besitzt. Nun werden diese Daten (=Transaktionsbestätigung für den Kunden) an den Käufer weitergeleitet und die Auslieferung (wobei der Händler auch einen Übergabenachweis benötigt) kann beginnen.

Vorteile dieses Systems sind, daß der Händler keinen Zugriff auf die Kreditkartennummer erhält, während die Kreditkartenfirma nur erfährt, bei wem der Kunde um welchen Betrag eingekauft hat (unerläßlich), jedoch nicht was er dort gekauft hat.

In Bezug auf Agenten ergeben sich keine neuen Probleme, lediglich Zertifikate sind zu speichern (unproblematisch) und ein privater Schlüssel zu sichern, z. B. durch Trennung in zwei Agenten, wobei der „Heimat-Agent“ diesen enthält. Ein größeres Problem stellt dar, daß die notwendigen Protokolle relativ komplex sind und derzeit keine frei verfügbaren Implementierungen existieren.

3.2.3 Bezahlung durch Agenten

Für den Vorgang der eigentlichen Bezahlung von Agenten stellen sich sowohl programmtechnische als auch organisatorische Probleme: Wogegen muß ein Agent bei welcher Zahlungsart geschützt werden, bzw. wo kann sie Einsatz finden.

3.2.3.1 Schwierigkeiten bei der Programmierung

Bei der tatsächlichen Implementierung der Zahlung durch Agenten stellen sich spezifische Probleme welche über die „normalen“, wie etwa die Geheimhaltung von Daten, hinausgehen:

? Verständnis der Protokolle: Je nach Zahlung werden unterschiedliche Protokolle verwendet. Dies kann einerseits direkt von der Zahlungsart abhängen (z. B. SET), sich jedoch auch aus dem Kauf ergeben (z. B. Einkauf in einem Internet-Shop erfordert das Parsen der Formulare, um die Daten in den entsprechenden Feldern einzufügen; auch muß der Agent hierzu das HTTP-Protokoll beherrschen und Formulare bedienen können).

? Kryptographische Verfahren erforderlich: Manche Zahlungsarten setzen die Verfügbarkeit von Kryptographie voraus. Aufgrund deren Komplexität ist ein Agent meist darauf angewiesen, diese in einer lokalen Bibliothek vorzufinden. Ist sie dort etwa nicht vorhanden oder unterstützt sie das konkrete Verfahren oder die vorhandene Schlüssellänge nicht, so kommt dem Agenten die Zahlungsmöglichkeit ohne sein Verschulden abhanden. Um sicherzugehen, müßte der Agent daher *alle* erforderlichen Programme und Bibliotheken mit sich führen.

? Unterschiedlichkeit der Zahlungsverfahren: Die möglichen Zahlungsverfahren sind sehr unterschiedlich. Es ist daher für jede unterstützte Methode eine eigene Implementierung erforderlich, welche auf beiden Seiten vorhanden und kompatibel sein muß. Intelligente Agenten könne hier zwar eine Anpassung in gewissem Umfang vornehmen, doch zumindest die Grundstruktur muß festgelegt sein.

? Teilweise approbierte Software nötig: Bei der Verwendung von SET ist eine Überprüfung der Software erforderlich, bevor das SET-Zeichen verwendet werden darf ([SETCO]). Dies erfordert einen zusätzlichen Aufwand und kann eventuell weitere Probleme hervorrufen, da SET nicht für automatische Bedienung (z. B. durch Agenten), spezifiziert wurde.

3.2.3.2 Eigenschaften von Zahlungsarten aus Agenten-Sicht

Bei der direkten Bezahlung durch Agenten ist die wichtigste Abgrenzung, ob dieser Vorgang auch auf einem unsicheren Server (= dem kein Vertrauen entgegengebracht wird) erfolgen kann. In diesem Abschnitt wird nur der eigentliche Zahlungsvorgang betrachtet. Die Entscheidung ob bzw. was gekauft wird, bleibt außer Betracht. In der Tabelle 2 ist für jede Zahlungsart aufgeführt, welche für Agenten wichtigen Eigenschaften gegeben sind, oder für die Verhinderung von Angriffen gegeben sein müssen. Da bei Nachnahme und Überweisung die Bezahlung in zwei aktive Teile zerfällt (Bankeinzug erfolgt durch den Akzeptierenden), die Mitteilung der Zahlungsart und die eigentliche Erfüllung, werden diese hier getrennt dargestellt. Da die Erfüllung der Nachnahme nicht von Agenten durchgeführt werden kann wurde diese Zeile in der Tabelle ausgelassen.

Die Bedeutung der einzelnen Spalten wird vorab erläutert:

1. Von Agenten verwendbar: Können Agenten diese Zahlungsart verwenden oder nicht? Physische Zahlungsarten wie Bargeld oder die Zahlung mit Varianten von Chip- oder Magnetstreifenkarten sind nicht möglich.
2. Für unsichere Server geeignet: Kann diese Zahlung auch auf einem unsicheren Server ohne Gefahr durchgeführt werden? Die Negation dieser Spalte ergibt, ob geheime Daten erforderlich sind, welche anderen Agenten oder insbesondere dem Server nicht bekannt werden dürfen.
3. Geheime Daten: Sind geheime Daten erforderlich, welche der Zahlungsempfänger nicht erfahren darf *außer* den zu übermittelnden oder zusätzlichen unabhängigen Daten (z. B. weiteres Geld) für die Zahlung, welche bis zu diesem Zeitpunkt bzw. vor anderen Agenten/dem Server geheim bleiben müssen? Ein Beispiel hierfür ist der private Schlüssel für SET-Zahlungen, welcher vom privaten Schlüssel des Agenten unterschiedlich ist.
4. Programmaufwand: Wie hoch ist der Aufwand der zur Programmierung dieser Zahlungsart erforderlich ist? Besonderes Augenmerk wird hierbei auf notwendige Bibliotheken bzw. die Größe des Codes gelegt, welche bei mobilen Agenten besonders bedeutsam ist. So ist etwa beim SET-Protokoll eine hoher Aufwand für Kryptographie erforderlich.
5. Agenten-Secret-Key ausreichend: Findet man mit dem geheimen Schlüssel des Agenten das Auslangen oder sind weitere kryptographische Informationen erforderlich (z. B. zusätzliche Schlüssel, TANs, …)?
6. Abhörschutz erforderlich: Muß die Kommunikation zwischen dem Zahlendem und dem Empfänger vor Abhören geschützt werden, oder sind die Informationen für andere nicht

(mehr) verwertbar? Bei Zahlung mit elektronischem Geld ist dies erforderlich, da sonst der Abhörende mit der Einreichung dem tatsächlichen Empfänger zuvorkommen könnte.

7. Zahlungs-Wiederholung möglich: Ist es möglich, die „Zahlung“ beliebig oft zu wiederholen, ohne daß es zu einer Bereicherung des Empfängers kommt? So kann etwa die Zahlung per Nachnahme (=Auswahl dieser aus den möglichen Zahlungsarten) beliebig oft erfolgen, genauso wie die Kreditkartennummer dem richtigen Empfänger beliebig oft mitgeteilt werden kann.
8. Zerstörungs-Schutz notwendig: Muß der Agent vor Zerstörung geschützt werden bzw. eine Sicherheitskopie der Daten vorhanden sein? So sind etwa TANs (meist) auch schriftlich vorhanden, die Kontonummer ohne Probleme wiederbeschaffbar, der private Schlüssel (SET) jedoch nicht, da nur elektronisch vorhanden.

Zahlungsart	**Von Agenten verwendbar**	**Für unsichere Server geeignet**	**Geheime Daten erforderlich**	**Programm-Aufwand**	**Agenten-Secret Key ausreichend**	**Abhörschutz erforderlich**	**Zahlungs-Wiederholung möglich**	**Zerstörungsschutz notwendig**
Bargeld, Quick	Nein	-	-	-	-	-	Nein	Ja
Nachnahme	Ja	Ja	Nein	Minimal	Ja	Nein	Ja	Nein
Überweisung (Bezahlung)	Ja	Ja	Nein	Minimal	Ja	Nein	Ja	Nein
Überweisung (Durchführung)	Ja	Nein	Ja	Sehr hoch	Nein	Nein	Nein	Nein
Bankeinzug	Ja	Ja	Nein	Gering	Ja	Ja	Ja	Nein
Elektronisches Geld	Ja	Nein	Nein	Hoch	Ja	Ja	Nein	Ja
Gutschein (Namentlich ausgestellt)	Ja	Ja	Nein	Gering	Ja	Nein	Ja	Ja
Daten, Inhabergutschein	Ja	Nein	Nein	Gering	Ja	Ja	Ja	Ja
Kreditkarte (Nummer, Ablaufdatum)	Ja	Nein	Nein	Gering	Ja	Ja	Ja	Nein
Kreditkarte (SET)	Ja	Nein	Ja	Sehr hoch	Nein	Nein	Nein	Ja

Tabelle 2: Eigenschaften von Zahlungsarten in Hinsicht auf Durchführung durch Agenten

3.2.4 Ein Protokoll zum sicheren Austausch von Daten

Will ein Agent bestimmte Daten von einem Server kaufen auf dem er sich befindet, und zwar gegen die Lieferung von anderen Daten (Tausch, aber auch z. B. E-Cash), so kann nachfolgender Algorithmus verwendet werden. Er garantiert die folgenden Eigenschaften:

1. Der Server kann nicht abstreiten daß er bereit ist, genau bestimmte Daten zu verkaufen.
2. Hat der Agent (bzw. dessen Heim-Server) angenommen, so kann er die Bezahlung nicht mehr verweigern oder zurücknehmen.
3. Nur der Server kann die gelieferten Zahlungsdaten verwenden.
4. Wurde das Angebot angenommen, so kann der Server die Lieferung nicht mehr aus irgendeinem Grund verweigern.
5. Der Agent kann jederzeit nachprüfen, ob die gelieferten Daten auch tatsächlich dem Angebot entsprechen.

Eine Erweiterung um zeitliche Elemente ist möglich, doch ist hierzu eine trusted third party notwendig welche Zeitstempeldienste anbietet. In diesem Fall kann der Server auch eine Frist für die Annahme des Angebotes setzen.

Problematisch ist bei diesem Protokoll, daß die Lieferung der Daten durch den Server nicht erzwungen werden kann: Es kann lediglich im Nachhinein mit Sicherheit festgestellt werden, daß er zur Lieferung verpflichtet gewesen wäre. Ist auch eine Garantie der Lieferung gewünscht, so ist jedenfalls eine vertrauenswürdige dritte Partei notwendig, welche als Treuhänder fungiert und die *vollständigen* Daten (anstatt nur zweier Schlüssel) zwischenzeitlich speichert.

Dieses Protokoll ist analog dem Problem der „fair non-repudiation“ (siehe [Zhou] und [Asokan et al. 1996]), mit dem Unterschied daß es sich hier um eine zweiseitige (und nicht eine einseitige) Transaktion handelt (nicht nur A ? B, sondern auch B ? A, beides muß gleichzeitig erfolgen und miteinander untrennbar verknüpft sein).

Voraussetzung für einen gerechten Ablauf ist, daß beide Parteien jederzeit Informationen von der dritten Partei *abrufen* können.

Der letzte mögliche Zeitpunkt für einen einseitigen Abbruch ist vor Schritt 3. Erfolgte dieser von beiden, so kann die Transaktion nicht mehr vorzeitig beendet werden.

Folgende Notation wird bei der Beschreibung verwendet:

? A, B: Kommunikationspartner

? TTP: Vertrauenswürdiger Dritter

? „a ? b: x“: a sendet x an b

? „a ? b: x“: a holt x von b ab

? TK_A, TK_B: Transaktions-Schlüssel von A bzw. B, welche zur Verschlüsselung der Nachrichten verwendet werden

? AK_A, AK_B: Agenten-Schlüssel von A bzw. B, welche zur Signierung von Elementen verwendet werden

? $eTK_x(y)$: Mit dem Transaktions-Schlüssel TK_x verschlüsselter Inhalt y

? $sAK_x(y)$: Mit dem Agenten-Schlüssel AK_x signierter Inhalt y, welcher extrahierbar ist (=externe Signatur)

? M_A, M_B: Nachricht von A bzw B an den anderen Partner (=Ausgetauschte Leistungen)

? $C_x=eTK_x(M_x)$: Mit dem Schlüssel TK_x verschlüsselte Nachricht M_x

? $L_x=H(M_x,TK_x)$: Verbindung zwischen Nachricht und Schlüssel (Anwendung einer sicheren Hashfunktion auf die Konkatenation der beiden Elemente)

? $L=H(L_A,L_B)$: Verbindung der beiden auszutauschenden Nachrichten (Leistung und Gegenleistung)

? $UB_A=sAK_A(B,L,C_A)$: Ursprungsbeweis für B, daß A die Nachricht M_A an B gesendet hat (=Einverständnis mit Austausch)

? $EB_A=sAK_A(B,L,C_B)$: Empfangsbeweis für B, daß A die Nachricht M_B des B empfangen hat (=vorläufige Empfangsbestätigung)

Das Protokoll läuft in folgenden Schritten ab, wobei Teile a und b gleichzeitig erfolgen können, numerierte Schritte jedoch erst nach Abschluß des vorherigen Schrittes (siehe auch Abbildung 45):

1a) A ? B: L_A, C_A: A sendet an B die verschlüsselte Nachricht sowie die Verbindung zwischen der Nachricht und dem Transaktionsschlüssel

1b) B ? A: L_B, C_B: B sendet an A ebenso die entsprechenden Daten

2a) A ? B: UB_A, EB_A: A berechnet aus dem eigenen L_A und dem empfangenen L_B L. Hieraus werden der Empfangsbeweis und der Ursprungsbeweis erstellt und an B geschickt.

2b) B ? A: UB_B, EB_B: B handelt analog zu A.

3a) A ? TTP: UB_B, $sAK_A(B,L,TK_A)$: A schickt an die dritte Partei (TTP) die Transaktionsdaten: Mit wem die Transaktion durchgeführt wird (B), die Verbindung der beiden Leistungen L, sowie den Transaktionsschlüssel TK_A zur Entschlüsselung seiner Leistung C_A. UB_B ist notwendig, um die TTP vor einer denial of service-Attacke durch A zu schützen, indem diese sehr viele solcher Nachrichten schickt (sofern TTP die Signatur durch B überprüfen kann).

3b) B ? TTP: UB_A, $sAK_B(A,L,TK_B)$: Auch B schickt diese Daten, nur mit seinem Transaktionsschlüssel TK_B.

4) $PUB_L=sK_{TTP}(sAK_A(B,L,TK_A),sAK_B(A,L,TK_B))$: Sobald sowohl A als auch B ihre Schlüssel bei TTP hinterlegt haben, überprüft diese die Übereinstimmung von L und berechnet PUB_L, welches mit dem privaten Schlüssel der TTP signiert wird.

5a) A ? TTP: PUB_L: A holt nun von TTP PUB_L ab, woraus TK_B extrahiert wird.

5b) B ? TTP: PUB_L: B extrahiert TK_A.

6) A/B können nun C_B/C_A entschlüsseln und erlangen so M_B/M_A, die Leistung der jeweils anderen Partei.

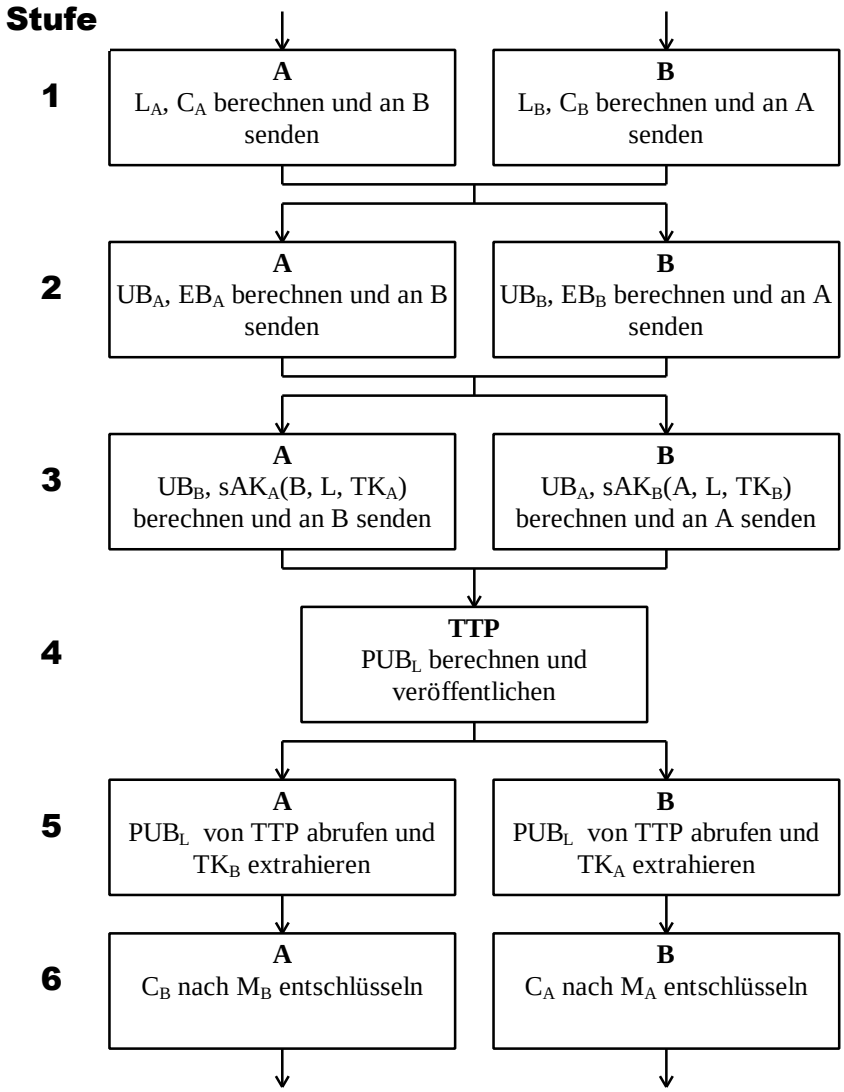

Abbildung 45: Ablaufdiagramm des Protokolls

Streitet eine der Parteien (z. B. A) ab, die Nachricht abgeschickt zu haben, so kann ein unparteiischer Dritter (z. B. die TTP) dies folgendermaßen überprüfen:

B stellt bereit: UB_A, AK_A, M_A, L_B, PUB_L

Der Dritte überprüft die Signatur von UB_A mittels AK_A. Er weiß damit, daß C_A von A stammt, an B gerichtet war und zur Transaktion gehört, die durch L gekennzeichnet ist. Durch die Prüfung von PUB_L und die Prüfung der Gültigkeit von $sAK_A(B,L,TK_A)$ ist auch sichergestellt, daß TK_A von A stammt und zur selben Transaktion gehört. Läßt sich C_A mittels TK_A nach M_A entschlüsseln, so ist auch sichergestellt, daß dies die zugehörige Nachricht ist. Aus L_B, M_A und TK_A läßt sich L berechnen und damit überprüfen, daß der Schlüssel von Anfang an zu dieser Transaktion gehörte und nicht später ausgewechselt werden konnte. Ein anderes L_B kann nicht verwendet werden, da B hierfür aus L und L_A dieses berechnen müßte (Hashfunktion = Einwegfunktion!).

Streitet eine der Parteien (z. B. A) ab, die Nachricht von B empfangen zu haben, so kann auch dies von einem Dritten überprüfen werden:

B stellt bereit: EB_A, AK_A, L_A, PUB_L

Der Dritte überprüft die Signatur von EB_A mittels AK_A. Er weiß damit, daß C_B von B an A gesendet, dort auch tatsächlich empfangen wurde, und dies im Rahmen der Transaktion L erfolgte. Da PUB_L von TTP signiert ist, ist damit sichergestellt, daß A jederzeit Zugang zum Schlüssel TK_B hatte, und daher C_B jederzeit entschlüsseln konnte. Die restlichen Prüfungen erfolgen wie beim obigen Fall.

Für den Sonderfall daß ein Agent mit dem Server auf dem er sich befindet solch ein Protokoll durchführen möchte, ergeben sich einige Besonderheiten da z. B. der Schlüssel des Agenten sich nicht auf diesem Rechner befinden darf (der Rechner könnte darauf Zugriff erlangen). Ein Teil des Protokolles muß daher über einen *vierten* Rechner durchgeführt werden, welchem *nur der Agent* vertrauen muß. Das Agentensystem hat zwar Zugriff auf den Agenten und damit auf die zu empfangende Nachricht, doch kann auf diese Weise weder eine Sendebestätigung des Agenten für die Nachricht noch eine Empfangsbestätigung für die Zahlung gefälscht werden. Der Agent muß daher alle seine Nachrichten über einen (für ihn vertrauenswürdigen) anderen Rechner senden, welcher alleiniger Besitzer des geheimen Schlüssels des Agenten ist. Für diesen Rechner stellt sich jedoch die Frage, ob der Wunsch auf Austausch auch tatsächlich vom Agenten stammt: Dies kann er nicht mit Sicherheit feststellen (Fälschung durch den Server ist möglich). Dennoch hat dieser Ansatz einen Vorteil: Der Transaktionspartner des Agenten kann dann nicht mehr abstreiten diese Transaktion durchgeführt zu haben und der Agent muß beim Verlassen dieses Rechners im Besitz der zu empfangenden Informationen sein.

3.2.5 Nachweis der Übergabe von Informationen an den Agenten

Im Zusammenhang mit dem vorigen Protokoll (aber auch unabhängig davon, siehe oben) stellt sich die Frage, wie ein Server beweisen kann, daß er einem Agenten bestimmte Informationen übergeben hat (z. B. eine Quittung oder die gekauften Daten). Da ein Server in der Lage ist, über einen Agenten die volle Kontrolle auszuüben, könnte der Agent jederzeit die Daten löschen und behaupten, der Server hätte sie ihm nie übergeben oder sie später durch direkten Zugriff auf seinen Speicherbereich gelöscht.

Das Protokoll im vorigen Abschnitt kann zwar nicht verhindern daß ein System eine Transaktion gegen den Willen des Agenten vortäuscht, doch ist das System dann auch gezwungen, die

Daten dem Agenten zu übergeben. Da die Spezifikation der Daten ebenso über den sicheren (vierten; siehe oben) Rechner erfolgt kann auch geprüft werden, ob die gelieferten Daten diesen entsprechen. Enthält nun ein Agent einen Speicherbereich, auf den der Agent selbst nur Lesezugriff (aber keinen Schreibzugriff!) hat, so kann ein Rechner die Daten dort hineinstellen und sicher sein, daß der Agent sie nicht löscht und so den Nachweis der tatsächlichen Übergabe führen. Um sich gegen Modifikationen durch nachfolgende Rechner abzusichern, auf denen sich der Agent später aufhält, ist es notwendig, daß die Weitergabe des Agenten ebenfalls gesichert erfolgt. Hierfür ist keine so komplexe Transaktion nötig: Das Agentensystem kann sich immer eine lokale Kopie des Agenten aufheben und, falls es die signierte Empfangsbestätigung nicht erhält, die Kopie jederzeit erneut (oder an einen anderen Rechner, z. B. den Heimatrechner) übertragen, bis er schließlich eine solche erhält.

Dies bedeutet gleichzeitig auch eine gewisse Absicherung des Agenten: Aus den (signierten und daher unveränderlichen) Anfangsparametern und dem Zustand beim Erreichen eines Servers lassen sich bestimmte Ergebnisse sicher verneinen, sodaß etwa die Übergabe (und im Gegenzug erfolgte Übernahme von Geldeswert) von bestimmten Informationen ausgeschlossen werden kann, und daher auf Server-Manipulationen beruhen muß. Dieser kann aufgrund der read-only-Eigenschaft nicht behaupten, die Daten wären vom Agenten erzeugt worden, bzw. (wegen des Hand-offs) die Verantwortung nicht auf nachfolgende Rechner abschieben.

3.3. Ausgewählte Rechtsfragen im Zusammenhang mit Agenten

In Zusammenhang mit Agenten stellen sich viele Rechtsfragen, auf die bisher noch wenig Augenmerk gelegt wurde. Mit der steigenden Bedeutung insbesondere im Bereich des E-Commerce ist es jedoch notwendig, auch diesen Bereich näher zu untersuchen und schon beim Entwurf von Agentensystemen miteinzubeziehen.

Die folgenden Erörterungen beziehen sich auf die österreichische Rechtslage, doch haben sie großteils eine weitere Gültigkeit, da z. B. sowohl Signatur- als auch Datenschutzgesetz auf EU-Richtlinien beruhen und daher EU-weit sehr ähnliche bis identische Regelungen bestehen.

Für eine Zusammenfassung wichtiger Aspekte dieses Abschnitts siehe [Sonntag 2002a].

3.3.1 Rechtliche Klassifizierung von Handlungen von Agenten

Eine Einordnung von Handlungen in den rechtlichen Rahmen erfordert zunächst eine Feststellung, was ein Agent im *Rechtssinne* ist. Da es sich nicht um eine natürliche Person handelt,

könnte es höchstens eine juristische sein. Hierbei handelt es sich jedoch entweder um Personengesamtheiten oder Gebilde, denen vom Gesetz ausdrücklich die Rechtsfähigkeit zugeordnet wird (z. B. GmbH, Gemeinden). Da beide Alternativen nicht zutreffen, kann es sich nur mehr um ein Werkzeug handeln, dessen sich eine (natürliche oder juristische) Person bedient, selbst wenn es sich hierbei um sehr komplexe Vorgänge handelt (welche viele Benutzer unter Umständen nicht in allen Details kennen, und deren Ergebnisse sie nicht vorhersagen können; was sich jedoch nicht von vielen anderen komplexen Systemen unterscheidet).

Ein Agent ist daher auch nicht im Rechtssinne handlungsfähig und seine Handlungen müssen immer einer Person zugerechnet werden, welche dafür verantwortlich ist. In Frage kommen demnach, neben Anderen, insbesondere der „Besitzer“ bzw. der Betreiber des Servers, auf welchem sich der Agent befindet. Um eine zuverlässige Identifizierung dieser Person für Dritte zu ermöglichen, ist ein Agent mit einem elektronischen Identität (z. B. Zertifikat) auszustatten, aus welcher der „Besitzer“ hervorgeht. Handelt es sich hierbei um eine natürliche Person so ergeben sich keine weiteren Probleme, doch ist für juristische Personen (zumindest in Österreich, siehe [Sonntag 2000]) die Ausstellung von Zertifikaten für rechtsgültige Unterschriften nicht vorgesehen. In diesem speziellen Fall ist dies jedoch unerheblich, da lediglich eine tatsächliche Zuordnung zu einer Person erforderlich ist, welche durch eine digitale Signatur hergestellt werden kann. Da hierbei nicht das Erfordernis der Schriftlichkeit besteht (diese müßte explizit angeordnet sein; Grundsatz der Formfreiheit § 883 ABGB), kann dies auch direkt durch eine juristische Person erfolgen.

Bei praktisch allen nützlichen Agenten wird jedoch noch eine weitere maßgebende Person hinzukommen: der Hersteller des Agenten. Auch er muß eindeutig identifizierbar sein (zumindest für den Besitzer des Agenten), sodaß eine Abgrenzung der Verantwortlichkeit möglich ist. Es ist daher erforderlich, daß der Hersteller den Programmcode signiert. Dadurch ist eine jederzeitige Überprüfung seiner Identität und besonders der Unverändertheit des Codes möglich. Die genaue Abgrenzung zwischen diesen beiden Personen ist jedoch schwierig: Daten (=Besitzer) können auch als Programm (=Programmierer) gesehen werden und umgekehrt.

Eine Abgrenzung kann dadurch erfolgen, daß der Programmierer genau zu definieren hat, welchen Wert und welche Bedeutung die Eingaben haben können bzw. müssen. Für deren konkreten Inhalt ist der Benutzer verantwortlich, für die darauf erfolgenden Aktionen hingegen der Programmierer. Weiters sind dem Benutzer jedoch auch noch alle Folgen zuzuordnen, welche sich gemäß der Dokumentation des Agenten aus den Eingaben sowie den empfangenen Daten

ergeben. Hierbei wird auf die Verständlichkeit der Beschreibung für den Adressaten (=Benutzer) abzustellen sein, wobei ein verständiger Computernutzer zum Vergleich heranzuziehen sein wird. Inkonsistenzen könnten in Analogie zu AGBs gesehen werden: Beide werden einseitig vom Anbieter formuliert und zur Grundlage des Vertrages (Kauf- oder Benutzungsvertrag) gemacht, und fallen daher dem Programmierer/Hersteller (als Ersteller der Dokumentation) zur Last. Besondere Eigenschaften und Verhalten sowie wichtige Entscheidungen sollten, wegen des großen Gesamtumfangs, besonders hervorgehoben werden. Im Gegensatz zu normalen Software-Verträgen, wo die Dokumentation bloß ein notwendiger Produktbestandteil ist, handelt es sich hier viel stärker um einen echten Teil der Hauptleistung des Vertrages. Es ergibt sich ein Übergang des verbindlichen Teils von der Spezifikation (relativ geringer Umfang) hin zur Dokumentation (vollständige Beschreibung): Nur die wenigsten Agenten werden speziell erstellt werden, sodaß eine exakte Spezifikation vorliegt. Vielmehr werden Benutzer aus einer Anzahl erhältlicher den gewünschten Agenten auswählen.

Fraglich ist auch die Stellung des Programmierers gegenüber Dritten Personen, insbesondere Betreibern von Rechnern, auf denen von ihm erstellte Agenten ausgeführt werden. In den meisten Fällen wird keine Vertragsbeziehung zwischen diesen beiden bestehen, sodaß lediglich eine rein deliktische Haftung verbleibt. Eine Möglichkeit wäre, einen Vertrag mit Schutzwirkung zugunsten Dritter anzunehmen. Dies setzt jedoch voraus, daß aus dem Vertrag zwischen Programmierer und Agentenbesitzer andere Personen stark gefährdet sind, dies vorhersehbar ist, und diese Personen der Sphäre eines Vertragspartners zugehören. Im letzten Punkt ergeben sich jedoch Probleme, da zwar vorhersehbar ist, daß Server-Betreiber gefährdet werden können, doch ob diese der Sphäre des Agentenbesitzers zugehören (Schutzpflichten dieses gegenüber dem Betreiber?) ist zweifelhaft. Auch ist deren Anzahl (potentiell weltweit) und deren Unübersehbarkeit (es könnte jeder sein) zu weit, um diese Konstruktion noch annehmen zu können. Darüber hinaus würden keine bloßen Vermögensschäden ersetzt werden sondern nur Schäden an absoluten Rechtsgütern (Personenschäden, Sachschäden, ...). Der Programmierer ist daher nur gegenüber dem Besitzer des Agenten verantwortlich (siehe sogleich); gegenüber Dritten aber nur nach allgemeinen Regeln. Der wohl wichtigste Fall dürfte die absichtliche sittenwidrige Schädigung gemäß § 1295 Abs. 2 ABGB sein: z. B. Trojaner, Viren, Würmer. Die konkrete Kenntnis des Geschädigten ist nicht erforderlich.

Die Abgrenzung zwischen dem Besitzer des Agenten und dem Betreiber des Servers, auf dem der Agent seine Handlungen setzt, ist hingegen weniger problematisch: Der Besitzer ist für alle Handlungen verantwortlich, der Betreiber nur für das ordnungsgemäße Funktionieren des

Agentensystems (z. B. die korrekte Zustellung von Nachrichten). Problematisch ist hier die Beweisfrage: Wie soll ein Agent nachweisen, daß das Agentensystem falsch funktionierte? Dies führt in dem meisten Fällen zu einem Absturz oder ist für den Agenten nicht erkennbar.

Doch auch für einen Server-Betreiber könnte eine Haftung entstehen: Im Kontrast zu einem Netzwerks-Provider besteht auch eine Haftung für den Inhalt der gespeicherten Daten. Gemäß § 16 ECG/Art. 12 EC-RL ([ECG], [EC-RL]) ist ein Hosting-Provider nicht haftbar, wenn er keine positive Kenntnis von illegalen Aktivitäten besitzt und auch keine Fakten oder Umstände kennt, welche offensichtlich in diese Richtung deuten. Erlangt er Kenntnis, so ist er verpflichtet, sofort den Zugang zu sperren, oder die betroffenen Daten zu löschen. In Verbindung mit Agenten bedeutet dies, daß der Server-Betreiber nicht für Daten *in* Agenten haftet, solange er sie nicht kennt (z. B. solange er nicht versucht, diese zu kaufen oder wenn ein gefälschtes Zertifikat präsentiert wird). „Wissen" des Servers wird dem Betreiber zugerechnet. Daher müssen entsprechende Hinweise, welche einem Menschen auffallen würden, vom Agentensystem ebenfalls geprüft werden. Es ist darauf hinzuweisen, daß dies auch für Anbieter gilt, welche ihre Dienste unentgeltlich oder ohne Vertragsbeziehung mit dem Benutzer anbieten (§ 20 Abs. 2 EC-RL). Da die Handlungen eines Agenten nur eine „verlängerte Hand" des Besitzers darstellen, ist eine illegale Aktion desselben auch Grund für eine derartige Reaktion (da dann auch der Benutzer illegal handelt). Der Standard für die Prüfung wird jedoch eher niedrig anzusetzen sein da keine Überwachungsverpflichtung besteht (§ 19 Abs. 1 EC-RL: Ein Anbieter ist nicht verpflichtet, seine Benutzer zu überwachen oder nach illegalen Aktivitäten zu forschen). Gewissen Hinweisen, z. B. wiederholte abgelehnte Zugriffsversuche, muß jedoch nachgegangen werden. Eine große Schwierigkeit besteht darin, daß hier der Betreiber aufgerufen ist, die Illegalität einer Handlung oder von Daten zu beurteilen: Dies ist normalerweise Aufgabe von Gerichten. Eine falsche Beurteilung könnte auch zu Konsequenzen für ihn führen, da diesfalls eine Vertragsverletzung (aufgrund seiner Reaktion; siehe unten) vorliegen wird. Eine etwas andere Interpretation könnte hier Abhilfe schaffen: Die erforderliche Kenntnis betrifft nicht nur die Handlung bzw. die Daten sondern auch deren Gesetzwidrigkeit. Ein anderes Problem liegt in den zu ziehenden Konsequenzen: Der Betreiber soll die Daten entfernen oder den Zugang zu ihnen sperren. Die Idee hinter dieser Regelung sind Webseiten, welche ohne Probleme gesperrt werden können. Im Hinblick auf Agenten ist dies nicht so ohne weiteres möglich. Als Optionen stehen zur Verfügung: Den Agenten belassen wie er ist (entspricht nicht dem Gesetz), ihn zu seinem Ursprungs-Rechner zurückzuschicken (ebenfalls nicht erlaubt), seine Threads zu beenden, oder ihn vollständig zu zerstören. Die letzten beiden Variationen

sind möglich. Vollständige Zerstörung entspricht dem Löschen, während die Terminierung aller Threads dem Verhindern des Zugriffs entspricht (jedoch noch potentielle Beweismittel hinterläßt). Letzteres ist daher zu bevorzugen und ein Agent sollte nur dann komplett gelöscht werden, wenn er vor längerer Zeit beendet wurde, oder die Archivierung der Daten eine zu große Belastung darstellen würde. Eine Benachrichtigung des Besitzers ist nicht erforderlich aber anzuraten: Falls die Beurteilung falsch war und der Agent lediglich gestoppt wurde, kann er wieder gestartet und somit der Schaden minimiert werden.

In manchen Fällen kann auch die Staatsangehörigkeit eines Agenten von Bedeutung sein (siehe etwa das Französische Urteil im Fall Yahoo in welchem verboten wird, Nazi-Andenken französischen Bürgern anzubieten; dies würde auch für französische Agenten gelten [Yahoo-F]). Die Staatsangehörigkeit ist nach dem Besitzer des Agenten zu beurteilen, wobei jedoch dieser oft nicht feststellbar sein wird. So kann etwa aus der E-Mail Adresse kaum ein Rückschluß auf die Staatsangehörigkeit gezogen werden, da es sowohl generische Adressen gibt (z. B. „.net“, „.org“), als auch nationale Adressen, welche aber *nicht* nach der Staatsangehörigkeit vergeben werden, sondern meistens nach Standort des Providers (also nicht einmal gemäß *dessen* Staatsangehörigkeit). Ein gewisser Rückschluß auf die Staatsangehörigkeit ist möglich, wenn im Zertifikat des Besitzers der vollständige Wohnort angegeben ist: Dieser kann oftmals ausreichen (z. B. für oben angeführtes Urteil, welches hauptsächlich auf den Wohnsitz und weniger auf die tatsächliche Staatsangehörigkeit abstellt). Ansonsten bleibt nur die Möglichkeit eines staatlichen Zertifikates oder Attributzertifikates, analog zu einem Reisepaß, sich auf Angaben des Kunden zu verlassen, oder derartige Artikel überhaupt nicht anzubieten.

3.3.2 Elektronische Signaturen durch Agenten

Es stellt sich die Frage, ob eine elektronische Signatur durch einen Agenten rechtsgültig ist oder nicht (siehe dazu auch [Sonntag 2000] sowie [Sonntag 2002b]). Dies ist besonders im Bereich von E-Commerce wichtig. Hierbei ist zu differenzieren: Einerseits in Unterschriften, welche für den Agenten gelten (Unterschrift im eigenen Namen des Agenten), und andererseits in Unterschriften, welche der Agent als Werkzeug für den Besitzer vornimmt.

Gemäß der EU Signatur-Richtlinie [Sig-RL] können nur Personen als Unterschreibende auftreten. Es ist daher für einen Agenten nicht möglich, für sich selbst zu unterschreiben. Dies entspricht auch seiner mangelnden Rechtsfähigkeit. Entsprechendes ist bereits, oder wird erst, in den nationalen Umsetzungen der Richtlinie festgeschrieben (Österreich: [SigG], Deutschland [DSigG]).

Für einen *mobilen* Agenten ist es jedoch auch unmöglich, eine rechtsgültige Unterschrift für seinen Besitzer zu erstellen. Er kann zwar genauso als Werkzeug gesehen werden wie ein lokaler Rechner (der für eine Signaturerstellung immer benötigt wird, da eine tatsächlich händische Erstellung der Signatur praktisch unmöglich ist). Das Problem liegt vielmehr darin, daß eine *sichere* elektronische Signatur gem. §2 Z 3 lit. c SigG mit Mitteln erstellt werden muß, welche der Signator unter seiner alleinigen Kontrolle halten kann. Da ein Agent zum Zwecke der Signatur den privaten Schlüssel des Besitzers mitführen müßte und dieser vor einem anderen Rechner nicht gesichert werden kann (sehr hohe gesetzliche Anforderungen), kann lediglich eine „normale“ Unterschrift erfolgen, welche jedoch nicht mit einer handschriftlichen gleichgestellt ist. Auch darf der Auslösecode für eine Signatur nicht gespeichert werden (§ 7 Abs. 3 Sig-VO, [SIGVO]), was ebenfalls ein Hindernis ist.

All dies ist jedoch kein Hindernis für einen Agenten, „normale“ digitale Signaturen zu erstellen. Diese können die gleiche Stärke aufweisen, müssen jedoch andere Schlüssel verwenden. Da nur sehr selten ein Schriftformerfordernis besteht, dürfte dies kein Hindernis für die Verwendung von Agenten sein (z. B. Grundstückskäufe durch Agenten werden selten sein). Von größerer Bedeutung ist jedoch, daß die gesetzliche Vermutung des Inhalts der Erklärung bei Vorhandensein einer Unterschrift nicht mehr anwendbar ist. Derartige Signaturen können zwar immer als Beweismittel verwendet werden, doch sind zwei Fälle zu unterscheiden: Signaturen welche auf dem Heimatrechner erstellt wurden und solche, welche auf anderen Servern erzeugt wurden. Letztere sind vermutlich suspekt, da auch der Rechner die Signaturerstellung ausgelöst haben könnte. Dies führt auch zum Problem des Nachweises, auf welchem Rechner eine Signatur erstellt wurde. Möglichkeiten hierfür sind sichere Zeitstempel und Identifizierung des Host-Rechners zu diesem Zeitpunkt durch Logs, oder Gegensignaturen durch diesen Rechner.

In eingeschränkten Fällen ist es dennoch möglich, durch einen Agenten eine sichere elektronische Signatur erstellen zu lassen:

1. Sie erfolgt auf dem lokalen Rechner (immobile Agenten, oder mobile Agenten auf Rechnern, welche ausschließlich der Signator kontrolliert) unter direkter Kontrolle des Besitzers sowie Auslösung durch diesen.
2. Der Agent befindet sich auf einem anderen Rechner, besitzt jedoch nicht den privaten Schlüssel. Zur Erstellung von Signaturen schickt er das zu signierende Dokument (bzw. dessen Hash-Wert) zu seinem (sicheren) Heim-Rechner, wo die Signatur vorgenommen und das Ergebnis anschließend an den Agenten zurückgesendet wird.

Die ansonsten mögliche Verschleierung des Codes (Vermischung von Programm, Datenverarbeitung und Signiervorgang) ist hier nicht ausreichend, da es ein anderer Rechner immer in der Hand hat, den Agenten mit bestimmten Eingaben zu versorgen, sodaß im Endeffekt bestimmte Daten signiert würden und ein Signaturvorgang ausgelöst werden könnte, ohne daß dies vom Besitzer beabsichtigt ist. In diesem Fall ist das Gesetz nach seinem Zweck auszulegen: Ein Angreifer erhält zwar keinen Zugriff auf den privaten Schlüssel, doch ist der Zweck dieser Bestimmung eindeutig der, dem Inhaber die Möglichkeit zu geben, ausschließlich über die Erstellung einer Signatur zu verfügen, indem er jede andere Person davon ausschließen kann. Eben dies kann, bis auf den nachher erläuterten Sonderfall, mit diesem Vorgehen nicht gewährleistet werden.

Dieser Sonderfall besteht darin, erst bei Empfang genau bestimmter Daten einen vorbestimmten signierten Datensatz herauszugeben. Dies beruht auf dem Gedanken die signierten Daten mit den zu empfangenden Daten (oder deren Prüfsumme) zu verschlüsseln, sodaß erst mit Empfang dieser eine Entschlüsselung und damit ein Zugriff auf die signierten Daten möglich ist. Die Nachteile dieses Verfahrens liegen jedoch klar auf der Hand: Die zu signierenden Daten müssen exakt vordefiniert sein (z. B. kein Einfügen des Datums bei Verwendung als Empfangsbestätigung) und die zu empfangenden Daten müssen ebenfalls genau bekannt sein (keine Bestätigung für nicht oder nur nach ihrer *Spezifikation* bekannte Daten; im Hinblick auf die Spezifikation selbst ist dies jedoch durchführbar).

Gegenüber der EU existieren in den USA eine Vielzahl von Gesetzen welche elektronische Signaturen zum Inhalt haben. Größtenteils werden auch dort nur Personen als Signatoren akzeptiert. Es gibt jedoch auch Ausnahmen: Der Illinois Electronic Commerce Security Act [Illinois EC Security Act] etwa definiert ein Zertifikat u. A. als „names or otherwise identifies its subscriber or a device or electronic agent under the control of the subscriber“. Hier ist also explizit vorgesehen, daß auch die Unterschrift eines Agenten direkt rechtsverbindlich sein kann da er ein entsprechendes Zertifikat, welches auf ihn (und nicht seinen Besitzer!) ausgestellt ist, innehaben kann.

Im Electronic Signatures in Global and National Commerce Act [US Signature Act] ist eine Gleichstellung zu handschriftlichen Signturen für Signaturen durch Agenten vorgesehen:

> **Title 1, SEC. 101, (h) Electronic Agents.—**
>
> A contract or other record relating to a transaction in or affecting interstate or foreign commerce may not be denied legal effect, validity, or enforceability solely because its formation, creation, or delivery involved the action of one or more electronic agents so long as the action of any such electronic agent is legally attributable to the person to be bound.

Es sind jedoch hierbei zwei wichtige Einschränkungen zu beachten:

? Sie bezieht sich lediglich auf Verträge und Urkunden welche zwischen(bundes-)staatlichen und internationalen Handel betreffen. Dies ist in der nur hierfür ausreichenden Kompetenz des Kongresses begründet.

? Alle Handlungen des Agenten müssen rechtlich der Person zugeordnet werden können, welche gebunden werden soll. Wie diese Zurechnung, insbesondere die Abgrenzung zum Hersteller des Agenten oder zu sonstigen Personen, welche auf die Handlungen Einfluß haben, erfolgt, wird nicht ausgeführt. Es ist jedoch nicht vorausgesetzt, daß dieser Agent auch im Besitz dieser Person sein muß: Auch Agenten der anderen Partei oder Dritter können Verwendung finden, doch müssen ihre Handlungen dann aus einem besonderen Grund (z. B. vorheriger Vertrag, AGBs) der zu bindenden Person zuordenbar sein.

Diesen Einschränkungen steht ein erweiterter Anwendungsbereich gegenüber: Die Vorschrift bezieht sich nicht ausschließlich auf Signaturen, sondern auf jedwede Handlungen im Laufe des Lebens von geschäftlichen Daten („record“). Auch der bloße Transport von Daten durch Agenten wird explizit als kein Grund für eine Nichtbeachtlichkeit derselben erklärt. Ebenso ist einbezogen, daß Verträge, welche auf Grund von Verhandlungen von Agenten entstehen, sehr wohl rechtsverbindlich sind (sofern ihr Inhalt innerhalb des Geltungsbereiches liegt).

3.3.3 Handlungsort eines Agenten und Konsumentenschutz

Für die Feststellung der Rechtsordnung, welche z. B. auf einen Vertrag anzuwenden ist, muß das anwendbare Recht festgestellt werden, dem sich ein Agent durch seine Handlungen unterwirft. Da diese immer seinem Eigentümer / dem Hersteller zugerechnet werden, kann er als bloßes Transportmittel (Erklärungsbote; das Risiko für Fehler liegt beim Besitzer/Programmierer) angesehen werden. Das anwendbare Recht ist also grundsätzlich analog zu dem Fall zu bestimmen, in dem der Besitzer direkt handelt (z. B. über einen Webbrowser).

Besonderheiten können sich jedoch für mobile Agenten ergeben: Für Webseiten gilt die Regel, daß eine Webseite mit spezieller Ausrichtung auf ein Land (z. B. Domain-Name, Gestaltung mit Landesfahne oder explizite Erwähnung, aber auch Sprache) als Betätigung in diesem Land gilt, und daher den dortigen Anforderungen zu entsprechen hat. Im Falle von mobilen Agenten begibt sich jedoch der Agent zum Anbieter hin, um dort in einer einheitlichen Sprache mit dem Anbieter zu kommunizieren. Dies bedeutet, daß keine Tätigkeit im Ursprungsstaat des Agenten mehr vorliegt, sondern ausschließlich im Anbieterstaat.

Da Agenten auch anonym Geschäfte abschließen können, ergibt sich eine zusätzliche Differenzierung. Für einen Anbieter ist es dann nicht erkennbar, ob es sich um einen Konsumenten handelt oder nicht. Lediglich aus dem Inhalt der Bestellung könnte er Vermutungen ableiten.

Nach dem Europäischen Vertragsstatutübereinkommen (Art. 5 EVÜ; [EVÜ]) besteht bei Verbraucherverträgen (diese Definition unterscheidet sich von der im KSchG [KSchG]) zwar eine freie Rechtswahl, doch sind Bestimmungen, welche dem Verbraucher den Schutz im Staat seines gewöhnlichen Aufenthalts entziehen, unwirksam (unter einigen zusätzlichen Bedingungen). Eine ähnliche Bestimmung findet sich auch im KSchG in § 13a Abs. 2. Für E-Commerce ist hier die Klausel wichtig, daß dies gilt, wenn dem Vertragsabschluß ein ausdrückliches Angebot oder eine Werbung in diesem Staat vorausgegangen ist und der Verbraucher dort die zum Abschluß des Vertrages notwendigen Rechtshandlungen vorgenommen hat.

Aus den oben festgestellten Elementen kann man daher schließen, daß bei Verwendung eines mobilen Agenten dem Konsumenten dieser Schutz verloren geht: Er gibt seine Erklärung in Österreich ab und läßt sie vom Agenten zum Anbieter befördern. Daß der Besitzer des Agenten nicht die komplette Erklärung in allen Details abgibt ist hier unschädlich, selbst wenn der Agent eine größere Freiheit hat. Auch bei einem sonstigen Transportmittel (Telefon, aber auch WWW) werden die zu übertragenden Daten vielfach zwischenzeitlich in andere Formate umgewandelt. Dem geht jedoch in den meisten Fällen kein Angebot und keine Werbung in Österreich voraus (wenn doch, besteht eventuell Schutz für den Konsumenten; will man nicht den Agenten als Transporteur anders behandeln als etwa die Post, da es sich um einen direkt dem Partner unterstehenden Boten handelt). Im Gegensatz zu Webseiten läßt sich auch aus der verwendeten Sprache für die Kommunikation kein Anhaltspunkt gewinnen. Auch der Domainname des Anbieters hilft vermutlich nicht weiter, da dieser dann eher nach dem Inhalt des Angebots ausgewählt wird, als nach dem Land an das sich dieses Angebot richtet (also eher „.com“ als „.de“, „.at“, „.fr“, etc.).

Handelt es sich um einen Anbieter und einen Konsumenten innerhalb der Europäischen Union, so findet auch die Fernabsatz-Richtline [Fernabsatz-RL] Anwendung auf Geschäfte die von Agenten geschlossen werden (da ausschließlich Fernkommunikationsmittel Verwendung finden; auch ein Agent, welcher lokal Erklärungen abgibt, ist ein Fernkommunikationsmittel, da er als Bote des entfernten Besitzers fungiert). Hierbei bestehen insbesondere zwei getrennte Informationspflichten:

1. Bevor der Konsument seine Vertragserklärung abgibt: Diese Informationen müssen entweder auch direkt abrufbar sein, sodaß der Konsument sie lesen kann bevor er seine Erklärung abgibt (= den Agenten losschickt), oder der Agent muß die Möglichkeit haben vor Abschluß des Vertrages mit diesen Daten zurückzukehren. Da jeder Anbieter die Möglichkeit zur Informationsabfrage ohne Kaufzwang bietet (was wird zu welchem Preis verkauft, welche Eigenschaften besitzt die Ware, …) stellt sich hier kein Problem. Kann der Agent diese Daten abfragen, so hat er auch die Möglichkeit mit diesen zurückzukehren und seinen Besitzer zu informieren. Dies reicht aus.
2. Während der Erfüllung, aber spätestens mit Lieferung der Ware: Der Konsument muß zusätzliche Informationen erhalten, insbesondere über sein Rücktrittsrecht. Diese Informationen müssen ihm explizit zugesandt werden und zwar auf einem dauerhaften Datenträger. Die Anforderungen hierfür sind jedoch sehr niedrig angesetzt: Hat der Konsument die Möglichkeit die Informationen auszudrucken (z. B. bei einer E-Mail), so ist dies bereits ausreichend. Analog dazu wird es genügen die Informationen dem Agenten zu übergeben. Wird hierzu der oben erläuterte Ansatz des Read-only-Speichers beim Agenten verwendet, so kann der Anbieter diese Übergabe auch nachweisen. Analog zur E-Mail wird dies, da der Besitzer nach Rückkehr des Agenten Zugriff darauf hat und die Daten ausdrucken kann, für die Dauerhaftigkeit ausreichen.

3.3.4 Erklärungsbewußtsein

Durch die Autonomie von Agenten kann dessen Besitzer nicht jede Aktion genau voraussehen und planen. Es stellt sich daher die Frage, ob diese Äußerungen überhaupt rechtlich bindend sind. In Österreich ist ein explizites Erklärungsbewußtsein nicht erforderlich, wenn der sich Äußernde die Erklärung adäquat verursacht hat und hätte vermeiden können, daß der Empfänger sie als Erklärung auffaßt. Der Start eines Agenten und ihn mit einer Aufgabe zu betrauen ist eine ausreichende Verursachung. Durch die Nicht-Verwendung des Agenten hätte auch vermieden werden können, beim Partner einen Anschein zu erwecken. Verläßt sich daher eine

anderer Agent oder ein Server auf eine Äußerung eines Agenten, so ist sie als Erklärung anzusehen und rechtlich relevant.

3.3.5 Zugang von Erklärungen

Von rechtlicher Seite aus gesehen ist es von Bedeutung ob und wann eine Erklärung oder Nachricht einem Agenten (bzw. dessen Besitzer) zugegangen ist. Dies hat insbesondere für Fristen eine hohe Bedeutung (z. B. Angebotsfrist).

Allgemein kann gesagt werden, daß ein Agent als Empfangsbote seines Besitzers gilt: Dieser setzt ihn zur Entgegennahme von Erklärungen ein und ist daher für die darauf folgende Übertragung verantwortlich (Verlust bzw. Verzögerungen fallen ihm zur Last). Der Zugang liegt allgemein dann vor, wenn die Erklärung so in den Machtbereich des Empfängers gelangt ist, daß dieser sich nach allgemeiner Verkehrsauffassung Kenntnis davon verschaffen kann (§ 11 Abs. 3, § 13 Abs. 2 ECG). Insbesondere ist dies von der Uhrzeit unabhängig (siehe auch § 13 AVG), während früher die Meinung vertreten wurde, daß Zugang nur während der Geschäftszeiten möglich ist ([Zankl 2001]). Da ein Agent seinem Server „ausgeliefert“ ist, kann nicht klar gesagt werden, ob der bloße Empfang durch den Agenten bereits ausreicht. Allerdings kann der Server auch nicht für die Handlungen der nachfolgenden Hosts auf dem Rückweg des Agenten verantwortlich gemacht werden. In letzterem Fall ist der Agent ein Bote seines Besitzers und die Nachricht reist daher auf seine Gefahr. Der kritische Zeitpunkt für den Empfang ist daher das Verlassen des Rechners, auf dem die Nachricht empfangen wurde: Der Agent hat die Möglichkeit direkt nach Hause zurückzukehren und die Nachricht abzugeben. Tut er dies nicht, so ist dies sein Risiko.

Für den Nachweis des genauen Zeitpunktes des Zugangs ist eine Bestätigungs-Nachricht des Agenten sowie das Log über ein Hand-off an den nächsten Rechner erforderlich, sofern nicht eine dritte Partei (welche dann den Zeitpunkt bestätigen kann) zur Zustellung verwendet wird.

§ 11 ECG verlangt vom einem Diensteanbieter eine umgehende Bestätigung des Erhalts einer elektronischen Vertragserklärung. Dies betrifft Agenten jedoch nicht, da Abs. 4 eine Ausnahme gewährt, falls der Vertrag ausschließlich über Fernkommunikationsmittel abgeschlossen wird. Eine Bestätigung (Antwort-Nachricht) ist jedoch erforderlich, falls der Agent im Hinblick auf eine Offline-Werbung oder Benachrichtigung seine Erklärung abgibt.

3.3.6 Agenten und Datenschutz

Ein wichtiges Element in Bezug auf automatische Datenverarbeitung ist der Datenschutz. Dieser besitzt besonders in Verbindung mit mobilen Agenten eine große Bedeutung: Um sinnvolle Arbeit durchführen zu können, müssen Agenten viele personenbezogene Daten besitzen und wären daher ein interessantes Ziel für Angriffe in dieser Hinsicht. Aber auch unter einem anderen Aspekt ist der Datenschutz für sie wichtig: Da es sich eben nur um Agenten handelt, wenn auch teilweise um intelligente, kann es durchaus vorkommen, daß zu viele Informationen preisgegeben werden. In einem solchen Fall sollte der Empfänger nicht einfach alles Bekanntgegebene verwenden können, sondern nur diejenigen Daten, welche zur Aufgabenerfüllung erforderlich sind. Doch auch ein dritter Punkt ist zu berücksichtigen: Da Agenten Daten verarbeiten, könnten ein Benutzer durch sie das Datenschutzgesetz verletzen.

3.3.6.1 Allgemeines

Das österreichische Datenschutzgesetz ([DSG]) geht auf eine EU-Richtlinie zurück ([DSRL]), und ist somit inhaltlich in der gesamten EU gültig (da fast keine Abweichungen zur Richtlinie bestehen; diese sind hier auch nicht relevant).

Es werden ausschließlich personenbezogene Daten geschützt, wobei noch eine Differenzierung in direkt und indirekt (= für den konkreten Verarbeiter ist es unmöglich, die Daten mit legalen Mitteln und vertretbarem Aufwand einer bestimmten Person zuzuordnen) Personenbezogen erfolgt. Bestimmte Daten sind besonders geschützt (Sensible Daten: Rasse, Gesundheit, politische Meinung, etc.; in Verbindung mit Agenten fast immer nur von geringer bis keiner Bedeutung; sowie Strafrechtsdaten).

Allgemein ist das Minimalitätsprinzip zu beachten, welches einer der allgemeinen gesetzlichen Grundsätze für die Verwendung von personenbezogenen Daten ist: Daten dürfen nur insoweit verwendet werden, als es für den Zweck der Verarbeitung erforderlich ist (welcher aber frei festgelegt werden kann; Zustimmung des Betroffenen erforderlich).

3.3.6.2 Zustimmung zur Verarbeitung durch den Agenten

Eine Zustimmung zur Verwendung von Daten durch Agenten ist zwar möglich, doch nur in eingeschränktem Ausmaß. Sie liegt nur dann vor, wenn der Betroffene in Kenntnis der Sachlage in die konkrete Verwendung seiner Daten einwilligt. Diese kann auch konkludent erfolgen, solange es sich nicht um sensible Daten handelt (hier ist eine explizite Genehmigung notwendig). Diese Zustimmung zur Verarbeitung ist immer auf einen konkreten Zweck zu beziehen,

welcher jedoch relativ weit gefaßt sein kann. Eine generelle Zustimmung für jede Art von Verarbeitung ist aber nicht möglich. Ein Betroffener kann daher seine Einwilligung auch über einen Agenten abgeben, doch wird es sich hierbei meist nur um eine konkludente Erklärung handeln: Durch die Beauftragung des Agenten wird die Zustimmung zur Verarbeitung der erforderlichen Daten gegeben.

Gibt der Agent jedoch Daten weiter, an welche der Besitzer nicht dachte (und eventuell auch nicht denken mußte), so geht dies zu seinen Lasten, da er sich des Agenten bedient hat. Die Verarbeitung erfolgt daher rechtlich gesehen mit Zustimmung des Betroffenen, der eventuell diesen Erklärungsirrtum nachträglich geltend machen kann. Demgegenüber hat jedoch der Empfänger der Erklärung die nötige Sorgfalt zu beachten: Ist es möglich, daß der Besitzer die Zustimmung zur Verwendung dieser Daten gegeben hat, wenn die Aufgabe des Agenten offensichtlich eine ganz bestimmte Andere ist? Handelt es sich auch beim Empfänger um einen Agenten, so haftet jeder für die Programmierung und Parametrisierung seines Agenten, wobei der Maßstab jedoch eher großzügig festzusetzen sein wird.

3.3.6.3 Verletzungen des Datenschutzes durch Agenten

Auch durch einen Agenten kann der Datenschutz verletzt werden, falls dieser personenbezogene Daten verarbeitet, ohne daß der Besitzer des Agenten hierzu berechtigt ist. Zu beachten ist jedoch, daß vielfach kein Vorsatz vorliegen wird, wenn Agenten ihre „Befugnis“ überschreiten (da der Besitzer in der Regeln nichts davon weiß, zumindest beim ersten Auftreten und Bekanntwerden, und es auch nicht in Kauf nehmen wird). Wurde er jedoch bereits einmal auf diese Überschreitung *hingewiesen*, so wird Vorsatz in Form von wenigstens dolus eventualis vorliegen. Besonders zu berücksichtigen ist die Meldepflicht für Datenverarbeitungen. Ist keine Ausnahme gegeben, so ist auch eine Verarbeitung durch Agenten zu melden.

Es existiert jedoch auch ein Straftatbestand bei dem kein Vorsatz vorausgesetzt wird: Die gröbliche Außerachtlassung der erforderlichen Sicherheitsmaßnahmen. In diesem Fall ist es nicht einmal erforderlich, daß tatsächlich der Datenschutz verletzt wurde, sondern lediglich eine Sorgfaltsverletzung und die hiermit einhergehende *Gefahr* für Verletzungen reicht aus (Gefährdungsdelikt). Es sind daher auch bei Agenten Sicherheitsvorkehrungen zu treffen und der Weg und die Verarbeitung von personenbezogenen Daten genau zu dokumentieren und zu überprüfen. Da nur die *gröbliche* Außerachtlassung von Sicherheitsvorschriften strafbar ist, sind die Anforderungen nicht allzu hoch. Vorauszusetzen ist wohl als Mindestmaß:

? Bei mobilen Agenten die Verschlüsselung bei der Übertragung

? Isolation der Agenten voneinander

? Sicherung von Daten auf dem Server vor unberechtigtem Zugriff durch Agenten

3.3.6.4 Schaffung geschützter Daten durch Agenten

Bei Agenten handelt es sich zwar nicht um natürliche Personen, doch können sie Informationen über solche besitzen. Vom Agenten über eine Person weitergegebene Daten sind daher geschützt und ganz normal zu behandeln: Lediglich wenn sie keine Person betreffen sind sie nicht geschützt.

Zusätzlich zu diesen „externen" Daten entstehen jedoch auch „interne" Daten: Daten über das Vorgehen und Verhalten des Agenten selbst. Diese können einerseits auf Anweisungen des Besitzers zurückgehen (z. B. Log der Nachrichten des Agenten oder der besuchten Server) und sind dann jedenfalls ebenso geschützt, oder nicht. In letzterem Fall kann es sich nur um zufällig entstandene Daten handeln. Da aber über den Agenten und dessen Identität immer noch Rückschlüsse auf den Besitzer möglich sind, fallen auch diese Daten unter den Schutz. Je „weiter" die Daten jedoch vom Besitzer entfernt sind, desto geringer wird in der Regel das Geheimhaltungsbedürfnis sein. Fällt dieses vollständig weg, so besteht auch kein Schutz mehr. Da eine Entscheidung darüber jedoch für einen Agenten selbst praktisch unmöglich ist, sind bei Verarbeitung durch Agenten in der Praxis wohl alle Daten über andere Agenten als geschützt anzusehen.

3.3.6.5 Agenten anonymer Besitzer

Tritt ein Agent unter Verwendung eines Pseudonym-Zertifikates auf, d. h. ist der Besitzer darin nicht angegeben, so stellt sich die Frage, ob es sich noch um direkt oder nur mehr indirekt personenbezogene Daten handelt. Da über die Zertifizierungsstelle die Identität noch feststellbar ist, dies jedoch durch den konkreten Verarbeiter im Regelfall nicht möglich ist, liegen nur mehr indirekt personenbezogene Daten vor. Zwar kann jederzeit über die Eröffnung eines Gerichtsverfahrens die Identität festgestellt werden, doch da es sich hierbei um Rechtsmißbrauch handelt (es ist gar kein Verfahren beabsichtigt), ist dies kein rechtmäßiges Mittel und erzeugt daher keinen direkten Personenbezug.

Wird überhaupt kein Zertifikat verwendet und textuell ein Pseudonym angegeben, so handelt es sich dennoch nicht um anonyme Daten (welche ungeschützt wären). Zwar kann niemand mehr direkt oder alleine eine Beziehung zu einer Person herstellen, aber über die Nachverfol-

gung des Weges und Anfrage bei Internet-Providern läßt sich der Ursprungsrechner des Agenten feststellen (sofern Agentensysteme Logs führen, welche Agenten sich wann bei ihnen aufhielten und woher sie kamen). Es handelt sich also ebenfalls um indirekt personenbezogenen Daten. Wird diese Kette an Logs unterbrochen, so handelt es sich endlich um anonyme Daten.

Zu berücksichtigen sind hier zwei Sonderfälle:

1. Nicht oder nur indirekt personenbezogene Daten können nachträglich zu solchen werden, wenn der Agent seine Anonymität aufgibt und z. B. ein anderes Zertifikat präsentiert oder eine Lieferadresse angibt. Ab diesem Zeitpunkt sind auch die bereits vorher entstandenen Daten geschützt. Es besteht also eine Rückwirkung.
2. Auch bei im obigen Sinne vollständig anonymen Agenten gilt dies nicht für alle Fälle: Etwa ein Internet-Provider, welcher ein Agentensystem zur Verfügung stellt, kennt auch dann die Identität des Besitzers eines Agenten, wenn dieser vollständig anonym ist. Er kann selbst feststellen, über welche Verbindung (und damit von welchem Benutzerrechner aus) der Agent zu seinem Server gelangte. Solange diese Informationen gespeichert werden, handelt es sich insgesamt um geschützte Daten. Werden sie entfernt (z. B. Verbindungsaufbau ist beendet, daher keine Verwaltungsdaten mehr nötig, und diese werden gelöscht), so wird der Agent mit allen seinen enthaltenen Daten anonym.

Es sind daher beide Fälle möglich: Ein anonymer Agent kann sich identifizieren und damit bisher ungeschützte Daten plötzlich dem Datenschutzgesetz unterstellen, aber ein Agent kann auch plötzlich anonym werden, wenn keine Rückverfolgung mehr möglich ist. Hiermit fallen seine Daten aus dem Schutzbereich heraus. Besonders wegen des ersteren Falles sollten *alle* Daten als geschützt behandelt werden, um spätere Probleme bei plötzlich zu gewährendem Schutz zu vermeiden.

3.3.6.6 Anwendbares Recht bei mobilen Agenten

Der Umfang des Geltungsbereiches des österreichischen Datenschutzgesetzes beschränkt sich nicht auf das Staatsgebiet, sondern reicht ein wenig darüber hinaus, in manchen Fällen jedoch auch etwas weniger weit. Es wird für die Anwendung auf den Auftraggeber der Verarbeitung abgestellt: Ist dies ein Inländer und erfolgt die Verarbeitung im EU-Ausland, ohne daß dort eine Niederlassung besteht, so ist österreichisches Recht anzuwenden. Spiegelbildlich ist von österreichischen Gerichten ausländisches Recht anzuwenden, wenn ein Auftraggeber aus dem EU-Raum eine Verarbeitung in Österreich durchführt oder durchführen läßt, ohne hier eine

Niederlassung zu besitzen. Ist der Auftraggeber nicht aus der EU, so ist jedenfalls das Recht des Staates anzuwenden, in welchem die Verarbeitung stattfindet.

Da ein mobiler Agent keine Niederlassung begründet (hierfür sind feste Einrichtungen oder besondere Rechtsbeziehungen, z. B. Gesellschaftsgründung, als Minimum erforderlich), findet innerhalb der EU immer das „Heimatrecht" des Agenten Anwendung. Dies bedeutet, daß Agenten von österreichischen Auftraggebern auf ihrem gesamten Weg das österreichische Datenschutzgesetz einzuhalten haben (sofern keine Auslandsniederlassungen bestehen). Obwohl es auf einer EU-Richtlinie basiert, kann dies von Bedeutung sein. So schützt etwa die Richtlinie nur die Daten natürlicher Personen, während das österreichische DSG zusätzlich auch diejenigen von juristischen Personen unter Schutz stellt.

3.3.6.7 Automatisierte Einzelentscheidungen

In der Datenschutzrichtlinie (und dadurch auch im Datenschutzgesetz) findet sich noch eine Sonderbestimmung, die insbesondere für intelligente Agenten von besonderer Bedeutung sein kann. Danach darf niemand einer Entscheidung zum Zweck der Bewertung einzelner Aspekte seiner Person unterworfen werden, die ausschließlich aufgrund einer automationsunterstützten Datenverarbeitung getroffen wurde, wenn dies rechtliche Folgen für ihn nach sich zieht oder ihn erheblich beeinträchtigt. Beispiele hierfür sind Zuverlässigkeit oder Kreditwürdigkeit.

Da jedoch vielfach ein starkes Bedürfnis nach solchen Entscheidungen besteht (insbesondere automatische Bonitätsprüfungen im E-Commerce), wurden einige Ausnahmen geschaffen, welche das Verbot stark einschränken:

1. Ist die automatisierte Einzelentscheidung gesetzlich explizit vorgesehen, so ist sie erlaubt.
2. Ergeht die Entscheidung im Rahmen des Abschlusses oder der Erfüllung eines Vertrages *und* wurde dem Ersuchen des Betroffenen auf Abschluß oder Erfüllung des Vertrages entsprochen (Bsp.: automatische Überprüfung der Kreditwürdigkeit bei Vertragsabschluß; wird diese jedoch verneint und der Vertrag daher *nicht* abgeschlossen, so *muß* eine Nachkontrolle durch einen Menschen stattfinden!).
3. Kein Verbot besteht, wenn die berechtigten Interessen des Betroffenen durch geeignete Maßnahmen garantiert werden. Da als Beispiel die Möglichkeit angeführt wird, daß der Betroffene seinen Standpunkt geltend machen kann, erlaubt dieser Punkt sehr weite Einschränkungen (So kann u. U. mit einer Mitteilung des Verweigerungsgrundes das Auslangen gefunden werden).

Für Agenten bedeutet dies, daß gewisse Aufgaben nicht oder nur unter besonderen Vorkehrungen durchgeführt werden dürfen. Besonders im Bereich des E-Commerce wird etwa in vielen Fällen eine Bonitätsprüfung unabdingbar sein. Ist das Ergebnis positiv, so entstehen keine Probleme, da dem Ersuchen des Kunden auf Vertragsabschluß entsprochen wird. Wird jedoch abgelehnt, so müssen die Interessen des Betroffenen in geeigneter Weise gesichert werden, um in den Genuß der Ausnahme nach Punkt 3 zu gelangen. Ein Beispiel wäre etwa, dem Agenten den genauen Grund mitzuteilen und ein Verfahren einzurichten, in welchem der Besitzer des Agenten Reklamationen oder Richtigstellungen von Daten anbringen kann (dieses muß aber nicht durch Agenten erfolgen können). Es ist daher bei jeder Ablehnung eines Vertrages durch einen Agenten eine Möglichkeit zur Reklamation vorzusehen.

4. Das Agentensystem POND

In diesem Kapitel wird das implementierte Agentensystem beschrieben, wobei wiederum besonders auf die Punkte Sicherheit und Bezahlung von Leistungen eingegangen wird. Auch die Vorkehrungen, welche für die erläuterten rechtlichen Aspekte von Bedeutung sind, insbesondere für eine Beweissicherung, werden erläutert. Zuerst erfolgt noch eine genauere Abgrenzung des Projektes durch die Beschreibung der Zielsetzungen und der Einschränkungen. Abschließend werden die sonstigen Teile des Frameworks erläutert, wie die Mobilität von oder die Kommunikation zwischen Agenten. Es handelt sich hierbei nur um die Basisfunktionalität für Agenten; konkrete Implementierungen von Agenten und Systemen von Agenten werden in Kapitel 1 erläutert (zur Abgrenzung zwischen Agentensystemen und Systemen von Agenten siehe Abschnitt 2.1.4).

4.1. Vorbemerkungen

In diesem Abschnitt wird erläutert welche Ziele bei der Implementierung verfolgt wurden, sowie welche Einschränkungen sich daraus ergeben: Teilweise als Folge (etwa kein Naming-System, da nur Server-lokale Kommunikation möglich ist), teilweise als absichtliche Entscheidung (z. B. Message-Passing aber kein Methodenaufruf). Es handelt sich hierbei um grundlegende Designentscheidungen, welche sowohl die Struktur, als auch die Funktionen des Agentensystems determinieren.

4.1.1 Zielsetzung

Grundlage für den Entwurf des Agentensystems waren mehrere Aspekte, welche in einem Framework vereinigt werden sollten:

1. Mobilität: Es soll eine schwache Mobilität ermöglicht werden, indem Agenten von einem Benutzer (aber auch durch den Agenten selbst) auf einen anderen Rechner verlagert werden können. Dies sollte zumindest als Möglichkeit für Agenten offenstehen, auch wenn ein Agent dies nicht ausnutzen muß (und einer Verlagerung durch den Bediener, welcher sich vom Benutzer eventuell unterscheidet, auch widersprechen kann). Der Agent hat hierbei seinen gesamten Zustand und seinen Code (inklusive sonstiger Daten) mitzunehmen. Dies beinhaltet auch etwaige zusätzliche Daten (Bilder, Daten-Dateien, etc.). Zu diesem Zweck hat der Programmierer des Agenten alle benötigten Dateien in ein JAR-Archiv zu packen.

Andernfalls versucht der Computer dies automatisch selbst zu erledigen, hierbei kann jedoch auch etwas übersehen werden (in der Regel wird jedoch viel zu viel eingpackt). Dies bedeutet eine Unabhängigkeit des Agenten von anderen Rechnern, sowohl von vorhergehenden Hosts, als auch vom Heim-Rechner.

2. Message-Passing: Die Kommunikation zwischen einzelnen Agenten erfolgt über Nachrichten. Dies beruht hauptsächlich darauf, daß es in diesem Falle einfacher ist, die Trennung von Agenten im Hinblick auf Sicherheit zu gewährleisten, als wenn jeder Agent beliebige Methoden von anderen Agenten aufrufen kann. Ein weiterer Vorteil besteht darin, daß es sich um eine asynchrone Kommunikation handelt, d. h. nicht eine sofortige Antwort unter Blockierung des Weiterarbeitens erwartet wird. Daher sind auch Callback-Funktionen vermeidbar, welche in einem anderen Kontext (= Thread) aufgerufen würden, was weitere Sicherheitsprobleme bewirken könnte. Diese Vorgangsweise ermöglicht eine Entkoppelung der Agenten sodaß keine Synchronität erforderlich ist. Damit ist sichergestellt, daß ein eventuelles User-Interface ohne größere Probleme immer verfügbar bleibt und ein Agent nicht durch einen anderen blockiert werden kann (z. B. wenn eine Exception in einem anderen Agenten auftritt würde der Thread des aufrufenden Agenten beendet werden). Diese nur lockere Kopplung der Agenten ermöglicht auch eine leichtere Integration fremder Kommunikationsprotokolle, bzw. die Interaktion von Agenten verschiedener Hersteller: Es muß nur das Datenformat bekannt sein, nicht die genaue Schnittstelle der einzelnen Agenten (was etwa bei RMI [DCE-RPC], [DCE] erforderlich wäre).

3. Sicherheit: Ein weiterer wichtiger Gesichtspunkt war die Sicherheit der Agenten voreinander bzw. des Rechners vor Agenten. Ein Agent soll keinerlei Zugriff auf Methoden oder Felder eines anderen Agenten erreichen können, auf welche Art auch immer er es versuchen mag. Ebenso soll es dem Server möglich sein, einzelne Agenten getrennt auf verschiedene Sicherheitsstufen einzuschränken. Diese Stufen können dynamisch (während der Laufzeit des Agenten) verändert werden: Sowohl selbsttätig (z. B. auf verdächtige Aktionen hin werden Berechtigungen entzogen) als auch auf Anfrage des Agenten (z. B. Kauf von zusätzlichen Berechtigungen). Die Berechtigungen eines Agenten basieren sowohl auf dem Hersteller des Programmcodes (festgestellt durch Code-Signaturen), als auch auf der Identität des Benutzers, welcher den Agenten verwendet. Das System kann relativ einfach verändert werden, etwa durch Subklassen. Doch auch eine dynamische Neukonfiguration ist vom Prinzip her denkbar.

4. Vertrauen: Gegenseitiges Vertrauen der Agenten ist ein wichtiges Element, da nicht gegen alle Probleme vorbeugende Maßnahmen möglich sind. Es wird daher ein System implementiert, wonach Agenten Aussagen über andere Agenten machen können. Diese werden dazu verwendet, Informationen über die Vertrauenswürdigkeit und Fähigkeiten von anderen Agenten zu sammeln. Durch einen Vergleich mit eigenen Erfahrungen wird wiederum ein Rückschluß auf den aussagenden Agenten getroffen. Dieses System ist eine Folge der offenen Konzeption: Jeder Agent soll grundsätzlich teilnehmen können und es wird keine geschlossene Gruppe angenommen. Dies hat zur Folge, daß völlig neue Agenten von unbekannten Besitzern auftauchen können. Auch über diese sollten alsbald Informationen zur Verfügung stehen (nachdem diese einige Transaktionen durchgeführt haben), um den existierenden Agenten die Auswahl aus den potentiellen Kommunikations- und Transaktionspartnern zu erleichtern.

5. „Heavy-weight“ Agenten: Bei jenen Agenten, für die das System entworfen wurde, handelt es sich um größere Agenten für die selbständige Durchführung komplexer Aufgaben. Systeme aus einer Vielzahl sehr kleiner Agenten werden nicht besonders unterstützt: Sie sind zwar möglich, doch ist die Performanz dann aufgrund der Kommunikation über Nachrichten eher gering. Der Grund für diese Wahl war, daß ein Agent alleine bereits für den Benutzer sinnvolle Ergebnisse bringen soll. Dies vor allem deshalb, weil dann die Konfiguration einfacher ist, als wenn erst eine Vielzahl von Agenten zu erstellen und anschließend zu integrieren/konfigurieren ist (Herstellung der Kommunikationsbeziehungen, etc.). Auch sind größere Agenten besser geeignet, intelligentes Verhalten zu zeigen, da komplexere Entscheidungssysteme möglich sind.

6. Lokalitätsprinzip: Agenten sollen ihre Aufgabe wenn möglich lokal (= auf dem Rechner, auf dem sie ausgeführt werden; im Folgenden immer in diesem Sinne verwendet) erledigen und sich nur selten auf einen anderen Rechner verlagern. Der Vorgang der Verlagerung kann daher auch eine längere Zeit in Anspruch nehmen, insbesondere für die Verschlüsselung (und damit Sicherung) während der Übertragung. Analog dazu wurden besondere Hilfsmittel entworfen, um die Anbindung an das Internet zu erleichtern: Ausfüllen von Web-Formularen durch Agenten, Unterstützung für das Parsen von Webseiten und ein detailliertes Sicherheitssystem für die Server welche ein Agent kontaktieren darf.

7. Java: Die Implementierung soll in JAVA erfolgen, sodaß eine portable Plattform entsteht, die gleichartig auf verschiedensten Systemen eingesetzt werden kann. Dies hat auch zur

Folge, daß möglichst keine Spezial-Bibliotheken Verwendung finden, sondern mit Standard-Funktionen das Auslangen gefunden werden soll. Ansonsten ergeben sich unter Umständen Probleme bei der Portabilität, da ein Agent sich dann zwar auf einen anderen Rechner begeben kann und dort ausgeführt wird, aber keine eigentliche Tätigkeit entfalten kann, da die notwendigen Bibliotheken nicht vorhanden sind. Die Alternative, Bibliotheken selbst mitzuführen ist zwar möglich, hat aber einige Nachteile. So unterliegt eine mitgeführte Bibliothek den selben Einschränkungen wie der Agent selbst (ansonsten eine Sicherheitslücke), während eine installierte Bibliothek beliebig eingestuft werden kann und meistens alle Berechtigungen erhält. Weiters handelt es sich bei Bibliotheken meist um größere Dateien, welche die Übertragungszeit von Agenten vergrößern und lokalen Speicherplatz belegen.

8. Praktische Beispiele: Ein besonderes Ziel beim Entwurf des Systems war auch, daß praktische Anwendungen Teil des Ergebnisses sind. Es sollte kein rein akademisches Forschungsprojekt erfolgen, sondern auch eine praktische Verwertung der Ergebnisse möglich sein. Ziel war insbesondere die Unterstützung von Kommunikation (Filterung / Behandlung von E-Mails oder anderen Nachrichten), die Benachrichtigung des Benutzers von besonderen Ereignissen (Reminder, SMS versenden), verschiedene Utilities (Website-Downloader; auch als Hilfs-Agenten für Agenten mit anderen Aufgaben) sowie die Erstellung eines Web-Portals, welches auch für Agenten zugänglich ist. Letzteres weist einen stärker explorativen Charakter auf, da dort insbesondere auch die Integration von verschiedenen Zugangsweisen sowie die Verwendung von XML angestrebt wurden.

4.1.2 Einschränkungen

Einige Einschränkungen mußten beim Entwurf und später bei der Implementierung des Systems gemacht werden. Einerseits um den Umfang des Projekts in einem realistischen Rahmen zu halten, andererseits weil sie aufgrund der Ziele nicht unbedingt notwendig sind. Diese sind im Einzelnen:

1. Keine graphische Konfiguration: Die Konfiguration mehrerer Agenten für eine Zusammenarbeit erfolgt nicht graphisch sondern über die Programmierung oder Parameter, welche den einzelnen Agenten übergeben werden. Eine solche graphische Konfiguration hätte den Vorteil, daß der Benutzer leichter den Zusammenhang zwischen Agenten und die Vollständigkeit seines Systems überblicken kann. Problematisch ist hierbei jedoch die meist notwendige automatische Plazierung der Agenten (erschwert durch deren unterschiedliche

Größe) und daß ein eigenes Protokoll für die Herstellung der Beziehungen erforderlich wäre. Auch könnte hierdurch eine Sicherheitslücke entstehen, da ein Agent zuerst die Autorisierung des Herstellers der Verbindung überprüfen müßte, was nicht ohne weiteres möglich ist.

2. Nur lokale Kommunikation: Kommunikation zwischen Agenten ist nur auf dem selben Rechner (=lokal) möglich. Eine Nachricht kann daher *nur* von *einem* lokalen Agenten an einen anderen *lokalen* Agenten gesendet werden. Dies gilt ebenso für Broadcasts, welche nur lokal verteilt werden (keine Weiterleitung an bestimmte Rechner oder gar das gesamte Netzwerk). In letzterem Fall ist dies eine notwendige Konsequenz der Offenheit des Frameworks, da ein Broadcast begrenzt werden muß. Sonst müßte er unter Umständen über die ganze Welt verteilt werden, was sowohl Bandbreitenprobleme als auch sonstige Schwierigkeiten bedeuten würde (Implementation: An welche Rechner ist er weiterzuschicken, sodaß jeder Rechner den Broadcast nur einmal erhält; Logik: Ist ein Broadcast wirklich für alle Agenten in allen Agentensystemen interessant?). Dies entspricht auch dem Lokalitätsprinzip, wonach jeder Agent seine Aufgabe möglichst lokal erfüllen soll (nur lose Koppelung der Agentensysteme). Einzelne Nachrichten auch gezielt an entfernte Agenten zu versenden wäre zwar technisch ohne großen Aufwand möglich und bringt keine logischen Widersprüche, doch entspricht dies nicht dem Lokalitätsprinzip. Dem steht jedoch auch ein praktisches Hindernis entgegen: Da kein zentraler Verzeichnis-Dienst existiert (siehe nächster Punkt), ist es für einen Agenten unmöglich, ihm bis dahin unbekannte Agenten auf anderen Rechnern zu finden. Er könnte also nur mit ihm persönlich bekannten Agenten eine Kommunikation aufnehmen und müßte auch wissen, auf welchem Rechner sie sich gerade befinden. Ansonsten wäre eine Weiterleitung von Nachrichten (wie etwa beim System Aglets [Oshima et al. 1998]) erforderlich, was jedoch nicht in allen Fällen funktionieren kann (Problem etwa beim Terminieren eines vorhergehenden Systems). Eine Modellierung dieser Funktionalität ist jedoch über Boten möglich: Ein Agent erzeugt einen Sub-Agenten welcher sich auf den anderen Rechner begibt und dort die Nachricht lokal absendet.

3. Kein Naming-Service: Aufgrund der dezentralen Organisation (jeder Server ist eigenständig) existiert kein zentrales Verzeichnis; weder der Server noch der darauf befindlichen Agenten. Auch ist kein lokales Verzeichnis vorgesehen. Dieses existiert zwar intern im Agentensystem doch haben Agenten darauf keinen Zugriff: Ein Agent kann daher nicht feststellen, welche anderen Agenten sich auf dem gleichen Server befinden (auch daraus

ließen sich Informationen gewinnen, welche eventuell privat bleiben sollen). Es gibt nur zwei Möglichkeiten, Agenten zu kontaktieren: eine Nachricht an einen bestimmten Agenten zu senden (in diesem Fall muß die Identität genau bekannt sein) und zu hoffen, daß dieser Agent vorhanden ist, sowie Broadcasts. Existiert der Ziel-Agent nicht, so wird eine Fehlermeldung zurückgeliefert. Ansonsten bleibt es dem Empfänger überlassen ob er antworten möchte (z. B. mit einer zum vorigen Fall identischen Fehlermeldung) oder nicht. Die zweite Alternative ist, einen Broadcast auszusenden, weshalb die Identität nicht bekannt sein muß. Dieser erreicht alle lokalen Agenten, denen es jedoch wieder freisteht, ob sie darauf antworten möchten (bzw. können). Ein lokales oder globales Verzeichnis könnte über spezialisierte Agenten nachgebildet werden. Dies setzt jedoch voraus, daß die Agenten sich dort freiwillig anmelden (bzw. bei Verlagerungen ummelden).

4. Keine vollständige Public-Key-Infrastruktur: Das Agentensystem verwendet zwar asymmetrische Kryptographie, stellt jedoch selbst keine vollständige PKI zur Verfügung. Die Identität von Agenten wird über Zertifikate modelliert, doch fehlen etwa Widerrufslisten. Diese Einschränkung erfolgt hauptsächlich zur Verringerung des Aufwands: Es handelt sich hier um keinen essentiellen Teil der unbedingt benötigt wird, sondern um zusätzliche Elemente, welche mit einem sehr hohen Implementierungsaufwand verbunden wären. Auch wurden die Verschlüsselungs- bzw. Signatur-Algorithmen und die zusätzlichen Klassen / Interfaces / Implementierungen für die verwendete Kryptographie nicht selbst erstellt. Es wird die (teilweise: z. B. für Forschungszwecke) öffentlich erhältliche Bibliothek IAIK ([IAIK]) verwendet.

5. „Intelligenz“ von Agenten ist nicht im Framework enthalten: Für die Intelligenz von Agenten ist keine besondere Unterstützung im Framework vorgesehen. Dies ermöglicht es Entwicklern auch einfache und kleine Agenten zu erstellen sowie für komplexere Aufgaben jeweils das am Besten geeignete System zu verwenden. In den Beispiels-Agenten ist relativ wenig Intelligenz zu finden, in den komplexeren implementierten Systemen werden teilweise regelbasierte Systeme verwendet. Diese Einschränkung wurde vorgenommen, um die Agenten universell anwendbar zu machen und den Umfang des Projektes nicht zu groß werden zu lassen.

4.2. Das Sicherheitssystem

Das implementierte Sicherheitssystem (für eine Beschreibung siehe [Sonntag/Hörmanseder 2000]) beruht auf dem Gedanken, daß Agenten „wertvolle“ Information besitzen, wie etwa E-Cash oder Kreditkartendaten. Sie müssen daher voreinander wie auch vor dem Host geschützt werden. Auch während der Übertragung auf einen anderen Rechner muß eine Verschlüsselung erfolgen, ebenso wie eine Sicherung gegen Verlust. Dieser Besitz von geldwerten Daten erlaubt es Agenten für Ressourcen, Dienstleistungen oder Informationen auf Servern zu bezahlen, welche ansonsten für sie nicht zugänglich wären.

In vielen Fällen wird dies jedoch nicht sofort erfolgen, sondern ein Agent wird zuerst anonym (und mit geringem Einsatz oder ohne Bezahlung) eine Suche durchführen. Erst nachdem ein gewünschter Anbieter gefunden ist, wird er seine Identität preisgeben und die Leistung tatsächlich in Anspruch nehmen, sowie dafür bezahlen. Dies erfordert, daß das Sicherheitssystem auch dynamische Veränderungen erlaubt, also ein Agent nach Übergabe von wertvollen Informationen zusätzliche Berechtigungen erhalten kann, ohne daß zwischenzeitlich ein Transfer auf einen anderen Rechner oder sonstige aufwendige Aktionen erforderlich sind.

Das implementierte Sicherheitssystem basiert auf dem Ansatz, daß jede Berechtigung ihren Preis hat: Jede Ressource kann bewertet werden (unabhängig von ihrer derzeitigen Verfügbarkeit) und für jedes Risiko kann ein entsprechender Preis festgesetzt werden. Manche sind sehr billig (z. B. gratis für lokal erzeugte Agenten), andere wiederum teuer (etwa für bisher unbekannte Agenten), oder überhaupt nicht erhältlich (z. B. die Berechtigung den Rechner abzuschalten). Der Preis bzw. die Erhältlichkeit von Berechtigungen wird aufgrund des Herstellers des Programmcodes und des Besitzers (=Verwenders) des Agenten festgesetzt. Sie enthält also zwei Dimensionen. Die Einführung von weiteren unabhängigen Dimensionen wäre ohne Probleme möglich (z. B. Uhrzeit oder derzeitige Auslastung führt zu Systemen ähnlich elektronischen Märkten wie etwa [Bredin et al. 1998] oder [Yemini et al. 1998]). Ein wichtiges Beispiel hierfür ist noch, die Zahlungsweise ebenfalls als Dimension zu integrieren: Wird mit E-Cash bezahlt, so kann etwa der Preis für Berechtigungen niedriger angesetzt werden, als bei Kreditkarten, wo eine Gebühr an die kartenausgebende Stelle abzuführen ist. Die Identität von beiden Personen wird durch Zertifikate überprüft. Kann (oder will) ein Agent ein Zertifikat nicht preisgeben, so wird er in der Regel nur sehr geringe oder auch gar keine Berechtigungen erhalten.

Berechtigungen, welche nicht strikt durchgesetzt werden können oder sollen, sind besonders für eine dynamische Zuweisung geeignet: So etwa die Menge an Harddisk-Speicherplatz die einem Agenten zur Verfügung steht. Auch der Agent selbst kann unter Umständen nicht im Vorhinein wissen, wieviel er schließlich tatsächlich benötigen wird. In diesem Fall wird er z. B. ein fixes Minimum erwerben und erhält eine optionale Quote zur Überschreitung, die von dem ihm zugebilligten Vertrauen abhängt. Ab diesem Limit muß der Agent zusätzlichen Platz erwerben, bzw. zuerst für den bereits belegten bezahlen. In der Zwischenzeit jedoch vertraut das Agentensystem (und damit dessen Betreiber), daß der Agent nach Abschluß seiner Tätigkeit für den tatsächlich verwendeten Platz bezahlen wird. Tut er dies nicht, so kann über das Zertifikat des Besitzers dessen Identität festgestellt und eine direkte Rechnung ausgestellt werden. Für den hierfür notwendigen Aufwand und das Risiko von dessen Realisierung muß der Agent einen Aufpreis bezahlen: Der Preis für den optionalen Speicherplatz ist höher, als wenn dieser direkt und fix gekauft wird. Im Gegenzug ist es für den Agenten natürlich günstiger, wenn er den optionalen Platz nicht vollständig ausnützt.

4.2.1 Vorteile

Gegenüber einem binären Modell für die Zuteilung oder Verweigerung von Berechtigungen welches auf festen Klassen beruht (eigene Agenten, vertrauenswürdigen Agenten, fremde Agenten, …), besitzt dieses Modell einige Vorteile:

? Granulare Sicherheit: Jede Berechtigung wird entsprechend dem Risiko ihres Gebrauchs bzw. Mißbrauchs durch bestimmten Code oder bestimmte Personen bewertet. Als Beispiel hierfür kann dienen, daß viele Provider auch unbekannten Agenten Berechtigungen zugestehen werden, falls diese hierfür mehr bezahlen, als die Kosten für zusätzlichen Speicherplatz betragen oder ihm durch die Ablehnung anderer Agenten entgehen, wenn die Ressourcen kurzzeitig erschöpft sind. Hierdurch sind auch geringfügige Unterschiede zwischen Gruppen von Agenten möglich (z. B. etwas höherer Preis für Agenten bei denen statistisch gesehen öfter Probleme auftraten), was bei bloßer Unterscheidung zwischen Erlaubnis oder Ablehnung nicht möglich ist.
? Einfach zu verstehen: Die Zuteilung von Berechtigungen ist leicht zu verstehen, da die Kosten für eine Berechtigung den tatsächlichen Kosten und dem geschätzten Risiko entsprechen. Dies ist auch z. B. für Manager leichter verständlich, welche sonst nicht aktiv im Aufbau einer Sicherheits-Policy involviert sind, aber oft Bewertungen von Produkten/Leistungen und Risken vornehmen.

? Integrierte Abrechnung: Dieses Modell ist gut für die Abrechnung von Leistungen geeignet. Berechtigungen können nicht nur daraus bestehen, Zugriff auf bestimmte Elemente zu erhalten (Verwendung von Spezial-Hardware oder bestimmten Interfaces), sondern sie können auch bestimmte Ausmaße (z. B. Größe von temporären Dateien) und Qualitäten (höhere Priorität der Threads einzelner Agenten) umfassen, ohne daß hierzu ein gesondertes System erforderlich ist. Auf diese Weise können Server ihre Ressourcen bestmöglich ausnutzen, da diese nicht für Agenten reserviert werden müssen, welche sie eventuell nie verwenden werden (u. U. ist sogar eine Versteigerung möglich). Währenddessen bezahlen Agenten nur für diejenigen Elemente, welche sie auch tatsächlich benötigen und verwenden. Eine Erweiterung dieses Systems kann noch dahin erfolgen, daß Agenten auch das Ausmaß der Nutzung in Rechnung gestellt wird (etwa die Dauer oder das Ausmaß der Verwendung; zu unterscheiden von im Vorhinein garantierten Werten).

4.2.2 Nachteile

Trotz dieser Vorteile existieren auch Nachteile bei einer Vergabe von Berechtigungen in dieser Weise:

? Ungeeignet für die Klassifizierung nach der Kreditwürdigkeit: Agenten können mit diesem System nicht nach ihrer Kreditwürdigkeit klassifiziert werden. Es ist sinnlos von einem Agenten einen höheren Betrag zu fordern, wenn es wahrscheinlicher ist, daß er *überhaupt nicht* bezahlen wird. Zahlungsmethoden müssen daher entweder unmittelbar erfüllenden Charakter haben (z. B. E-Cash, Abbuchung von einem Kundenkonto), oder Agenten müssen ein bestimmtes Mindestvertrauen genießen, bevor sie überhaupt zusätzliche Berechtigungen gegen kreditgewährende Zahlungen erwerben können.

? Große Anzahl von Entscheidungen bei der Konfiguration: Da zwei Punkte den Preis einer Berechtigung definieren (Code-Hersteller und Verwender) und viele verschiedene Berechtigungen (bzw. zusätzlich in verschiedenen Ausprägungen) existieren, ist die Konfiguration des Sicherheitssystems nicht trivial. Werden etwa auf jeder Achse nur vier Zustände unterschieden (wie implementiert), so sind pro Berechtigung 16 Werte zu bestimmen für die jeweils eine Beurteilung stattzufinden hat. Aufgrund dieser hohen Anzahl ist eine sinnvolle Vorkonfiguration und eine Vereinfachung der tatsächlichen Durchführung bzw. Anpassung notwendig (implementiert wurde z.B. eine Gruppierung von Berechtigungen).

? Keine Delegation: Ein Agent kann keine seiner eigenen Berechtigungen an einen anderen Agenten weitergeben: Jeder Agent wird getrennt für sich alleine beurteilt. Dies ist dem Modell inhärent, auch aufgrund der strengen Trennung zwischen einzelnen Agenten, selbst wenn sie dem selben Besitzer gehören. Die einzige Möglichkeit ist, daß ein Agent selbst einen Sub-Agenten erzeugt, welcher die gleichen Basis-Berechtigungen erhält (sofern er aus demselben Code-Paket stammt und für den gleichen Besitzer aktiv wird). Zusätzlich erworbene Berechtigungen gehen jedoch auch hier nicht auf den Sub-Agenten über. Aufgaben sollten daher so auf Agenten aufgeteilt werden, daß ein Agent eine besondere Aufgabe (welche eventuell eine Zusatzberechtigung benötigt) immer zur Gänze selbst erfüllt und lediglich unabhängige Aufgaben an andere Agenten zur Bearbeitung übergibt. Wird Delegation vorgesehen, so ergeben sich auch Schwierigkeiten mit einer eventuellen Abrechnung nach der Benutzungsintensität: Wie sollten die Kosten aufgeteilt werden? Dieses Problem ist jedoch nicht so hinderlich, da es jedem Agenten freisteht, selbst zusätzliche Threads und beliebige Objekte zu erzeugen um besondere Aufgaben durchzuführen. Es kann sich hierbei sogar um Agenten handeln, sofern diese als Objekte behandelt werden (und daher keine unabhängig Verlagerung durchführen und nicht selbst Nachrichten empfangen können).

4.2.3 Basis-Zuteilung von Berechtigungen

Um sowohl statische als auch dynamische Aspekte zu erfassen, basiert die Zuordnung von Berechtigungen auf zwei Elementen: Der Herkunft des Programmcodes (statisch; signierter Code) und dem Besitzer des Agenten (dynamisch; Identität des Agenten). Dies ist deshalb notwendig, da bei größerem Einsatz von Agenten davon auszugehen ist, daß ein Großteil der Benutzer fremd-hergestellte Agenten verwenden werden.

Der Programmcode einer bekannten Firma wird mehr Berechtigungen erhalten können, als ein solcher von unbekannten Unternehmen oder Privat-Personen da er aufgrund der größeren Verbreitung vermutlich von höherer Qualität ist (weniger Fehler, weniger Möglichkeiten zum Mißbrauch). Auch ist die Wahrscheinlichkeit, daß ein solcher Code von einem Virus infiziert ist geringer, da professionelle Firmen extensive Sicherheitsmaßnahmen gegen derartige Probleme unternehmen (eine spätere Infizierung ist aufgrund der Signierung des Programmcodes leicht zu entdecken).

Da ein Agent ein Ziel besitzen muß, ist es notwendig, daß vom Benutzer ein solches vorgegeben wird. Diese Parametrisierung ist jedoch wiederum eine Fehlerquelle: Viele vollkommen harmlosen Programme können durch eine entsprechende Parametrisierung dazu gebracht wer-

den, schädliche Auswirkungen zu zeigen. Ein Beispiel wäre etwa einen Shopping-Informationsagenten loszuschicken, welcher sich nach bestimmten Artikeln und deren Preis erkundigt. Diese an sich harmlose Tätigkeit kann bei Verwendung einer falschen Identität und Angabe illegaler Produkte zu großen Problemen führen!

Aus diesem Grund sind beide Aspekte bei der Vergabe von Berechtigungen zu berücksichtigen. Problematisch für die Implementierung hierbei ist, daß nicht einfach eine Schnittmenge der Berechtigungen des Codes und der Berechtigungen des Besitzers als Ergebnis gewählt werden kann: Bei niedrigen Sicherheitsstufen kann die sinnvolle Menge an Berechtigungen geringer sein als die Schnittmenge, während sie bei höheren Stufen auch unter Umständen etwas größer sein kann. So hat eventuell ein Agent eines besonders gut bekannten und zuverlässigen Besitzers auch dann einige Rechte, wenn es sich ausnahmsweise um einen unsignierten Programmcode handelt, welcher sonst überhaupt nicht zugelassen werden würde.

Um nicht für jede Person Rechte zuweisen zu müssen werden sowohl Code-Signierer als auch Besitzer von Agenten jeweils in Gruppen eingeteilt:

? Produzent des Programmcodes: Die Gruppierung erfolgt nach dem Zertifikat dessen zugehöriger privater Schlüssel zur Signierung des Programmpaketes verwendet wurde. Manche Server mögen auch unsignierten Code akzeptieren (z. B. vom lokalen Rechner), obwohl hierfür die Berechtigungen meist eher restriktiv ausfallen werden. Die Gruppierung kann sehr fein unterteilt sein, etwa nach der ausstellenden Zertifizierungsstelle (welche Garantie jeweils abgegeben wird). Auch eine individuelle Zuteilung von Berechtigungen zu bestimmten Paketen ist möglich.
? Besitzer des Agenten: Die Gruppen unterscheiden sich dadurch, ob der Agent ein Zertifikat seines Besitzers preisgeben kann oder will, worauf wieder eine ähnliche Einteilung wie bei den Code-Zertifikaten erfolgt. Auch hier können wiederum einzelnen Besitzern gesondert Rechte zugewiesen werden. Ebenso kann die Einteilung wieder nach den Garantien der Zertifizierungsstellen erfolgen, aber auch etwa nach der Unternehmenszugehörigkeit (auch diese Information kann in einem Zertifikat enthalten sein; direkt im Namen, aber auch als Attribut oder getrennt davon Attributzertifikate).

Zu jedem Zeitpunkt gehört ein Agent zu jeweils einer Code- und einer Besitzer-Gruppe. Beide können sich mit der Zeit verändern (z. B. wenn der Agent zusätzliche Beweise für seine Identität preisgibt (etwa das Zertifikat des Besitzers), wenn die Überprüfung der Code-Signatur erst später erfolgt, oder die Überprüfung des Zertifikat-Widerrufs erfolgreich verlaufen ist,

weshalb eine dynamische Zuordnung erforderlich ist. Diese dynamische Veränderung ist der stärkste Kontrast des neuen Sicherheitssystems gegenüber dem vollkommen statischen System, welches standardmäßig von Java (siehe [Gong]) verwendet wird.

Berechtigungen innerhalb des Schnittpunkts eines Code-Herstellers und eines Besitzers werden noch einmal unterteilt:

? Basis-Berechtigungen: Ein Agent der Gruppe besitzt *immer* die hier enthaltenen Berechtigungen, welche ein Mindestmaß darstellen. Sie müssen für die normale Arbeit des Agenten ausreichen. Hierfür wird ein negativer Preis vorgesehen, um zu kennzeichnen, daß diese Berechtigungen beim Start eines Agenten automatisch zugewiesen werden.

? Optionale Berechtigungen: Diese Berechtigungen kann ein Agent zusätzlich erwerben, typischerweise gegen Bezahlung oder temporär für eine kurze Zeit. Diese können für manche Agenten gratis, für andere billig oder teuer, und für wiederum andere überhaupt nicht erhältlich sein.

Auch bei ohne Kosten erhältlichen Berechtigungen (Preis ist 0) hat es Sinn diese in die Optional-Guppe einzuordnen, da hierdurch ein Agent angehalten wird diese nur dann zu reservieren, wenn er sie tatsächlich benötigt. Dies erlaubt es etwa einem Rechner seine Ressourcen besser auszulasten. Welche Berechtigungen nun im Einzelnen zugeteilt werden ergibt sich aus einer Matrix (siehe Abbildung 46), welche entsprechend dem Vertrauen in Code-Hersteller und Besitzer aufgebaut ist und zwischen Basis- und optionalen Berechtigungen differenziert.

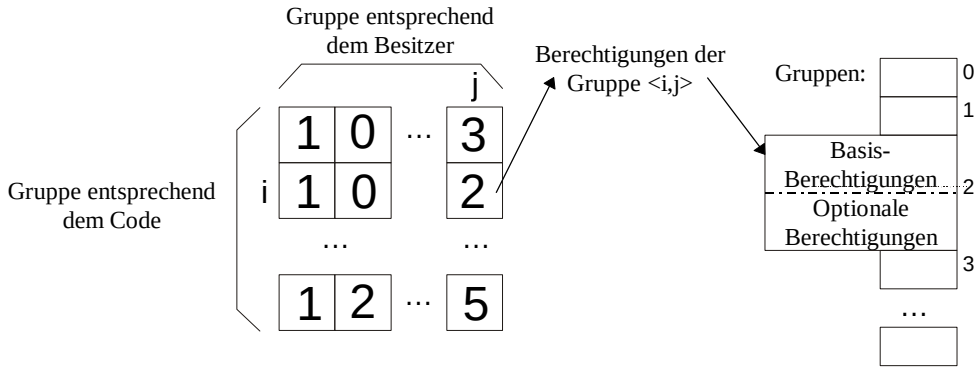

Abbildung 46: Bildung von Gruppen für Berechtigungen

4.2.4 Erweiterte Zuteilung von Berechtigungen

Um mehr Flexibilität zu ermöglichen kann ein Agent zusätzliche Informationen bereitstellen, um in eine „bessere“ Berechtigungsklasse eingeordnet zu werden. Dies kann einerseits als eine

zusätzliche Dimension für die Berechtigungsmatrix modelliert werden, andererseits aber auch anderweitig erfolgen, wie etwa als generelle Preissenkung oder indem überhaupt unabhängig vom Sicherheitsmodell einzelne Berechtigungen zugeordnet werden. Auf diese Weise sind auch „negative Berechtigungen“ möglich: Verhält sich ein Agent verdächtig, so können ihm einzelne Berechtigungen entzogen werden. Auf diese Weise könnte auch eine Vergangenheits-basierte Zugriffskontrolle, wie von [Acharya/Edjlali 1997] beschrieben, implementiert werden. Die hierfür notwendige eindeutige Identifizierung von Agenten kann einerseits über deren Identität erfolgen, andererseits aber auch über Code-Prüfsummen, da in Java kein selbst-modifizierender Code erlaubt ist. Durch die explizite Modellierung als „Zusatzinformation“ wird dies auch für den Agenten transparent und er kann Gegenmaßnahmen unternehmen, wie etwa eine Verhaltensänderung oder die Freigabe zusätzlicher Informationen.

Beispiele für zusätzliche Klassifikationen und folgende Konsequenzen sind beispielsweise:

- Stammkunden: Agenten oder auch Besitzer, welche schon öfters Berechtigungen erfolgreich gekauft haben, können mehr oder andere Berechtigungen erhalten oder auch einen Preisabschlag.
- Mitgliedskarte: Agentensysteme können eine Art „Mitgliedskarte“ in Form eines Zertifikates ausstellen, dessen Präsentation besondere Vergünstigungen mit sich bringt.
- Rückvergütung: Wird einem Agenten Geld zurückerstattet (oder sonst ein Bonus gewährt, z. B. Mengenrabatt), so kann dies auch in Form eines Gutscheins erfolgen, welcher später gegen Berechtigungen eingetauscht werden kann.

Eine Erweiterung des Sicherheitssystems ist noch in die andere Richtung möglich, daß nicht alle Prüfungen unbedingt sofort durchgeführt sondern manchmal auf einen späteren Zeitpunkt verschoben werden. Ein Agent kann daher sofort mit der Arbeit beginnen (wenn auch mit sehr geringen Berechtigungen) und erhält erst zu einem späteren Zeitpunkt diese, nachdem die Prüfung durchgeführt wurde (eine Basis-Prüfung sollte jedoch auf jeden Fall vorab erfolgen). Die Klassifikation kann dann in die folgenden Zustände eingeteilt werden:

- Überprüfung angefordert: Dies ist insbesondere für zusätzliche Informationen, die ein Agent zur Verfügung stellt, relevant. Der Agent fordert das Agentensystem auf diese zu überprüfen und ihm weitere Berechtigungen zuzuteilen.
- Unbekannte Klassifikation: Der Agent stellte zwar zusätzliche Informationen zur Verfügung, aber dem Agentensystem sind diese unbekannt. Deshalb kann auch keine Überprüfung erfolgen. Da der Programmcode vom Agenten mitgebracht wird, kann das Agentensy-

stem zwar Methoden aufrufen etc. (Syntax ist bekannt), doch da die Semantik nicht festgelegt ist, kann eine Beurteilung des Werts der Informationen nicht erfolgen.

? Überprüfung derzeit nicht möglich: Nicht immer ist das Agentensystem in der Lage, alle Zusatzinformationen vollständig zu prüfen. So kann etwa die Überprüfung des Widerrufs eines Zertifikats längere Zeit erfordern. Oder es ist zur Zeit keine Internet-Verbindung möglich. Hierbei handelt es sich jedoch um ein temporäres Problem, sodaß später automatisch (oder auf neues Ansuchen hin) eine weitere Überprüfung stattfindet.

? Überprüfung undurchführbar: Die Überprüfung kann dauerhaft nicht durchgeführt werden. Ein Beispiel hierfür wäre, daß ein Zertifikat zwar vollständig überprüft werden kann, aber von einer unbekannten Zertifizierungsstelle stammt. Eine nochmalige Überprüfung wird dasselbe Ergebnis liefern.

? Abgelehnt: Die Überprüfung konnte durchgeführt werden, schlug jedoch fehl. Im Gegensatz zur undurchführbaren Überprüfung waren hier alle Prüfvorgänge möglich, doch war das Ergebnis nicht zufriedenstellend (z. B. mathematischer Fehler bei der Signatur-Prüfung, Gültigkeitszeitraum abgelaufen, …).

? Akzeptiert: Die Zusatzinformation wurde akzeptiert, ohne daß eine Aussage über die Prüfung getroffen wurde (welche auch erst später erfolgen kann). Aus diesem Zustand ergibt sich nicht unbedingt auch, daß zusätzliche Berechtigungen zugeteilt werden (z. B. falls der Agent bereits mehr besitzt, als hierdurch erworben werden können).

4.2.5 Exkurs: Das Sicherheitssytem von JDK 1.2 im Überblick

Das Sicherheitssystem des JDK ab Version 1.2 [Gong] beruht auf sogenannten ProtectionDomains. Darunter versteht man die Menge aller Objekte welche direkt für einen Besitzer (hier eine Einheit mit bestimmten Berechtigungen, z. B. ein Programm oder ein Applet) zugänglich sind. Hierbei ist wiederum in System-Domains (inkludiert in der Implementierung das Agentensystem) und Anwendungs-Domains (jeweils eine pro Agent; sonst einheitlich pro JVM) zu unterscheiden. Alle geschützten externen Ressourcen wie das Dateisystem, Netzwerksinterfaces oder Bildschirm/Tastatur dürfen nur durch den System-Domain erreichbar sein. Welche Berechtigungen ein Protection-Domain erhält, wird in der Policy festgelegt. Jeder Klasse/Instanz wird genau ein Domain und jedem Domain eine Menge an Berechtigungen zugewiesen. Ein Domain besteht normal aus einem CodeSource Objekt welches den URL zum Ursprung des Codes und die zugehörigen Zertifikate enthält (im Agentensystem ersetzt um auch andere Aspekte berücksichtigen zu können). Die Zuordnung von Code zu Domains erfolgt durch den Classloader, wobei alle Klassen, welche von einem Classloader geladen werden,

potentiell miteinander interagieren können (was mit Klassen aus anderen Classloadern nicht möglich ist).

Classloader sind dafür zuständig den Bytecode von Klassen zu finden, in den Speicher zu laden, und ihm Berechtigungen zuzuweisen. Es können beliebig viele Classloader existieren welche in einer Baumstruktur angeordnet sind. Die Wurzel ist der Primordial-Classloader welcher nicht in Java geschrieben ist (sondern meist C) und die notwendigen Daten auf eine systemabhängige Art aus dem Dateisystem lädt. Alle Systemklassen (aus dem package java.*) müssen, andere können von diesem Classloader geladen werden. Diese Klassen haben keinen zugehörigen Classloader, sondern eine *null*-Referenz, welche sie als vom Primordial-Classloader geladen, kennzeichnet (historische Gründe; auf den Primordial-Classloader ist Zugriff nur über native-Code-Aufrufe möglich).

Soll eine Klasse geladen werden, so wird zunächst in einem Cache überprüft, ob diese Klasse bereits von diesem Classloader geladen wurde (keine Klasse darf von einem Classloader zwei Mal geladen werden; bei verschiedenen Classloadern ist dies ohne weiteres möglich, selbst wenn diese aus unterschiedlichem Bytecode bestehen). Besitzt der Classloader einen Parent-Classloader, so wird dieser rekursiv aufgerufen, andernfalls der Primordial-Classloader bemüht. Führt auch dies nicht zum Erfolg sucht der Classloader die Klasse selbst. Schlägt auch dies fehl, wird eine Exception ausgeworfen und die Klasse kann nicht verwendet werden. Dieser Mechanismus mußte für das Agentensystem abgeändert werden (siehe unten).

Ist zu überprüfen, ob eine bestimmte Aktion durchgeführt werden darf, so wird das entsprechende Permission-Objekt erzeugt und der AccessController zur Überprüfung aufgefordert. Verläuft diese erfolgreich, so kehrt die Methode zurück, andernfalls wird eine SecurityException ausgeworfen. Ein Beispiel für eine solche Überprüfung ist unten angeführt (es wird geprüft ob auf alle Dateien des Root-Verzeichnis, exklusive Subverzeichnisse, sowohl Lese- als auch Schreibzugriff möglich ist):

```
FilePermission perm=new FilePermission("/*","read,write");
AccessController.checkPermission(perm);
```

Die Überprüfung erfolgt durch Untersuchung der Berechtigungen aller Programmstücke des Stacks des Threads in welchem die Überprüfung durchgeführt wird. Besitzt irgendein Codestück auf dem Stack die Berechtigung nicht, so wird der Zugriff verweigert. Ausnahmen bestehen für privilegierten Code. Ist es beispielsweise notwendig, daß ein Code eine Methode aufruft, welche zusätzliche Berechtigungen benötigt, so kann dies hiermit erfolgen. Ein Beispiel ist

die Bildschirmausgabe eines Applets. Es selbst ist hierzu nicht berechtigt. Ruft es jedoch eine Systemmethode auf, so soll der Zugriff erlaubt werden. Hierzu kann ein Code-Block als privilegiert gekennzeichnet werden, was zur Folge hat, daß bei der Überprüfung der Berechtigungen der Methoden auf dem Stack die Überprüfung eingestellt wird. Die Methode checkPermission beginnt bei dem diese Methode direkt aufrufenden Objekt und überprüft dieses. Ist die Prüfung erfolgreich, so wird eine Stufe weiter nach oben gestiegen und dort die Prüfung durchgeführt. Ist nun eine solche Methode als privilegiert gekennzeichnet, so stoppt die Prüfung *nach* dieser Methode.

4.2.6 Implementation

Die Implementierung des Sicherheitssystems basiert darauf, daß jeder Agent von einem eigenen Classloader geladen wird, ebenso wie Bibliotheken, welche er verwendet. Lediglich die JDK-Klassen und die Klassen des Agentensystems werden vom gemeinsamen Classloader geladen. Dies ermöglicht es einerseits jedem Agenten andere Berechtigungen zuzuteilen, andererseits verhindert es, daß ein Agent auf Methoden und Felder anderer Agenten zugreifen kann, selbst wenn er eine Referenz darauf erlangen sollte. Hiermit ist auch noch ein weiterer Vorteil verbunden: Jeder Agent kann seine eigenen Klassen mit sich führen und diese müssen keinen eindeutigen Namen haben. Es können bei zwei Agenten problemlos zwei Klassen existieren, welche (inklusive den packages) exakt gleichen Namen besitzen, jedoch eine vollständig verschiedene Schnittstelle (oder auch nur eine andere Implementierung) besitzen.

Durch das Abstellen auf den Classloader sind keine Veränderungen an der virtuellen Maschine (VM) erforderlich, sodaß die Portabilität sichergestellt ist: Auf jeder Plattform, welche zumindest die Version 1.2 des JDK unterstützt (Version 1.1 verwendet ein stark unterschiedliches Sicherheitssystem), kann daher das Agentensystem eingesetzt werden.

Da das Standard-Sicherheitssystem keine dynamischen Berechtigungen kennt, mußten auch einige zusätzliche Klassen implementiert bzw. abgeändert werden. Hierbei stellte sich das Problem, daß manche davon als „final“ spezifiziert sind. Bei diesen mußte der Sourcecode genommen und eine Parallel-Implementierung durchgeführt werden, da durch dieses Schlüsselwort Subklassen verboten werden. Die Abbildung 47 verdeutlicht die Zusammenarbeit der einzelnen Klassen des Sicherheitssystems.

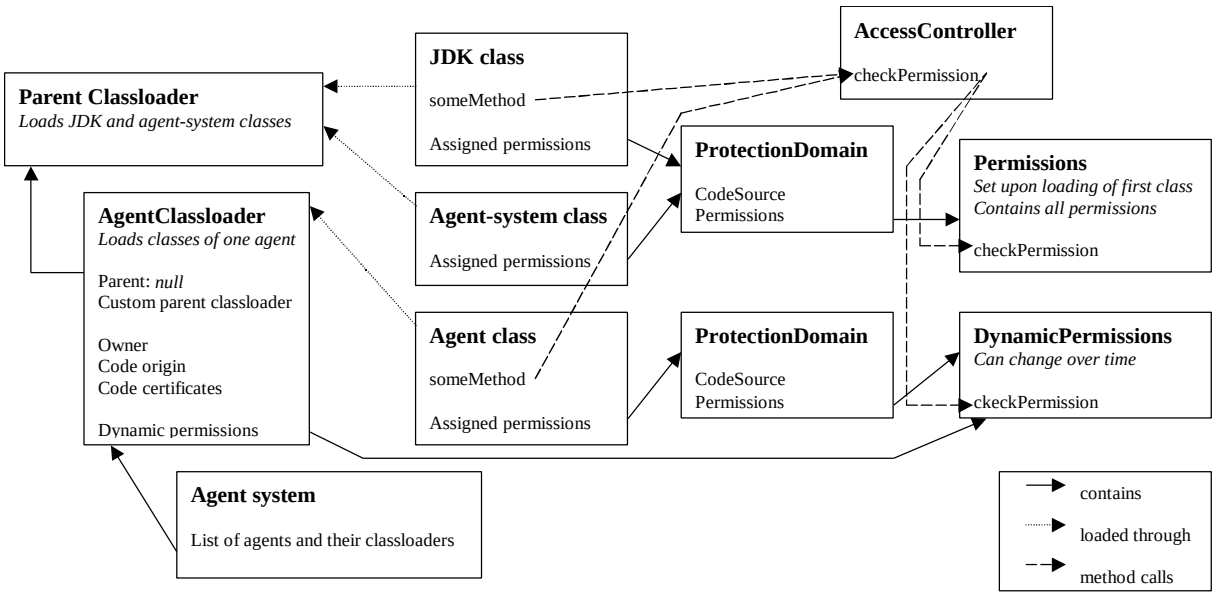

Abbildung 47: Das Sicherheitssystem im Überblick

Über den „Parent Classloader“ werden die Klassen des JDK und des Agentensystems geladen. Es handelt sich hierbei um den normalen Classloader, welcher auch den Standard-CLASSPATH zum Suchen der Code-Dateien verwendet. Allen hierdurch geladenen Klassen wird derselbe ProtectionDomain zugeordnet, welcher den Ursprung der Klassen enthält (URL bzw. Verzeichnis auf der Festplatte) und eine Kollektion der Berechtigungen dieses Codestükkes als Permissions Objekt. Dieses Berechtigungen werden schon beim Laden der Klasse durch den Classloader gesetzt und können später nicht mehr verändert werden (read-only). Agenten-Klassen werden im Gegensatz dazu von jeweils einem AgentClassloader geladen (für jeden Agenten wird ein Eigener vom Agentensystem erzeugt, welcher eine Liste über diese und sonstige Informationen in der Struktur AgentData speichert). Auch diesen Klassen wird ein ProtectionDomain zugewiesen, welcher jedoch ein besonderes Permissions-Objekt enthält. Dieses ist vom Typ DynamicPermissions und erlaubt auch Veränderungen in den Berechtigungen zu einem späteren Zeitpunkt. Da auf dieses Objekt später nicht mehr direkt zugegriffen werden kann, wird eine Referenz darauf (privat, daher kein Zugriff von anderen Klassen/Objekten) im Classloader gespeichert, welcher dann auch die Modifikationen der Berechtigungen übernimmt. Ein Agent kann zwar auf seinen eigenen Classloader Zugriff erlangen, doch kann er dort keine besonderen Methoden ausführen. Insbesondere ist ihm ein typecast auf Agent-Classloader oder ein Aufruf von dessen Methoden verwehrt. Auch kann er keinen anderen Classloader erreichen: Dies wäre sehr gefährlich, da er dann Code mit anderen Berechtigungen laden könnte.

Wird eine Methode aufgerufen, welche nur mit besonderen Berechtigungen ausgeführt werden darf, so ruft diese AccessController.checkPermission mit der benötigten Berechtigung als Parameter auf. Diese Methode überprüft den gesamten Stack, ob alle darauf befindlichen Methoden diese Berechtigung besitzen. Hierzu wird von jeder Methode die Klasse bestimmt und deren Permissions Objekt inspiziert. Gelangt die Prüfung bis zur obersten Stufe (=entweder das Runtime-System, welches die main-Methode aufrief, oder die Methode run eines eigenen Threads) so wird der Zugriff gewährt, ansonsten eine Exception ausgeworfen und damit die Ausführung des folgenden Codes verhindert. Da die Prüfung also nur auf dem statischen Aspekt des Programmcodes beruht, muß in das DynamicPermissions Objekt die dynamische Komponente mit integriert werden.

Die Verwendung eines eigenen Classloaders bedingt jedoch, daß der Programmcode tatsächlich selber gesucht und geladen werden muß. Dies wird nicht direkt vom Classloader durchgeführt sondern von der Klasse ResourcePackage, wovon jeweils ein Objekt für jeden Classloader existiert. Es repräsentiert entweder das Verzeichnis, in welchem der Programmcode abgelegt ist, bzw. die JAR-Datei, in welcher er enthalten ist. Welches Paket bzw. Verzeichnis verwendet wird kann beim Start eines Agenten angegeben werden. Ansonsten wird ein definierten Suchpfad verwendet: Zuerst wird der Standard-Classpath durchsucht, danach das Verzeichnis (bzw. die Dateien), welches in der Environment-Variable AGENT_LIBRARY_CLASSPATH angegeben ist, anschließend das aus der Variable AGENT_CLASSPATH. In diesen Pfaden wird sowohl direkt nach dem Programmcode gesucht (als .class - Datei), wie auch in allen JAR und ZIP Dateien. Konnte die Klasse nicht gefunden werden, so wird eine Exception ausgeworfen. Ist der Programmcode in einer JAR-Datei eingepackt, wird beim ersten Zugriff darauf die Signatur (sofern vorhanden) überprüft. Hierbei findet ein konservativer Ansatz Verwendung: Es werden nur jene Code-Zertifikate akzeptiert, mit welchen *alle* Dateien des Archivs signiert wurden und bei denen auch für *alle* Dateien die Signatur erfolgreich überprüft werden konnte. Dies bedeutet zwar eine gewisse Einschränkung, hat aber einen großen Vorteil: Alle Klassen eines Agenten besitzen dieselben Berechtigungen. Dieser muß also nicht darauf achten, welcher Code welchen Teil aufruft und daher bestimmte Aktionen (nicht) durchführen darf. Nachteilig ist, daß in ein Paket nicht von einem zweiten Hersteller weitere Klassen eingefügt bzw. ersetzt werden können, selbst wenn dieser (wenn gesondert verwendet) die gleichen Berechtigungen erhalten würde (was aber nicht für jedes Agentensystem korrekt sein muß). Dies kann jedoch einfach umgangen werden indem die ursprüngliche Signatur entfernt wird und alle Dateien neu signiert werden (was auch ohne Kenntnis des Sourcecodes möglich ist). Nicht

jeder Hersteller wird dies jedoch besonders gerne durchführen, da er hierbei für fremden Code garantieren muß.

Eine weitere Folge des Sicherheitsmodells ist, daß jeder Agent nur für sich selbst beurteilt wird, also keine Delegation von Berechtigungen möglich ist (siehe auch oben). Wenn daher ein Agent einen Sub-Agenten erzeugt, so werden diese von verschiedenen Classloadern geladen. Selbst bei Austausch von gegenseitigen Referenzen ist kein Methodenaufruf möglich. Auch in diesem Fall von zwei besonders eng zusammengehörenden Agenten ist eine Kommunikation über Nachrichten erforderlich.

Die Kommunikation zwischen Agenten erfolgt über Nachrichten, welche über das Agentensystem ausgetauscht werden. Da auch für die darin enthaltenen Objekte das selbe Sicherheitssystem gilt, können auch sie nicht direkt von einem Agenten an den anderen weitergegeben werden: Der Empfänger könnte die Daten nicht auslesen. Daher ist eine Konvertierung in Objekte des Empfänger-Agenten erforderlich. Dies erfolgt durch Serialisierung der Nachricht im Kontext des Senders und anschließender Deserialisierung im Kontext des Empfängers. Hierdurch werden Objekte aus dem Sicherheitsbereich des Senders in Objekte des Sicherheitsbereiches des Empfängers (=über dessen Classloader geladen) umgewandelt. Dies hat den Nachteil, daß jede Nachricht serialisierbar sein muß, was jedoch in den meisten Fällen kein großes Problem darstellt (Markierung als serialisierbar durch Implementierung des leeren Interfaces java.io.Serializable reicht aus). Dies hat jedoch auch Vorteile: So ist durch die Konvertierung sichergestellt, daß der Empfänger-Agent mit der Nachricht zumindest potentiell etwas anfangen kann: Er kennt die zugehörigen Klassen und diese sind zumindest sehr ähnlich. Weiters ist es hierdurch möglich alte Versionen von Objekten durch Neue zu ersetzen sodaß Abwärtskompatibilität erreicht werden kann (hierzu müssen jedoch bei der neueren Version die De-/Serialisierungs-Methoden überschrieben werden).

Für zusätzliche Klassifikationen (siehe 4.2.4) wurden in dem package „Classification“ einige Klassen implementiert, welche jedoch nur als Beispiel gedacht sind. So existiert eine Klasse für alllgemeine Zertifikate sowie eine für Stammkunden. Besondere Überprüfungen finden hierbei nicht statt. Als Resultat bei erfolgreicher Überprüfung wird der Preis für Berechtigungen generell um 10 % reduziert.

Basis-Sicherheitssystem:

? AgentClassloader: Hierbei handelt es sich um das Hauptelement des Sicherheitssystems, welches auch dafür zuständig ist, benötigte Klassen eines Agenten zu definieren (für den

Such- und Ladevorgang ist die Hilfsklasse ResourcePackage, siehe unten, verantwortlich). Dieser Classloader hat keinen Parent-Classloader, sondern ein solcher muß explizit zusätzlich verwaltet werden. Dies ist erforderlich, da wir Klassen nicht durch den System-Classloader laden dürfen (solche Klassen würden immer alle Berechtigungen erhalten), sondern höchstens durch den Primordial-Classloader. Auf diesen erhält man nur Zugriff über die Basisklasse SecureClassloader, welche jedoch vorher (was unerwünscht ist) den Parent-Classloader bemühen würde. Dieser Zugriff ist erforderlich um Basis-Klassen des JRE zu laden (z. B. java.lang.*) wobei auch für diese die Berechtigungen entsprechend gesetzt werden müssen. Im Gegensatz dazu müssen Klassen des Agentensystems *immer* vom selben Classloader geladen werden und *immer* volle Berechtigungen erhalten (dies ist z. B. für die Klasse AgentBase von Bedeutung welche als Basisklasse eines Agenten von jedem Classloader benötigt wird). Diese Klassen werden daher über den expliziten Parent-Classloader (wobei es sich um den System-Classloader handelt) geladen. Würden auch diese Klassen jeweils durch den AgentClassloader geladen, so könnte weder das Agenten-System auf den Agenten zugreifen noch umgekehrt. Der Classloader überprüft auch Package-Sealing, d. h. ob weitere Klassen in diesem package definiert werden dürfen (potentieller Angriff: Eine eigene Klasse in ein package einfügen erlaubt Zugriff auf package-private Daten und Methoden). Über diese Klasse werden auch zusätzliche Berechtigungen vergeben bzw. solche wieder entzogen indem das zugehörige DynamicPermissions-Objekt manipuliert wird. Bei jeder Änderung wird der Agent über diese durch Aufruf seiner Methode firePermissionChange informiert.

? PolicyByValue: Die Basisklasse für die Zuteilung von Berechtigungen nach dem erläuterten System. Berechtigungen werden zu Gruppen zusammengefaßt und jedem Agenten eine Gruppe zugeordnet. In der Klasse werden auch Abfrage-Funktionen definiert um dem Classloader die Arbeit zu erleichtern: Basis- und zusätzliche Berechtigungen, die billigste Berechtigung, welche eine gewünschte inkludiert (falls diese nicht direkt selbst erhältlich ist), etc. Um unnötige Käufe zu vermeiden, können auch Gruppen von Berechtigungen erfragt werden. Bei der Berechnung des billigsten Sets für eine Menge an Berechtigungen wird nur eine Annäherungslösung gesucht, da es sich um eine exponentielle Zeitkomplexität handelt (die Anzahl der potentielle Berechtigungen, minimal 17, ist zu groß um dies noch in angemessener Zeit berechnen zu können). Existiert eine Lösung so wird diese mit dem implementierten Verfahren auch gefunden. Der Algorithmus sucht für jede gewünschte Berechtigung die billigste mögliche Berechtigung und entfernt abschließend all jene, welche von anderen ebenso enthaltenen impliziert sind. Dies ist nicht unbedingt die optimale Lö-

sung, da eventuell eine teurere (derzeit nicht enthaltene) Berechtigung existiert, welche mehrere der enthaltenen Berechtigungen ersetzen könnte und billiger ist als die Summe derer Preise. Es handelt sich hierbei um eine abstrakte Klasse, welche nur die allgemeine Funktionalität zur Verfügung stellt. Welche Berechtigungen zugeteilt werden und nach welchen Kategorien Agenten beurteilt werden ist in einer abgeleiteten Klasse zu implementieren (Siehe unten AgentPolicy)

? DynamicPermissions: Dies ist eine Erweiterung der Klasse java.security.Permissions. Da diese jedoch als final deklariert ist und somit Subklassen unmöglich sind wurde der Programmcode kopiert und die zusätzlich notwendige Funktionalität integriert. So ist es hiermit nicht nur möglich Berechtigungen hinzuzufügen, sondern auch wieder zu entfernen. Die Speicherung von Berechtigungen erfolgt gruppiert, wobei auch bei den Einzel-Kollektionen ein Löschen nicht vorgesehen ist (außer bei der hier ebenso selbst implementierten). Dieses erfolgt daher (bei anderen Kollektionen) auf die Weise, daß ein neues Objekt erstellt wird, in welches alle außer der zu löschenden Berechtigung kopiert werden.

? ResourcePackage: Dies repräsentiert die Code-Basis eines Agenten, wobei es sich entweder um ein Archiv (JAR oder ZIP) oder ein Verzeichnis handelt. Diese Klasse ist dafür zuständig den Bytecode einer Klasse sowie sonstige Ressourcen zu finden und zu laden (siehe ebenso oben). Hierbei werden enthaltene Archive durchsucht (nur eine Ebene; nicht rekursiv). Beim Laden der ersten Klasse, welches immer die Hauptklasse des Agenten ist, wird der Code-Ursprung eingetragen falls er nicht schon beim Erzeugen des Agenten angegeben wurde. Hierbei wird das erste Verzeichnis bzw. Archiv verwendet in dem eine passende Klasse (=Package- und Klassen-Name korrekt; Vaterklassen werden nicht geprüft) gefunden wird. Beim Laden der ersten Klasse werden auch die Code-Zertifikate überprüft und anschließend gespeichert, sodaß diese Prüfung nur einmal erfolgt. Wenn ein Agent auf einen anderen Rechner verlagert werden soll, so muß er hierzu seinen gesamten Code mitnehmen. Ist dieser in einem Archiv gespeichert, so stellt sich kein Problem. Andernfalls ist diese Klasse auch dafür zuständig die notwendigen Dateien in ein Archiv zusammenzupacken. Hierzu werden alle Dateien des Start-Verzeichnisses (wo die Basisklasse des Agenten gefunden wurde) und alle Sub-Verzeichnisse rekursiv eingepackt. Es werden zwar Digests erstellt um Veränderungen erkennen zu können, doch erfolgt keine Signierung. Beispiel: Die Agentenklasse ist MyAgentSystem.MainAgent und wurde als Datei an der Stelle C:\AgentHome\MyAgentSystem\MainAgent.class gefunden. Dies bedeutet daß das Startverzeichnis (=Pfad zum Suchen von weiteren Klassen/Ressourcen) C:\AgentHome\ ist. Soll

der Agent verlagert werden, so werden alle Dateien aus C:\AgentHome\ sowie allen Subverzeichnissen darin eingepackt.

Implementierung:

? AgentPolicy: Hierbei handelt es sich um eine Implementierung von PolicyByValue, welche zusätzlich noch einen Preis-Faktor inkludiert (z. B. Änderung durch zusätzliche Klassifikationen). Die hier implementierten Standard-Sicherheitseinstellungen werden weiter unten erläutert (siehe 4.2.10). Enthalten ist auch die Überprüfung der Code- bzw. Besitzer-Zertifikate. Hierbei wird nicht nur die korrekte Signatur des Zertifikates selbst überprüft, sondern auch ob es von einer bekannten Zertifizierungsstelle stammt oder nicht. Dazu wird die gesamte Zertifikatskette schrittweise überprüft (jedes Zertifikat muß im Prüfzeitpunkt gültig sein; für eine andere Prüfmethode siehe [Hammer 2000]) und das letzte Zertifikate muß eines der vom Agentensystem als vertrauenswürdig eingestuften CA-Zertifikate oder zumindest mit einem solchen signiert worden sein. Dies wird entweder durch Zertifikats-Vergleich festgestellt (falls in der Prüfkette enthalten) oder über den Aussteller (bzw. Subject des vertrauenswürdigen Zertifikates), wobei in letzterem Fall noch eine Überprüfung der letzten Signatur durchgeführt wird.

? ValuedPermissions: Dies ist eine Implementierung von DynamicPermissions, wobei Berechtigungen jeweils noch zusätzlich mit einem Wert versehen sind. Dies wird verwendet, um den Preis für die Berechtigung zu speichern.

Berechtigungen:

? CreateAgentPermission: Eine neue Berechtigung die festlegt, ob ein Agent einen weiteren Agenten erzeugen darf oder nicht. Dies ist ein Ansatz zur Verhinderung von Angriffen durch Ausnutzung aller Ressourcen (konkret dem Überschwemmen mit weiteren Agenten).

? FileLimitPermission: Die Berechtigung Dateien bestimmter Länge auf der Festplatte speichern zu dürfen. Sie ist etwas komplexer, da sie logischerweise auch alle kürzeren Dateien einschließen muß.

? RuntimePermission(„grantPermissions“), RuntimePermission(„revokePermissions“): Die Berechtigungen einem Agenten zusätzliche Berechtigungen zu gewähren bzw. zu entziehen. Sie werden vorsichtshalber überprüft bevor eine dieser Aktionen durchgeführt wird. Sie dienen zur Absicherung falls es einem Agenten doch gelingen sollte Zugriff auf den Classloader zu erreichen. Zu beachten ist, daß diese Berechtigungen nie vergeben werden. Um diese Aktionen daher durchführen zu können, muß der gesamte aufrufende Code die Be-

rechtigung AllPermission besitzen, welche alle (technisch möglichen) Rechte darstellt. Sollte ein Agent dies erreichen so wären die besonderen Berechtigungen nutzlos, da er dann bereits jedwede Aktion ohne Einschränkung durchführen könnte.

? RuntimePermission(„manageAgentSystem“): Siehe unten. Sie dient dem Schutz des Agentensystem, sodaß nur besondere Agenten das Agentensystem beenden können.

Agenten-Information:

? PermissionChangeListener: Eine einfache Interface-Klasse, welche ein Agent implementieren muß, ist er an Informationen über Veränderungen seiner Berechtigungen (nach deren Durchführung) interessiert. Zum Erhalt von Notifikation muß das entsprechende Objekt beim Agenten registriert werden.

? PermissionEvent: Das Objekt, das zur Benachrichtigung von Änderungen bei Berechtigungen verwendet wird. Es enthält die Berechtigung und einen Hinweis darauf, ob diese zugeteilt oder entzogen wurde.

Hilfsklassen:

? AgentData: Diese Klasse dient als Wrapper für die Meta-Informationen über einen Agenten, welche das Agentensystem für die Verwaltung benötigt. Im Hinblick auf das Sicherheitssystem ist hier hauptsächlich eine Liste der erfolgreich überprüften Code-Zertifitake, der Pfad/Dateiname der Agentenklassen, die Berechtigungsgruppe des Agenten, der generelle Preisfaktor, die überprüften zusätzlichen Klassifikationen, sowie eine Referenz auf den zugehörigen Classloader enthalten.

? LimitedFileOutputStream: Hierbei handelt es sich um einen speziellen OutputStream, welcher zusätzlich eine Beschränkung der Länge der Ausgabedaten ermöglicht. Hierdurch ist eine Begrenzung des von einem Agenten verwendeten Festplattenplatzes möglich. Um eine Umgehung zu verhindern, muß die Verwendung der normalen FileOutputStream-Klasse unterbunden werden. Hierbei stellt sich ein besonderes Problem, da es in der Standard-Konfiguration nicht möglich ist eine angeforderte Klasse durch einen anderen Code zu ersetzen. So ist es ohne Veränderung der JVM nicht möglich, bei Anforderung eines normalen FileOutputStreams einen LimitedFileOutputStream (bzw. auch nicht bloß dessen Bytecode) zu liefern. Als Ansatz wurde daher gewählt, daß grundsätzlich ein FileOutputStream nicht erzeugt werden kann (führt zu einer Exception). Agenten *müssen* daher die Ersatzklasse verwenden, wollen sie Daten in einer Datei speichern. Um die Performanz nicht zu stark zu beeinträchtigen wird nicht bei jedem Schreibvorgang die Berechtigung geprüft,

sondern standardmäßig nur alle 1024 Bytes (Konstante). Hierzu wird eine FileLimitPermission mit der aktuellen Länge vor dem tatsächlichen Schreibvorgang verwendet.

? UnresolvedPermission, UnresolvedPermissionCollection: Diese beiden Klassen wurden identisch aus dem Java Framework Source-Code kopiert. Dies war notwendig, da sie einerseits als final deklariert sind (= keine Subklassen möglich), andererseits nur packageweite Sichtbarkeit besitzen (package-private). Sie können daher nur im package java.security verwendet werden. Um eine Neuimplementierung zu umgehen, wurden diese Dateien kopiert und mit einem anderen Package-Namen versehen. Ihre Aufgabe ist es als Platzhalter für Berechtigungen zu dienen, welche noch nicht geladen wurden, während die Policy bereits erzeugt wurde. Spätestens bei einer tatsächlichen Prüfung werden diese Objekte durch die tatsächliche Klasse der Berechtigung ersetzt.

4.2.7 Angriffe und deren Abwehr

Zur Überprüfung des Sicherheitssystems wurde ein Agent geschrieben (Programmcode siehe Anhang) der auf verschiedenste Arten versucht, zusätzliche Berechtigungen zu erlangen. Gleichzeitig dient er auch dazu die verschiedenen Gruppen von Berechtigungen (je nach Code-Zertifikaten, Besitzer, …) zu überprüfen. Die versuchten Angriffsarten sind:

? Zugriff auf den System-Classloader: Es wird versucht Zugriff auf den System-Classloader zu erhalten. Über diesen könnten dann Klassen geladen werden, welche andere Berechtigungen hätten als solche, die über den Agenten-Classloader geladen werden.
? Zugriff auf den System-Classloader über die Superklasse: Dasselbe wird versucht, indem der Zugriff über die Superklasse (java.security.SecureClassLoader) versucht wird.
? Berechtigungen ergänzen über Classloader: Hier wird versucht, über das Reflection-API zusätzliche Berechtigungen zu erlangen, indem die Methode „grantAdditionalPermission“ aufgerufen wird. Dies ist direkt nicht möglich, da ein typecast auf AgentClassloader schon beim Kompilieren als fehlerhaft (keine public Klasse) ausgewiesen wird.
? Zugriff auf den Classloader des Agentensystems: Da das Agentensystem für jeden Agenten zugänglich ist könnte versucht werden auf dessen Classloader zuzugreifen. Dies ist sehr gefährlich, da das Agentensystem vom System-Classloader mit allen Berechtigungen geladen wird.
? Direktzugriff auf den ProtectionDomain: Über das Klassen-Objekt kann auch direkt auf den Protection-Domain zugegriffen werden (was jedoch nicht erlaubt ist und daher fehlschlägt). Wäre dies erfolgreich, könnten dort direkt Berechtigungen hinzugefügt werden.

? Zugriff auf den ProtectionDomain über Reflection-Methoden: Dasselbe wird über das Reflection-API versucht, da hierdurch eventuell Zugriff erlangt werden könnte.
? Lesen von C:\autoexec.bat: Bei diesem und den folgenden Angriffen handelt es sich mehr um Tests, ob die Berechtigungen auch tatsächlich durchgesetzt werden. Konkret wird hier versucht die Datei autoexec.bat zu lesen (Stellvertretend für sonstige sicherheitskritische Dateien).
? Schreiben auf c:\Test.out.txt: Es wird versucht, auf eine Datei zu schreiben ohne hierzu die erforderlichen Berechtigungen zu besitzen. Hierdurch könnten sowohl Dateien verändert als auch erzeugt werden (besonders in Hinsicht auf Viren gefährlich).
? Starten eines neuen Prozesses: Dieser Angriff versucht das Sicherheitssystem zu umgehen indem ein neuer Prozeß gestartet wird. Da es sich tatsächlich um einen Prozeß und nicht bloß einen Thread handelt, unterliegt er nicht dem Java-Sicherheitssystem, sondern lediglich den Betriebssystem-Einschränkungen. Hierdurch könnte sogar ein zweites Agentensystem gestartet werden.

Alle diese Angriffe werden in verschiedenen Programm-Abschnitten bzw- Zuständen ausgeführt:

? Normalzustand: Die Angriffe werden als normaler Code ausgeführt.
? Nach dem Versuch „AllPermission“ zu erwerben: Es wird versucht die Berechtigung für alle Aktionen zu erwerben. Schlägt dies fehl, so ergibt sich kein Unterschied zum ersten Versuch. Ansonsten können alle Aktionen durchgeführt werden.
? In einem privileged-Block: Auch als privilegierter Code darf sich keine Änderung ergeben, da der Programmcode des Agenten logischerweise den identischen Berechtigungen unterworfen ist wie er selbst (dies wäre nur sinnvoll in Verbindung mit einer Bibliothek welche vom Agenten aufgerufen wird).
? In einem eigenem Thread: Es wird ein gesonderter Thread gestartet, in welchem der Test durchgeführt wird. Das Sicherheitssystem darf nicht vom Ausführungs-Thread abhängen.

Schließlich wurde noch ein kurzer Spezial-Agent (Programmcode im Anhang) geschrieben, welcher versucht das Agentensystem zu beenden (sowohl direkt als auch in einem privileged-Block). Dies ist theoretisch möglich, da der Agent jederzeit darauf zugreifen kann um z. B. eine Nachricht abzusenden (die zugehörigen Methoden müssen aus implementationstechnischen Gründen öffentlich sein). Dies muß auf jeden Fall verhindert werden. Hierzu wurde die Berechtigung RuntimePermission(„manageAgentSystem“) eingeführt. Sie wird überprüft,

wenn ein Agentensystem erzeugt oder beendet werden soll. Da diese Berechtigung nicht vergeben wird, sind nur Agenten mit der Berechtigung AllPermission (alle Berechtigungen) hierzu in der Lage. Auch aus diesem Grund sollte diese Berechtigung daher nur an besonders vertrauenswürdige Agenten, wenn überhaupt (dann auch bei Programmfehlern keine Probleme möglich), vergeben werden.

4.2.8 Nicht abwehrbare Angriffe

Nicht alle Angriffe können durch das Agentensystem abgewehrt werden. Gegen die folgenden Attacken ist derzeit keine Abwehr möglich. Teilweise auch deshalb, da im Java Runtime Environment keine entsprechende Vorkehrungen möglich sind (z. B. Rechenzeit-Beschränkung für Threads).

Ein möglicher Angriff wäre beliebig viele Threads zu erzeugen bis die Rechenleistung sinkt oder schließlich (eventuell) die JVM abstürzt. Dies könnte verhindert werden doch müßte hierzu die Klasse Thread ersetzt werden, was wiederum ebenso wie bei LimitedFileOutputStream voraussetzt, daß alle von Agenten verwendeten Threads davon abgeleitet werden müßten. Dies ist jedoch schwieriger, da sie vielfach in Bibliotheken verwendet wird, welche nicht abgeändert werden können aber dennoch dieser Beschränkung unterliegen.

Allgemein kann gegen Ressourcen-Erschöpfungs-Angriffe nichts unternommen werden, da dies nicht unterstützt wird. So ist etwa eine Beschränkung des verwendeten Speicherplatzes unmöglich: Ein Agent kann jederzeit beliebig viele neue Objekte anlegen. Analog dazu ist der Versuch, das Agentensystem dadurch zu überlasten indem Unmengen an Nachrichten verschickt werden (eventuell auch Broadcasts da bei diesen die Verteilung aufwendiger ist).

Die Verwendung von verschiedenen Classloadern als Sicherheitssystem bedingt noch eine weitere Schwierigkeit: So kann ein Agent zwar keine Methoden von Objekten aus anderen Classloadern als seinem eigenen und dem System-Classloader aufrufen, doch kann er sehr wohl dessen Schnittstelle herausfinden. Über das Reflection-API können von public, protected und package-private Methoden (und Feldern) sowohl Name als auch Parameter bzw. Typ festgestellt werden. Tatsächlicher Aufruf oder Zugriff ist jedoch nicht möglich. Dies bedingt zwar keine Sicherheitslücke doch können hiermit immerhin Informationen gesammelt werden, welche als solche schon Bedeutung besitzen könnten: Entweder für sich oder als Hilfestellung um andere Angriffe durchführen oder verbessern zu können.

Mangelnde Ressourcen-Kontrolle in Java:
?? Thread-Überlast
?? Ressource-Exhaustion

Mangelnde Berechtigungen für Reflection-API:
?? Schnittstelle ausspähen

Fehlende Zertifikats-Infrastruktur:
?? Man-in-the-middle Angriff beim Agenten-Transfer
?? Keine Widerrufs-Prüfung von Zertifikaten

Nicht implementiert:
?? Beenden **aller** Threads eines Agenten

Abbildung 48: Einteilung nicht-abwehrbarer Angriffe

4.2.9 Die Kryptographie-Infrastruktur

Die erforderlichen Kryptographie-Methoden wurden nicht selbst implementiert, sondern die Bibliothek „IAIK“ der TU Graz verwendet. Dennoch wurden einige Elemente zusätzlich spezifiziert und implementiert, welche direkt für das Agentensystem von Bedeutung sind und am Besten durch kryptographische Mittel abgebildet werden. Ein Beispiel hierfür ist die Identität eines Agenten, welche als Zertifikat abgebildet wird. Diese Identitäten werden gespeichert, um später Agenten einfacher erzeugen zu können (es braucht nur das Zertifikat ausgewählt zu werden). Die Zertifikate, zugehörige private Schlüssel, sowie das Server-Zertifikat werden in einer eigenen Datei (=PersonalSecurityStore) gespeichert. Diese wird zur Zeit mit einem Standard-Paßwort verschlüsselt, sodaß die Daten nicht direkt daraus ausgelesen werden können. Um eine höhere Sicherheit zu erreichen und falls mehrere Benutzer an einem Rechner arbeiten und jeweils ihr eigenes Agentensystem starten möchten, müßte dem Start des Systems eine Abfrage von Name und Paßwort vorausgehen: An Hand des Namens würde dann die Datei ausgewählt und mit dem Paßwort diese nachfolgend entschlüsselt.

Um zwischen mehreren Systemen sicher Agenten übertragen zu können besitzt jedes Agentensystem ein eigenes Zertifikat mit dem Namen „Server Certificate“. Code, dessen Besitzer durch dieses Zertifikat repräsentiert wird, besitzt in der Standard-Einstellung besondere Berechtigungen (siehe unten). Existiert beim Start des Agentensystems noch kein SecurityStore, so wird ein neuer erzeugt und in diesen auch gleich ein neues Server-Zertifikat eingefügt. Da die Erzeugung von Zertifikaten Kryptographie-Provider-pezifisch ist (nur Lesen und Verwenden ist allgemein definiert) wurde eine eigene Subklasse geschaffen, welche Zertifikate mit der IAIK-Bibliothek erstellen kann. Soll ein anderer Provider verwendet werden, so müßte diese Klasse angepaßt oder eine Parallel-Implementierung geschaffen werden.

Für die Implementierung des eigenen Zertifikates ist ein eigener Kryptographie-Provider erforderlich. Dieser ist dafür zuständig für ein gewünschtes Kryptographie-Objekt (in diesem Fall nur Zertifikate, allgemein aber auch für Schlüssel, Hashverfahren, etc.) die entsprechende Factory-Klasse zu definieren. Diese findet dann Anwendung, wenn ein serialisiertes Zertifikat wiederhergestellt werden soll (das Speicherformat ist zwar definiert doch kann auf jedem Rechner eine andere Implementierung (=Klasse) verwendet werden). Es wurden nur die beiden Agenten-Identitäts-Zertifikate sowie das Zertifikate zur Speicherung öffentlicher Schlüssel hier integriert. Das Server-Zertifikat kommt nicht vor, da es sich um ein Standard-X.509-Zertifikat handelt, welches vom normalen Kryptographie-Provider wiederhergestellt wird.

Die Identität eines Agenten wird durch ein Zertifikat modelliert. Es enthält Informationen über den Namen des Agenten, dessen Besitzer, den Ursprung (URL des Heimat-Agentensystems), eine (hoffentlich) eindeutige ID (analog zu UUID‘s) und den öffentlichen Schlüssel; jedoch keine Informationen über den Programmcode. Hierdurch kann eine Identität mit jedem beliebigen Code verwendet werden (Informationen darüber, z. B. dessen Signator, sind in der JAR-Datei enthalten). Um die Arbeit mit Agenten einfacher und schneller zu gestalten, existieren zwei Untergruppen von Agenten-Identitäten: Eine einfache Form, welche nur Strings enthält und nicht signiert ist, sowie eine vollständige Form, welche ein Zertifikat des Besitzers enthält und auch signiert werden kann. Um zwei weitere Angriffe abwehren zu können kann der Ersteller eines Agenten auch noch sogenannte deploy-Signaturen integrieren. Durch die Code-Signatur kann die Identität an bestimmten Code gebunden werden, sodaß dieser bei mobilen Agenten nicht auf dem Weg zu einem anderen Agentensystem ausgewechselt werden kann. Mittels der Initialisierungs-Signatur kann der Besitzer den Agenten nach Parametrisierung signieren, sodaß der Anfangszustand beweisbar (und damit bei bekannter Eingabe auch die Ausgabe ohne äußere Einwirkungen) wird. Um Agenten-Identitäten problemlos speichern und austauschen zu können, wird für ihre Speicherung eine Plattformunabhängige Form (Definition ASN.1, DER-Kodierung) verwendet (wie bei X.509-Zertifikaten). Dies hat aber auch zur Folge, daß Zertifikate nicht mehr einfach serialisiert werden dürfen. Dadurch ist es möglich verschiedene Implementierungen auf verschiedenen Systemen zu verwenden. Enthält ein Agent etwa zusätzlich die Identitäten anderer Agenten (Partner- oder Sub-Agenten), so darf die Serialisierung nicht mehr automatisch erfolgen. Vielmehr muß die Identität als „transient“ markiert werden und bei der Serialisierung die Zeile

```
out.writeObject(otherAgentsIdentity);
```

ersetzt werden durch:

```
out.writeObject(otherAgentsIdentity.getType());
out.writeObject(otherAgentsIdentity.getEncoded());
```

Hierbei wird zuerst der Typ des Zertifikates (hier einer Agenten-Identität) ausgegeben und anschließend die Identität selbst in codierter Form. Analog dazu muß das Einlesen erfolgen:

```
String type=(String)in.readObject();
byte[] buf=(byte[])in.readObject();
CertificateFactory cf=CertificateFactory.getInstance(type);
ByteArrayInputStream bais=new ByteArrayInputStream(buf);
otherAgentsIdentity =(AgentIdentity)cf.generateCertificate(bais));
```

Für die Speicherung zusätzlicher Elemente in der Agenten-Identität steht die Erweiterung AgentIdentityExtension zur Verfügung. Hierbei können beliebige serialisierbare Daten unter einem zugehörigen String-Schlüssel gespeichert werden. Die Codierung erfolgt wieder in einem unabhängigen Format doch muß darauf geachtet werden, daß die Nutzdaten selbst nur mittels Java-Serialisierung gespeichert und wiederhergestellt werden können (da keine Informationen über deren Typ bekannt sind).

Für die Erzeugung dieser beiden Zertifikate ist eine eigene Factory-Klasse erforderlich. Diese erzeugt ein neues Objekt (je nach Typ) und ruft die entsprechende Methode zum Einlesen der Daten auf. Ist das Zertifikat signiert, so wird auch noch anschließend die Signatur eingelesen, überprüft, und gespeichert. Es wurde nur die Methode implementiert ein einziges Zertifikat aus einem Stream wiederherzustellen. Mehrere Zertifikate aus einem Stream sowie Zertifikats-Widerrufslisten wurden, da nicht benötigt, auch nicht implementiert.

Um im SecurityStore auch einzelne Schlüssel bzw. Schlüssel-Paare speichern zu können, ist ein eigenes Zertifikat erforderlich, da die verwendeten Standard-Speicher nur private Schlüssel und Zertifikate speichern können. Bei einem Schlüsselpaar wird daher ein leeres unsigniertes Zertifikat erzeugt, welches nur den öffentlichen Schlüssel und den Namen des Schlüsselpaares enthält. Dieses wird dann zusammen mit dem privaten Schlüssel abgelegt. Auch für Public-Key-Zertifikate ist eine eigene Factory notwendig doch werden hier die Werte direkt in der Factory eingelesen und eingetragen.

4.2.10 Standard-Sicherheitseinstellungen

Die Standard-Policy ist in der Klasse AgentPolicy implementiert und basiert auf dem Besitzer (welcher der Identität des Agenten entnommen wird), der Signatur des Programmcodes, sowie

einem generellen Faktor für Preise von Berechtigungen (siehe oben 4.2.3). Die Einteilung von Agenten in Berechtigungsgruppen ist in der Abbildung 49 dargestellt.

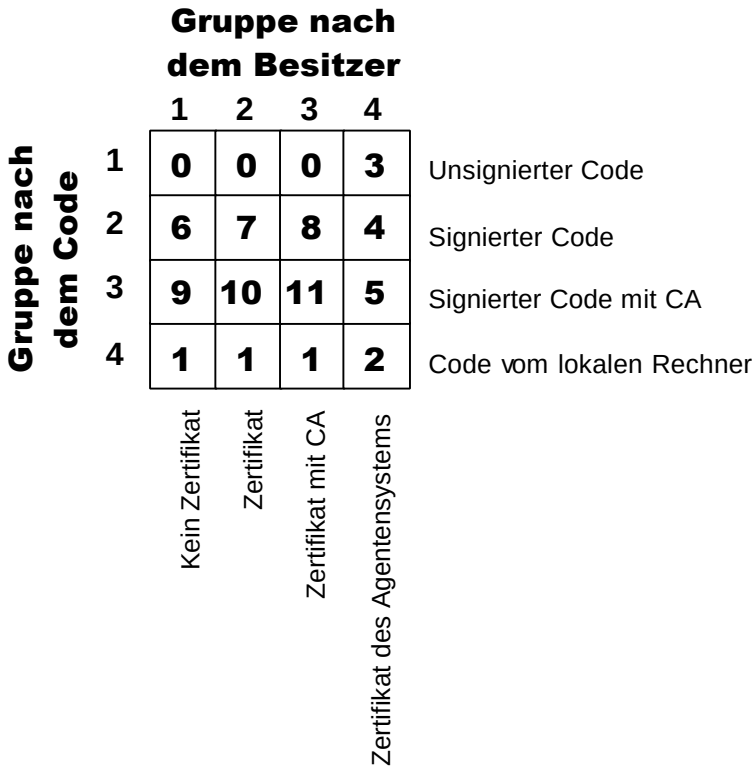

Abbildung 49: Einteilung von Agenten in Gruppen nach Besitzer und Code-Hersteller

Welche Berechtigungen mit welcher Gruppen in der Implementierung als Standard verbunden sind, kann der Tabelle 3 entnommen werden (bei den optionalen Berechtigungen ist auch der Preis angeführt):

Nr.	Basis-Berechtigungen	Optionale Berechtigungen
0	Keine	Keine
1	Standard-Berechtigungen Alle Properties lesen und schreiben	Alle Berechtigungen (0,1) Lesen von allen Dateien auf dem Agenten Such-Pfad (0,0)
2	Alle Berechtigungen	Keine
3	Standard-Berechtigungen	(Sub-) Agenten erzeugen (0,9)
4	Standard-Berechtigungen Kein Warn-Banner	Alle Berechtigungen (0,1)
5	Standard-Berechtigungen Kein Warn-Banner Alle Properties lesen und schreiben	Alle Berechtigungen (0,0)
6	Standard-Berechtigungen	Verbindung zu anderen Rechnern (0,3)
7	Standard-Berechtigungen	Verbindung zu anderen Rechnern (0,3) 50 kB temporäre Datei (0,5) (Sub-) Agenten erzeugen (1.0)
8	Standard-Berechtigungen	Verbindung zu anderen Rechnern (0,3) 100 kB temporäre Datei (0,7) (Sub-) Agenten erzeugen (0,9)
9	Standard-Berechtigungen	Verbindung zu anderen Rechnern (0,2)
10	Standard-Berechtigungen	Verbindung zu anderen Rechnern (0,2) 50 kB temporäre Datei (0,4) (Sub-) Agenten erzeugen (0,9)
11	Standard-Berechtigungen	Alle Properties lesen und schreiben (0,1) Verbindung zu anderen Rechnern (0,2) 100 kB temporäre Datei (0,6) (Sub-) Agenten erzeugen (0,8)

Tabelle 3: Standard-Berechtigungen nach Gruppen in POND

Unter den Standardberechtigungen sind diejenigen Berechtigungen zu verstehen welche normalerweise jedem Applet zugestanden werden. Es handelt sich hierbei nur um Berechtigungen zum Auslesen einiger Properties (z. B. Version des JDK) sowie den eigenen Thread zu stoppen. Hinzu kommen noch einige Berechtigungen welche für das Agentensystem speziell sind:

? Jeder Agent darf die IP-Adresse und den Namen des lokalen Rechners feststellen. Dies dient hauptsächlich dazu, damit Agenten feststellen können wo sie sich gerade befinden (Alternative: Extraktion dieser Informationen aus der Adresse des Agentensystems welche den Agenten immer zur Verfügung steht).

? Jeder Agent darf alle Dateien aus dem Bibliotheks-Suchpfad und allen Subverzeichnissen darin lesen. Dies ist zusätzlich notwendig, da sich dort oft nicht nur Programmcode sondern auch Konfigurationsdateien oder sonstige Ressourcen (Bilder für Buttons, Daten, etc.) befinden.

? Damit Agenten die Swing-Bibliothek verwenden können muß ihnen die Berechtigung „accessEventQueue“ gegeben werden. Dies ist zwar eine potentielle Sicherheitslücke, da dann

ein Agent u. U. auf Events anderer Agenten zugreifen kann, doch ist dies erforderlich, da die entsprechenden Aktionen in der Swing-Bibliothek nicht in privileged-Blöcke eingebunden sind, wie es eigentlich sein sollte (dann wäre dies nicht erforderlich).

Im Gegensatz zu Applets besteht jedoch für Agenten bei den Standardberechtigungen noch eine wichtige Einschränkung: Applets können auf allen Ports über 1024 auf Verbindungen warten. Dies ist Agenten normalerweise nicht erlaubt.

Die genaue Liste der Standard-Berechtigungen ist:

```
ValuedPermissions perms=new ValuedPermissions();
// Common properties which are always assigned
perms.add(new java.lang.RuntimePermission("stopThread"),-1);
perms.add(new java.util.PropertyPermission("java.version","read"),-1);
perms.add(new java.util.PropertyPermission("java.class.version","read"),
     -1);
perms.add(new java.util.PropertyPermission("os.name", "read"),-1);
perms.add(new java.util.PropertyPermission("file.separator","read"),-1);
perms.add(new java.util.PropertyPermission("path.separator","read"),-1);
perms.add(new java.util.PropertyPermission("line.separator","read"),-1);
// Not so common properties that will have to be acquired (but are free)
perms.add(new java.util.PropertyPermission("os.version","read"),0);
perms.add(new java.util.PropertyPermission("os.arch","read"),0);
perms.add(new java.util.PropertyPermission("java.vendor","read"),0);
perms.add(new java.util.PropertyPermission("java.vendor.url","read"),0);
perms.add(new java.util.PropertyPermission("java.specification.version",
     "read"),0);
perms.add(new java.util.PropertyPermission("java.specification.vendor",
     "read"),0);
perms.add(new java.util.PropertyPermission("java.specification.name",
     "read"),0);
perms.add(new java.util.PropertyPermission("java.vm.specification.version",
     "read"),0);
perms.add(new java.util.PropertyPermission("java.vm.specification.vendor",
     "read"),0);
perms.add(new java.util.PropertyPermission("java.vm.specification.name",
     "read"),0);
perms.add(new java.util.PropertyPermission("java.vm.version","read"),0);
perms.add(new java.util.PropertyPermission("java.vm.vendor","read"),0);
perms.add(new java.util.PropertyPermission("java.vm.name","read"),0);
// Allow all agents to get the IP address of the agent-system host
// (mostly needed for checking where they are)
perms.add(new java.net.SocketPermission("localhost","resolve"));
// All agents may read all libraries (needed e. g. for mailcap
// files in mail.jar or pop3.jar)
String libraries=System.getProperty("AGENT_LIBRARY_CLASSPATH");
if(!libraries.endsWith(""+java.io.File.separatorChar))
    libraries+=java.io.File.separatorChar;
perms.add(new java.io.FilePermission(libraries+"-","read"),-1);
// For testing, so agents can use Swing
perms.add(new java.awt.AWTPermission("accessEventQueue"),-1);
```

Wird die Standard-Policy geändert so muß auf die in der Datei AgentPolicy angeführten Berechtigungen geachtet werden, welche Agenten nie erhalten dürfen bzw. nicht erhalten sollten. Andernfalls könnte es für Agenten möglich sein, das Sicherheitssystem zu umgehen. In diese

Gruppe gehört auch die Berechtigung AllPermission, welche trotzdem im Standard-Modell in mehreren Gruppen enthalten ist. Hierbei wird davon ausgegangen, daß, wenn der Code-Hersteller bzw. Besitzer durch ein Zertifikat eindeutig feststellbar ist, auch größere Risiken eingegangen werden können.

4.3. Bezahlung von Leistungen

Die Bezahlung von Leistungen im Agentensystem POND erfolgt in zwei Kategorien: Einerseits die Bezahlung für Berechtigungen, welche mit dem Agentensystem selbst abgewickelt wird, andererseits die Bezahlung für beliebige Daten, Leistungen, Waren, etc., welche mit anderen Agenten durchgeführt wird. Dementsprechend erfolgen die Erläuterungen in drei Teilen: Das allgemeine Zahlungs-System, die Bezahlung für Berechtigungen und die allgemeine Bezahlung zwischen Agenten an Hand des Sportportals.

4.3.1 Das Zahlungs-System

Das Subsystem zur Bezahlung von Leistungen und Berechtigungen basiert auf frei erweiterbaren Klassen, welche Technologieunabhängig sind. Zur praktischen Verwendung wurden die wichtigsten Zahlungsarten implementiert. Zu den üblichen Zahlungsweisen wie etwa Kreditkarte wurde noch eine zusätzliche hinzugefügt: Die Verwendung von (lokalen) Gutscheinen. Deren Bedeutung insbesondere für die Bezahlung für Berechtigungen wird im nächsten Abschnitt näher erläutert.

Alle in diesem Kapitel erläuterten Klassen können auch in XML codiert und aus XML wiederhergestellt werden. Es handelt sich um eine Unterart der Serialisierung, welche einen Strom von Zeichen erzeugt der einer XML-Datei entspricht. Zu beachten ist hierbei jedoch, daß die Codierung jeweils nur ein einzelnes XML-Element zurückgibt. Um daher ein gültiges XML-Dokument zu erhalten müssen diese Elemente in ein leeres Dokument eingefügt werden. Diese Konstruktion erleichtert es, mehrere Objekte zusammenzufassen und gemeinsam zu versenden: Es wird ein Rahmendokument erstellt und alle enthaltenen Objekte in ihrer XML-Form hinzugefügt. Auf diese Weise wird etwa der Inhalt komplexer Nachrichten im Sport-Portal im ebXML-Format erzeugt.

4.3.1.1 Rechnung und Bezahlung

Eine Bezahlung erfolgt immer im Hinblick auf eine bestimmte Rechnung („Invoice“) und muß mit dem darin angegebenen Preis exakt übereinstimmen. Sowohl Rechnung als auch Bezahlung können elektronisch signiert sein, was in den meisten Fällen wohl vorausgesetzt werden wird.

? Invoice: Hierbei handelt es sich um ein Interface, welches die allgemeinen Eigenschaften einer Rechnung spezifiziert. Eine Rechnung ist mit einem Datum und einem Titel versehen und enthält eine Reihe von Positionen (Klasse InvoiceItem). Diese sind durch einen Text und einen zugehörigen Preis näher gekennzeichnet. Weiters existiert eine Funktion zur Berechnung des Gesamtpreises. Eine Rechnung wird immer von einer bestimmten Person (bzw. Agenten oder Agentensystem) ausgestellt, welche durch einen Text, ein Zertifikat, oder beides gekennzeichnet sein kann (keine anonymen Verkäufer; dies könnte bei Bedarf durch Pseudonym-Zertifikate realisiert werden). Weiters existieren noch die Methoden für die Signierung bzw. die Prüfung der Signatur.

? AnonymousInvoice: Hierbei handelt es sich um eine direkte Implementierung des Interfaces Invoice ohne zusätzliche Elemente. Die Anonymität bezieht sich hierbei auf den Empfänger der Rechnung, dessen Name nicht enthalten ist.

? NamedInvoice: Bei dieser Erweiterung wird zusätzlich noch der Rechnungs-Empfänger festgelegt: Entweder als Text, als Zertifikate, oder beides. Diese Information wird auch mitsigniert, sodaß der Empfänger später nicht verändert werden kann.

? Payment: Dies ist das Analogon zur Rechnung, die Bezahlung einer solchen. Auch hier wird wiederum die Signierung des gesamten Objektes unterstützt. Es enthält die zugehörige Rechnung. Zur Unterscheidung der verschiedenen Zahlungsarten ist jede mit einer Charakteristik zu kennzeichnen (z. B. „Creditcard“ für Kreditkarten-Zahlungen).

? PaymentBase: Hierbei handelt es sich um eine abstrakte Klasse, welche die wichtigsten Methoden von Payment implementiert, insbesondere die Funktionen zur Erstellung und Prüfung von Signaturen. Alle hier implementierten Zahlungsarten sind von dieser einen Klasse abgeleitet.

4.3.1.2 Implementierte Zahlungsarten

Vier verschiedene Zahlungsarten wurden implementiert, wobei eine (EmptyPayment) eigentlich keine Zahlungsart ist. Sie kann dafür verwendet werden auch Gratis-Angebote mit denselben

Protokollen abzuwickeln, z. B. zur Verhandlung über das gewünschte Datenformat oder die Art der Lieferung.

? CreditCardPayment: Die Zahlung per Kreditkarte ist durch die Angabe der Kartenfirma, des Namens auf der Karte, der Kartennummer und des Ablaufdatums gekennzeichnet. Insbesondere beim Datum muß bei der XML-Codierung darauf geachtet werden daß eine eindeutige Repräsentierung erfolgt, sodaß beim Parsen auch wieder ein identisches Objekt entsteht. Ansonsten würde die eventuell vorhandene Signatur ungültig werden.

? VoucherPayment: Hierbei handelt es sich um die Zahlung mittels eines Gutscheines, welcher eine Subklasse von java.security.cert.Certificate sein muß. Dies wurde deshalb gewählt, da ein elektronischer Gutschein ohnehin nur mit einer digitalen Signatur des Ausstellers Sinn macht und daher die zugehörige Infrastruktur verwendet werden kann. Beim enthaltenen Schlüssel handelt es sich um den öffentlichen Schlüssel dessen, für den der Gutschein ausgestellt wurde. Dadurch sind sowohl anonyme wie auch persönliche Gutscheine möglich, welche jedoch immer für eine bestimmte Identität ausgestellt werden (den Inhaber des zugehörigen privaten Schlüssels). Solche Gutscheine sind daher nicht frei übertragbar (keine Inhabergutscheine), sondern können immer nur vom rechtmäßigen Besitzer eingelöst werden (sofern der Empfänger die Kenntnis des privaten Schlüssels überprüft).

? DataPayment (=Tausch): Diese Klasse dient dazu den Tausch zwischen zwei Personen auf dieselbe Weise abwickeln zu können, wie einen Kauf. Hiermit können auch alternative Zahlungsarten emuliert werden ohne daß die Implementierung einer eigenen Klasse notwendig wäre (z. B. Übergabe von E-Cash). Die Daten können auf eine von drei Arten vorliegen: Als String, als Array von Bytes, oder als serialisierbares Objekt (ausschließlich; Kombinationen sind nicht möglich). Für die Codierung in XML werden nicht-textuelle Elemente als Hexadezimal-String codiert.

? EmptyPayment (=Gratis): Diese sehr einfache Klasse enthält keine weiteren Daten.

4.3.1.3 Hilfs-Klassen

Von den für die Implementierung erforderlichen Hilfs- und Zusatzklassen sollen nur die drei wichtigsten kurz erläutert werden. Diese finden auch in den Beispiels-Agenten Verwendung.

? Currency: Diese Klasse repräsentiert eine Währung mit ihren zusätzlichen Eigenschaften wie Kurz- und Langbezeichnung, Abkürzungen, Stellung (vor- oder nachgestellt), sowie

eines Wechselkurses zu einer Referenzwährung (Verwendet: Euro). Zusätzlich kann auch noch angegeben werden nach welcher Locale (Orts-Informationen in Java) die Ausgabe bzw. das Parsen erfolgen soll (so verwendet etwa Österreich den Beistrich als Groschen-Abteiler, während bei US$ hierfür der Punkt Verwendung findet). In der Klasse ist noch eine statische Liste von vordefinierten Währungen enthalten. Diese kann auch verändert werden (Hinzufügen bzw. Entfernen von Währungen, Update des Wechselkurses).

? Price: Hiermit wird ein Preis modelliert, welcher aus einem Betrag und einer Währung besteht. Beim Vergleich von zwei Preisen erfolgt bei unterschiedlichen Währungen eine Umrechnung in die gemeinsame Referenzwährung.

? FIMVoucher: Diese Klasse dient zum Test der Bezahlung mittels Gutscheinen, kann jedoch auch praktischen Einsatz finden. Sowohl die ausgebende Person wie auch der Inhaber werden wiederum als Zertifikat modelliert (hier können etwa AgentIdentity-Objekte oder X.509 Besitzer-Zertfiikate eingesetzt werden). Enthalten ist noch ein Preis (=Wert des Gutscheins) und ein Verfallsdatum, welches bei Erstellung in der Zukunft liegen muß.

4.3.2 Bezahlung für Berechtigungen an das Agentensystem

Im Agentensystem erfolgen einige Einschränkungen, welche jedoch in Subklassen ohne weiteres wieder erweitert werden können. So werden zur Zeit ausschließlich Zahlungen per Kreditkarte und Gutschein (sowie Gratis-Zahlungen falls der Preis 0 ist) akzeptiert. Gutscheine müssen darüber hinaus vom Agentensystem selbst ausgestellt sein. Es muß auch der genaue Preis übergeben werden, d. h. es wird nicht herausgegeben (was bei Kreditkarte ohnehin kein Problem ist).

In Hinsicht auf die Sicherheit wird jedoch vorausgesetzt, daß der Agent alle seine Zahlungselemente auch tatsächlich signiert: Er muß daher die komplexe Form der Identität besitzen (d. h. inklusive öffentlichem Schlüssel) und den dazugehörigen privaten Schlüssel kennen. Spiegelbildlich hierzu signiert auch das Agentensystem seine Rechnungen mit dem eigenen privaten Schlüssel. Für Codefragmente hierzu siehe Anhang.

Der Ablauf des Kaufes einer Berechtigung wird folgendermaßen durchgeführt:

1. Der Agent entscheidet sich für die Berechtigung oder die Gruppe der Berechtigungen, welche er erwerben möchte. Mit diesen fragt er beim Agentensystem an wie hoch der Preis ist.

2. Das Agentensystem überprüft ob der Agent diese Berechtigungen erhalten kann und welcher Preis hierfür zu entrichten ist. Dementsprechend wird ein Rechnung zusammengestellt und signiert. Ist die Berechtigung nicht erhältlich, so ist der Preis unendlich (Double.POSITIVE_INFINITY).

3. Der Agent kann sich nun entscheiden ob er die Berechtigung erwerben möchte oder nicht. Falls ja, so muß er ein Zahlungs-Objekt zusammenstellen, welches die erhaltene Rechnung und die Zahlungsdaten enthält. Dieses ist anschließend zu signieren. Das gesamte Objekt wird dem Agentensystem übergeben.

4. Das Agentensystem prüft nun noch einmal den Preis (er könnte sich in der Zwischenzeit verändert haben) und die Signatur des Agenten. Auch wird festgestellt, ob die vom Agenten gewählte Zahlungsart erlaubt ist.

5. Die Zahlungsdaten werden in einer lokalen Datei gespeichert, sodaß selbst bei einem Systemabsturz die erhaltenen Werte gesichert sind. Hierbei wird das gesamte Objekt in XML-Form gespeichert, sodaß aufgrund der Signatur des Agenten später ein Nachweis möglich ist (über die ebenfalls enthaltenen Berechtigungen und die Signatur des Agentensystems kann auch der Preis überprüft werden).

6. Die entsprechenden Berechtigungen werden dem Agenten zugeteilt und die Kauf-Methode kehrt zurück. Der Agent kann nun die Berechtigungen verwenden.

In Bezug auf die Sicherheit muß hierbei festgestellt werden, daß der Agent nicht vor dem Rechner geschützt werden kann. So wäre es daher denkbar, daß eine Rechnung ausgestellt wird und dann die Signatur des Agenten durch das Agentensystem selbst angebracht wird. Dieses Problem läßt sich zwar nicht umgehen, doch kann es durch den Einsatz von Gutscheinen verringert werden. Der Erwerb von Gutscheinen kann auf einem vertrauenswürdigen dritten Rechner erfolgen, auf welchem das Agentensystem (bei dem der Gutschein verwendet werden soll) durch einen Agenten vertreten ist, der Gutscheine verkauft. Hierfür werden konventionelle Zahlungsmittel wie etwa eine Kreditkarte verwendet. Diese Transaktion ist „gefährlicher“, da hierbei ein fast unlimitierter Betrag auf dem Spiel steht (wenn die Kreditkartennummer bekannt wird oder ein höherer Betrag eingesetzt würde) und erfolgt daher aug diesem sicheren Rechner. Mit diesen Gutscheinen können dann auf dem Zielrechner Berechtigungen erworben werden.

Der Vorteil der Verwendung von Gutscheinen gegenüber anderen Zahlungsarten ist, daß diese Gutscheine für andere Agenten wertlos sind, da sie auf einen bestimmten Agenten ausgestellt

sind: Ein anderer Agent oder ein anderes Agentensystem können sie *nicht* verwenden. Hierzu wäre es notwendig auch den privaten Schlüssel des Besitzers zu kennen (welcher jedoch niemals mit einem Agenten mitgeschickt wird), der den Anfangszustand des Agenten signiert hat. Diese Signatur kann bei einem veränderten Agenten nicht mehr wiederhergestellt werden, sodaß ein anderer Agent mit einer kopierten Identität die Gutscheine ebenfalls nicht einlösen kann. Der Vorteil der Verwendung von Gutscheinen auf dem Zielrechner ist, daß hiermit zumindest eine Risiko-Minimierung möglich ist: Mehr als die mitgeführten Gutscheine kann nicht verloren gehen. Daher sollte diese Art der Bezahlung vorgezogen werden.

Ein weiterer Vorteil der Bezahlung mit Gutscheinen ist, daß in Verbindung mit den Methoden zur Beweissicherung eine lückenlose Nachverfolgung möglich ist. Durch den Nachweis des empfangenen Zustandes sowie des Zustandes beim Verlassen des Agentensystems (inklusive des Read-only-Speichers) kann Gewißheit über den Verbleib bzw. die Verwendung der Gutscheine erlangt werden (zum Großteil bzw. in den meisten Fällen auch über die Korrektheit deren Verwendung). Im Gegensatz hierzu ist dies bei Zahlung durch Kreditkarte oder etwa Abbuchung nicht möglich, da auch jederzeit später noch eine Verwendung möglich ist. Gutscheine hingegen können ausschließlich während des lokalen Aufenthalts eingesetzt werden und sind daher nach Verlassen des Servers wieder „sicher“.

4.3.3 Bezahlung zwischen Agenten untereinander

Die Bezahlung zwischen Agenten untereinander wurde nicht im Framework selbst implementiert, sondern ist Teil des Sport-Portales. Die Implementierung ist jedoch unabhängig davon und kann überallhin unverändert übernommen werden.

Hier wird davon ausgegangen, daß der Käufer mit einem Wunsch an den Verkäufer herantritt. Ist eine umgekehrte Rollenverteilung erwünscht, so muß der Inhalt des Angebots verändert werden: Anstatt Ware gegen Geld zu erwerben, muß die Formulierung „Geld gegen Warenlieferung“ zum Ausdruck bringen (In diesem Fall ist die Reihenfolge Lieferung – Zahlung genau entgegengesetzt der Bezeichnung: Lieferung ist Geld und Zahlung ist Übergabe der Ware).

Das Protokoll zum Kauf von Dienstleistungen/Produkten ist generisch und muß durch Subprotokolle konfiguriert werden. Einzustellen sind die folgenden Elemente:

1. Identifizierungs-Protokolle: Auf welche Weise kann sich der Partner identifizieren?
2. Lieferungs-Protokolle: Wie kann die Lieferung erfolgen?

3. Zahlungs-Protokolle: Welche Zahlungsarten werden akzeptiert?
4. Abfolge Zahlung – Lieferung (Vorleistungspflicht): Muß der Verkäufer zunächst die Ware liefern und wird anschließend der Preis übergeben, oder umgekehrt?
5. Inhalts-Verhandlung (inklusive Preis): Was soll zu welchem Preis verkauft werden?

Hierdurch kann das gemeinsame (und daher auch wahrscheinlich allen Agenten bekannte) Protokoll den Wünschen und Besonderheiten der konkreten Fälle angepaßt werden. Durch die Anführung mehrerer Protokolle können sowohl Spezial-Protokolle Verwendung finden (falls der Partner diese versteht), doch ist immer noch ein Rückzug auf allgemeine Elemente möglich. Es können daher jederzeit zusätzliche neue Protokolle entworfen werden, welche mit bestehenden Agenten eingesetzt werden können, ohne daß eine Neukompilierung erforderlich ist. Im Gegenteil, sogar eine dynamische Konfiguration zur Laufzeit ist möglich.

4.3.3.1 Das Kauf-Protokoll (Klasse SellProtocol)

Das Kauf-Protokoll läuft nach folgendem Schema ab. Hierbei ist zu beachten daß bei Fehlschlagen eines Schrittes das Protokoll abgebrochen wird (Ausnahmen sind extra erwähnt), wobei nach Möglichkeit hiervon auch der andere Partner informiert wird.

1. Zuerst wird dem Partner ein Identifizierungs-Protokoll vorgeschlagen. Dieses kann er entweder annehmen oder ablehnen. Bei einer Ablehnung wird das Nächste vorgeschlagen, bis keines mehr vorhanden ist. Ansonsten wird das gewählte Protokoll durchgeführt. Ein direkter Start eines Identifizierungs-Protokolls ist nicht möglich, da verschiedene davon mit der selben Nachricht beginnen könnten und daher eine eindeutige Identifizierung des vom anderen Partner gestarteten Subprotokolls nicht mehr möglich wäre.
2. Anschließend wird das Subprotokoll zur Vereinbarung des Gegenstandes und des Preises gestartet.
3. Nun werden die Präferenzen bezüglich der Vorleistungspflicht ausgetauscht. Da die Berechnung symmetrisch ist, erfolgt kein abwechselnder sondern ein gleichzeitiger Versand. Verlangt eine Seite eine bestimmte Reihenfolge so setzt sich diese gegenüber einer Präferenz durch. Bei widersprechenden Verlangen wird das Protokoll abgebrochen. Widersprechend sich hingegen nur Präferenzen, so ist das Ergebnis der Wunsch des Verkäufers.
4. Anschließend wird eine Liste der verfügbaren/akzeptablen Lieferprotokolle vom Verkäufer an den Käufer geschickt. Dieser kann dann auswählen. Dies erfolgt in einer eigenen Me-

thode sodaß Subprotokolle hier eines besondere Startegie bzw. Präferenz einbauen können. Die Standardimplementierung wählt einfach das erste verfügbare gemeinsame Protokoll aus.

5. Analog zur Lieferung wird auch die Bezahlung vereinbart. Diese erfolgt nach der Lieferprotokoll-Vereinbarung um die verfügbaren Zahlungsprotokolle bei Bedarf an die gewählte Lieferung anpassen zu können (so ist beispielsweise Nachnahme nur in Verbindung mit Postzustellung möglich).
6. Zu diesem Zeitpunkt sind alle Elemente der Transaktion bekannt und vereinbart aber bis auf die Identifizierung noch nicht durchgeführt. Zur Bestätigung erhält der Käufer eine zusammenfassende Nachricht über die gewählten Punkte: Inhalt, Reihenfolge, Lieferung, und Zahlung. Diese wird vom Verkäufer signiert. Der Käufer hat nun eine letzte Möglichkeit sich zu entscheiden ob er diese Transaktion durchführen möchte. Falls ja muß er die Nachricht selbst signieren und an den Verkäufer zurückschicken, worauf der Austausch initiiert wird. Falls nicht kann der Käufer über den weiteren Verlauf des Protokolls entscheiden. Er kann dieses entweder abbrechen oder zu einem früheren Zeitpunkt zurückkehren um neue Bedingungen zu vereinbaren. Eine Rückkehr ist zur Vereinbarung der Vorleistungspflicht, der Lieferung, sowie der Zahlung möglich. Die Vereinbarung des Gegenstandes wurde hier ausgespart, da diese ja selbst in Form einer komplexen Verhandlung erfolgt.
7. Je nach der gewählten Reihenfolge werden nun Lieferungs- bzw. Zahlungs-Protokoll (und anschließend das jeweils andere Protokoll) gestartet.

Durch die signierte (und gegengezeichnete) Nachricht haben beide Seiten einen Nachweis was mit dem anderen Partner vereinbart wurde. Da es sich um zwei getrennte Agenten handelt hat aufgrund des Sicherheitssystems keiner die Möglichkeit des Zugriffs auf den anderen, sodaß diese Daten fälschungssicher sind. Tritt daher bei der tatsächlichen Durchführung ein Problem auf (z. B. keine physikalische Lieferung jedoch Abbuchung des Preises), so kann jede Seite hiermit Beweis für die Zustimmung des Partners führen.

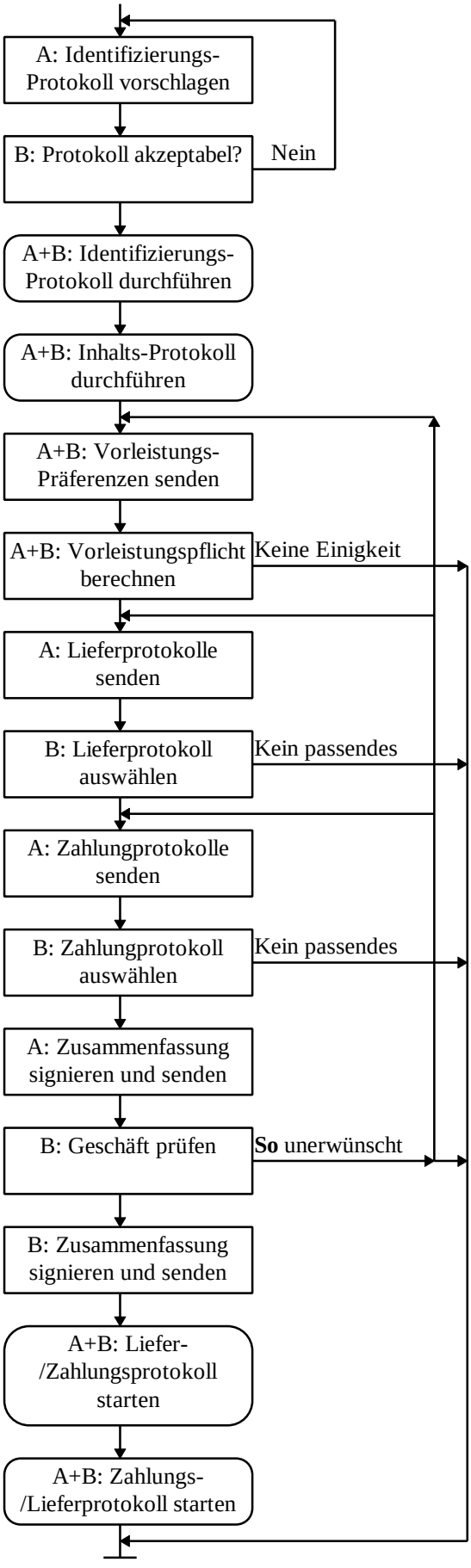

Abbildung 50: Ablauf des Kauf-Protokolls

4.3.3.2 Inhalts-Verhandlung (Klasse ContentNegotiationProtocol)

Für den Gegenstand der Transaktion verantwortlich ist das Protokoll zur Inhalts-Verhandlung. Dieses muß je nach der Ware angepaßt werden. Die Verhandlung erfolgt derart, daß der Käufer zuerst ein Angebot macht, welche Ware er um welchen Preis erwerben möchte. Anschließend erfolgt immer abwechselnd ein Gegenangebot bis entweder eine Seite die Verhandlung abbricht oder eine Übereinstimmung erzielt wurde (wenn das Gegenangebot dem Angebot entspricht). Im ersten Fall wird der Verkauf abgebrochen, ansonsten fortgeführt.

Sowohl Preis als auch Ware sind jeweils in einem Angebot enthalten. Wichtig festzustellen ist, daß kein Format festgelegt ist, in welcher Weise der Inhalt beschrieben wird oder nach welchen Regeln die Verhandlung verläuft. Lediglich das Grobkonzept abwechselnder Äußerungen ist festgelegt.

Ein besonderer Punkt ist noch, daß dieses Protokoll auch dafür zuständig ist den Gegenstand schlußendlich zur Verfügung zu stellen. Dieser wird unverändert an das Lieferprotokoll weitergegeben, welches damit umgehen können muß.

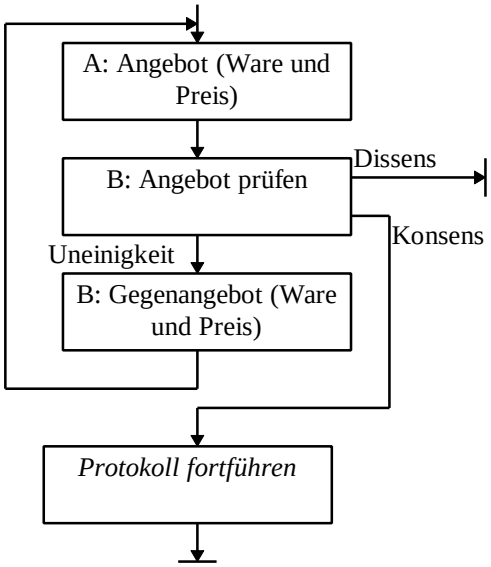

Abbildung 51: Ablauf der Inhalts-Verhandlung

4.3.3.3 Beispiel für die Konfiguration eines Kauf-Protokolles

Im folgenden wird die Konfiguration eines Kauf-Protokolles im Programmcode dargestellt. Dieser Teil ist sowohl auf der Käufer wie auch auf der Verkäufer-Seite erforderlich (mit Ausnahme der Konfiguration der Daten der Kreditkarte).

Die gewünschte Transaktion umfaßt die folgenden Elemente:

1. Zahlung vor Lieferung wird verlangt, nicht nur gewünscht
2. Identifizierung der Partners über signierte Daten oder Paßwörter erlaubt
3. Lieferung erfolgt entweder per Post oder direkt in XML-Form
4. Zahlung entweder per Kreditkarte, Nachnahme, Daten oder Gutschein oder ist Gratis
5. Die Signierung der Rechnung bzw. des Zahlungs-Objektes erfolgt unter der Identität des Agenten und mit dessen privatem Schlüssel
6. Der Inhalt der Transaktion wird durch das Inhalts-Protokoll (“ContentProtocol “) festgelegt, welches gesondert implementiert ist.

```
// Identifikations-Protokolle
IdentificationProtocol[] idProts=new IdentificationProtocol[2];
idProts[0]= new SignDataIdentificationProtocol(this);
idProts[1]= new PassphraseIdentificationProtocol(this);

// Zustell-Protokolle
DeliveryProtocol[] delProts=new DeliveryProtocol[2];
delProts[0]=new XMLDocDeliveryProtocol(this);
delProts[1]=new PostDeliveryProtocol(this);

// Zahlungs-Protokolle
PaymentProtocol[] payProts=new PaymentProtocol[5];
    // Kreditkartenzahlung
    Calendar cal=Calendar.getInstance();
    cal.setTime(new Date());
    cal.add(Calendar.YEAR,1);
    byte[] number={4,0,0,0,0,8,1,5,4,7,1,1,0,3,3,3,3};
payProts[0]=new CreditCardPaymentProtocol(this,"ACME-Card",
          "Jane Doe",number,cal.getTime());
    // Zahlung er Nachnahme
payProts[1]=new CODPaymentProtocol(this);
    // Zahlung mittels Daten
    DataWrapper[] data=new DataWrapper[1];
    data[0]=new DataWrapper("Nothing","-",
          new Price(Currency.findCurrency("EUR"),1.0));
payProts[2]=new DataPaymentProtocol(this,data);
    // Keine Zahlung
payProts[3]=new NoPaymentProtocol(this);
    // Zahlung mittels Gutschein
    VoucherWrapper[] vouchers=new VoucherWrapper[1];
    vouchers[0]=new VoucherWrapper("Special Discount",null,
          new Price(Currency.findCurrency("EUR"),1.0));
payProts[4]=new VoucherPaymentProtocol(this,vouchers);

// Kauf-Protokoll erzeugen
SellProtocol sellProt=new SellProtocol(this,idProts,
     new ContentProtocol(this),delProts,payProts,
     new NamedKeyAndCertificate("Sell-Protocol",
     getKeyPair("Agents own keys").getKeyPair().
          getPrivate(),getIdentity()));
// Zahlung vor Lieferung gefordert
sellProt.setPriority(SellProtocol.REQUIRE_PAYMENT_FIRST);

// Registrieren des Protokolles beim Agenten
```

```
registerConversation(sellProt);
```

4.4. Rechtliche Aspekte

Rechtliche Aspekte als solche können in der Software keine Implementierung finden. Es werden daher hier diejenigen Aspekte der Agentensystems POND erläutert, welche rechtliche Bedeutung erlangen können. Insbesondere sind dies Nachweise, daß Agenten zu einem gewissen Zeitpunkt einen bestimmten internen Zustand besaßen. Hieraus läßt sich (zumindest in eingeschränkter Weise) ablesen, wo eine Modifikation des Agenten erfolgte, bzw. ob dieser Endzustand ohne äußere Beeinflussung überhaupt erreichbar ist.

4.4.1 Nachweis der Datenübergabe an Agenten

Zum Nachweis der Übergabe von Daten an einen Agenten dient ein spezieller Speicher, welchen ein Agent implementieren kann. Hierzu dienen die Klassen HasReadOnlyStore (Interface) und ReadOnlyStore (Implementierung). Die Implementierung erfolgt über eine Hashtabelle, sodaß nur eindeutige Schlüssel verwendet werden können. Die Methode zum Entfernen eines Elementes wurde deaktiviert indem eine IllegalStateException ausgeworfen wird. Um auch eine nachträgliche Veränderung von Objekten durch dasjenige Objekt zu verhindern, welches die Daten hier speicherte (und daher eventuell noch eine Referenz darauf besitzt), werden die Daten als Bytearray in serialisierter Form (und daher als unabhängige tiefe Kopie) gespeichert. Dies garantiert die Unverändertheit nach dem ursprünglichen Speichervorgang und sichert gleichzeitig auch die Serialisierbarkeit des gesamten Objektes für etwaige Transfers des umgebenden Agenten. Um auch die Wiederherstellung garantieren zu können wird bei der Speicherung eines Objektes dieses auch temporär wiederhergestellt. Bei einem Zugriff auf das Objekt wird dieses wieder deserialisiert und zurückgegeben. Dies hat zur Konsequenz, daß bei zwei Abfragen auf die selben Daten unterschiedliche Objekte zurückgegeben werden. Ein Vergleich kann daher nur mehr mittels „equals“ durchgeführt werden, nicht mehr durch direkten Vergleich („==“). Dies ist zwar ein Nachteil, garantiert aber, daß eine Veränderung auch durch den Agenten nicht möglich ist (und zwar auch nicht in Sub-Objekten eines gespeicherten Objektes). Als Alternative müßte jedes Objekt auf seine read-only Eigenschaft geprüft werden (z. B. Implementierung eines Marker-Interfaces) und alle enthaltenen Objekte ebenfalls rekursiv untersucht werden (dies könnte über das Reflection API erfolgen; komplex, da es sich nicht um einen Baum handeln muß, sondern ein allgemeiner Graph mit Kreisen vorliegen kann).

Als Erweiterung, in Hinsicht auf die Möglichkeit des Agenten die Übergabe abzulehnen (z. B. da bei großen Datenmengen die Transferzeit steigt), existiert noch das Interface HasVetoable-

ReadOnlyStore. Hier wird noch zusätzlich eine Methode definiert, bei welcher der Agent anhand des Schlüssels, der Länge der Daten, und der Identität des beantragenden Agenten (null beim Agentensystem selbst) entscheiden kann, ob er die Daten annehmen will. Hierfür besteht natürlich keine Garantie, doch zumindest das Agentensystem respektiert derartige Wünsche. Ein Agent, welcher das Agentensystem darum bittet einem anderen Agenten Daten auf diese Weise zu übergeben, erhält diesfalls eine Fehlermeldung zurück.

4.4.2 Übergabenachweis bei Agenten-Transfers

Der „Hand-Off“ beim Transport eines Agenten auf einen anderen Server dient dazu einen Nachweis für die Übergabe des Agenten in einem bestimmten Zustand zu besitzen. Dies erfolgt technisch durch eine erste Signatur durch den Quell-Server, welche zum Ziel-Rechner geschickt wird. Dieser überprüft die Signatur und sendet bei Erfolgt seine eigene Signatur der empfangenen Daten zurück. Wird diese vom Quell-Rechner akzeptiert, so erfolgte der Transfer erfolgreich. Ansonsten wird ein Fehler zurückgegeben. Die Signaturen (sowohl die gesendete als auch die empfangene), die Identität des Agenten, das Zertifikat, sowie der Digest werden vom Agentensystem in einer Datei gespeichert. Dies erfolgt in XML-Form, bzw. falls dies nicht möglich ist (z. B. kein XML-Parser verfügbar oder ein Fehler bei der Umsetzung), als Klartext (in Hexadezimal-Codierung für die Binärdaten).

Dies kann logischerweise erst nach Transfer der aktuellen Daten erfolgen (wie könnte der Empfänger sonst eine Prüfung vornehmen), sodaß ein Angriff in der Weise immer noch möglich ist, den Empfang nicht zu bestätigen, den Agenten aber dennoch zu starten.

Optional ist es noch möglich die Speicherung der kompletten Daten zu veranlassen, sodaß ein voller Beweis über den Zustand im Transferzeitpunkt möglich ist. Dies sollte jedoch nur sparsam verwendet werden, da hierbei große Datenmengen anfallen können. Technisch erfolgt die Realisierung durch eine besondere Funktion im Agentensystem, welche anhand der Identität eines Agenten feststellt, ob dieser bei einem Transfer vollständig zu protokollieren ist. Dies erlaubt eine Auswahl, sodaß eine Beschränkung auf einzelne und wichtige Fälle möglich ist. In Subklassen wäre hier auch eine dynamische Veränderung der Richtlinien möglich, z. B. auf Anforderung des Besitzers des Agentensystems hin, oder etwa als Reaktion bei verdächtigen Aktionen von Agenten.

4.4.3 Nachweis der Anfangs-Konfiguration

Der Nachweis des Anfangszustandes ist in der Abwicklung komplexer, da er nicht zu einem bestimmten Zeitpunkt (z. B. Transfer des Agenten) erfolgen kann. Daher erfolgt die Signierung des Zustandes auf Aktion des Benutzers hin durch Auswählen eines Menüpunktes. Bis dahin kann daher eine ausführliche Konfiguration erfolgen und der Agent auch (z. B. auf internen Servern) transferiert werden. Dies setzt voraus, daß sich der Agent noch auf einem Rechner befindet, welcher Zugriff auf den privaten Schlüssel des Besitzers hat. Die Prüfung ist auch nur dann möglich, wenn der Besitzer eine vollständige Kopie aufgehoben hat bzw. diese wiederherstellen kann (der Code muß daher meist nicht dupliziert werden).

Die Signierung von Agenten, welche nicht vollständig serialisierbar sind, ist hier nicht von Bedeutung, da diese dann ihren Quell-Rechner ohnehin nicht verlassen können. In diesem Fall stellt sich jedoch auch nicht das Beweisproblem und ein Nachweis gegenüber Dritten würde ohnehin nur dann Bedeutung besitzen, wenn eine Gegen-Signatur durch eine unabhängige Stelle erfolgen würde. Andernfalls ist jederzeit eine Fälschung durch andere Daten und eine neue Signatur möglich.

Da nicht nur der Zustand des Agenten (=Anfangskonfiguration) von Bedeutung für etwaige Verträge, Haftung, oder den Beweis des Anfangs ist, sondern auch der Anfangsbestand an Code (Austausch oder Modifikation von Klassen; enthaltene Konstanten, etc.), muß auch dieser vom Benutzer signiert werden. Die Signatur durch den Programmierer alleine ist hier nicht ausreichend, da sonst z. B. eine ältere Version des Codes (mit bestimmten noch enthaltenen Fehlern) substituiert werden könnte. Es ist jedoch nicht erforderlich den gesamten Code zu signieren. Die Signatur der Code-Signatur alleine ist ausreichend (da diese wiederum eindeutig den Code festlegt sofern sie korrekt/überprüfbar ist und den gesamten Code betrifft).

Zu berücksichtigen bei der Signatur ist, daß alle Daten signiert werden. Dies bedeutet, daß auch andere (für die Überprüfung unwesentliche) Daten enthalten sind. Bei einer Überprüfung durch andere Agenten bzw. Personen kann dies zu Schwierigkeiten führen. Um die Prüfung durchführen zu können, muß der gesamte Agent in diesem Zustand übergeben werden (was daher auch diese zusätzlichen Informationen beinhaltet). Hierbei könnte es sich jedoch z. B. um Kreditkartendaten handeln. Ein Agent wird losgeschickt einen bestimmten Artikel zu kaufen: Nicht nur die für die Überprüfung wesentliche Beschreibung des Produktes, sondern auch die Zahlungsdaten sind hier enthalten und müssen bekanntgegeben werden. Eine Alternative hierzu wäre geheimzuhaltende Daten erst nach der Signatur in den Agenten einzubringen oder eine

derartige Nachprüfung nur vertrauenswürdige dritte Stellen durchführen zu lassen auf deren Geheimhaltung vertraut wird.

Um das vorher dargestellte Problem zu lösen besteht noch eine weitere Möglichkeit. Analog zu der Markierung von Klassen als serialisierbar durch die Implementierung des (leeren) Interfaces java.io.Serializable könnte ein weiteres Interface Verwendung finden. Alle derartigen Objekte würden dann in die Signatur miteinbezogen. Die Implementierung könnte über das Reflection-API erfolgen, da hiermit zusätzlich zu den Elementen auch alle enthaltenen Referenzen eines Objektes identifiziert werden könnten. Dies müßte jedoch rekursiv erfolgen und es stellen sich weitere Fragen: Ist ein Sub-Objekt derart markiert ist dann das (nicht-markierte) Eltern-Objekt auch zu signieren oder nicht? Was ist mit nicht-markierten Sub-Objekten eines Objektes? Eine Vereinfachung auf ein bestimmtes Objekt (z. B. über eine Methode identifizierbar) wäre daher auf jeden Fall sinnvoll und dann eine wünschenswerte Erweiterung (was nebenbei auch zur Beschleunigung führen würde).

Die Durchführung der Signaturen erfolgt über das Agentensystem auf Anforderung des Benutzers durch ein Menü hin. Es wird der gesamte Zustand des Agenten serialisiert und mit dem privaten Schlüssel des Besitzers verschlüsselt. Anschließend wird das Manifest des Codes (darin enthalten sind die Signaturen der einzelnen Dateien) ebenso signiert. Beide Signaturen sind Teil der Identität des Agenten und werden daher bei Transfers mitübertragen, sodaß zumindest bei der Code-Signatur der entgegennehmende Host diese auch überprüfen kann. Eine Signierung nur eines Teils des Zustandes ist in der derzeitigen Implementierung nicht möglich, könnte aber durch Subklassen jederzeit eingeführt werden.

4.5. Weitere Elemente des Agentensystems

In diesem Abschnitt werden sonstige Teile des Agentensystems beschrieben, welche von Bedeutung sind, aber nicht zum Hauptteil der Arbeit gehören. Hierzu zählt insbesondere die Kommunikation zwischen Agenten untereinander sowie zwischen Agenten und Benutzern, sowie Mobilität.

4.5.1 Kommunikation zwischen Agenten

Die Kommunikation zwischen Agenten erfolgt ausschließlich über Nachrichten, welche über das Agentensystem ausgetauscht werden. Ein gegenseitiger Aufruf von Methoden ist wegen des Sicherheitssystems nicht möglich und auch nicht erwünscht. Als Alternative besteht nur die

Möglichkeit über gemeinsam zugängliche Dateien oder über eine Socket-Verbindung miteinander zu kommunizieren (vorausgesetzt beide Agenten sind hierzu berechtigt). Die Kommunikation über Nachrichten besitzt einen besonderen Vorteil: Es können jederzeit beliebig viele Kommunikationsvorgänge mit anderen Partnern überlappend ausgeführt werden. Hierbei kann es sich sowohl um verschiedene Kommunikationen mit verschiedenen Agenten handeln, als auch um gleichartige Kommunikationen mit unterschiedlichen Agenten, bis hin zu mehreren gleichen Kommunikationen mit einem einzelnen Agenten. Dies setzt aber auch voraus, daß der Agent während der Verarbeitung einer Nachricht keine langen Berechnungen vornimmt, sondern die Kontrolle bald wieder zurückgibt. Ist eine zeitaufwendige Arbeit erforderlich, so sollte dies in einem eigenen Thread erfolgen. Ein solcher Kommunikationsvorgang wird im Folgenden als Konversation bezeichnet (analog dazu siehe die Klassen Conversation bzw. Protocol).

Für die Kommunikationsmuster ist zu beachten, daß nur lokale Kommunikation innerhalb eines Agentensystems möglich ist (Lokalitätsprinzip). Soll eine Nachricht an einen Agenten in einem anderen System verschickt werden (grundsätzliches Problem: Dessen Identität und Ort muß bekannt sein), so muß hierzu ein Agent in das andere System geschickt werden, welcher die Nachricht dann am Zielrechner lokal absendet.

4.5.1.1 Nachrichten

Eine Nachricht ist immer von einem einzelnen Agenten an einen ganz bestimmten anderen Agenten gerichtet. Die Identität beider Agenten ist in jeder Nachricht in Form zweier AgentIdentity-Objekte enthalten. Jede Kommunikationsverbindung (i. e. Conversation bzw. Protocol) ist durch eine eindeutige Nummer gekennzeichnet. Da diese auf beiden Seiten unterschiedlich sein kann, wird auch sie für jeden Agenten getrennt mitgeführt. Ebenso wird der Typ der Conversation gespeichert. Um an Hand einer Nachricht alleine entscheiden zu können welcher Agent welche Rolle in der Kommunikationsbeziehung übernimmt, wird auch gespeichert, ob dies eine Nachricht vom Initiator der Kommunikation ist oder nicht.

?? Konversations-Typ
?? Absender
?? Konversations-ID (Eigene)
?? Adressat
?? Konversations-ID (Partner)
?? Vom Initiator der Konversation (Richtung)
?? *Inhalt (Serialisierbar)*

Abbildung 52: Elemente einer Nachricht

Der eigentliche Inhalt jeder Nachricht muß serialisierbar sein, da ansonsten eine Zustellung nicht möglich ist (siehe oben; zum Wechsel der Classloader-Sicherheitsbereiche muß eine Umwandlung durchgeführt werden). Der Inhalt ist in konkreten Subklassen zu definieren. Dient eine Nachricht lediglich als Flag, so kann auch ein leerer Inhalt verwendet werden: Allein der Empfang einer Nachricht eines bestimmten Typs kann dem Empfänger schon als Information genügen (z. B. Befehl zum Abbruch einer Aktion).

Bei diesen Nachrichten handelt es sich um ungesicherte Informationen: Zwar überprüft das Agentensystem, ob der Absender einer Nachricht auch tatsächlich der Agent ist, der sie verschickt, doch ist dies keine Garantie für den Inhalt. Mit der Klasse SignedMessage ist ein abweichendes Konzept möglich: Der Absender kann die Nachricht digital signieren und so die Korrektheit und seine Absendereigenschaft garantieren. Es handelt sich hierbei um eine Wrapper-Nachricht, welche jede beliebige andere Nachricht als Inhalt haben kann: Die notwendigen Informationen wie z. B. der Adressat werden dieser entnommen. Beim Empfang ist jedoch darauf zu achten, daß das Auspacken durch den Empfangsagenten selbst erfolgen muß: Andernfalls würden die Signatur-Informationen verlorengehen. Bei der Wiederherstellung der Inhalts-Nachricht (welche als Byte-Array gespeichert wird) muß darauf geachtet werden den richtigen Classloader (=des Empfänger-Agenten) zu verwenden. Die Wiederherstellung erfolgt durch das Agentensystem und daher wäre die Nachricht für den Agenten sonst nicht mehr zugänglich. Daher ist der Inhalt explizit unter Verwendung eines anderen -Classloaders zu deserialisieren.

Ist der Empfänger einer Nachricht zur Zeit des Empfangs auf die Festplatte persistiert (siehe dazu Näheres unter 4.5.2) so wird er wieder in den Speicher geladen und ihm die Nachricht anschließend zugestellt. Nach Verarbeitung der Nachricht bleibt er in diesem Zustand, wird also nicht wieder automatisch persistiert.

Soll eine Nachricht unabhängig von einer Conversation oder einem Protokoll verschickt werden, so sind die folgenden notwendigen Elemente selbst zu setzen (entweder über den Konstruktor oder über die separaten Methoden): Absender, Adressat, IDs der Konversationen des Senders und des Empfängers, sowie eine Konversations-Typen-ID. Bei Versand im Rahmen einer Konversation bzw. eines Protokolls werden alle diese Werte automatisch eingetragen. In diesem Fall kann der Konstruktor ohne Parameter verwendet werden. Die angeführten Werte können nur ein einziges Mal gesetzt werden. Spätere Änderungen sind aus Sicherheitsgründen nicht mehr möglich, lediglich eine Abfrage dieser Werte.

Im Agentensystem sind einige gebräuchliche bzw. mit besonderer Bedeutung versehene Nachrichten (z. B. Akzeptieren/Ablehnen eines Angebots, oder einfaches Ergebnis als String) schon vordefiniert. Es existieren jedoch auch einige mit spezieller Bedeutung die hier kurz erläutert werden:

? TerminateAgentMessage: Diese Nachricht dient dazu einem Agenten mitzuteilen, daß er sich selbst beenden soll. Um Mißbräuchen vorzubeugen akzeptiert die Standard-Implementation in TerminateAgentConversation diese Nachricht nur von sich selbst (z. B. aus einem anderen Thread). Durch Einbindung einer unterschiedlichen Implementation können weitere oder andere Prüfungen durchgeführt werden wie beispielsweise ein Vergleich der Identität des Besitzer-Zertifikates des Absenders der Nachricht mit dem Besitzer dieses Agenten.
? ReplyToLateMessage: Diese Nachricht dient als Antwort auf eine Nachricht, welche zu spät eingelangt. Ist eine Konversation bereits beendet (jedoch archiviert), so wird diese Nachricht zurückgeschickt. Die Konversation ist zwar noch verfügbar aber vermutlich nicht im richtigen Zustand um diese Nachricht noch verarbeiten zu können. Sie kann jedoch auch unabhängig hiervon verwendet werden, wenn eine Antwort auf eine Nachricht zu spät einlangt. Die zu späte Nachricht ist als Datum enthalten.
? UnknownConversationMessage: Mit dieser Nachricht wird der Sender einer Nachricht darauf hingewiesen, daß die laufende Konversations-Nummer ungültig ist. Dies kann einerseits davon herrühren, daß eine falsche Nummer eingetragen wurde, oder dadurch entstehen daß die Konversation beendet aber nicht archiviert wurde. Auch hier ist die falsche Nachricht wieder enthalten.
? UnknownConversationTypeMessage: Hierbei handelt es sich um einen schwerwiegenden Fehler: Die Konversations-Typennummer ist unbekannt. Es wurde daher entweder eine falsche eingetragen oder der Agent ist grundsätzlich nicht in der Lage diese Art von Konversation zu führen. Wiederum ist die falsche Nachricht enthalten.

Sollte es sich in den letzten drei Fällen bei der ursprünglichen Nachricht um einen Broadcast handeln, so wird keine derartige Nachricht gesendet, sondern der Broadcast einfach ignoriert: Ein Agent der einen Broadcast aussendet würde sonst mit derartigen (in diesem Fall auch meist sinnlosen) Rückmeldungen überflutet werden, da naturgemäß nur ein Teil der existierenden Agenten diesen verstehen kann (und er auch nur diese betrifft) bzw. die Rückmeldung sinnlos ist.

Zur Zeit werden die drei Fehler-Nachrichten beim Empfang ignoriert. Im Debug-Modus wird eine Meldung ausgegeben. Dieses Verhalten kann in Subklassen geändert werden. Hierbei ist jedoch darauf zu achten, daß keine Endlosschleifen entstehen. Wird etwa eine ReplyToLate Message abgeschickt und ist zwischenzeitlich auch die Konversation beim Absender der Nachricht beendet, so darf auf die ReplyToLate-Nachricht nicht wieder eine ReplyToLate-Nachricht zurückgeschickt werden.

4.5.1.2 Broadcasts

Ein Broadcast enthält die gleichen Daten wie eine gerichtete Nachricht wobei jedoch die Addressaten-Informationen erst beim Empfang eingesetzt werden (und daher bei jedem Empfänger korrekterweise die eigenen Werte enthalten sind). Im Gegensatz zu gerichteten Nachrichten existiert bei Broadcasts beim Senden kein Empfänger, da sie an alle lokalen Agenten verschickt werden. Sollte im Zeitpunkt des Versandes ein Agent persistiert sein, so wird er übergangen. Im Gegensatz zu einer direkten Nachricht wird er daher nicht depersistiert und der Broadcast ist für ihn verloren. Dies hat insbesondere den Grund, daß ansonsten die Persistenz kein große Bedeutung hätte: Es ist immer wieder mit Broadcasts zu rechnen, sodaß diese Agenten sich dauernd selbst wieder-persistieren müßten. Ein Agent muß sich daher entscheiden, ob er Broadcasts empfangen will (keine Deaktivierung) oder nicht.

Vom Sender gesetzt:
?? Konversations-Typ
?? Absender
?? Konversations-ID (Eigene)
?? Vom Initiator der Konversation (Richtung)
?? *Inhalt (Serialisierbar)*
Vom Empfänger gesetzt:
?? Adressat
?? Konversations-ID (Partner)

Abbildung 53: Setzen der Elemente eines Broadcasts

Broadcasts spielen im Agentensystem eine bedeutende Rolle, da keine für Agenten zugängliche zentrale Registrierung von Agenten mit ihren Fähigkeiten und Eigenschaften existiert. Möchte daher ein Agent einen Anderen ausfindig machen, um mit ihm eine Kommunikation aufzubauen, so kann dies nur über einen Broadcast erfolgen. Alle interessierten Agenten (eine Untermenge derer die den Broadcast verstehen; andere Agenten ignorieren ihn einfach) können daraufhin eine *gerichtete* Nachricht an den Absender schicken, welcher anschließend aus den Anmeldungen seine Auswahl trifft.

Soll ein Broadcast unabhängig von einer Konversation verschickt werden, so sind der Absender, die Konversationsnummer des Absenders, und der Konversations-Typ direkt anzugeben. Ansonsten werden auch hier alle diese Informationen automatisch eingesetzt.

4.5.1.3 Implementierung

Bei der Implementierung besteht bei den Nachrichten eine Besonderheit: Sowohl gerichtete Nachrichten als auch Broadcasts haben viele Elemente gemeinsam (z. B. Absender, aber auch Adressat). Sie sollten daher eine gemeinsame Basisklasse besitzen. Darüber hinaus sollte es jedoch auch möglich sein später weitere Subklassen zu erzeugen, welche andere Eigenschaften gemeinsam haben. Ein Beispiel hierfür sind die ebXML-Nachrichten. Daher kann für die gemeinsamen Elemente keine gemeinsame Basisklasse verwendet werden, da ansonsten später die Hierarchie verändert werden müßte, um zusätzliche Gemeinsamkeiten einzufügen. Es werden daher die Gemeinsamkeiten in einem Interface spezifiziert, welches sowohl von Nachrichten wie auch von Broadcasts implementiert wird. Dies hat allerdings den Nachteil, daß keine Implementierungs-Vererbung möglich ist (Interfaces enthalten nur Deklarationen aber keinen Code). Dies vermeidet auch das Problem, daß eine Klasse von der anderen abgeleitet werden müßte (entweder ist ein Broadcast eine Subklasse von Message oder umgekehrt; von der Semantik her gesehen kein gute Implementierung). Daraus ergibt sich die in Abbildung 54 dargestellte Hierarchie wobei ebXMLMessageType ein Sub-Interface von MessageType ist und von den neuen Nachrichtentypen jeweils implementiert wird (welche wiederum Subklassen der „normalen“ Nachrichtenimplementationen sind).

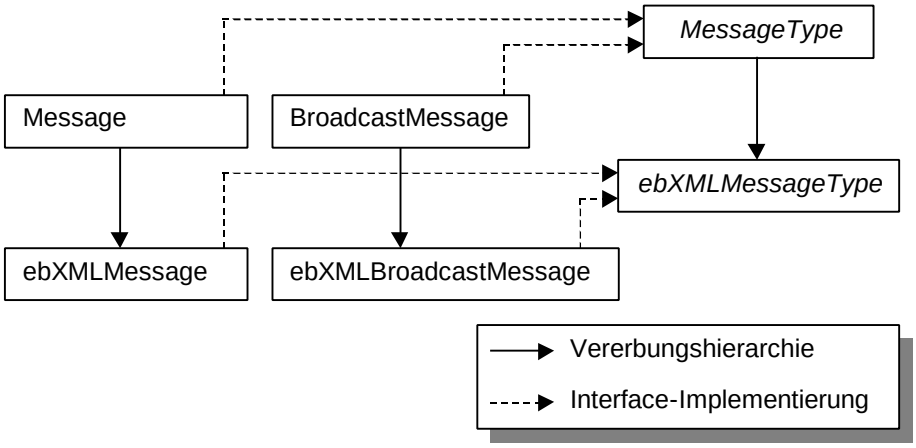

Abbildung 54: Klassenhierarchie von Nachrichten und Broadcasts

4.5.1.4 Konversationen

Zur Erleichterung der Kommunikation mit anderen Agenten existieren sogenannte „Konversationen“. Hierbei handelt es sich um einen Austausch von Nachrichten mit (meistens) einem einzelnen anderen Agenten. Im Rahmen einer Konversation werden zu versendende Nachrichten selbständig mit den notwendigen Parametern gefüllt und empfangene Nachrichten automatisch an das passende Objekt zur Behandlung durchgereicht. Eine Konversation kann auch als ein zusammengehöriger Kommunikations-Vorgang beschrieben werden. Die Kapselung ist erforderlich, da gleichartige Nachrichten-Austausche zum selben Zeitpunkt mit verschiedenen Agenten möglich sind. Ebenso wie unterschiedliche Kommunikationen mit dem selben (oder anderen) Agenten. Dies setzt voraus, daß ein Kommunikationsvorgang nicht blockierend implementiert ist, sodaß die Behandlung von eintreffenden Nachrichten auch verzahnt erfolgen kann. Hieraus ergibt sich, daß jede Konversation einen internen Zustand benötigt, welcher bei Eintreffen einer Nachricht (oder durch Zeitablauf) geändert wird.

Ein Konversationsmuster (entspricht meist einem Objekt, kann aber z. B. auch in einen Client- und einen Server-Teil aufgespalten werden) ist durch eine eindeutige Nummer gekennzeichnet. Hierbei stellt sich das Problem, daß bei einem offenen System diese theoretisch weltweit eindeutig sein müßte. Da dies nicht garantiert werden kann, muß zusätzlich überprüft werden, ob eine empfangene Nachricht verstanden werden kann (falls nicht ist es wohl eine falsche Konversation) und auch zu dieser Nachricht paßt (sonst handelt es sich um ein leichte Abart). Agenten müssen daher damit rechnen, daß auch falsche bzw. unpassende Nachrichten empfangen werden und in der Lage sein damit umzugehen (was meistens sinnvollerweise durch einfaches Ignorieren erfolgen kann). Anhand dieser Nummer wird auch beim Empfang einer Nachricht, welche noch nicht zu einer Konversation gehört (initiale Nachricht), festgestellt, von welcher Konversations-Klasse sie behandelt werden soll. Hierzu sind alle Konversationen, welche ein Agent kennt, bei diesem mit einem leeren Prototyp zu registrieren. Wird eine entsprechende Nachricht empfangen, so wird eine Kopie des Prototypen erstellt, die entsprechenden Felder ausgefüllt (z. B. Partner), und die Nachricht zur Behandlung übergeben. In Java ist zwar jede Klasse mit einer eindeutigen Nummer versehen (welche aus der Schnittstelle der Klasse generiert wird), doch kann diese keine Verwendung finden. Eine kleine Schnittstellen-Änderung würde eine unterschiedliche Nummer bewirken, sodaß mit früheren Implementierungen keine Kommunikation mehr möglich wäre. Dadurch wären auch Parallel-Implementationen (z. B. durch einen anderen Anbieter) unmöglich, ebenso wie eine Trennung

in verschiedene Objekte (wie oben etwa in Client und Server). Diese Nummer ist folgendermaßen spezifiziert:

```
public final static long conversationTypeID=……;
```

Eine identische Zeile mit einer anderen Nummer ist in jeder Konversations-Klasse einzubauen. Da es sich um eine statische Konstante handelt, ist ein Überschreiben nicht möglich. Auf sie kann auch ohne aktuelles Objekt zugegriffen werden (z. B. notwendig bei der Erzeugung einer neuen Konversation; diese wird durch die Typen-Nummer spezifiziert). Die Überprüfung des Wertes eines konkreten Objekts erfolgt über das Reflection-API, da ansonsten immer der Wert der Klasse, in welcher die Prüfung durchgeführt wird, zurückgegeben würde.

Jedes Konversations-Objekt ist auch durch eine (nur für diesen Agenten und diesen Konversations-Typ) eindeutige Nummer gekennzeichnet. Hiermit werden gleichartige Konversationen (u. U. sogar mit dem gleichen Partner) voneinander unterschieden.

In der Klasse Conversation (Implementierung einer einfachen Konversation) werden die folgenden Werte gespeichert:

- Identität des Kommunikationspartners: Diese ist schon bei der Erzeugung der Konversation anzugeben. Handelt es sich um eine Konversation ohne bestimmten Partner (z. B. Aussenden eines Broadcasts um geeignete Partner zu finden), so bleibt dieser Wert null.
- Laufende Konversationsnummer des Partners: Die eindeutige Nummer dieses Objektes des Typs der Konversation. Sie wird automatisch von der Konversation vergeben.
- Laufende Konversationsnummer: Hier wird die eigene laufende Nummer gespeichert.
- Referenz zum Agenten: Um in der Konversation auf den Agenten, zu dem sie gehört, zugreifen zu können.
- Einzelner Partner: Ob es sich bei dieser Kommunikation um eine Konversation mit einem einzelnen Partner handelt oder nicht. Dies wird benötigt um zu entscheiden, ob beim Empfang der ersten Nachricht die laufende Konversationsnummer des anderen Agenten lokal eingetragen werden soll oder nicht. Erfolgte dies bei mehreren Partnern, so würde beim Empfang der zweiten Nachricht eine Exception ausgeworfen werden, da die laufende Nummer sich von der bereits durch die erste Antwort eingetragenen unterscheidet.
- Initiator: Ob dieser Agent der Initiator der Konversation ist. Hierdurch kann die Rollenverteilung geprüft werden und jeder Agent feststellen, welchen Part er durchzuführen hat.

? Archivierung: Durch dieses Flag wird gekennzeichnet, daß die Konversation nach ihrem Ende gespeichert werden soll. Hierdurch ist auch nach deren Ende noch ein Zugriff auf interne Daten möglich. Andernfalls wird das Objekt freigegeben und ist verloren.

Das Senden einer gerichteten Nachricht oder eines Broadcasts erfolgt durch einfachen Aufruf der entsprechenden Methoden einer Konversation, in denen zuerst die notwendigen Werte (Absender, Adressat, etc.) eingesetzt werden. Daraufhin werden die entsprechenden Methoden im Agenten selbst aufgerufen (welche nur ein paar Fehlerprüfungen vornehmen und die tatsächliche Verteilung dann durch das Agentensystem durchführen lassen).

Im Gegensatz zum Absenden von Nachrichten ist der Empfang etwas komplexer. Hierzu dient die Methode handleMessage. In Subklassen muß diese überschrieben werden, um die tatsächliche Behandlung durchzuführen. Es ist dabei darauf zu achten zuerst die Methode der Superklasse aufzurufen. Liefert diese als Rückgabewert „Wahr“ zurück, so wurde die Nachricht bereits weiter oben in der Hierarchie behandelt und es sollten keine weiteren Aktionen mehr stattfinden. Ansonsten wird überprüft, ob die Nachricht auch tatsächlich von dem Partner dieser Konversation stammt (nur bei eindeutigem Partner), sowie ob die laufende Konversationsnummer korrekt ist. Eine inhaltliche Behandlung findet jedoch nicht statt.

Eine Hilfs-Methode zur Erzeugung einer neuen Konversation ist die Methode newConversation, welche sowohl in der Klasse Conversation (Kopie der Konversation für welche die Methode aufgerufen wird), als auch in AgentBase (neue Konversation entsprechend der angegebenen Konversations-Typennummer) existiert. Eine Konversation kann auch direkt als Objekt erzeugt werden, doch ist dann darauf zu achten, daß die entsprechende Konversationstype auch im Agenten registriert ist. Es wird nur überprüft, ob eine entsprechende Nummer im Agenten registriert ist. Ob es sich hierbei um die gleiche Klasse bzw. Implementierung handelt wird nicht geprüft. Anschließend existiert die Konversation, allerdings bisher nur als Objekt. Es kann nun die entsprechende Konfiguration vorgenommen werden (z. B. eintragen von Parametern, Anfangszustand setzen, Callbacks eintragen, …). Erst durch den Aufruf von startConversation wird die Konversation tatsächlich gestartet und der Status auf „laufend“ gesetzt. Hierbei wird die Methode handleMessage mit dem Parameter „null“ aufgerufen (das einzige Mal, daß diese Methode ohne eine Nachricht als Parameter aufgerufen wird). Durch Aufruf von waitEnded kann ein Agent (oder sonst ein beliebiger Thread) auf das Ende der Konversation warten. Hierbei handelt es sich um ein blockierendes Warten, das *nicht* unterbrochen werden kann. Zum Beenden einer Konversation muß die Methode endConversation aufgerufen werden. Dies

hat zur Folge, daß die Konversation aus der Liste der laufenden Konversationen entfernt und (falls gewünscht) archiviert wird. Weiters werden alle wartenden Threads vom Ende informiert und können weiterarbeiten. Später eintreffende Nachrichten für diese Konversation erhalten eine Fehlermeldung (siehe unter 4.5.1.1; spezielle Nachrichten).

4.5.1.5 Protokolle

Die Klasse „Protocol“ ist eine Erweiterung einer Konversation. Hier ist eine Möglichkeit zur Speicherung des Status der Abwicklung (Zustandsautomat) bereits vordefiniert. Zusätzlich besteht jedoch auch die Möglichkeit ein anderes Protokoll als Sub-Protokoll zu starten. Hierdurch ergibt sich nicht nur eine statische Hierarchie von Protokollen (Vererbung), sondern auch eine dynamische (Aufruf-Hierarchie). Dies ermöglicht eine gesteigerte und einfachere Wiederverwendung von existierenden Protokollen. So kann etwa ein beliebiges Protokoll zur Überprüfung der Identität des Partners an den Anfang gestellt werden ohne dieses selbst programmieren zu müssen.

Bei Verwendung von Subprotokollen ist jedoch eine Besonderheit zu beachten: Die verwendeten Nachrichten *müssen* mit dem leeren Konstruktor erzeugt werden. Bei Verwendung eines Subprotokolls ist es erforderlich die Konversations-Type und laufende Nummer des Subprotokolls der des äußeren Protokolls anzupassen. Dies ist nur dann möglich, wenn diese Werte vorher nicht explizit angegeben oder gesetzt werden. Dies sollte jedoch kein Problem sein, da die Angabe dieser Werte beim Erzeugen einer Nachricht nur in einem einzigen Fall notwendig ist: Wenn diese ohne Verwendung einer Konversation versandt werden soll. Die Anpassung ist erforderlich, weil sonst die empfangene Nachricht nicht an das Hauptprotokoll (und in diesem dann weiter an das Subprotokoll) weitergeleitet, sondern eine neue Konversation erzeugt würde. Dies fällt insbesondere dann auf, wenn es die erste Aktion eines Protokolls ist, ein Subprotokoll zu erzeugen, und somit die erste Nachricht einer Konversation nicht von dem Protokoll, sondern von dem Subprotokoll stammt. In diesem Fall müssen beim Empfänger zwei Protokolle erzeugt werden: Das Haupt- und das Subprotokoll und die Nachricht an letzteres zur Verarbeitung weitergeleitet werden. Aus diesem Grund müssen Nachrichten auch über die Methoden sendMessage und broadcastMessage der Konversation verschickt werden (welche in Protocol überschrieben wurden) und dürfen nicht direkt über die entsprechenden Agenten-Methoden verschickt werden. Der Methode handleMessage der Konversation entspricht hier doProtocol welche anstatt der Ersteren zu überschreiben ist.

Beim Start eines Subprotokolls ergeben sich keine Besonderheiten: Es wird ganz normal gestartet indem es mit dem Parameter „null“ anstatt einer Nachricht aufgerufen wird. Im Gegensatz dazu ist beim Beenden eines Subprotokolls und dem Übergang der Kontrolle an das übergeordnete Protokoll eine zusätzliche Besonderheit zu beachten: Ein Protokoll hat zwar meist nur einen Eingang, aber praktisch immer mehrere Endpunkte (zumindest: Erfolgreich - Fehler). Es stellt sich daher die Frage, wo nach dem Ende eines Subprotokolls im Hauptprotokoll weitergearbeitet werden soll. Dies beruht in der Regel auf dem Endzustand des Subprotokolls. Hierzu dient die Methode adjustStateAfterSubprotocolTerminated, welche in Klassen, die Subprotokolle verwenden, überschrieben werden muß. Darin ist der Status des Haupt-Protokolls (=aktueller Zustand) entsprechend dem Status des Subprotokolls neu zu setzen. In der einfachsten Form wird nur überprüft, ob das Subprotokoll mit einem Fehler endete und der Zustand des Hauptprotokolles entsprechend auf „Endzustand - Fehler“ gesetzt (ansonsten bleibt der Status unverändert). Bei Erfolg wird einfach im Hauptprotokoll weitergearbeitet.

Ein Protokoll kann zu jedem Zeitpunkt immer nur *ein* aktives Subprotokoll besitzen: Alle Nachrichten werden an dieses weitergeleitet. Ein Verteilen von Nachrichten auf mehrere verzahnte Subprotokolle bzw. die zwischenzeitliche Abarbeitung Einzelner davon im Hauptprotokoll ist nicht möglich. Es ergibt sich daher eine lineare Hierarchie, welche allerdings höher oder tiefer (Subprotokolle von Subprotokollen sind ohne weiteres möglich) bzw. aus verschiedenen Elementen bestehen kann.

Das Verhältnis von Haupt- und Subprotokoll wird in Abbildung 55 näher dargestellt (wobei eine Beschränkung auf ein einzelnes Subprotokoll erfolgt):

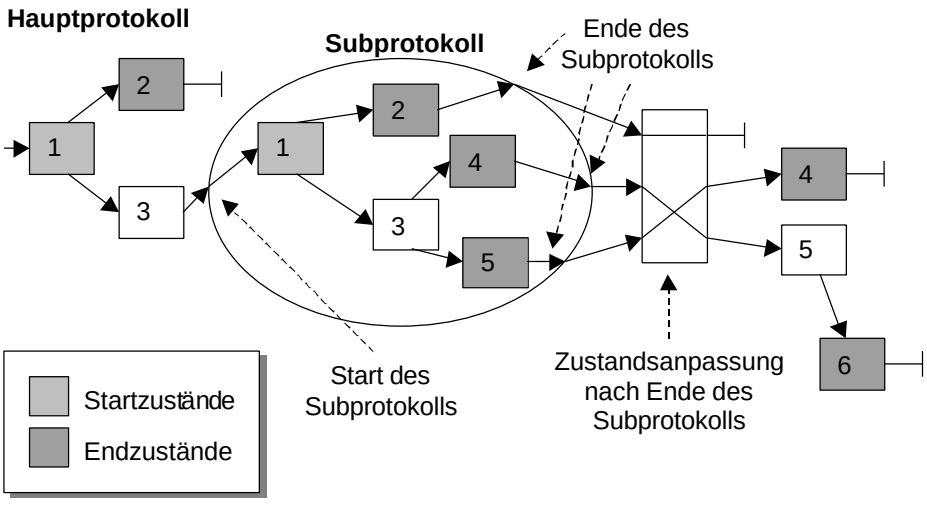

Abbildung 55: Beispiel des Verhältnisses zwischen Haupt- und Subprotokoll

4.5.1.6 ebXML Nachrichten

Im Rahmen des Sport-Portal Projektes wurde das Agentensystem um Nachrichten im ebXML-Format erweitert, um universelle Einsetzbarkeit zu ermöglichen. Diese Nachrichten werden anschließend als String kodiert und an den Empfänger-Agenten weitergeleitet, wo sie wieder in das ebXML-Format zurückgewandelt werden. Eine Erweiterung in die Richtung, daß das AgentenSystem direkt ebXML-Nachrichten empfangen könnte und diese dann an Agenten weiterleitet, wäre problemlos möglich.

Bei ebXML [ebXML] handelt es sich um ein Set von Spezifikationen für ein modulares aber dennoch komplettes E-Commerce Framework, welches von den Vereinten Nationen (UN/CEFACT) und OASIS gefördert wird. Es basiert auf XML und besitzt einen besonderen Fokus auf Geschäftsprozesse. Hierbei soll es sich um den Nachfolger von EDI (Electronic Data Interchange) handeln, für den insbesondere geringere Entwicklungskosten, Flexibilität und einfache Benutzung sprechen. Die grundlegenden Spezifikationen wurden Mitte Mai 2001 beschlossen (daher sind keine Änderungen an der Basis mehr zu erwarten; außer im Rahmen der Weiterentwicklung) und man bemüht sich um eine Standardisierung durch internationalen Organisationen.

Die ebXML Architektur stellt Möglichkeiten zur Verfügung, Geschäftsprozesse und deren zugehörige Nachrichten sowie deren Inhalt zu definieren, selbige zu registrieren und aufzufinden, Firmenprofile und Geschäftsvereinbarungen festzulegen, sowie spezifiziert eine einheitliche Nachrichten-Transportschicht.

Der Vorteil dieses Ansatzes ist der, daß sowohl die Syntax exakt definiert ist (Nachrichten-Details sowie Inhalt sind XML und besitzen DTD‘s) als auch die Semantik (für ein Beispiel siehe [Huemer 2001]), insbesondere für die Inhaltsdaten und die Geschäftsprozesse. Dies erlaubt es, Software zu implementieren bzw. konfigurieren ohne besondere zusätzliche Informationen des zukünftigen Geschäftspartners zu besitzen. Zu diesem Zweck ist auch eine gemeinsame Datenbank (Repository) vorgesehen in der Unternehmen ihre (externen) Geschäftsprozesse ablegen bzw. nach solchen suchen können (sowohl nach Partner-Profilen als auch Prozess-Modellen und Nachrichtenformaten). Auf diese Weise können Informationen über die elektronischen Kommunikationsfähigkeiten potentieller Geschäftspartner gewonnen werden: Welche Prozesse sie verwenden und wie sie eintreffende Nachrichten vom Inhalt her interpretieren werden.

Als Basisklasse für Agenten, welche das ebXML Protokoll nützen möchten wurde eine neue Agentenklasse geschrieben (ebXMLAgent). Diese ist dafür zuständig ebXML Nachrichten (inklusive Broadcasts) in Strings und eigene Nachrichten zu verpacken (ebXMLWrapperMessage sowie ebXMLBroadcastWrapperMessage) bzw. beim Empfang solcher Nachrichten diese wieder in normale ebXML-Nachrichten auszupacken. Ansonsten werden keine besonderen Aufgaben übernommen. An Hilfsklassen existiert noch MessageParsingHandler. Bei XML und Verwendung von DTD‘s ist es erforderlich, daß diese DTDs‘ vom Parser gefunden werden. Da dies bei mobilen Agenten ein Problem darstellen kann, ist diese Klasse dafür zuständig die in Nachrichten angegebenen DTD‘s in tatsächliche Streams aufzulösen (dies erfolgt über den Classloader des Agenten welcher die DTD-Datei im Code-Paket bzw. Verzeichnis des Agenten sucht), von welchen die Strukturdaten anschließend eingelesen werden können.

Wie bereits im Abschnitt über Nachrichten (4.5.1.1) erläutert, wurde eine Erweiterung des Interfaces MessageType vorgenommen, in welchem die zusätzlichen Methoden für ebXML-Nachrichten spezifiziert werden. Eine ebXML-Nachricht besitzt zusätzlich einen Typ (Normal, Fehlermeldung, Bestätigung), eine eindeutige ID (nicht für die Konversation sondern für die Nachricht), sowie eine freie textuelle Beschreibung der Nachricht. Letztere hat keine Bedeutung für den Computer, sondern dient nur dazu einem Menschen (falls er die Nachricht zu Gesicht bekommt) zusätzliche Informationen zu geben (z. B. bei Übertragung über E-Mail und Übertragungsproblemen kann der Administrator hieraus auf die Bedeutung der Nachricht schließen). Die Basisklassen ebXMLMessage and ebXMLBroadcastMessage dienen dazu eine komplette ebXML-Nachricht zusammenzustellen: Aus den Benutzerdaten, den Daten aus der Konversation, und den Zusatzdaten ist ein komplettes XML-Dokument zu bilden. Gleicherweise ist ein solches Dokument wieder in die einzelnen Teile aufzuspalten, die notwendigen Objekte zu erzeugen, und diese mit Daten auszufüllen. Um dies zu ermöglichen, ist bei allen ebXML-Nachrichten eine kleine Einschränkung notwendig: Jede derartige Klasse *muß* einen öffentlichen Konstruktor ohne Parameter besitzen. Dies ist erforderlich, weil (je nach dem Nachrichteninhalt) dynamisch eine entsprechende Klasse erzeugt werden muß, welche dann die Decodierung der Inhaltsdaten vornimmt. Für diese Inhaltsdaten wird vorausgesetzt, daß sie ebenfalls im XML-Format existieren. Die Methoden zum Extrahieren der Werte bzw. dem Konvertieren in XML hat jede Nachricht selbst zu implementieren. Dies kann entweder komplex erfolgen, indem jedes Datenelement gesondert in ein XML-Element verpackt wird (das XML-Dokument entspricht dem Datenmodell), oder indem der Inhalt serialisiert wird (und als Hexadezimal-String als einziger Inhalt gespeichert wird).

Enthält eine Nachricht rekursiv ein XML-Dokument als Inhalt, so ist besonders zu beachten, daß dieses als normaler Text inkludiert werden muß: Beim Parsen der Nachricht würde z. B. ein Fehler auftreten, falls mitten im Dokument eine DTD-Definition angetroffen wird, welche zu einem enthaltenen Sub-Dokument gehört. In diesem Fall ist das Dokument als CDATA-Sektion einzubinden. Um keine Probleme mit darin enthaltenen CDATA-Sektionen zu provozieren, ist ein zusätzliche Kodiervorgang notwendig (siehe dazu die Methoden CDATAQuoteText und CDATAUnquoteText in der Klasse XMLUtils).

4.5.1.7 Beispiels-Protokoll

Das Zusammenspiel von Protokollen und Subprotokollen soll hier an einem Beispiel erläutert werden (der vollständige Programmcode der Klassen befindet sich im Anhang). Das Beispiel ist eine Anfrage an einen anderen Agenten wobei keinerlei Verhandlung stattfindet: Es wird entweder die Antwort geschickt oder eine Ablehnung. Sowohl der Teil des Anfragenden als auch des Antwortenden ist unten dargestellt. Zu beachten ist insbesondere, daß zuerst ein Subprotokoll aufgerufen wird (ContactProtocol), bevor eine eigene Nachricht versendet wird. Dieses dient lediglich dazu, die Identitäten der beiden Teilnehmer auszutauschen. Eine besondere Anpassung nach dem Ende dieses Subprotokolls ist nicht erforderlich. Lediglich bei einem Fehler wird auch im Hauptprotokoll der Zustand auf einen Fehler gesetzt und das Protokoll beendet. Die Funktionen sendInquiry, receiveAnswer, etc. dienen zum tatsächlichen Versand von Nachrichten entsprechender Klasse bzw. Inhalts.

Die einzelnen Abschnitte in dem switch-Statement entsprechen dem internen Zustand beim Empfang der Nachricht. Dies entspricht nicht den Performativen (Arten der Kommunikationselemente in der Sprech-Akt-Theorie zur Agentenkommunikation, z. B. Bitte, Warnung, Informierung; siehe [Nwana/Wooldridge1996] für einen Überblick über ACLs und Ontologien), was bei unterschiedlichen Nachrichten besonders klar wird: Je nach Antwort wird eine andere Aktion durchgeführt (siehe etwa Zustand 2 des Anfragenden). Den Performativen entsprechen die einzelnen Aktionen wie die Methoden receiveRejection oder answerInquiry.

Teil des Anfragenden:

```
switch(state.curState)
{
     case 0:
          if(inquiry==null)
          {
               // Error: We need an inquiry!
               state.curState=17;
               endProtocol();
          }
          else
          {
               state.curState=1;
               startSubProtocol(new ContactProtocol(getAgent(),
                    getCounterpartIdentity(),true),null);
          }
     break;
     case 1:
          sendInquiry();
     break;
     case 2:
          if(msg instanceof AnswerMessage)
          {
               receiveAnswer((AnswerMessage)msg);
          }
          else if(msg instanceof RejectMessage)
          {
               receiveRejection((RejectMessage)msg);
          }
          else
               return null;
     break;
     default:
          endProtocol();
     break;
}
```

Teil des Antwortenden:

```
switch(state.curState)
{
     case 0:
          state.curState=1;
          startSubProtocol(new ContactProtocol(getAgent(),
                    getCounterpartIdentity(),false),msg);
     break;
     case 1:
          state.curState=2;
     break;
     case 2:
          if(msg instanceof InquiryMessage)
               answerInquiry((InquiryMessage)msg);
          else
               return null;
     break;
     default:
          endProtocol();
     break;
}
```

Zustands-Aktualisierung nach Ende des Subprotokolls:

```
public void adjustStateAfterSubprotocolTermin ated(Protocol subprotocol)
{
     // State 2 in ContactProtocol is a success, all others are
     // failures / errors
     if(subprotocol.getState().curState!=2)
     {
          state.curState=18;
          endProtocol();
     }
}
```

4.5.2 Mobilität und Persistenz

In diesem Abschnitt wird erläutert wie Mobilität im Agentensystem unterstützt wird. Hiermit eng verbunden ist ebenso Persistenz (das zeitweilige Stillegen eines Agenten wobei sein Zustand auf einem Hintergrundspeicher erhalten bleibt). Ausgespart wird das Hand-off, wenn ein Agent auf einen anderen Rechner verlagert wird.

4.5.2.1 Serialisierung des Zustands von Agenten

Um den Zustand eines Agenten auf einen anderen Rechner übertragen zu können ist es erforderlich, diesen vollständig zu serialisieren. Hierzu wird das Framework von Java verwendet. Dies bedingt, daß jeder Agent das (leere) Interface java.io.Serializable implementieren muß. Ein Agent kann dann ohne weiteres in einen Stream umgewandelt werden. Dies setzt voraus, daß auch *alle enthaltenen Elemente* serialisierbar sind. Dies ist etwa bei Zertifikaten nicht der Fall. Da auch die Agenten-Identitäten als Zertifikate modelliert sind, sind diese händisch zu serialisieren (was über zwei im Java-Framework vorgesehene Methoden möglich ist).

Besondere Probleme ergeben sich, wenn ein Agent aus mehreren Threads besteht. Sofern der Agent noch eine Referenz auf diese Threads besitzt, werden diese automatisch mit-serialisiert, ansonsten (komplett unabhängig oder ausschließlich eine Rück-Referenz) nicht. Dies bedeutet auch, daß ein Agent (nachdem er wiederhergestellt wurde) benötigte Threads wieder neu erzeugen und starten muß.

Analog hierzu sind auch User-Interface-Objekte (z. B. Fenster) nicht serialisierbar und müssen als transient deklariert bzw. vor der Serialisierung explizit zerstört werden. Es ist daher bei einem Zugriff hierauf immer zuerst die Existenz zu prüfen (und eventuell neue Objekte zu erzeugen). Alternativ können derartige Objekte auch in speziellen Funktionen wiederhergestellt werden, welche nach Ankunft bzw. Depersistierung aufgerufen werden (siehe unten).

Teilweise kann die Speicherung/Erzeugung von Threads automatisiert werden, wie anhand der Klasse FIM.Util.Threads.Timer vorgeführt wird. Diese Klasse stellt einen Timer dar (einmalig oder wiederkehrende Benachrichtigung), der serialisiert werden kann und bei Deserialisierung automatisch wieder vollständig erzeugt wird, inklusive einem Thread zur Benachrichtigungs-Auslösung, falls erforderlich (d. h. wenn der Timer bei der Serialisierung aktiv war).

4.5.2.2 Mobilität

Die Mobilität von Agenten erfolgt in der Weise, daß ein Agent am Quell-Rechner serialisiert wird, diese Daten des derzeitigen Zustands sowie sein Programmcode zu einem anderen Rechner übertragen werden, und der Agent dort neu gestartet wird. Es liegt schwache Mobilität vor, da die Ausführung nicht bei der folgenden Anweisung fortgesetzt wird, sondern beim normalen Startpunkt des Agenten. Für die Fortsetzung der Ausführung an der passenden Stelle ist der Agent selbst verantwortlich.

Die Übertragung der Daten erfolgt über ein eigens definiertes Protokoll „amp“ (Agent Movement Protocol). Auf der Client-Seite wird es durch die Klasse AMPURLConnection implementiert, auf der Server-Seite durch die Klasse ServingThread. Für jede ankommende Verbindung wird ein neues Objekt erzeugt, um die Wartezeit möglichst zu verkürzen. Es können daher mehrere Agenten gleichzeitig empfangen werden, was aufgrund der mitunter langen Dauer (kryptographische Überprüfungen) wichtig ist.

In der Klasse AgentSystem kann eingestellt werden, ob sichere oder unsichere Verbindungen hergestellt werden sollen. Bei sicheren Verbindungen erfolgt die Datenübertragung verschlüsselt (Diffie-Hellman Schlüssel-Austausch), und die Identität der beiden Rechner wird mittels Zertifikaten (Challenge-Response Protokoll) überprüft.

Bevor der Agent tatsächlich gesendet wird, erfolgt auf dem Zielrechner eine Überprüfung, ob dieser Agent grundsätzlich zugelassen wird. Hierzu werden der Name des Agenten, seine Hauptklasse (Name inklusive packages), die Code-Zertifikate, sowie seine Identität übertragen. Es werden die folgenden Werte überprüft:

? Keine doppelten ID-Werte: Existiert lokal ein Agent, dessen Identität dieselbe ID wie der zu Empfangende besitzt, so wird der Agent nicht zugelassen. Eine Überprüfung, ob es sich um einen identischen Agenten handelt, findet nicht statt (es bestehen fast immer geringe Unterschiede; auch zeigt eine Duplizierung meist Programmierfehler an).

? Agenten mit nicht signierter Identität (=einfache Version) werden nur vom lokalen Rechner als Quelle akzeptiert (d. h. nur von Agentensystemen auf anderen Ports). Dies dient ausschließlich zum Testen. Derartige Agenten-Identitäten sollten nicht in fertigen Systemen verwendet werden.

? Es wird überprüft, ob die lokale Policy das Erzeugen von Agenten dieser Klasse erlaubt. Hiermit können suspekte Agenten-Klassen vollkommen ausgeschlossen werden. Dies ist nur ein sehr grober Schutz, weil er schon durch ein Umbenennen der Klasse umgangen werden könnte. In Verbindung mit signierten Identitäten ist dies allerdings wichtiger, da dort die Klasse ebenfalls enthalten ist. Es wäre daher auch eine neue Identität erforderlich, die vom Aussteller neuerdings signiert werden muß.

Beim eigentlichen Agententransfer wird zuerst der Programmcode übertragen (hierbei ist die Verwendung eines Archivs für die Zusammenfassung des Codes und der Daten eines Agenten wichtig da ansonsten der gesamte Verzeichnis-Unterbaum zusammengepackt wird; siehe auch oben), anschließend der Agent selbst. Hierdurch entsteht ein Duplikat des Agenten am Zielrechner. Erst anschließend wird (nach erfolgter Bestätigung) der Agent am Quell-Rechner zerstört. Hierdurch wird sichergestellt, daß der Agent nur dann terminiert wird, wenn die Verlagerung auch tatsächlich erfolgreich abgeschlossen wurde.

Zwei Hilfsklassen werden benötigt, um das Protokoll „amp“ zu implementieren: AMPStreamHandlerFactory und AMPStreamHandler. Diese sind dafür zuständig, bei Bedarf ein Objekt der Klasse AMPURLConnection zu erzeugen.

Ereignis-Methoden

Um dem Agenten die Möglichkeit zu geben bei Persistierung bzw. Verlagerung auf einen anderen Server hin spezielle Vorkehrungen zu treffen (z. B. schließen/öffnen von Fenstern), wurden spezielle Methoden eingebaut.

? onBeforeMoving: Diese Methode wird aufgerufen bevor der Agent für die Verlagerung serialisiert wird. Durch das Auswerfen einer eigenen Exception kann der Agent dies verhindern. In dieser Exception kann er auch einen Zeitraum angeben, nach welchem das Agentensystem es noch einmal versuchen soll. Dies gibt einem Agenten etwa die Möglichkeit, eine Aufgabe fertigzustellen bevor er verlagert wird. Da diese Funktionalität jedoch für das Agentensystem auch gefährlich sein kann (ein Agent kann nicht mehr entfernt werden und verhindert damit z. B. das Terminieren des gesamten Systems), wurden zusätzliche

Einschränkungen vorgesehen (welche in Subklassen des Agentensystem abgeändert werden könnten). So kann ein Agent dies nur einmal verlangen: Bei der zweiten Verzögerung wird der Agent statt dessen terminiert. Auch ist die maximale Verzögerung durch ein Konstante auf 2 Minuten festgelegt.

? onAfterMoving: Wenn ein Agent nach einer Verlagerung am Zielrechner angekommen ist, wird diese Methode aufgerufen. Sie dient insbesondere auch dazu neue Aktionen anzustoßen, welche auf dem nun erreichten Rechner ausgeführt werden sollen.

? onBeforePersist: Hiermit wird dem Agenten signalisiert, daß er auf den Hintergrundspeicher deaktiviert werden soll. Diese Methode ist analog zu onBeforeMoving. Auch hier kann dies vom Agenten verweigert werden (mit den selben Folgen).

? onAfterDePersist: Diese Methode wird aufgerufen, nachdem ein Agent von einem Hintergrundspeicher wiederhergestellt wurde.

4.5.2.3 Persistenz

Um Speicherplatz und Rechenzeit zu sparen, kann ein Agent durch das GUI (aber auch durch den Agenten selbst) auf den Hintergrundspeicher ausgelagert werden. Dies bedeutet, daß sein Zustand zwar vollständig erhalten bleibt, er jedoch nicht aktiv ist (kein laufender Thread). Der Zusammenhang mit Mobilität ist darin zu sehen, daß die selben Mechanismen verwendet werden: Auch hier wird der Agent serialisiert, und auch hier erfolgt nur eine schwache Persistenz (der Agent wird nicht bei der folgenden Instruktion wiederhergestellt, sondern wie nach einer Verlagerung neu gestartet).

Analog zur Mobilität kann ein Agent auch hier der Persistierung widersprechen bzw. diese verzögern (siehe Ereignis-Methoden).

Ist ein Agent persistiert, so kann er nur auf zwei Arten wiederhergestellt werden: Durch den Benutzer, oder indem eine Nachricht direkt an ihn gesendet wird (Broadcasts reichen hierzu nicht aus). Dies könnte auch dazu verwendet werden, einen Snapshot eines Agenten zu erzeugen, sodaß ein früherer Zustand wiederhergestellt werden könnte. In diesem Fall wäre der einzige Unterschied, daß der Thread des Agentes nach der Speicherung nicht beendet würde. Bei der Wiederherstellung eines Snapshots wäre zu berücksichtigen, daß der Original-Agent zuerst zu beenden ist.

5. Anwendungen intelligenter Agenten als persönliche Assistenten

Die Verwendung von Agenten im Bereich des Internets kann von zwei großen Gruppen ausgehen: Agenten von Endverbrauchern (Konsumenten) und Agenten von Anbietern (sowohl von Informationen als auch Waren; gratis bzw. entgeltlich). Diese Unterscheidung ist wichtig, da fundamentale Interessensgegensätze bestehen und daher auch die Implementierung anzupassen ist. So sind Verbraucher-Agenten in der Regel darauf bedacht, viele Angebote von verschiedensten Anbietern zu sammeln und hierbei möglichst wenig Informationen über sich selbst preiszugeben, während Anbieter-Agenten die (potentiellen) Käufer möglichst zum sofortigen Abschluß bewegen und soviel Informationen als möglich sammeln wollen, bzw. an Menschen anstatt Agenten als Kunden interessiert sind. Auch müssen sie unter Umständen in der Lage sein nicht nur mit Agenten, sondern auch mit menschlichen Besuchern zu kommunizieren. Die für die Verhandlungen benötigten Daten werden dafür direkt und passend codiert zur Verfügung gestellt (Hauptaugenmerk: Kommunikation mit verschiedenen Interessenten). Währenddessen liegt das Hauptaugenmerk bei Konsumenten-Agenten auf dem Erfassen der notwendigen Informationen vom Konsumenten, da die Kommunikation mit den Verkäufern standardisiert ist (oder eben nicht stattfindet). Dies wird auch in Abbildung 56 dargestellt.

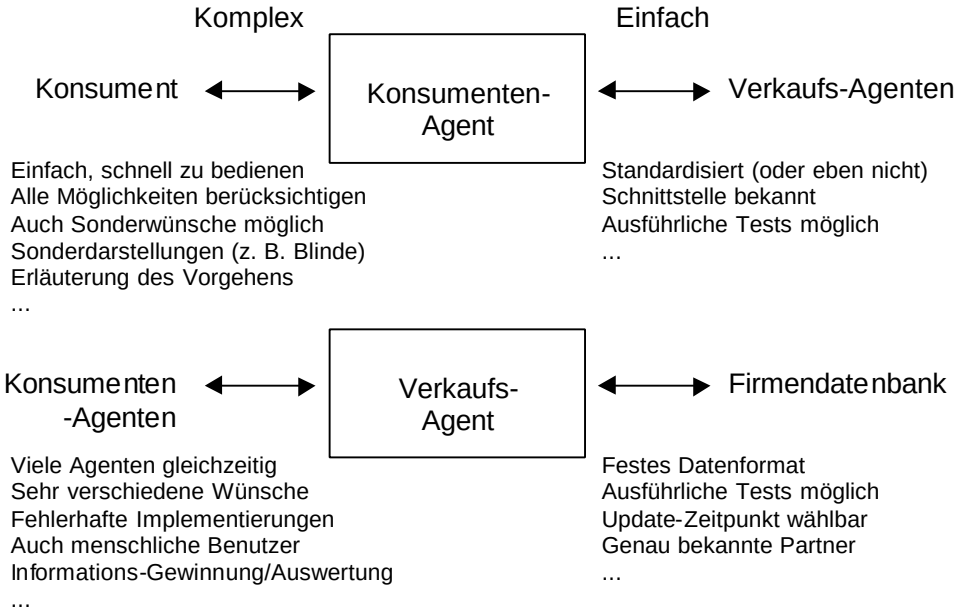

Abbildung 56: Aspekte von Konsumenten- und Verkaufs-Agenten

Neben dieser Einteilung nach dem Verwender in Hinsicht auf E-Commerce können Agenten auch in verschiedenen anderen Sachgebieten, wie etwa der Informationssammlung oder Personalisierung, tätig werden, wobei jeweils spezifische Anwendungsbereiche und Probleme existe-

ren. Einige ausgewählte werden jeweils im Folgenden dargestellt, wobei auch Ansätze für die Lösung oder Abmilderung der geschilderten Schwierigkeiten angeführt werden, insbesondere im Hinblick auf das Agentensystem POND.

5.1. Konsumenten-Agenten (Persönliche Sicht auf das Internet)

Bei Agenten für eine persönliche Sicht auf das Internet muß zwischen Realtime- und Offline-Agenten unterschieden werden. Erstere unterstützen den Benutzer direkt während seiner Informationssuche (z. B. Letizia [Letizia]), letztere bereiten diese nur vor oder führen selbständig längere oder aufwendigere Suchen durch (etwa im Sinne eines „aktiven Instrumentes“, wie es in [Schwehm 1998] als notwendig dargestellt wird). Realtime Agenten sind für diese Aufgabe (zumindest zur Zeit) noch weniger geeignet, da sie zusätzliche Bandbreite benötigen, um die Informationen zu sammeln, an Hand derer dem Benutzer anschließend Hinweise gegeben werden können. Auch ist die Analyse und Klassifizierung einer Webseite eine komplexe und langwierige Aufgabe (welche jedoch notwendig ist, bevor die Ergebnisse mit dem Interessensprofil des Benutzer verglichen werden kann). Im Gegensatz dazu können Offline-Agenten auch während der Nacht (billigere Verbindungen) und ohne Störung des Benutzers arbeiten. Dies ermöglicht es ihnen auch Umfeldsuchen durchzuführen und so einen größeren Bereich abzudekken, als eigentlich notwendig wäre.

Zu diesem Bereich gehört auch ganz allgemein Personalisierung, die ebenfalls in Zusammenhang mit Agenten Bedeutung besitzt (zu kollaborativem Filtern in Verbindung mit Agenten siehe etwa [Green at al. 1998] sowie [Delgado et al. 1998]).

Eine Kombination dieser zwei Grundkonzepte ist jedoch ebenso möglich: Ein Agent sucht sowohl während Arbeitspausen (z. B. während des Lesens einer längeren Seite), als auch Offline, nach Informationen und untersützt damit den Benutzer direkt bei der Online-Suche. Das Problem hierbei ist, daß dies nur funktioniert, wenn der Agent bereits relativ detailliert das Gebiet kennt, welches der Benutzer dann später untersuchen wird.

5.1.1 Aufgabenbereiche

Aufgabe von intelligenten Agenten ist in diesem Bereich die Sammlung von Daten verschiedener Anbieter und das Treffen einer Auswahl davon, um diese anschließend dem Benutzer zu präsentieren, wobei insbesondere das WWW sowie Archive von Mailinglisten oder Newsgruppen Bedeutung besitzen. Hierbei kann es sich einerseits um direkte Informationen handeln,

andererseits aber auch nur um vielversprechende Links, Organisationen, oder Personen. Wichtig hierbei ist, nicht nur eine Positiv-Auswahl zu treffen (was den Benutzer interessiert oder interessieren könnte), sondern auch eine Negativ-Auswahl: Was als uninteressant bekannt ist, sollte explizit entfernt werden, ebenso wie sehr ähnliche Dokumente, da es sich hierbei vermutlich um Kopien handelt. Dies verringert die Anzahl der Ergebnisse und erhöht die Qualität. Anhand ähnlicher Dokumente können auch Gemeinsamkeiten festgestellt werden, sodaß im Sinne einer rekursiven Selbstverbesserung eine bessere Beurteilung sowohl der Benutzerinteressen als auch anderer Seiten ermöglicht wird.

Eine anderer Aufgabenbereich ist sie Bewertung von (eventuell bereits vorhandener) Information: Der Benutzer erhält so Hinweise darauf, wie zuverlässig etwa die Daten auf einer Webseite sind. Eine solche Beurteilung ist jedoch gefährlich, da dies mit Sicherheit einen Angriffspunkt für gezielte Manipulationen darstellt sowie ein ethisches Problem aufwirft: Sollen Menschen bzw. deren Produkte durch rein maschinelle Bewertung (mit großen Auswirkungen!) beurteilt werden (siehe dazu auch Abschnitt 3.3.6.7)? Ansatzpunkte für eine Bewertung etwa von Webseiten können sein:

? Aufbau der verschiedenen Webseiten einer Site (anhand von als „benutzerfreundlich“ bekannten Merkmalen). Aber: Inhalt und Aufbau haben nur entfernt miteinander zu tun.
? Anzahl der Verweise auf externe Websites.
? Anzahl der Verweise von anderen *externen* Webseiten auf diese. Problematisch ist hierbei das Auffinden dieser Webseiten (z. B. Referenzliste oder über Suchmaschinen).
? Bewertung der Seiten, von welchen aus auf diese verlinkt wird.
? Links auf diese Seite von qualifizierten Seiten (z. B. große Portale, seriöse Zeitungen, moderierte Verzeichnisse, etc.), von denen bekannt ist, daß Links genau geprüft werden.
? Negativ-Elemente: Sind Wörter oder Elemente enthalten, von denen bekannt ist, daß sie *nicht* zum gesuchten Thema passen?

Dies ist ein besonders wichtiger Punkt, da mit dem andauernden Wachstum des WWW und der Einfachheit der Informationsbereitstellung das Bewertungs- und Vertrauensproblem immer drängender wird.

Mit der nächsten Gruppe, Konsumenten-Agenten für lokale Aufgaben, überlappt die Vergabe von Prioritäten für Nachrichten wie etwa Newsgruppen-Artikel oder E-Mails. Hierbei erfolgt eine Vorsortierung der neuen Nachrichten nach ihrer Wichtigkeit (Konfigurierbar oder durch

selbständiges Lernen; siehe auch Personalisierung) für den Empfänger. Zusätzlich können bei großer Sicherheit der Klassifikation auch bestimmte Reaktionen vorgeschlagen oder eventuell sogar automatisch durchgeführt werden.

5.1.2 Schwierigkeiten

Schwierigkeiten hierbei liegen insbesondere im Bereich des Datenschutzes: Um für den Benutzer gute Ergebnisse zu erzielen, benötigt der Agent viele Informationen über ihn, seine Vorlieben, Interessen, aber auch Abneigungen. Für Firmen wären diese Daten von hohem Wert sowohl für Werbung als auch für die Bewertung des eigenen Angebotes (seien es nun Webseiten oder Produkte). Agenten sind daher ein besonderes „Angriffs"-Ziel, da dort die gewünschten Informationen bereits vorverarbeitet und explizit vorliegen. Datenschutz kann auch im Sinne von Norman ([Norman 1994]) als Unterpunkt dessen gesehen werden, was er als „the feeling of control" beschreibt: Der Benutzer eines Agenten weiß, was der Agent an Aktionen vornimmt, welche Daten er preisgibt, und ist in der Lage dies auch effektiv zu kontrollieren und durchzusetzen.

Gewisse Informationen müssen jedoch auch dem Anbieter zugänglich gemacht werden, da ansonsten kein sinnvoller Service möglich ist. Analog dazu lassen sich auch aus dem Verhalten Daten ableiten, ohne daß diese explizit zur Verfügung gestellt werden müßten. Das Ziel eines Agenten hat es zu sein, diesen expliziten wie auch impliziten Informationstransfer zu minimieren, und zwar (soweit möglich) bis zu dem unvermeidlichen Ausmaß, welches auch dann erreicht wird, wenn der Benutzer selbst (d. h. ohne Agenten-Unterstützung) die Webseiten besucht. Dies ist zwar nicht einfach, doch können Agenten hier sogar eine Verbesserung ermöglichen: Durch Mobilität sind sie in der Lage Anfragen von verschiedenen Orten aus durchzuführen, sodaß eine Zuordnung IP-Adresse / URL ? Benutzer nicht mehr möglich ist. Auch kann ein Agent problemlos Cookies ([CookieCentral], [Cookie-Spec], [RFC 2109]) vor jedem Besuch einer neuen Webseite löschen. Eine Mühe die sich wohl kaum ein Mensch antun würde. Ebenso können automatische Filter für Cookies integriert werden (als Stand-alone Programm für unter Anderem diesen Zweck siehe etwa WebWasher [WebWasher]).

Ein weiteres Problemfeld ist die Sammlung von Informationen über den Benutzer durch den Agenten. Nur die wenigsten Programme besitzen ein Interface, über welches solche Daten zugänglich und die darauf ausgelegt bzw. für Agenten entworfen sind. Es bleibt daher in vielen Fällen nur ein Umweg: Entweder über eine low-level Beobachtung des Benutzers (verfolgen der Mausbewegungen: Problem des Erkennens der Bedeutung; die unter dem Zeiger befindli-

chen Daten sind nur sehr schwer zu identifizieren), das Erstellen eines Zwischenstücks oder Adapters (z. B. den Agenten als lokalen WWW Proxy-Server zu verwenden; dann sind jedoch bestimmte Informationen nicht mehr zugänglich, wie und an welcher Stelle eines Dokumentes der Benutzer länger verweilte; siehe beispielsweise [Nick/Themis 2001] für ein derartiges System), oder eine Kombination von beidem.

5.1.3 Lösungsansätze

Das Problem des Datenschutzes kann zwar durch Agenten nicht gelöst doch zumindest verbessert werden. Folgendes Ansätze bestehen:

? Anonymität: Wird nicht direkt der Benutzer identifiziert (z. B. durch ein Paßwort), so können anonyme Agenten verwendet werden, welche zusammen mit anderen Maßnahmen (Mobilität, Entfernung sonstiger Identifikationsmerkmale) keinerlei Rückschluß auf den Besitzer mehr zulassen. Im Agentensystem POND wird dies durch Agenten ohne Identifikation unterstützt: Sie besitzen zwar nur geringe Rechte, aber für reinen Lese-Zugriff auf den lokalen Webserver reichen diese wohl meist aus.

? Intessensgebiete-Trennung: Jeder Benutzer kann mehrere anonyme oder unter Pseudonym agierende Agenten erzeugen, welche unterschiedliche Identitäten besitzen. Auf diese Weise kann er als eine Vielzahl verschiedener Personen mit getrennten Interessensgebieten erscheinen. Dadurch sind zumindest Querverbindungen nicht mehr möglich. Dies ist für Agenten leichter möglich, da ein Benutzer kaum mehrere Rechner bzw. Browser verwenden wird. Auch können etwa im Agentensystem POND Agenten von sich selbst Kopien herstellen (mit anderer Identität) oder weitere Agenten starten, sodaß der Benutzer nur einen Agenten konfigurieren muß, aber dennoch im Ergebnis von einer Vielzahl repräsentiert wird.

? Zeitliche Mischung von Anfragen: Zusammengehörende Anfragen können zu verschiedenen Zeitpunkten gestellt werden wobei zwischenzeitlich andere durchgeführt werden (z. B. A – B – C – A – C – B, etc.). Dies erschwert es einem Anbieter sinnvolle Informationen aus dem Verhalten zu extrahieren, da die Mischung auch komplett zufällig erfolgen kann. Dies ist nur für Offline-Agenten sinnvoll.

? Ausnutzung von Mobilität: Manchmal werden Benutzer durch die IP-Adresse oder andere eindeutige Kennzeichnungen des Computers identifiziert. Durch die Ausnutzung von Mobilität (welche POND unterstützt) können Agenten von verschiedenen Orten aus Anfragen stellen, ohne daß diese als zusammengehörig erkannt werden (eine Art mobiler Proxy-

Server). Einzelne Anfragen müssen in vielen Fällen jedoch von jeweils einem Ort aus durchgeführt werden, da durch einen Server-seitigen Zustand sonst keine zusammenhängende Sitzung möglich ist.

? Beauftragung anderer Agenten: Arbeiten mehrere Agenten zusammen, so können Aufgaben getauscht werden. Auch dies erlaubt eine Verschleierung der Interessen, da Anfragen auf mehrere Agenten verteilt werden. Dies zu erreichen ist jedoch nicht ganz einfach, da nicht jeder Agent eine beliebige Anfrage für einen anderen ausführen wird: Der Inhalt könnte seinem Ruf abträglich oder gar verboten sein. Für diese Möglichkeit ist daher unter Umständen an eine Art „Agenten-Pool“ bei Providern oder Insitutionen zu denken, welche verschiedenste Anfragen entgegennehmen und ausführen. Der konkret handelnde Agent wird hierbei vielfach gewechselt (keine Zuordnung zwischen Agent und Besitzer). POND stellt hierfür das System zur Bewertung anderer Agenten zur Verfügung, wovon die Akzeptierung von Aufträgen abhängig gemacht werden kann.

Auch bei der Sammlung der Vorlieben des Benutzers kann durch die Integration von Agenten eine Qualitätsverbesserung erreicht werden. Es stellt sich jedoch bei Java das Problem, daß dazu ein Umweg über das Nativ-Code-Interface erforderlich ist (wodurch die Plattformunabhängigkeit verloren geht), wenn eine Verbindung zu bestehenden externen Programmen hergestellt werden soll, falls diese nicht durch Text Ein-/Ausgabe gesteuert werden.

? Agenten als Mittler: Ein Anwendungsbereich von Agenten ist die Verbindung zu Legacy-Applikationen. Stellen diese zumindest gewisse Beobachtungsmöglichkeiten bereit, so können Agenten als Zwischenstücke verwendet werden, um (nach entsprechender Vorauswertung bzw. Umwandlung) auch deren Informationen zu integrieren.

? Komponentenbasierte Software: Komponentenbasierte Software erlaubt Agenten einen viel leichteren Zugriff, indem sie sich als „Zwischenstück“ zwischen einzelne Programmteile einfügen kann. Sie erhalten dadurch Zugriff auf innere Vorgänge, sodaß die Beobachtung der Benutzeraktionen stark verbessert wird. Intelligente Agenten ermöglichen es dann in eingeschränktem Ausmaß, durch die Beobachtung mehrerer Komponenten übergeordnete Ziele des Benutzers zu identifizieren.

? Applikationsübergreifende Datenerfassung: Agenten sind unabhängig von einzelnen Programmen. Sie können daher aus mehreren Programmen Daten beziehen und diese zueinander in Relation setzen. Die Reaktionen auf bestimmte E-Mails können etwa mit der Suche in einem Web-Browser und den bearbeiteten Textverarbeitungs-Dokumenten verglichen werden. Es wird dadurch vermieden, aus einzelnen Aktionen zu große Schlüsse zu ziehen

und es wird mehr auf den Gesamtzusammenhang geachtet. Dies ist naturgemäß eine schwierige Aufgabe. Hier kommt wiederum die Intelligenz von Agenten ins Spiel.

? Zeitliche Unabhängigkeit: Da Agenten nicht mit einzelnen Applikationen verbunden sind, besitzen sie zeitliche Unabhängigkeit. Es können nicht nur (wie sonst auch möglich) Daten über mehrere Sitzungen hin gesammelt werden, sondern es kann eine unabhängige, eventuell zeitintensive, Auswertung erfolgen. Der Agent würde sich hierbei auf eine reine Protokollierung der Benutzer-Tätigkeit beschränken, sodaß dieser bei seiner Arbeit wenig gestört wird. In anschließenden Arbeitspausen erfolgt die Auswertung, aufgrund derer zu einem späteren Zeitpunkt die Informationen für eine Unterstützung des Benutzers zur Verfügung stehen (Online-Sammlung, Offline-Auswertung und Online-Unterstützung).

5.2. Konsumenten-Agenten (Lokale Aufgaben)

Für Konsumenten können Agenten auch lokal Aufgaben erfüllen. Obwohl es sich um Aufgaben handelt, bei welchen keine Mobilität notwendig ist, wird sie dennoch in vielen Fällen nützliche Zusatzfunktionen erlauben.

Bei den hier angeführten Beispielen ist zu beachten, daß viele davon bereits in einzelnen Programmen implementiert sind. Der Vorteil von intelligenten Agenten ist daher darin zu sehen, daß wieder eine anwendungsübergreifende Verarbeitung und Integration stattfindet:

? Die Daten stammen aus verschiedenen Applikationen.
? Mehrere Agenten können zusammenarbeiten, welche jeweils auf ein besonderes Gebiet spezialisiert sind.
? Die Ergebnisse der Verarbeitung werden an verschiedene Applikationen weitergeleitet.

5.2.1 Aufgabenbereiche

Beispiele hierfür sind rein lokale Aufgaben, welche darin bestehen, besondere Informationen zu besorgen, zu filtern, und anschließend den Benutzer darüber zu informieren („Trigger-Modell“). Der Vorteil dabei ist, daß relativ wenig Intelligenz erforderlich ist, was auch der Grund für die weite Verbreitung ist. Exemplarisch können genannt werden:

? Reminder: Erinnerung an bestimmte Termine oder Vorgänge. Dies kann von zeitabhängigen Alarmen bis hin zu komplexen Bedingungen reichen (eine bestimmte Datei wurde von einer Person bearbeitet, in einem besonderen Verzeichnis gespeichert, und eine gewisse Zeit lief ab). Ersteres wurde als Beispiel-Agent in POND implementiert.

? Aktien-Ticker mit Alarm: Dauernde/Regelmäßige Anzeige ausgewählter Aktienkurse auf verschiedenste Arten (E-Mail, SMS, Ticker-Fenster, etc.). Bei Erfüllung bestimmter Vorgaben (z. B. bestimmter Abfall/Anstieg, Einlangen von Nachrichten oder Analysen, …) wird der Benutzer gesondert alarmiert. Auch diese Anwendung ist in POND als komplexeres Beispiel vorhanden, wobei eine Integration in das System zur Nachrichten-Bearbeitung (siehe auch sogleich unten) erfolgte.
? Software-Updates: Der Agent stellt selbständig fest, welche Software installiert ist, und überprüft anschließend verschiedene Webseiten, ob neue Versionen bzw. Updates verfügbar sind (Hersteller, spezielle Update-Webseiten, Händler, etc.).

Auch im speziellen Bereich der Bearbeitung von Nachrichten können Agenten Anwendung finden. Besonderer bedeutungsvoll ist, daß die Nachrichten aus verschiedensten Quellen stammen können: E-Mails, Newgruppen, Mailinglisten, Webseiten, Notizen, etc. Aber auch aus Datenbanken oder Dateisystemen können zusätzliche Informationen zusammengestellt werden, sodaß für die Bearbeitung notwendige Daten gleich in Einem bereitgestellt werden.

? Filtern von E-Mails: Etwa zum Identifizieren und Entfernen von Spam oder Sortieren in Ordner nach Thema oder nach Dringlichkeit. Andere Möglichkeiten wären Spezial-Interfaces, um eine Zusammenfassung von E-Mails an Handys oder PDAs zu senden (nur die ersten Zeilen; keine Bilder oder Attachments, …), um auf diesem Weg Kommandos einzuholen, wie mit diesen zu verfahren ist, oder den Benutzer von dem Einlangen wichtiger Nachrichten zu alarmieren.
? Forwarder: Weiterleiten von Mails bzw. Kopien davon, basierend auf externen Informationen, beispielsweise an die anderen Mitarbeiter der Arbeitsgruppe (Integration in Groupware). Auch Rückmeldungen (wurde schon von jemandem beantwortet) können so integriert werden: Die E-Mail wird in einen anderen Ordner verschoben.
? Replier: Automatische Beantwortung von Nachrichten wie etwa bei Urlaub oder sonstiger Abwesenheit. Eine andere Anwendung ist die automatisierte Anfragebeantwortung, um etwa Standard-Fragen zu Produkten (Update erhältlich, Preis, FAQs, …) ohne Benutzereingriff zu bearbeiten oder zumindest vorzubereiten.

Alle drei erwähnten Punkte sind unter Verwendung von POND implementiert. Hierbei hat die Konfiguration jedoch noch durch (allerdings äußerst einfache) Programmierung zu erfolgen, da kein graphischer Editor zur Verfügung steht.

Dieser Anwendungsbereich ist besonders auch in Verbindung mit PDAs interessant, da es hier weniger auf Rechenleistung ankommt als auf dauernde Verfügbarkeit und regelmäßiges Durchführen einfacherer Aufgaben. Arbeitet der Benutzer nicht gerade am Gerät, so können Agenten die freie Rechenleistung für diese Aufgaben nutzen. Aber auch die Verwendung als Report-Medium ist sinnvoll: Agenten beginnen Aufgaben zu Hause oder auf einem sonstigen Server und begeben sich mit den Ergebnissen schließlich zum PDA (der sich hoffentlich beim Benutzer befindet), um diese dort zu präsentieren.

5.2.2 Schwierigkeiten

Problematisch kann sein, daß auch für Routine-Aufgaben manchmal ein sehr breites Allgemeinwissen notwendig ist. Dieses ist in Agenten ungleich schwieriger zu integrieren als detailliertes Spezialwissen für einzelne Aufgaben.

Ein anderer Punkt (besonders für PDAs) ist die Belegung von Rechenzeit. Es gibt keine Garantie, daß sie zu dem benötigten Zeitpunkt auch verfügbar ist: Es kann z. B. gerade der Benutzer direkt am Gerät arbeiten. Es bleibt daher nur mehr wenig bis gar keine Rechenzeit für den Agenten übrig. Problematisch ist besonders die mangelnde Vorhersehbarkeit, weniger der absolute Mangel. Eingeschränkt läßt sich dies durch Benutzerbeobachtung und vorgezogene Ausführung verbessern, doch wird dies besonders bei mobilen Geräten wenig qualitätsvoll sein (da oft gar keine Regelmäßigkeiten bestehen und daher Voraussagen unmöglich sind).

Eine weitere Schwierigkeit ist, daß die Konfiguration solcher Agenten nicht sehr einfach ist: Umso mächtiger und nützlicher sie werden, desto mehr Möglichkeiten zur Anpassung bestehen (insbesondere, da Agenten dieser Art eher geringere Intelligenz besitzen). Eine komplexe Konfiguration erhöht jedoch die Fehleranfälligkeit, sodaß die Benutzerzufriedenheit sinkt. Auch kann es Benutzern unter Umständen schwer vermittelbar sein, warum einzelne Optionen bestimmte Folgen nach sich ziehen: Komplexe Systeme enthalten auch „versteckte“ Beziehungen, welche (für den Benutzer) zu unerwünschten und überraschenden Ergebnissen führen können. Die richtige Abwägung zwischen Mächtigkeit/Nützlichkeit und Anpassungsfähigkeit zu finden stellt eine besondere Herausforderung dar.

Besonders wichtig ist die Interoperabilität und die Erweiterbarkeit: Nur wenn diese gewährleistet sind, haben Agenten einen Vorteil gegenüber einzelnen Programmen (oder dort eingebauten Assistenten/Programmteilen). Agenten müssen in der Lage sein sowohl mit verschiedensten

anderen Programmen (auch wenn diese nicht darauf ausgelegt sind), als auch mit anderen Agenten zu interagieren bzw. zu kommunizieren ([Charlton et al. 2001]).

5.2.3 Lösungsansätze

Ein Ansatz insbesondere für die Vereinfachung der Konfiguration ist, die Agenten mit einfachen Aktionen starten zu lassen. Der Benutzer kann später jederzeit zusätzliche Aufgaben durch Konfiguration weiterer Punkte zuteilen. Dieses schrittweise Herangehen reduziert den jeweils notwendigen Aufwand und die Komplexität: Wenn die bisherige Konfiguration funktioniert, sollten sich bei kleinen Änderungen/Hinzufügungen auch nur kleine Verhaltensänderungen ergeben. Diese inkrementelle Konfiguration ermöglicht es dem Agenten auch aus bisherigen Tätigkeiten Präferenzen des Benutzers herauszulesen und somit selbst Vorschläge für Verbesserungen oder eine Ausweitung der Tätigkeit zu machen sowie den Benutzer durch die Zusatz-Konfiguration zu führen und sinnvolle/passende Werte vorzuschlagen.

Bezüglich der Erweiterbarkeit und der Interoperabilität gibt es einige Punkte, die schon beim Entwurf der Agenten zu beachten sind. Diese geben zwar keine Garantie auf eine funktionierende Zusammenarbeit, doch besteht zumindest die Möglichkeit hierzu. Mit mehr oder weniger großem Aufwand ist es dann möglich aus einzelnen Elementen ein funktionierendes größeres System zusammenzusetzen. Die folgenden Punkte sollten daher schon beim Design mitberücksichtigt werden:

? Schnittstellen vorsehen: Feste Schnittstellen sowohl zu Daten wie auch zu Methoden sollten vorgesehen werden, sodaß ein Zugriff auch von anderen Stellen aus möglich ist. Da sich hierbei natürlich Sicherheitsfragen stellen, kann dies auch über ein besonderes Protokoll erfolgen. POND unterstützt die Definition von Protokollen in implementierungsunabhängiger Weise: Besitzen sie die selbe Nummer und ist ihr Verhalten kompatibel (äquivalente Nachrichten zum korrekten Zeitpunkt), so können Objekte bzw. Agenten beliebiger Hersteller zusammenarbeiten.
? Universelle Manipulationsmöglichkeiten: Der Agent sollte in der Lage sein seine Eingangsdaten aus verschiedenen Quellen zu beziehen: Direkt übergebene Daten aus denen herauszusuchen ist, aber auch selbständige Suche. Ideal ist eine Konfigurationsmöglichkeit durch optional Inanspruchnahme eines anderen Agenten: So können auch (noch) unbekannte Programme bedient und daraus Daten importiert werden.
? Keine stand-alone Lösungen: Ein Agent soll keine Aufgabe einfach alleine erledigen sondern es sollte eingeplant sein, daß er seinen Auftrag auch von einem anderen Agenten er-

halten kann (inklusive erforderlicher Konfiguration; siehe oben). Ebenso ist vorzubereiten daß die Ergebnisse nicht nur dem Benutzer direkt präsentiert werden (z. B. in einem Fenster), sondern daß sie wieder an einen oder mehrere andere Agenten weitergegeben werden können (den auftraggebenden Agenten, aber auch andere). Hierauf wurde bei der Implementierung von Beispiels-Agenten besonderer Wert gelegt (siehe etwa schon die Beispiels-Agenten Beeper und BeepSender).

? Silent mode: Die Aufgabe sollte, wenn einmal erteilt, ohne weitere Benutzerinteraktion im Hintergrund ausgeführt werden können. Dies beinhaltet auch, daß die Bearbeitung ohne Anzeige (z. B. eigenes Fenster) durchgeführt werden kann. Nur in diesem Fall kann die Verarbeitung auch auf andere Rechner hin verlegt werden. Ein „silent mode“, in welchem lediglich ein Hintergrund-Prozess abläuft, sollte daher zusätzlich zu einem normalen Modus (Anzeige des Fortschritts, Abbruchmöglichkeit, Eingriffe erlaubt, …) vorgesehen werden. Dies entspricht dem Standard-Verhalten, welches für Agenten bei POND vorgesehen ist: Sie erhalten nur ein kleines Fenster zur Anzeige innerhalb des Agentensystem,, können jedoch bei Bedarf zusätzliche Fenster öffnen.

5.3. Konsumenten-Agenten (E-Commerce)

Insbesondere im Bereich des E-Commerce können Agenten Anwendung finden und zwar in allen Phasen des Verkaufsprozesses: Pre-Sales, Sales und After-Sales. Sie können weiters dazu dienen, zusätzliche Anreize für Kunden („added value“) zum Umstieg von konventionellen Geschäften zu bieten (siehe auch [Sonntag/Reisinger 2001a]). Nicht alle der folgenden typischen Aufgaben können/sollten jedoch auch von Agenten erfüllt werden (oder sind überhaupt elektronisch durchführbar).

? Pre-Sales Phase: Beschaffung von Informationen, Anbietervergleich, Marktbeobachtung (Bezüglich der Informationssuche und –sammlung kann auf die obigen Ausführungen verwiesen werden).
? Sales Phase: Kaufabwicklung, Zahlungsabwicklung, Abholung, Qualitätprüfung, Verwendung von Kundenkarten, Reklamations-Abwicklung
? Post-Sales Phase: Service, Reparaturabwicklung, Problembehebung, Zusatzprodukte (Cross- und Up-Selling), Support, Ersatzteilbeschaffung, Altgeräterücknahme

Zu berücksichtigen ist in diesem Bereich natürlich die besondere Bedeutung der Aktionen des Agenten: Diese haben mehr (Kauf durch Agenten) oder weniger (Zusammenstellung von An-

bietern) direkte finanzielle Auswirkungen, weshalb auf Sicherheit, Verläßlichkeit und Robustheit großer Wert zu legen ist.

5.3.1 Aufgabenbereiche

Im einzelnen könnten Agenten die folgenden Aufgaben übernehmen:

? Suche nach Anbietern: Agenten können mehrere Suchmaschinen besuchen, dort nach Anbietern suchen, und anschließend eine Vorauswahl treffen, um tote Links, Duplikate, falsche Ergebnisse, usw. auszusortieren. Auch kann unter Umständen schon eine weitere Filterung vorgesehen sein: Der Agent kann zusätzlich bereits nach dem gewünschten Produkt suchen und Anbieter, welche dieses nicht führen oder bei denen es momentan nicht lieferbar ist, aus der Liste entfernen. Ebenso können Zusatzinformationen über den Anbieter erfaßt werden (etwa Bewertungen der Anbieter, exemplarisch [Gomez]).

? Anbietervergleich: Es existieren bereits Webseiten zum Vergleich von verschiedenen Anbietern ([Primanestor], [CNETShopper], [Geizhals]; für einen einfachen selbst erstellten Suchagenten siehe [Weigend 2000]). Agenten könnten diese Tätigkeit ebenso übernehmen, insbesondere auch einen Meta-Vergleich durchführen. Eine weitere Aufgabe in diesem Bereich ist das Nachprüfen der Informationen indem direkt auf den verwiesenen Webseiten kontrolliert wird, ob es sich tatsächlich um das richtige Produkt handelt und die Informationen korrekt sind. Auch kann der Agent diese Seiten speichern, sodaß er dem Benutzer nicht nur die Liste, sondern auch gleich die dazu passenden Detailinformationen präsentieren kann.

? Produktrezensionen: Für viele Produkte/Produktgruppen existieren Webseiten, auf welchen eine Beschreibung und manchmal auch ein Vergleich verschiedener Produkte angeboten wird (z. B. für PDAs: [PDAComparison]). Auch diese Informationen können von Agenten zusammengestellt und zu einem Gesamtvergleich mit Angeboten integriert werden.

? Order-Tracking: Agenten können auch die Überwachung von Bestellungen übernehmen. Dies kann einerseits in regelmäßiger Kontrolle des Lieferstatus (bestellt, bei Verpacken, ausgeliefert, verzögert, nicht mehr lieferbar, …) bestehen, als auch in der Verfolgung des Versandes (falls anwendbar und unterstützt, z. B. bei Paketdiensten). Dies erlaubt genauere Voraussagen, wann das Produkt tatsächlich eintreffen wird, bzw. wann in etwa die Abbuchung vom Konto oder der Kreditkarte erfolgt. Für letztere Fälle kann dies auch einer Überwachung gleichkommen indem der Besitzer alarmiert wird, falls eine Belastung stattfindet, ohne daß eine Auslieferung zumindest bevorsteht.

? Suche nach Zusatzprodukten: Dies ist ein besonders sinnvoller Bereich, da es bei Zusatzprodukten oft sehr große Preisunterschiede gibt. Weiters kann es schwierig sein festzustellen, ob das Add-On zum konkreten Basisprodukt kompatibel ist. Agenten können hier eine Vorauswahl treffen. Auch rein zur Information, welche Arten/Typen von Zusatzprodukten überhaupt erhältlich sind, kann eine Übersicht sinnvoll sein.

? Bedarfsdeckung: Bei einfachen und standardisierten Produkten ist es auch denkbar, Agenten die Bedarfsdeckung zu überlassen. Der Agent stellt den Stand fest, überprüft an Hand seiner Erfahrungen wie lange dieser ausreichen wird bzw. die Lieferung dauert, und bestellt selbsttätig rechtzeitig den notwendigen Ersatz. Hierbei werden einige der obigen Anwendungen integriert, wie etwa die Suche nach dem billigsten Anbieter und Order-Tracking, um den Besitzer bei Problemen benachrichtigen zu können.

? Abrechnungskontrolle: Bei komplexen Abrechnungen (z. B. Datenbankzugriffen welche nach Suche, gelieferten Ergebnissen, Textseiten, etc. bezahlt werden) können Agenten die Überwachung der Abrechnung übernehmen, insbesondere wenn es sich um Zugriffe von mehreren Personen handelt. Agenten können anschließend auch Auswertungen durchführen, um eine genauere Zuordnung der Kosten oder eine Vorausschau für später zu erstellen, sowie Hinweise auf eventuelle Einsparungsmöglichkeiten (doppelte/ähnliche Abfragen, etc.) geben. In diesen Anwendungen ist besonders auf den Datenschutz zu achten.

5.3.2 Schwierigkeiten

Neben den besonderen Problemen existiert in Bezug auf E-Commerce ein allgemeines, welches übergreifend für fast alle Anwendungsgebiete gilt: E-Commerce Anbieter sträuben sich oft gegen einen direkten Zugang von Agenten zu ihren Daten. Die Hauptbefürchtung ist, daß Konsumenten nur mehr den Preisvergleich wählen und andere Aspekte völlig außerachtlassen. Da dies für viele Geschäfte ein Problem darstellt, wird es zu verhindern versucht.

Für die einzelnen Anwendungsgebiete existieren jedoch auch spezifische Probleme:

? Die Ergebnisse von Anfragen bei einer Suchmaschine sind häufig nicht sehr zielführend. Einerseits unterscheiden sich die Ergebnisse mehrerer Anbieter stark, andererseits befinden sich darunter auch viele inaktuelle, inzwischen nicht mehr existierende, und auch falsch zugeordnete Ergebnisse. So ist es etwa gar nicht einfach alle (oder zumindest einige wichtige große) Internet Buch-Geschäfte zu finden: Bei einer Suche nach „Internet Book Shop“ auf

Altavista ([Altavista]), wird etwa von den bedeutenden Anbietern kein Einziger unter den ersten zehn Ergebnissen aufgeführt[6].

? Zusätzliche Informationen sind oft nur schwierig aus einer Webseite zu extrahieren. So ist etwa bei einem Buch die ISBN-Nummer zwar sehr einfach festzustellen (genaues Format bekannt; steht praktisch immer unmittelbar neben dem String „ISBN“) jedoch z. B. die voraussichtliche Lieferzeit schon sehr viel schwieriger (an verschiedensten Stellen, eventuell auf einer anderen Seite, etc.). Weiters besteht kein Standard, sodaß diese Informationen von jedem Anbieter auf eine andere Weise, in einem neuen Format, und an unterschiedlicher Stelle dargestellt werden. Hierunter fällt auch die Berechnung von Lieferzeit und Lieferkosten, insbesondere bei internationalen Bestellungen: Diese erfährt man entweder erst am Ende des Bestellvorganges oder die Informationen müssen selbst an Hand von Angaben auf (u. U. mehreren) Hilfe-Seiten berechnet werden.

? Bei der Suche nach Zusatzprodukten stellt sich die Frage in welcher Kategorie bzw. wo überhaupt gesucht werden soll. So ist etwa Zubehör eine sehr vielschichtige Gruppe, die Produkte verschiedenster Kategorien umfaßt (und daher auch u. U. in verschiedensten Geschäften zu finden ist). Die eine Gemeinsamkeit, die Kompatibilität mit einer bestimmten Version eines konkreten Produktes, kann jedoch nicht direkt zur Suche verwendet werden. Eine Möglichkeit ist (sofern angeboten) eine Liste beim Hersteller des Grundproduktes zu durchsuchen und anschließend nach ähnlichen Seiten Ausschau zu halten.

? Die automatische Deckung des Bedarfs durch Agenten ist naturgemäß riskant: Was passiert bei Fehlern des Agenten? Nicht so einfach zu lösen ist auch die Datenerfassung: Wie stellt der Agent die noch vorhandene Menge fest? Dies mag etwa bei der Versorgung mit elektronischem Bargeld (E-Cash) noch einigermaßen einfach zu bewerkstelligen sein, doch bei physikalischen Gütern, wo dies eine sehr praktische Anwendung wäre (z. B. Druckerpapier, Grundnahrungsmittel, etc.), ist dies nur kompliziert möglich.

? Bei der Abrechnung stellt sich wieder das Problem des Datenschutzes: Nicht nur die Abrechnung wird überwacht, sondern auch das Verhalten der Personen, welche die Kosten verursachen. Handelt es sich um mehrere Rechner, so stellt sich auch die Frage der Erfassung. Entweder der Agent besitzt auf jedem Rechner einen Sub-Agenten, welcher das lokale Logging übernimmt, oder der Agent muß sich auf einem „Engpaß“ wie z. B. einem Firewall befinden, welchen der gesamte Verkehr passieren muß.

[6] Dafür findet sich etwa folgendes darunter: „A-Ball Plumbing Supply - Internet Book Shop: Step by Step Guide Book on Home Plumbing“!

5.3.3 Lösungsansätze

Eine Abhilfe gegen das Problem der generellen Ablehnung könnte sein, dem/den Agenten umfassende Informationen in standardisiertem Format zur Verfügung zu stellen. Dadurch wird garantiert, daß Agenten nicht nur den relativ einfach festzustellenden Preis an den Benutzer weitergeben, sondern auch sonstige Zusatzinformationen. Da sich der Wettbewerb durch das Aussperren von Agenten ohnehin nicht verhindern sondern bestenfalls abschwächen läßt, könnte diese Strategie beiden Seiten Vorteile bringen. Dies ist insbesondere auch vor dem Hintergrund zu sehen, daß viele Kunden schon Probleme mit Online-Shops hatten (lange Lieferzeit, falsche Angaben auf Webseiten, Probleme bei fehlerhaften Produkten, …), und sich daher die anfängliche Euphorie nur beim Allerbilligsten zu kaufen bereits wieder etwas abschwächt. Es werden daher der Ruf eines Anbieters und die angebotenen Zusatzdienste wieder größere Bedeutung gewinnen. Diese Verringerung der Fokussierung auf den Preis alleine und hin zur Integration auch anderer Produktaspekte (Garantie, Lieferzeit, etc.) führt zu weiteren Problemen wie etwa schon dem Vergleich (z. B. welche Garantie ist besser?). Dies führt zur Entwicklung von Verhandlungs-Protokollen, welche Agenten durchführen, um mit dem Verkäufer über eine Vielzahl von Punkten Einigkeit zu erlangen (siehe [Lomuscio et al. 2000], wo auch eine Klassifikation für Verhandlungen im E-Commerce dargestellt wird; sowie [Jennings et al. 2001]).

Auch ohne diese „Weiterentwicklung“ ist es unmöglich, Agenten komplett von einem Online-Shop auszusperren, indem die Daten einfach über den normalen Web-Zugang abgefragt werden: Es bereitet keine Schwierigkeiten, einen normalen Benutzer zu simulieren. So etwa indem ein Agent auch Cookies akzeptiert, obwohl diese für ihn keine Bedeutung haben (nicht immer aber auch für Webseiten; deren Verwendung kann als Abwehr gegen Agenten und direkte Links verwendet werden!). Durch den Einbau von zufällig gesteuerten Verzögerungen und unnötigerweise abgerufenen Seiten ist auch eine Erkennung durch die hohe Zugriffsgeschwindigkeit und Zielsicherheit nicht mehr möglich. Genau dies wird etwa im System POND vom Agenten zum Versenden von SMS verwendet: Er muß Cookies akzeptieren (welche für die Funktionalität selbst keine Bedeutung besitzen; die Daten werden komplett über Formulare von Seite zu Seite weitergegeben!), um über bestimmte Webseiten SMS verschicken zu können.

Um die Ergebnisse von Anfragen an Suchmaschinen zu verbessern existieren mehrere simple und einige komplexere Möglichkeiten. Tote Links und falsche Seiten können relativ einfach entfernt werden. Ebenso ist die gemeinsame Auswertung der Ergebnisse von Anfragen an ver-

schiedene Anbieter kein größeres Problem. Um jedoch auch andere Webseiten zu finden, könnte eine Umfeldsuche gestartet werden. Dies beinhaltet die Suche nach anderen Begriffen, welche zum Gesuchten dazugehören (z. B. selbes Fachgebiet),,oder nur Synonyme sind (mit Verwendung eines Thesaurus). Letztere Techniken sind jedoch sehr aufwendig und bedürfen auch eines größeren Wissens über das zu untersuchende Gebiet. Sie sind daher nur durch intelligentere Agenten durchführbar.

Auch bei der Suche nach Zusatz-Informationen auf Webseiten sind intelligente Agenten gefragt. So können auch aus der optischen Nähe von Elementen Schlüsse gezogen werden. Da sich dies jedoch z. B. in HTML nicht unbedingt auch in Code-Nähe ausdrückt (Bsp.: Tabellen), ist hierzu ein größeres Verständnis notwendig. Eine simple Lösung ist, eine genaue Definition der Seite aufzustellen (welche Trennzeichen/-wörter/-zeichenfolgen enthält und beschreibt, an welcher Stelle in welchem Format welche Daten zu finden sind). Bei Verwendung von XML in Verbindung mit Stylesheets wäre dies besonders einfach: Es müßte lediglich das Datenmodell (DOM) bekannt sein, um sofort beliebige Daten extrahieren so können. Diese Spezifikationen sind jedoch immer auf einen einzigen Anwendungsfall hin zugeschnitten und daher nur äußerst eingeschränkt für andere Webseiten verwendbar. Weiters sind sie besonders anfällig gegen Layout-Änderungen. Wenn nicht ausschließlich die Bilder geändert werden (z. B. zu besonderen Feiertagen), so ist eine neue Spezifikation zu erstellen. Da Webseiten ein sehr dynamisches Medium sind, ist dies ein besonderes schwerwiegendes Hindernis. Bei den implementierten Beispielsagenten wird etwa versucht, eine Webseite soweit zu reduzieren (indem etwa Tags entfernt werden), daß optisch nahe Elemente auch textmäßig näherrücken.

Ein besonderer Ansatz zur Verbesserung der Gesamt-Systemleistung (also nicht der einzelner Agenten!) ist, daß Akteure auch altruistisch handeln: Wohlwollende Agenten (benevolent agents; [Huhns/Mohamed 1999]). Der Gedanke hierbei ist, daß Agenten für sie unnütze Aufgaben durchführen (z. B. Hindernisse aus dem Weg räumen), um später zu profitieren, wenn auch andere Agenten Situationen bereinigen, welche für sie Probleme darstellen. Bei einer Übertragung des Gedankens auf den Bereich von E-Commerce ist jedoch zu berücksichtigen, daß bei einem Großteil der Teilnehmer dies nur insoweit Anklang finden wird, als keine Zusatzkosten entstehen (kurze Verzögerungen, kleine Zusatzarbeit, etc.). Auch stehen dem eventuell rechtliche Probleme entgegen: Bezahlte Leistungen (z. B. Informationen, aber auch Programme) dürfen nicht einfach an andere Agenten weitergegeben werden, unabhängig davon wie großzügig der Agent ist. Dennoch ist es von Wert den Gedanken zu verfolgen, wenn auch nur in einem gewissen Umfeldbereich: Die eigenen Erfahrungen oder vorläufige Informations-

Sammlungen auch an andere Agenten oder die Öffentlichkeit weiterzuverteilen kann Vorteile und Ersparnisse für alle bringen. Dies wird derzeit schon unabhängig von Agenten praktiziert, indem positive und negative Erfahrungen in Newsgruppen oder zu Bekannten gepostet werden. Doch existieren auch institutionalisierte Webseiten für diesen Zweck (wenn auch in eher negativer Form, hauptsächlich für Beschwerden: [Diemucha]).

5.4. Anbieter-Agenten (Organisation, Website)

Auch für die Anbieter für Informationen können Agenten von Vorteil sein (speziell in Bezug auf E-Commerce: siehe nächster Abschnitt). Hierbei handelt es sich meist um stationäre Agenten welche von Kunden-Agenten besucht werden. Auf zwei sehr unterschiedliche Teilbereiche soll hier exemplarisch besonders eingegangen werden: die Wartung einer Webseite in umfassender Hinsicht und die Organisation von Veranstaltungen insbesondere Lehrveranstaltungen. In beiden Fällen handelt es sich um Agenten, welche vollkommen stationär sind und keiner Mobilität bedürfen.

5.4.1 Aufgabenbereiche

Bei der Vorbereitung von Lehrveranstaltungen auf der Universität (aber analog auch auf andere Lehrveranstaltungen und mit Adaptionen auf allgemeine Veranstaltungen anwendbar) können Agenten verschiedene Aufgaben übernehmen. Diese Bereiche beruhen besonders auf Erfahrungen mit mehreren Kursen, welche (bis auf vereinzelte Elemente) ausschließlich über das Internet abgehalten wurden ([I-LVA], [CBT-LVA], [TW-LVA]). Ein Problembereich bei der Abhaltung von LVAs über das Internet ist, wie Gruppen von Studenten gebildet werden können. Dies beginnt mit der Anmeldung, bei welcher Informationen über die Studenten abgefragt werden, welche anschließend teilweise auf öffentlichen Webseiten präsentiert werden. Dies kann zwar über eine Datenbank und Skripts auch durchgeführt werden, doch ist dann eine besondere Zugangskontrolle (welche Studenten aufgenommen werden können und welche nicht; überprüfen von Voraussetzungen wie abgelegten Prüfungen; etc.) nicht oder nur sehr eingeschränkt möglich, da Skripts nur Daten abspeichern und später präsentieren, jedoch keine selbständigen länger dauernde Aktionen durchführen können: Die Rückmeldung an den Benutzer muß sofort erfolgen.

Insbesondere bei Seminaren ist auch die Zuteilung der Themen nicht sehr einfach zu bewerkstelligen: Manche Themen sind überlaufen, andere werden eher ignoriert. Hier eine optimale Zuordnung zu finden, kann ebenso durch Agenten unterstützt werden. Allgemein handelt es

sich hier um Tätigkeiten, welche kein besonders hohes Maß an Intelligenz, jedoch höhere Autonomie, erfordern und daher auch für einfachere Agenten geeignet sind.

Ein weiterer Aspekt ist die Erstellung von Link-Listen welche als Ansatzpunkte für Studenten zu den Themen diesen könnten. Dies ist analog zur Informationssuche zu sehen. Hierbei besteht jedoch ein viel geringerer Zeitdruck, da diese Aufgabe schon sehr früh gestartet werden kann. Es ist daher für Agenten ohne weiteres möglich auch komplexe und langdauernde Auswertungen durchzuführen. Somit können den Studenten bereits detailliertere Informationen als Startpunkte zur Verfügung gestellt werden, während sie gleichzeitig auch lernen können mit unvollständigen (und teilweise eventuell auch falschen) Informationen umzugehen. Deren Feedback kann wiederum dem Agenten zugeführt werden, sodaß dieser bei späteren ähnlichen Aufgaben u. U. bessere Ergebnisse liefert.

Auch die allgemeine Betreuung (als sehr einfaches Beispiel siehe [Humbert 2000]) einer Webseite kann von Agenten profitieren. Hierbei handelt es sich hauptsächlich um Aufgaben, welche automatisch in regelmäßigen Abständen durchzuführen sind und eine übergreifende Kontrolle bzw. Überarbeitung ausführen:

? Suche nach toten Links: Gegenüber einer manuell ausgelösten Suche besteht hier der Vorteil darin, daß z. B. kurze Ausfälle ignoriert werden und die Behandlung teilweise automatisch erfolgen kann (Linklisten: Verschieben des Links an das Ende, z. B. in eine eigene Kategorie „Links no longer active“, …).
? Benachrichtigung von Benutzern bei bestimmten Änderungen auf den Webseiten. Dies ist eine Art von Personalisierung. Der Service kann über die eigene Webseite hinaus ausgedehnt werden indem auch externe Websites untersucht werden.
? Filtern von Beiträgen: Auf Diskussions-Webseiten („Webboards“) finden sich unter Umständen auch unpassende Beiträge, welche teilweise von Agenten automatisch herausgesucht und gelöscht/gesondert markiert/verschoben werden können. In diesem Fall braucht der Betreiber der Webseite nicht mehr alle neuen Beiträge durchlesen, sondern kann sich mit der Vorauswahl durch den Agenten begnügen.
? Einfaches Groupware-Tool: Aus der Kombination von Benachrichtigungen bei Änderungen und der automatischen Sammlung und Präsentation von Informationen läßt sich eine Art einfaches Groupware Tool bilden. Der Vorteil ist, daß Benutzer keine zusätzliche Software installieren müssen (nur der Anbieter) und die Bedienung relativ leicht ist (der Agent übernimmt einige Aufgaben und kann für die restlichen Hilfestellungen geben). So

ist etwa die Benachrichtigung der anderen Gruppenmitglieder bei Änderungen der gemeinsamen Webseite bzw. Integration von neuen Elementen einzelner Mitarbeiter auf diese Weise ohne händische Interaktion möglich.

5.4.2 Schwierigkeiten

Die starke Integration mit einem Webserver bringt auch Probleme mit sich: Entweder befindet sich das Agentensystem direkt auf dem Server, dann werden Rechenzeit und andere Ressourcen dafür benutzt, oder es befindet sich auf einem anderen Rechner. In letzterem Falle bedeutet dies jedoch, daß der Zugriff entweder über das Netzwerk erfolgen muß oder gar über den Webserver. In allen Fällen entsteht dadurch eine mehr oder weniger starke Zusatzbelastung für den Server. Erfolgt der Zugriff über den Webserver, so ist auch bei der Erstellung von Zugriffsstatistiken hierauf Rücksicht zu nehmen.

Einen weiteren Problempunkt stellt die Interaktion des Agenten mit den Webseiten dar: Wie können Informationen daraus gewonnen werden? Zusätzlich zu den oben unter 5.3.2 erläuterten Problemen ist hier jedoch zusätzlich zu beachten, daß die Seiten geändert werden können sollten. Für die Auswertung ist daher ein Datenmodell erforderlich, das eine äquivalente Transformation der Webseite darstellt. Ansonsten würde auch eine Zusammenfassung mit Ausscheiden unwichtiger Elemente ausreichen, welche sowohl leichter zu erstellen, als auch auszuwerten ist. Noch eine Stufe schwieriger ist die Auswertung bzw. Veränderung von dynamisch generierten Webseiten: Die abgerufene Seite zu verändern ist sinnlos, vielmehr muß entweder aus der erstellten Seite und dem Skript zusammen ein Schluß gezogen werden (extrem schwierig, selbst für einfache Aufgaben), oder ein direkter Zugriff auf die zugrundeliegende Datenbasis erfolgen. In dieser Hinsicht ist eine Modellierung in XML besonders nützlich, da Daten-Modifikation problemlos und unabhängig von der Präsentation (XSL) ist.

Bei der Organisation von Veranstaltungen ist es notwendig, dem Agenten ausführliche Informationen zu geben, damit er sinnvolle Ergebnisse liefert. Dies bedeutet entweder eine langwierige Konfiguration oder eine lange Einarbeitungszeit bei selbstlernenden Agenten. Dies gilt insbesondere bei der Suche nach Informationen zu festgelegten Themen, da hierfür der Agent auch Informationen über das Sachgebiet benötigt.

5.4.3 Lösungsansätze

Das Problem der Zusatzbelastung für den Webserver läßt sich zumindest teilweise dadurch lösen, daß der Agent Zugriff auf die Auslastungsinformation des Servers und des Webservers

erhält. Er kann dann seine Tätigkeit auf entsprechende Zeiten verlagern (z. B. während der Nacht). Dies ist jedoch nur für nicht-dringende Angelegenheiten möglich. Bei POND wird dies durch den integrierten Timer unterstützt: Ein Agent kann sich zu einem vorgegebenen Zeitpunkt „aufwecken“ lassen.

Um die Auswertung der Webseiten zu erleichtern kann eine Transformation von HTML ([HTML-Spec]) nach XML ([XML-Spec]) durchgeführt werden (falls die Seiten nicht ohnehin auf XML basieren und mit Stylesheets oder Preprozessoren nach HTML konvertiert werden). Dadurch wird die Auswertung erleichtert und sie ermöglicht auch eine Rücktransformation. So könnte eines der bestehenden Objektmodelle für HTML/XML verwendet werden (wie etwa DOM; Document Object Model [DOM]). Eine solche Vorgangsweise bedeutet unter Umständen einen Zugriff direkt auf die Quelldateien unter Umgehung des Webservers (was auch Auslastungsprobleme verringert) und sie kann in eingeschränktem Ausmaß auch für dynamische Webseiten eingesetzt werden. Werden die Daten aus einer XML-Datei entnommen bzw. solche für verschiedene Benutzer unterschiedlich formatiert, so kann durch deren Modifikation durch Agenten auch bei dynamischen Seiten eine Anpassung erfolgen. Bei hoher Qualität der Identifizierung des Agenten (etwa durch Zertifikate) ist es Agenten im System POND auch erlaubt direkt auf das Dateisystem zuzugreifen (wobei auch eine Beschränkung auf einzelne Pfade/Dateien bzw. auf Lesen/Schreiben möglich ist). Dadurch kann auch der sensible Bereich des direkten Zugriffs auf Dateien eines Webservers für Agenten freigegeben werden.

Um die Konfiguration zu vereinfachen existieren zwei große Ansätze: Einerseits die Intelligenz der Agenten zu verbessern, andererseits die Agenten mit Lernfähigkeit auszustatten (was auch als ein Subelement des ersten Ansatzes gesehen werden kann). Da beides nicht sehr einfach ist und weitere Rechenzeit benötigt, die in diesem Falle besonders knapp ist, kann nur auf eine ausführliche Konfiguration zurückgegriffen werden. Auch hier könnten wieder die bereits oben angeführten Konzepte der Hilfestellung durch den Agenten und der inkrementellen Konfiguration angewendet werden.

5.5. Anbieter-Agenten (E-Commerce)

Ebenso wie unter 5.4.2 dargelegt, stellt sich auch im E-Commerce für Agenten von Anbietern wieder das Problem, Daten aus Webseiten zu extrahieren. Zusätzlich ist jedoch (insbesondere bei B2B E-Commerce) die Kommunikation mit anderen Anbietern von Bedeutung. Diese kann entweder über ein „privates“ Protokoll, oder unter Verwendung von EDI oder eines anderen

Standards durchgeführt werden. Auf Grund der großen Komplexität von EDI bestehen Bestrebungen statt dessen XML für den Datentransfer zu verwenden. Es existieren viele Standardisierungs-Ansätze (siehe etwa [Glushko et al. 1999] für einen Überblick über anwendungsspezifische Sprachen), von denen ein besonderer ebXML ([ebXML]) ist. Hierbei wird insbesondere auf eine vollständige Integration in die Geschäftsprozesse Wert gelegt und nicht nur die Syntax von den auszutauschenden Nachrichten festgelegt, sondern auch die Semantik. Zu deren Bedeutung siehe [Smith/Poulter 1999], welche die Bedeutung von Ontologien hervorheben, die sie als Vokabular mit genau festgelegter Bedeutung verstehen. Auch wird ausgeführt, daß mit XML alleine nicht das Auslangen gefunden werden kann, sondern daß zusätzlich noch Schemata notwendig sind (etwa zur Definition von Datentypen und zum Festlegen von weiteren Einschränkungen). EbXML findet als Übertragungsprotokoll Einsatz beim implementierten Sport-Portal (siehe auch 6.6.2).

5.5.1 Aufgabenbereiche

Konkret können Agenten für Anbieter von Produkten, Dienstleistungen oder Informationen im Bereich von E-Commerce die folgenden Aufgaben übernehmen:

? Kreditprüfung: Bei der Überprüfung der Kreditwürdigkeit von Kunden können verschiedene Aspekte integriert werden wie etwa die Auskunft von Banken, Kreditschutzverbänden oder auch bisherigen Bestellungen. Zu berücksichtigen sind hier Datenschutz-Vorschriften (siehe 3.3.6). Da es sich um wichtige Entscheidungen handelt, ist die Fehlertoleranz von besonderer Bedeutung: Die Informationen müssen auch auf verschiedenen Wegen beschafft werden können. Auch bei (teilweise) falscher Schreibweise von Informationen (z. B. Adresse) sollten die Daten aufgefunden werden. Hierzu gehört auch die automatische Prüfung der Ediktsdatei, in welcher Insolvenzen, Ausgleiche, und Schuldenregulierungsverfahren aufgeführt sind. Aus praktischen Gründen (Anschlag an der Gerichtstafel am Wohnsitz/Sitz des Schuldners) wurde bisher auf eine derartige Prüfung verzichtet, sofern nicht eine Auskunft eines Kreditschutzverbandes eingeholt wurde. Nachdem diese Daten nunmehr im Internet veröffentlicht werden ([Edikte]), ist es auch für kleinere Betriebe durch den Einsatz von Agenten möglich, eine automatische Prüfung bei jeder Bestellung bzw. jedem Auftrag vorzunehmen.
? Umgehungsprüfung: Da es sich bei E-Commerce vielfach um eine rein elektronische Abwicklung der Geschäftsbeziehung handelt, ist eine Ausnutzung von Fehlern oder Schwächen besonders gefährlich. Agenten können dazu eingesetzt werden besondere Muster bei

Bestellungen einer Person (oder auch übergreifend) herauszufinden. So wird etwa die Werbung von Neukunden oft mit Geschenken belohnt. Ein beliebter Trick hierbei ist, ein anderes Familienmitglied zu werben, welches oft auch unter der selben Adresse wohnt. Im E-Commerce ist dies noch viel anfälliger, da vielfach nur E-Mail Adressen beworben werden. Welche Person hinter dieser tatsächlich steht ist nicht einfach festzustellen. Wird aber die Bestellung zwar von einem „geworbenen" Neukunden durchgeführt, die Lieferadresse und auch die Rechnungsadresse jedoch einem bestehenden Kunden gleicht, so ist dies ein guter Hinweis auf eine versuchte Umgehung. Diese und ähnliche Muster können von Agenten geprüft werden, wobei in manchen Fällen nicht einmal eine genaue Kenntnis der Umgehungsmethode bekannt sein muß: Aus allgemeinen Datenkorrelationen lassen sich u. U. bereits Schlüsse ziehen und es kann der Betreiber darauf hingewiesen werden, gewisse Vorgänge genauer zu untersuchen.

? Virtueller Kundenberater/Verkäufer: Agenten können sowohl für andere Agenten als auch für Menschen Zusatzfunktionalität beim Verkauf anbieten. Dies kann etwa das Angebot passender Zusatzprodukte (Cross-/Up-Selling) als auch die Identifikation bzw. Konfiguration des genauen Kundenwunsches sein (z. B. Zusammenstellung eines maßgeschneiderten Computers oder eines Autos).

? Hilfestellung für den Benutzer: Weniger auf den Inhalt der Produkte oder der Bestellung bezogen, können Agenten auch bei der praktischen Durchführung behilflich sein, indem sie den Bestellvorgang erläutern, dementsprechende Hilfe leisten, oder auch oft gestellte Fragen bzw. Probleme weiterleiten, sodaß diese in eine FAQ-Liste aufgenommen werden.

? Informationssammlung: Wie allgemein bei Anbietern und Kunden können Agenten Informationen aus anderen Webseiten extrahieren. Besonderes Augenmerk ist hier auf das Wettbewerbsrecht und Werbebeschränkungen zu legen. Die Praxis den Preis des Artikels der Konkurrenz anzuführen (wurde etwa früher bei Online-Buchshops durchgeführt) ist eine Möglichkeit die jedoch, so weit es möglich ist, zu verhindern versucht wird.

5.5.2 Schwierigkeiten

Schwierigkeiten in dem hier beschriebenen Bereich sind teilweise bei der Informationssammlung zu beobachten (siehe oben). Andererseits existieren spezifische Problemfelder:

? Bei der Suche nach nicht genauer spezifizierten Problemen oder Umgehungsversuchen kann keine Erfolgsgarantie gegeben werden. Auch wird sehr viel Zeit und Rechenleistung benötigt. Eine vollkommen freie Suche ist daher mit hohen Unsicherheiten behaftet. Es

sollte daher eine Einschränkung bzw. gezielte Suche nach möglichen Problemen mit zumindest eingegrenztem Bereich erfolgen.

? Die Behinderung von Agenten der Konkurrenz gewinnt hier eine besondere Bedeutung. Automatischer Preisvergleich (nur wenn die Konkurrenz teurer ist; für Kunden bestimmt) bzw. das Ausspähen von Lagerständen oder Lieferzeiten (als interne Information) kann hier zur Bedrohung werden, da es nun automatisch durchgeführt werden kann, ohne daß zusätzliche Arbeit notwendig ist oder Verdacht erregt wird.

? Durch die Verwendung von Agenten steigt auch die Anfälligkeit gegenüber Fehlern dieser Agenten. Da es sich um komplexe Software handelt, ist die Wahrscheinlichkeit für einen Programm- oder Bedienungsfehler höher. Die Bedeutung dieses Punkts steigt noch mehr, da intelligente Agenten auch mit wichtigeren und damit höherwertigen Aufgaben betraut werden sodaß der Schaden im Fehlerfall eher höher ist.

? Menschen kommunizieren nicht gerne mit Maschinen, wenn diese versuchen, sich als Menschen auszugeben. Beim Einsatz von Agenten als Kaufberater oder in unterstützender Funktion ist darauf zu achten. Ansonsten kann es leicht zu deren gänzlicher Ablehnung kommen (was das Gegenteil der Intention, dem Benutzer zu helfen, zum Ergebnis hat).

5.5.3 Lösungsansätze

Da bereits eines der Elemente von Agenten die Robustheit ist muß bei wichtigen Anwendungen besonderes Augenmerk darauf gelegt werden, z. B. indem mehrfache Lösungswege und Alternativen verwendet werden. Auch sollte eine Plausibilitätsprüfung für die Ergebnisse integriert werden. Hilfreich insbesondere für den Anfang ist, wenn Agenten ihre Arbeit fast vollständig erledigen und am Ende nur noch eine kurze Bestätigung einholen. Hierbei wird das Ergebnis präsentiert und der Benutzer muß lediglich mit einer einzigen Aktion bestätigen. Die eigentliche Durchführung übernimmt wieder der Agent. Ergeben sich keine Probleme so kann der Benutzer später diese Sicherheitsabfrage abschalten und dem Agenten vollständig die Kontrolle überlassen. Dies ist naturgemäß nur für relativ seltene Tätigkeiten durchführbar; bei oftmaligen Aktionen ist auch eine Aggregation (wobei mehrere Einzel-Aktionen auf einmal bestätigt werden) oder ein Trainings-Modus denkbar (der Agent vergleicht die von ihm potentiell erzielten Ergebnisse mit den tatsächlich ohne seine Einwirkung durch Benutzer-Aktionen erreichten).

Um unerwünschte Agenten fernzuhalten ist einerseits ein Sicherheitssystem geeignet (siehe 4.2 zur Erläuterung des bei POND implementierten Konzeptes). Andererseits kann auch z. B. eine

Limitierung von Anfragen stattfinden. Ein Benutzer wird eine Limitierung auf 50 Produkte pro Tag vielfach nicht einmal bemerken, während eine automatisierte Gesamtabfrage hierdurch zumindest stark behindert wird. Auch solche Berechtigungen sind in POND möglich (siehe etwa die Beschränkung des Festplatten-Speichers auf eine bestimmte Byte-Anzahl). Emulieren Agenten jedoch Benutzer-Aktionen, so werden keine wirklichen Abwehr-Strategien mehr möglich sein ohne nicht auch zumindest manchmal „echte“ Benutzer zu treffen. Erfolgen die Anfragen durch Agenten, so kann auch die Identifizierung der Herkunft in eingeschränktem Maße Anwendung finden. Wird von den Agenten ein anerkanntes Zertifikat verlangt (wie etwa bei POND für besondere Berechtigungen vorgesehen), so müssen eine Vielzahl von natürlichen Personen zusammenarbeiten, um als Gruppe Limitierungen umgehen zu können. Eine Einzelperson wird in diesem Fall nur dann behindert, wenn sie anonym auftreten will. Alle solchen Abwehrmaßnahmen sollten immer nur mit Vorsicht verwendet werden, da durch sie auch normale, d. h. nicht böswillige, Agenten in ihrem Verhalten gestört werden könnten (wie etwa Vermittler-Agenten oder Agenten von Zwischenhändlern; diese verursachen naturgemäß viele Anfragen und ein höheres Datenaufkommen).

In manchen Fällen wird es auch sinnvoller sein, wenn die Agenten mehr im Hintergrund tätig sind ohne direkt in Erscheinung zu treten. Dies verhindert auch, daß durch eine zu menschliche Darstellung falsche Vorstellungen geweckt werden, welche dann nicht erfüllt werden können ([Tsakiridou 2001]). Eine Alternative hierzu ist Agenten eher als Comic darzustellen: Es ist besser wenn der Kunde über sie lacht, als wenn er sich darüber ärgert.

6. Ausgewählte Fallbeispiele, implementiert mit POND

In diesem Kapitel werden mehrere Agenten bzw. Systeme von Agenten vorgestellt, welche sowohl zum Test des Systems, als auch zur praktischen Nutzung erstellt wurden. Bei den einzelnen Agenten wurde auf verschiedene Punkte Wert gelegt. So werden bei Beispielsagenten die Sicherheitsfunktionen nicht verwendet, um die Darstellung der grundlegenden Funktionen (Nachrichten-Verarbeitung, Mobilität) einfach zu halten oder den Blick nicht zu verstellen.

Bei den Systemen von Agenten wurde hingegen sowohl auf ein funktionelles Benutzer-Interface, als auch auf umfassende Sicherheitsvorkehrungen geachtet. Dies bedingt jedoch, daß eine relativ komplexe Installation erforderlich ist, bevor diese Agenten verwendet werden können: Es müssen Schlüssel erzeugt, der Code signiert, und Identitäten erzeugt werden.

Es werden hier nur die Funktion bzw. besondere Aspekte erläutert und teilweise Code-Fragmente angeführt, da der gesamte Programmcode zu lang wäre. Von einigen einfachen Agenten befindet sich jedoch der komplette Code im Anhang.

6.1. Basis-Agenten

Dieser Abschnitt enthält drei einfache Agenten, welche zwar zur Demonstration dienen, aber teilweise auch später in Systemen Verwendung finden.

6.1.1 Beeper

Dieser sehr einfache „Agent“ (seine Aufgabe alleine reicht wohl noch nicht für eine Qualifikation als echter Agent aus) hat lediglich eine Funktion: Klickt der Benutzer auf einen Button der Benutzerschnittstelle (Abbildung 57, Abbildung 58) so wird ein kurzer Ton ausgegeben. Dies kann auch durch einen anderen Agenten über eine Konversation ausgelöst werden (auch hier noch keine echte Agenten-Eigenschaft, da jede Anforderung jedes Agenten ohne Prüfung bearbeitet wird). Hierzu existiert ein zweiter „Agent“, welcher beim Klicken eine derartige Nachricht (Broadcast) abschickt. Eine zweite Version des ersten Agenten stellt kein UI zur Verfügung, sondern wartet nur auf eingehende Nachrichten.

Im folgenden ist der (hier komplette) Programmcode des Beeper-Agenten, gekürzt um Kommentare, Imports, sowie das enthaltene Testsystem (automatische Erzeugung des Agenten

beim Start des Agentensystems), angeführt. Der Programmcode der anderen Agenten, sowie der Konversation und der zugehörigen Nachricht befindet sich im Anhang.

```
package BaseAgents.Beeper;

public class BeeperAgent extends GUIAgentBase implements ActionListener
{
     protected transient JButton beep=new JButton("Beep!");

     public BeeperAgent()
     {
          registerConversation(new BeepConversation(this));
     }

     protected void showDialog()
     {
          doBeep();
     }

     protected void doBeep()
     {
          Toolkit.getDefaultToolkit().beep();
          logMessage("Beep!");
     }

     protected javax.swing.JPanel createVisualization()
     {
          javax.swing.JPanel visualization=new javax.swing.JPanel();
          visualization.setLayout(new BorderLayout());
          visualization.setBorder(new javax.swing.border.EtchedBorder());
          javax.swing.JPanel pane=new javax.swing.JPanel();
          pane.setLayout(new GridBagLayout());
          pane.setBackground(Color.white);
          pane.setOpaque(true);
          javax.swing.JLabel label=new javax.swing.JLabel(
                              "Beeper ("+getIdentity().getAgentName()+")");
          GridBagConstraints cons=new GridBagConstraints();
          cons.insets=new Insets(10,10,10,10);
          cons.fill=GridBagConstraints.HORIZONTAL;
          cons.anchor=GridBagConstraints.CENTER;
          cons.weightx=1.0;
          cons.weighty=0.0;
          cons.gridx=0;
          cons.gridy=0;
          pane.add(label,cons);
          cons.fill=GridBagConstraints.NONE;
          cons.ipadx=10;
          cons.ipady=10;
          cons.gridy=1;
          if(beep==null)
               beep=new JButton("Beep!");
          beep.setHorizontalAlignment(SwingConstants.CENTER);
          beep.addActionListener(this);
          pane.add(beep,cons);
          visualization.add(pane);
          return visualization;
     }

     public void actionPerformed(ActionEvent e)
     {
          showDialog();
     }

}
```

Abbildung 57: Agentensystem mit Beeper-Agent

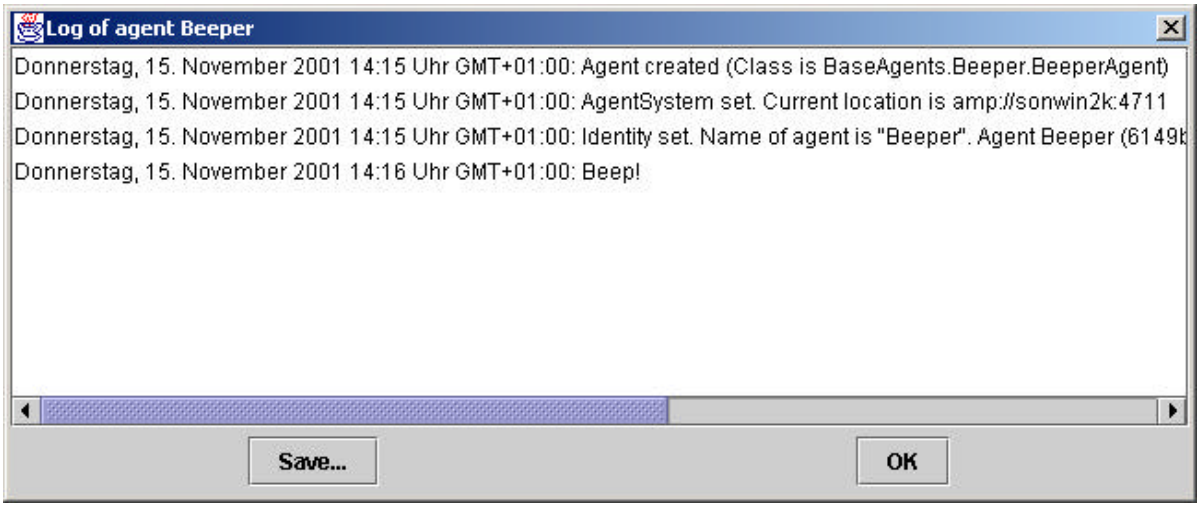

Abbildung 58: Log-Fenster des Beeper-Agenten

6.1.2 Kommando zur Rückkehr

Da Agenten lediglich lokal kommunizieren können, ist es nicht möglich eine Nachricht an einen Agenten auf einem anderen Rechner zu senden, um ihn zur Rückkehr (Verlagerung auf den Heimatrechner) aufzufordern (zum Ablauf siehe Abbildung 59). Dies ist sowohl für einen Abbruch der Tätigkeit als auch für weitere Konfigurationen, oder zur Inspektion des derzeitigen Zustandes nützlich. Der Besitzer eines Agenten kann daher (wenn er Kenntnis vom derzeitigen Server des Agenten hat) diesen auf seinen Heimat-Rechner zurückholen oder ihn zu einem dritten Server schicken.

Zu diesem Zweck muß ein Agent ein spezielles Interface implementieren und eine entsprechende Konversation registrieren. Um hierbei keine Sicherheitsprobleme zu produzieren (jeder könnte Agenten nach Hause schicken und damit die derzeitige Arbeit unterbrechen), muß die Nachricht vom Besitzer des zurückzuholenden Agenten signiert sein. Unsignierte Nachrichten (oder bei Fehlern während der Überprüfung) werden ignoriert.

Der Agent, der zum Transport der Nachricht dient (Kommando-Agent), wird auf dem Heim-Rechner (lokal) konfiguriert, indem ihm die Identität des zurückzuholenden Agenten (Ziel-Agent) und dessen Position (Ziel-Rechner) übergeben wird (Abbildung 60). Anschließend wird die Aufforderung signiert. Daraufhin transferiert er sich selbst auf den Ziel-Rechner und sendet dort die Nachricht an den Agenten ab (Abbildung 61; Kopfzeile des Agentensystem-Fensters). Anschließend wartet er auf eine Antwort (Erfolg, Fehler/Verweigerung, Agent unbekannt) und kehrt mit dieser zum Ausgangsrechner zurück (Abbildung 62). Hier ist er für eine neue Aufgabe bereit. Wurde der Befehl akzeptiert, so kehrt der zurückzuholende Agent selbständig auf den Rechner zurück, vom dem aus die Anfrage gestartet wurde. Dies muß nicht unbedingt sein Heimatrechner sein (andere Implementierungen mit Beschränkung auf den Heimat-Rechner sind möglich)!

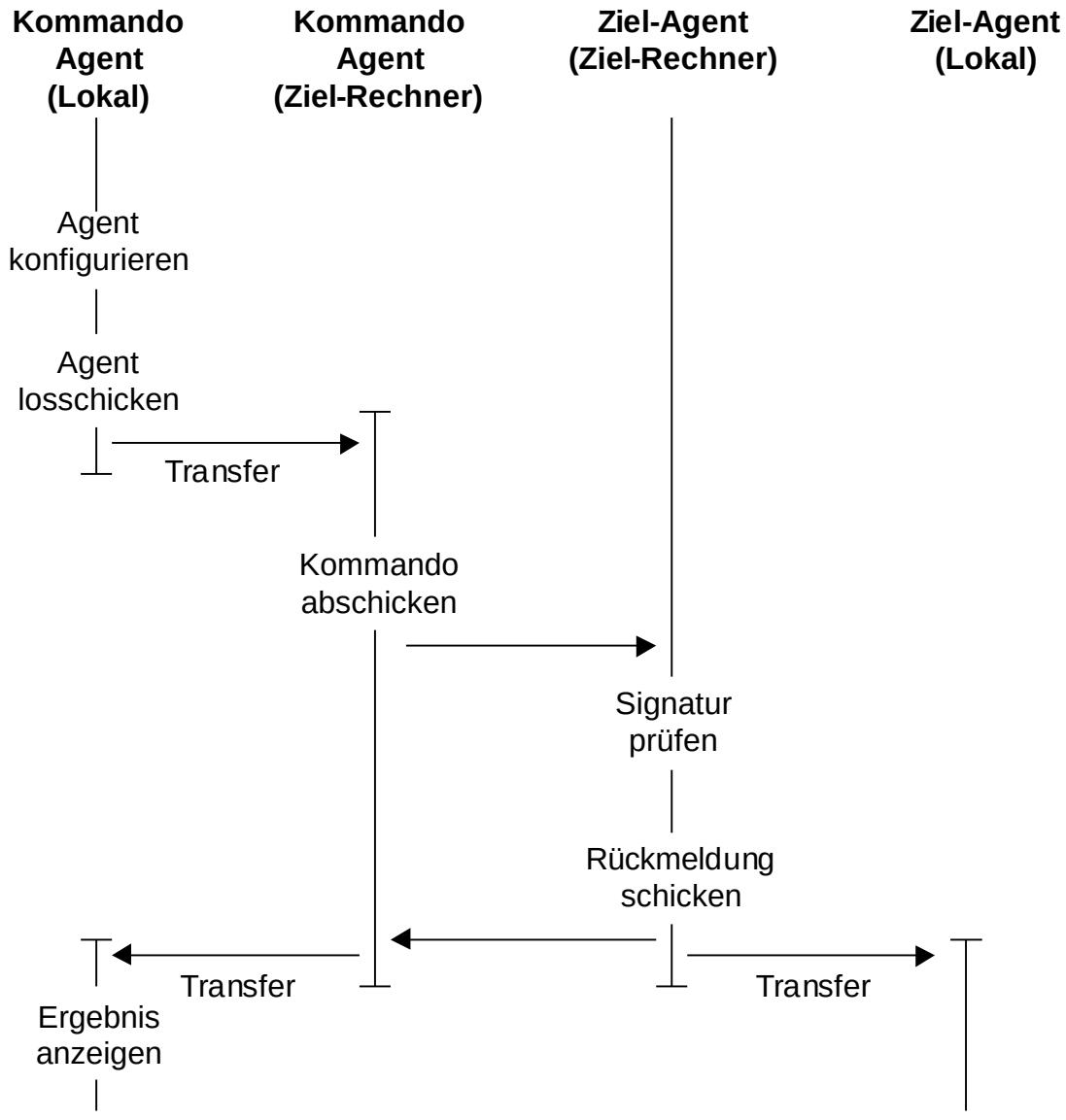

Abbildung 59: Ablaufdiagramm zur Rückkehr eines Agenten auf Kommando

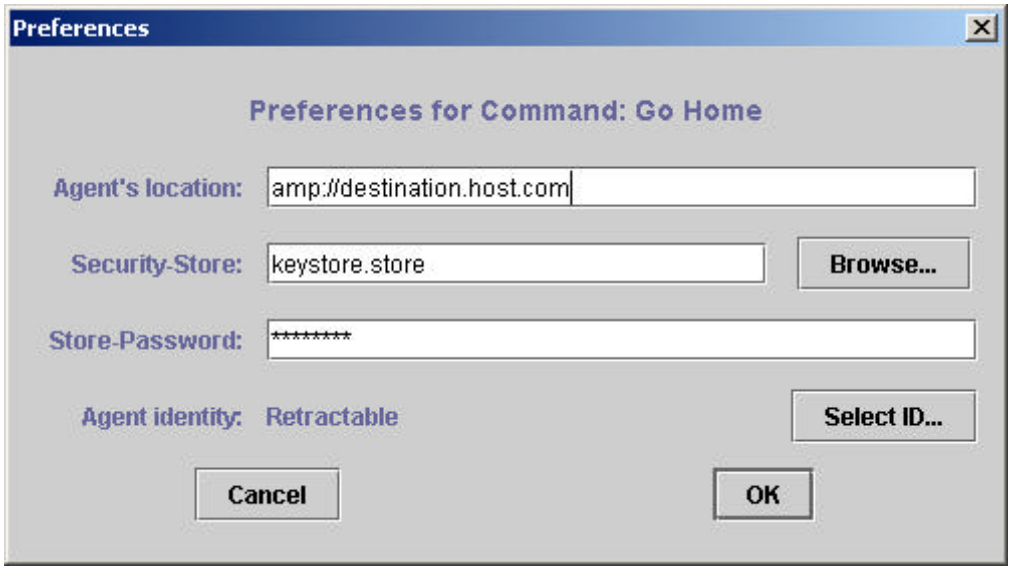

Abbildung 60: Konfigurationsfenster des Agenten zum Übermitteln des Heimkehr-Kommandos

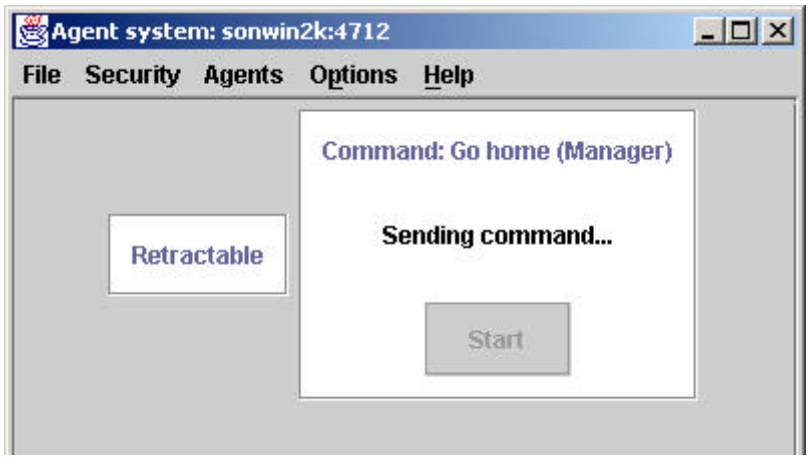

Abbildung 61: Agent auf Ziel-Host

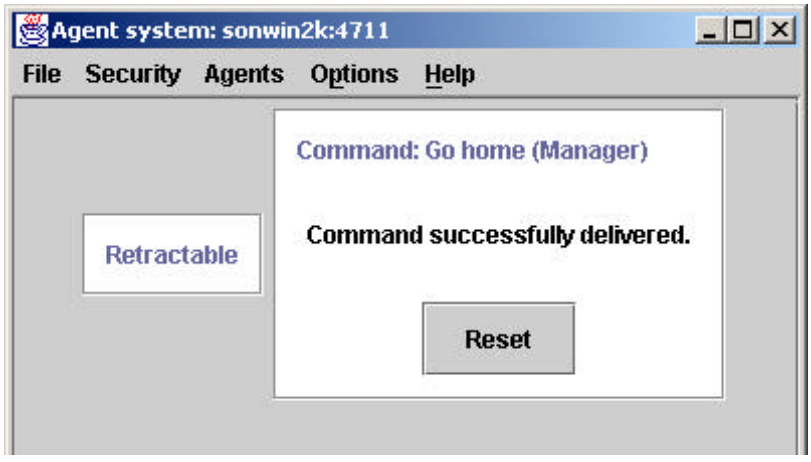

Abbildung 62: Beide Agenten auf den Ausgangs-Host zurückgekehrt

Bei dieser Anwendung besitzt das Sicherheitssystem bereits eine größere Bedeutung, da ein Agent abgeschirmt wird und so ausschließlich sein Besitzer Kommandos an ihn geben kann. Möglich wäre noch eine zusätzliche Verschlüsselung des Zieles mit dem öffentlichen Schlüssel

des Agenten selbst: Andere könnten nicht mehr herausfinden, wohin der Agent sich transferieren soll. Da jedoch der Agent das (lokale) Agentensystem benötigt, um auf einen anderen Rechner verlagert zu werden, ist eine Geheimhaltung höchstens gegen Abhören des Transfers des Kommando-Agenten möglich (was jedoch nur bei unverschlüsselter Übertragung Bedeutung besitzt).

Dieses Modell kann grundsätzlich auch verallgemeinert werden, sodaß ein Agent zum Transport beliebiger Nachrichten möglich ist. Auf diese Weise können globale Nachrichten implementiert werden solange bekannt ist, auf welchem Rechner sich der Empfänger-Agent befindet. Wegen des größeren Aufwands für den Transfer eines Agenten ist dies jedoch aus Effizienzgründen keine optimale Lösung.

6.1.3 Benachrichtigung per Message-Box

Dieser schon recht komplexe Agent (14 Klassen) dient dazu, auf einem anderen Rechner eine Messagebox anzuzeigen. Dies kann etwa zur Benachrichtigung des Besitzers verwendet werden. Zu berücksichtigen ist jedoch, daß dies nur dann sinnvoll ist, wenn der Besitzer des Agenten-Servers bekannt ist und es sich um einen interaktiven Rechner handelt (also kein bloßer Server, der nur gelegentlich oder bei Problemen verwendet wird).

Die Implementierung erfolgt derart, daß es sich um einen ständigen Service handelt: Ein Agent bleibt kontinuierlich auf dem selben Rechner und nimmt Aufträge entgegen. Zur Durchführung schickt er Sub-Agenten aus, welche sich auf den Zielrechner begeben und dort die Nachricht anzeigen (siehe Abbildung 63, Abbildung 65). Diese entscheiden nach ihrer Ankunft anhand des derzeitigen Standortes über die durchzuführende Aufgabe: Anzeige des Fensters oder Rückmeldung an den Management-Agenten.

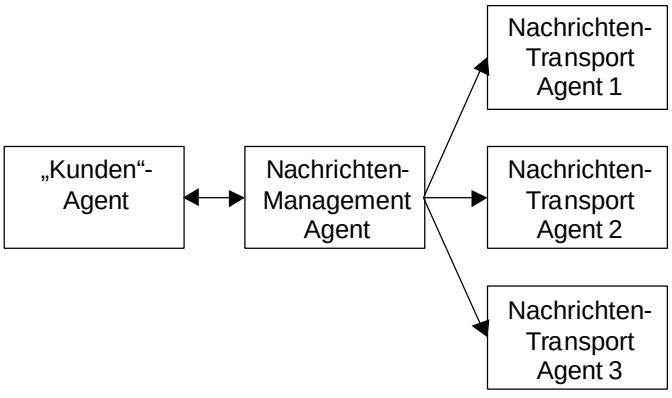

Abbildung 63: Benachrichtigung per Messagebox - Agenten-Modell

Um eine komplexe Konfiguration zu vermeiden, wurde ein Broadcast-Protokoll zum Finden von Agenten, die derartige Nachrichten/Subagenten bereitstellen, implementiert (Abbildung 64). Hierzu wird zuerst ein Broadcast ausgesendet und anschließend während einer festgelegte Zeit auf Antworten gewartet. Bei dieser Implementierung werden ausschließlich Agenten desselben Besitzers akzeptiert, sodaß keine Bezahlung für diese Leistung erfolgt. Um diese Eigenschaft sicherzustellen, müssen Teilnehmer an einer derartigen Transaktion ihre Identität durch Kenntnis ihres privaten Schlüssels nachweisen. Erst dann werden Sub-Agenten auf Anfragen hin ausgeschickt.

Für den Kunden handelt es sich um eine einfache Transaktion: Nur die Anfrage wird ausgeschickt. Eine Rückmeldung über den Erfolg/Mißerfolg der Anzeige wird nicht durchgeführt, könnte jedoch leicht integriert werden. (fire-and-forget Modell).

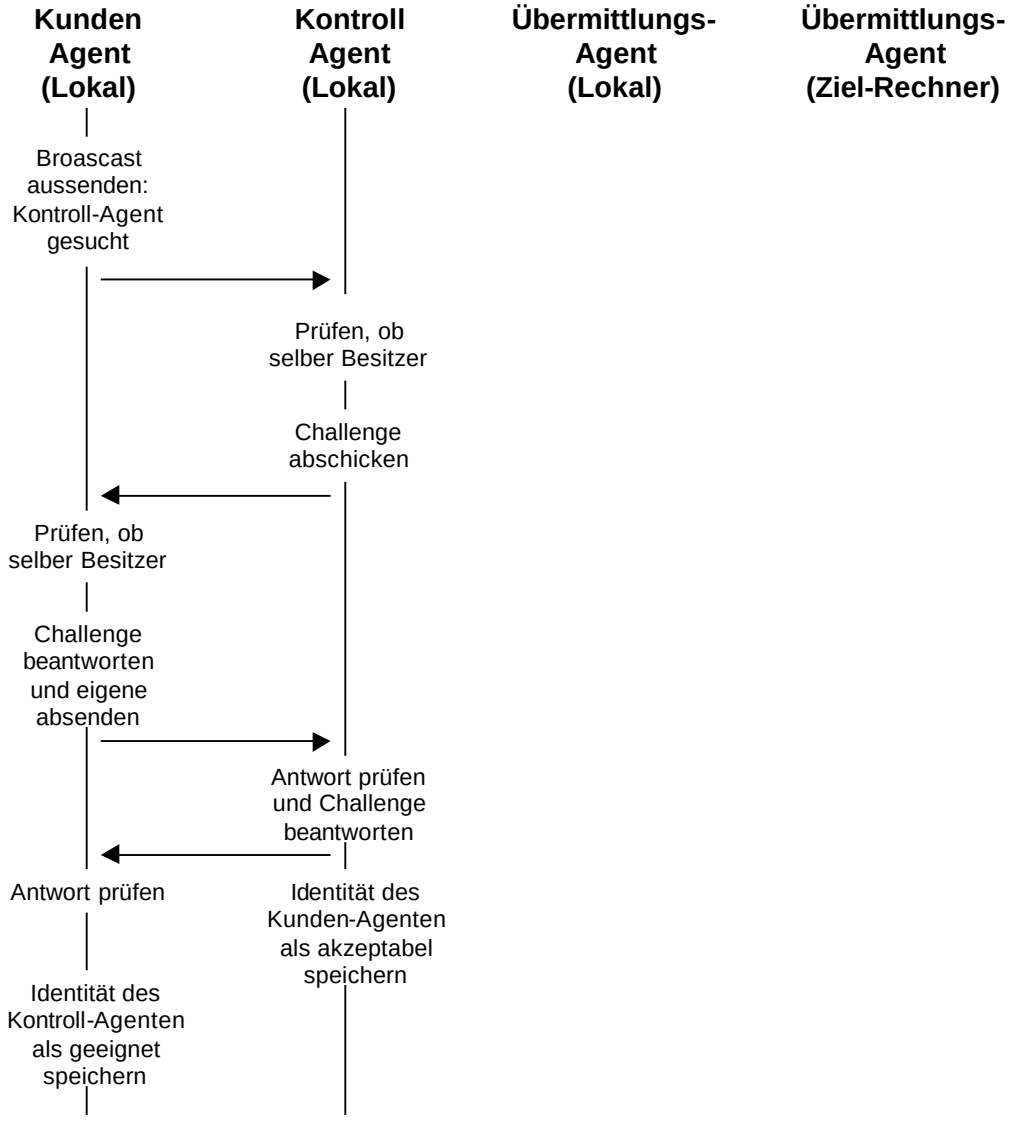

Abbildung 64: Ablaufdiagramm zur Anzeige einer Messagebox auf einem entfernten Rechner (Discovery)

Es können zu jeder Zeit beliebig viele Sub-Agenten existieren. Um den Überblick nicht zu verlieren wird eine Liste offener Aufgaben geführt, in welcher auch die Identitäten der Agenten verzeichnet sind. Für ein professionell einsetzbares System müßte noch hinzugefügt werden, daß nach einer gewissen Zeit (Timeout) entweder ein zweiter Agent nachgeschickt, oder die Aufgabe als fehlgeschlagen zurückgemeldet wird.

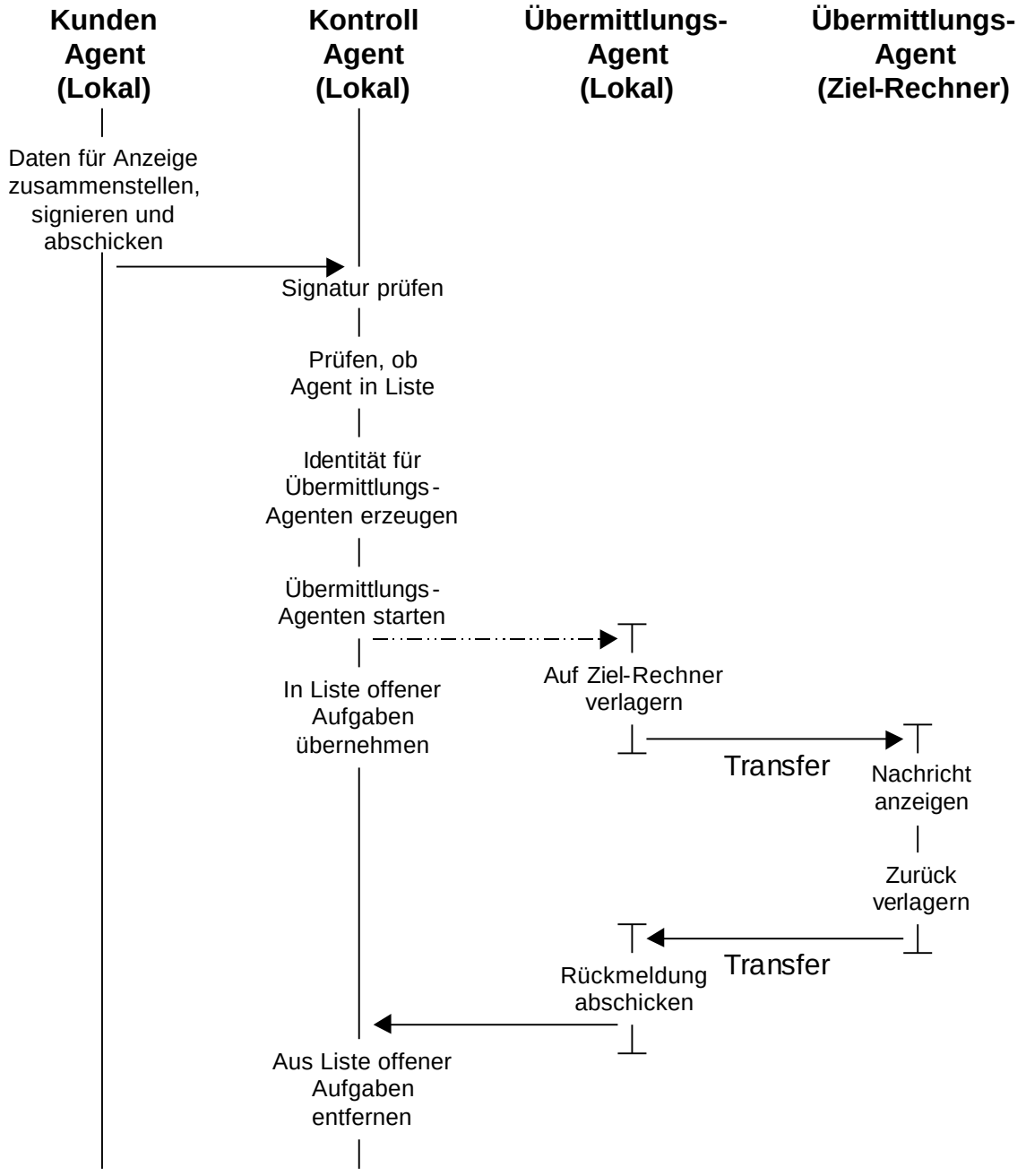

Abbildung 65: Ablaufdiagramm zur Anzeige einer Messagebox auf einem entfernten Rechner (Anzeige)

6.2. Das Demonstrations-System

Hierbei handelt es sich um ein System von Agenten, wobei zwei Agenten als „Zwischenhändler“ (Trader) Produkte von einem Produzenten auf einem Rechner, zu einem Konsumenten auf einem zweiten Rechner transportieren. Hierzu übernehmen sie ein (bis auf Seriennummer und Qualität leeres, jedoch serialisierbares) Objekt, verlagern sich auf den zweiten Rechner, liefern es dort ab, und kehren zurück zum Start-Rechner. Dies wird eine per Kommandozeile festgelegte Anzahl von Malen wiederholt. Für die Produkte wird bezahlt, doch da es sich um ein frühes System handelt, ungesichert und lediglich durch Austausch von Zeichenketten. Gleichzeitig wird auch die Qualität/Verläßlichkeit der einzelnen Agenten bzw. ihrer Produkte bewertet und im Erfahrungs-System abgelegt. Diese werden am Ende des Testlaufes als Dateien auf der Festplatte für eine Auswertung gespeichert.

Die Suche nach Produzenten (siehe Abbildung 66) bzw. Konsumenten erfolgt wieder über Broadcasts, wobei keine Speicherung des Transaktionspartners erfolgt: So kann etwa der Konsument[7] dynamisch ausgewechselt werden (oder zwei Konsumenten eingesetzt werden, wie in Abbildung 67).

Der Name des zweiten Hosts (auf den hin die Objekte zu transportieren sind) ist beim Start des Systems als Parameter anzugeben, da auf ein Benutzerinterface über die angezeigten Werte hinaus verzichtet wurde.

Abbildung 66: Demo-System: Produzenten-Seite

[7] Auswechseln des Produzenten würde zu Fehlern (kein ordnungsgemäßes terminieren) führen, da sich die Transportagenten beenden würden, ohne daß bereits alle Produkte des späteren Produzenten transportiert wären.

6.3. Konkrete Utility-Agenten

Es wurde eine Anzahl von kleineren Agenten erstellt, die konkrete Aufgaben erfüllen und auch praktisch eingesetzt werden. Bei ihnen wurde auch auf ein funktionelles User-Interface geachtet, sowie daß diese Agenten auch unter voller Sicherheit lauffähig sind (sofern ihnen Mindestberechtigungen zugestanden werden).

6.3.1 Reminder

Hierbei handelt es sich um eine Art Alarmuhr: Zu einem bestimmten Zeitpunkt wird der Benutzer mit einem vorher eingegebenen Text an ein Ereignis erinnert (siehe Abbildung 70 für die Konfiguration eines Ereignisses und Abbildung 68 für die Anzeige der aktuellen Alarme). Ereignisse können auch periodisch sein, sodaß nach Alarmierung automatisch der nächste Termin eingetragen wird.

Die Alarmierung kann auf verschiedene Weisen erfolgen (einheitlich für alle Alarme; siehe Abbildung 69):

? Anzeige einer Messagebox auf dem lokalen Rechner: Dies erfolgt durch den Agenten.

? Anzeige einer Messagebox auf einem anderen Rechner: Hierzu wird ein Hilfs-Agent zur Anzeige von Nachrichten auf anderen Rechnern verwendet (siehe 6.1.3).

? Versenden einer SMS: Dies erfolgt durch den Agenten wobei Hilfs-Klassen des Frameworks verwendet werden. Der tatsächliche Versand erfolgt über öffentliche Websites welche dieses Service anbieten.

? Versenden einer E-Mail: Auch hierzu ist ein weiterer Agent erforderlich (siehe 6.4.1) der den tatsächlichen Versand der E-Mail übernimmt.

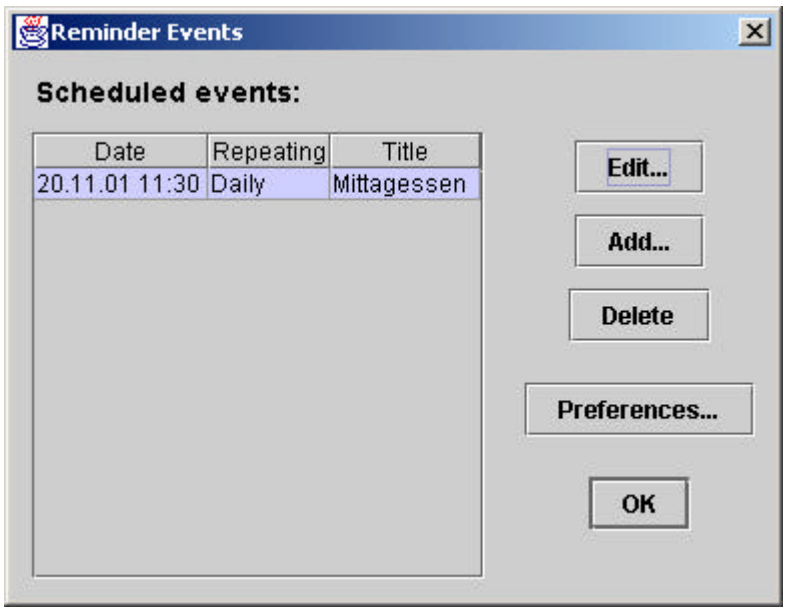

Abbildung 68: Anzeige der anstehenden Benachrichtigungs-Ereignisse

Die Implementierung erfolgt über Timer, sodaß keine Rechenzeit verbraucht wird. Zu beachten ist jedoch, daß es sich um einen Timer des Agenten und nicht des Agentensystems handelt. Wird der Agent daher persistiert, erfolgen keine Alarmierungen mehr. Da der im Framework implementierte Timer serialisierbar ist, schadet ein Transfer des Agenten auf einen anderen Rechner nicht (kann jedoch zu einer Verzögerung des Alarms führen, falls dieser währenddessen erfolgt wäre).

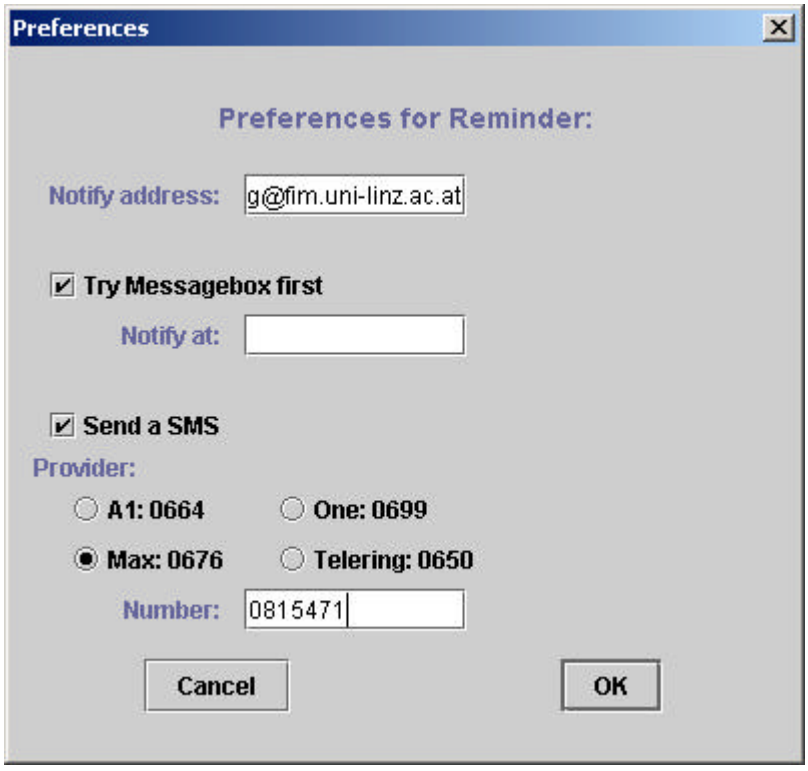

Abbildung 69: Konfigurations-Dialog: Benachrichtigungs-Möglichkeiten

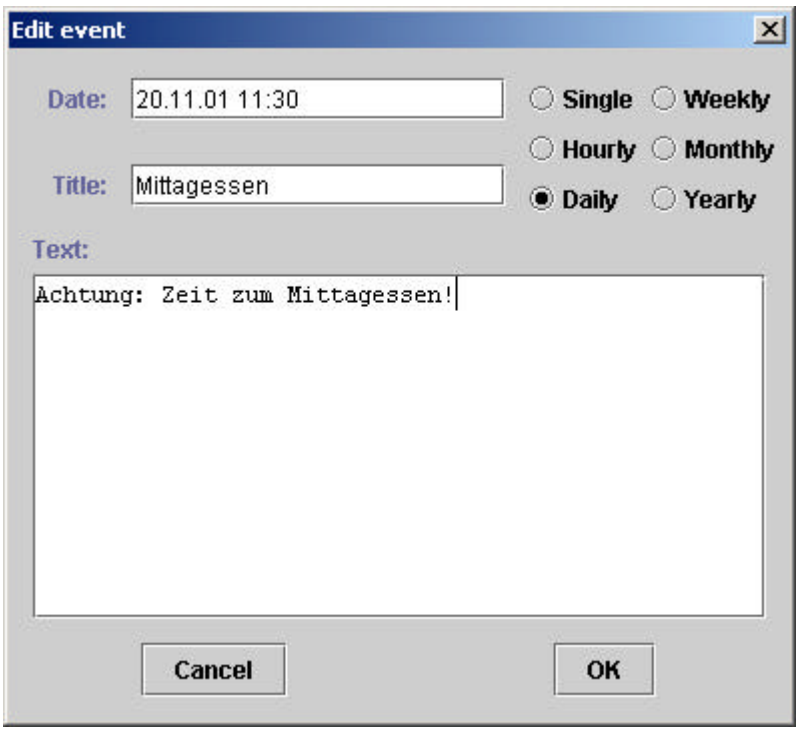

Abbildung 70: Benachrichtigungs-Ereignis editieren

6.3.2 Website-Downloader

Mit diesem Agenten kann eine gesamte Website rekursiv geladen werden, wobei zusätzliche Einschränkungen möglich sind: Alle Dateien, nur HTML-Dateien, oder alles außer HTML-Dateien (Dokumente, Bilder, Videos, …). Als Restriktion ist noch möglich, daß keine Links auf externe Server verfolgt werden. Die Grenze des Downloads wird über die Tiefe angegeben, wobei hier die Anzahl der zu verfolgenden Links gezählt wird (d. h. Tiefe 0 ist nur die Webseite selbst; Tiefe 1 die Webseite sowie alle Seiten, auf welche von der Startseite aus verlinkt wird). Die Ergebnisse können auf mehrere Arten gespeichert werden:

1. In einem lokalen Verzeichnis: Berechtigungen hierfür werden vom Agenten erworben.
2. Als ZIP-Datei eines bestimmten Namens auf dem lokalen Rechner: Auch hierfür wird die notwendige Berechtigung erworben.
3. Als ZIP-Datei in Form eines Streams (bzw. der Byte-Werte hiervon): Dies wird für die Übermittlung an andere Agenten verwendet. Gesonderte Dateien würden den Kommunikationsaufwand stark erhöhen und verkomplizieren, sodaß darauf verzichtet wurde (dafür wäre eine parallele Abarbeitung möglich).

Anforderungen zum Download können sowohl über das Benutzerinterface erfolgen (Konfiguration über Dialog; siehe Abbildung 71), bzw. über eine Konversation durch andere Agenten. In beiden Fällen wird das Benutzerinterface inaktiv geschaltet und der derzeitige Status ange-

zeigt (Details wie die derzeit bearbeiteten URLs werden mitprotokolliert und sind im Log ersichtlich). Ist eine Website in Bearbeitung, so werden keine weiteren Aufträge akzeptiert, auch nicht über Konversationen.

Die technische Durchführung erfolgt durch Starten eigener Threads für jedes zu ladende Element (HTML-Seite, Bild, etc.). Um Server nicht zu stark zu belasten, erfolgt eine zweifache Begrenzung: Es werden maximal 10 Threads gestartet (Schutz des lokalen Servers) wobei jedoch pro Quell-Server (von dem geladen wird) höchstens 2 Threads existieren dürfen. Die erste Begrenzung wird daher nur dann aktiv, falls auch externe Webseiten geladen werden.

Eine Auswertung der Seiten erfolgt nur in Hinsicht auf Image-Links und direkte Links („SRC“ und „HREF“; andere Links werden ignoriert, z. B. durch Scripts). Links auf geladene Seiten werden korrekt umgeschrieben, sodaß die geladenen Seiten auch lokal voll funktionsfähig sind. (Nunmehr) Externe Links werden ebenso angepaßt, sodaß sie auf den korrekten Server zeigen (z. B. wird bei relativen Links der Servername eingefügt). Dies erfordert eine Zweiteilung der zu ladenden Seiten in die letzte Ebene der Tiefe und alle weiter oben befindlichen: Die letzte Ebene darf erst dann geladen werden, wenn alle oberen Seiten bereits vollständig festgestellt sind, da sonst kein korrektes Setzen der Links auf bereits in höheren Schichten geladene Seiten möglich ist.

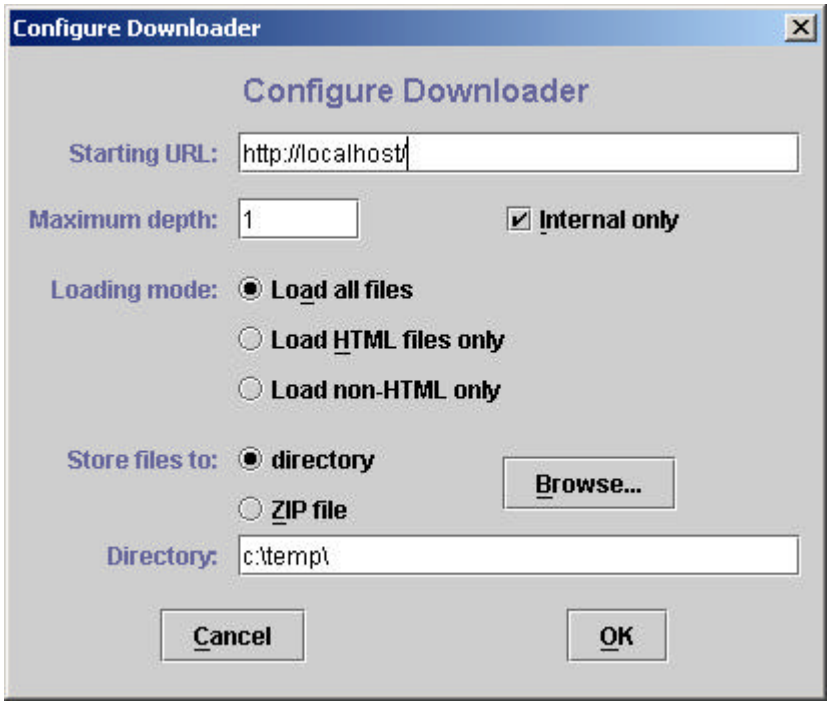

Abbildung 71: Konfigurations-Dialog: Website-Download-Agent

6.3.3 Site-Watcher

Dieser Agent dient als Verbindung eines Downloader-Agenten (siehe 6.3.2) zu den Agenten zur Mail-Bearbeitung (siehe 6.4): Es wird regelmäßig eine bestimmte (vom Benutzer zu konfigurierende; siehe Abbildung 72) Website geladen und über jede Webseite dieser Site ein Hash-Wert gebildet. Ändert sich dieser, so wird die neue bzw. geänderte Webseite in eine E-Mail umgewandelt und an subskribierte Agenten (beliebige Anzahl; sofern sie akzeptiert wurden, was hier bei identischem Besitzer der Fall ist) zur anschließenden Verarbeitung weitergeleitet. Diese können die Webseiten inhaltlich behandeln oder eine Benachrichtigung des Benutzers veranlassen, z. B. über SMS (siehe 6.4.6; siehe Beispiels-System in Abbildung 73).

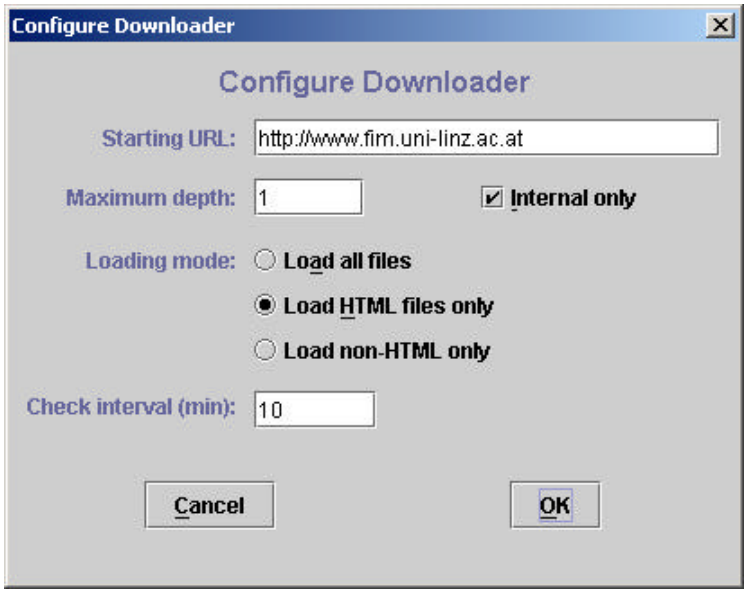

Abbildung 72: Konfigurations-Dialog: Zu überwachende Websites

Das Benutzerinterface des Agenten im Agentensystem zeigt lediglich die Anzahl der subskribierten Clients an (siehe Abbildung 73 rechts oben).

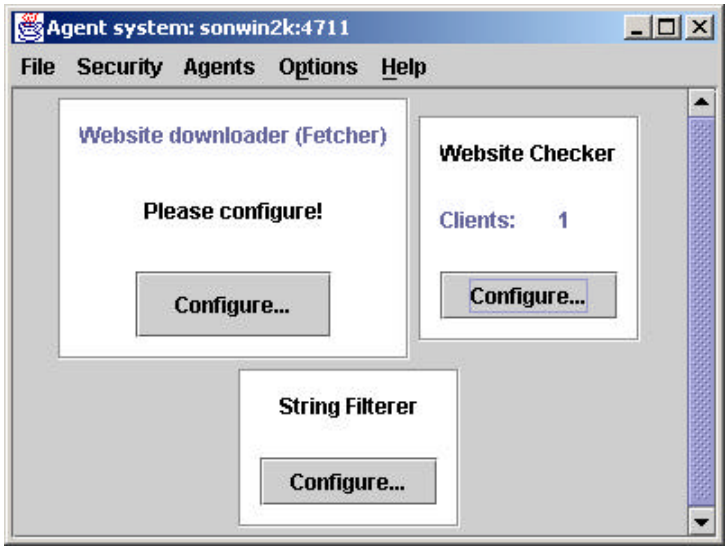

Abbildung 73: Beispiels-System von Agenten: Downloader, Site-Watcher und Benachrichtigung per SMS

6.3.4 Aktienkurs-Ticker

Dieser Agent lädt aktuelle Börsenkurse aus dem Internet und schickt sie an andere interessierte Agenten weiter. Da es sich um einen Demonstrations-Agenten handelt, ist das Symbol (z. B. „MSFT“) ebenso wie das Prüfungsintervall händisch zu konfigurieren. Das Parsen der Daten (Symbol, Kurs, Währung, Änderung, Umsatz) ist auf eine bestimmte Webseite (http://www.nasdaq.com/) ausgelegt. Auch hier wird ein Formular ausgefüllt und die Daten aus der Ergebnis-Webseite extrahiert. Der aktuelle Kurs wird auch in der Benutzeroberfläche des Agenten angezeigt (siehe Abbildung 74).

Abbildung 74: Aktienkurs-Ticker Agent mit aktuellem Börsenkurs

Der Konfigurationsdialog (Abbildung 75) zeigt hier eine Besonderheit: Am unteren Rand befindet sich die Warnleiste für Applet-Fenster. Dies bedeutet, daß dieser Agent beim Test unter verminderten Berechtigungen lief: Zwar genug, um die Webseite abfragen zu können, jedoch nicht genug, um Fenster ohne diese Warnzeile anzeigen zu dürfen.

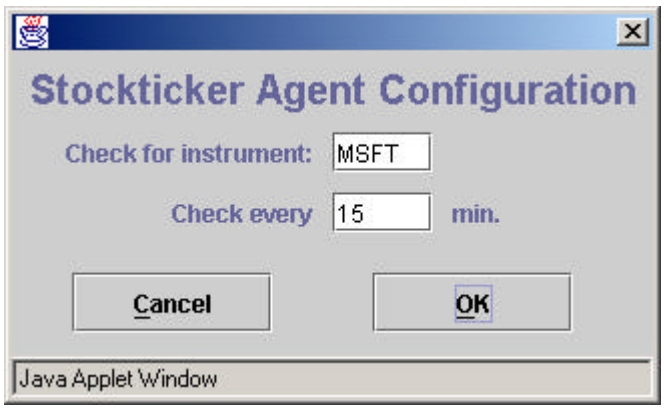

Abbildung 75: Konfigurationsdialog: Aktienkurs-Ticker-Agenten

6.4. Agenten zur Mail-Bearbeitung

Ein Framework für Agenten zur Bearbeitung von Nachrichten aller Arten wurde erstellt. Dies ermöglicht es einem Programmierer auf einfachste Art selbst Agenten zur Bearbeitung zu erstellen. Hierzu bestehen Regeln und Aktionen, welche bei Zutreffen der konfigurierten Regeln ausgeführt werden. Für einen Überblick über die Elemente siehe Abschnitt 6.4.8. Zwei kon-

krete Applikations-Agenten wurden implementiert (neben Agenten zum Abrufen bzw. Versenden von E-Mails): Einerseits Durchsuchen der Mails nach Schlüsselwörtern und anschließende Benachrichtigung des Besitzers, andererseits ein Agent zur generellen Weiterleitung. Dieses Framework bzw. einzelne Agenten daraus finden auch in anderen hier vorgestellten Systemen Anwendung.

6.4.1 Agent zum Versenden von E-Mails

Mit Hilfe dieses Agenten können andere Agenten E-Mails versenden. Es können die Felder für den Titel, den Inhalt, die Addressaten (TO, CC und BCC), die Absender-Adresse und (falls gewünscht) ein besonderer Mailserver angegeben werden. Eine Bedienung über ein Benutzerinterface (das vorhandene erlaubt nur die Einstellung des standardmäßig zu verwendenden Mailservers) ist nicht vorgesehen. Der Agent kann lediglich über Konversationen genutzt werden. Um die Identität dieses Agenten herauszufinden wird wiederum ein Broadcast-Protokoll unterstützt. Für den tatsächlichen Versand über SMTP werden die Klassen des JavaMail-Paketes von Sun sowie die konkreten Implementierungen von Drittanbietern (SMTP, NNTP; [JavaMailProviders]) verwendet.

6.4.2 Agent zum Abholen von Nachrichten

Mit diesem Agenten werden E-Mails von einem Server abgeholt (Konfiguration siehe Abbildung 76) und anderen Agenten zur Verfügung gestellt. Auch hier finden wieder die JavaMail Klassen Anwendung (jedes von diesen unterstütze Protokoll kann verwendet werden; IMAP, POP, NNTP, …).

Dieser Agent besitzt zwei Möglichkeiten, um mit anderen Agenten Kontakt aufzunehmen:

? Er kann wie die vorher beschriebenen Agenten über Broadcasts aufgefunden werden.
? Zusätzlich teilt er sein Entstehen bzw. seine Ankunft auf einem Rechner durch einen Broadcast mit, sodaß interessierte Agenten sich anschließend anmelden können.

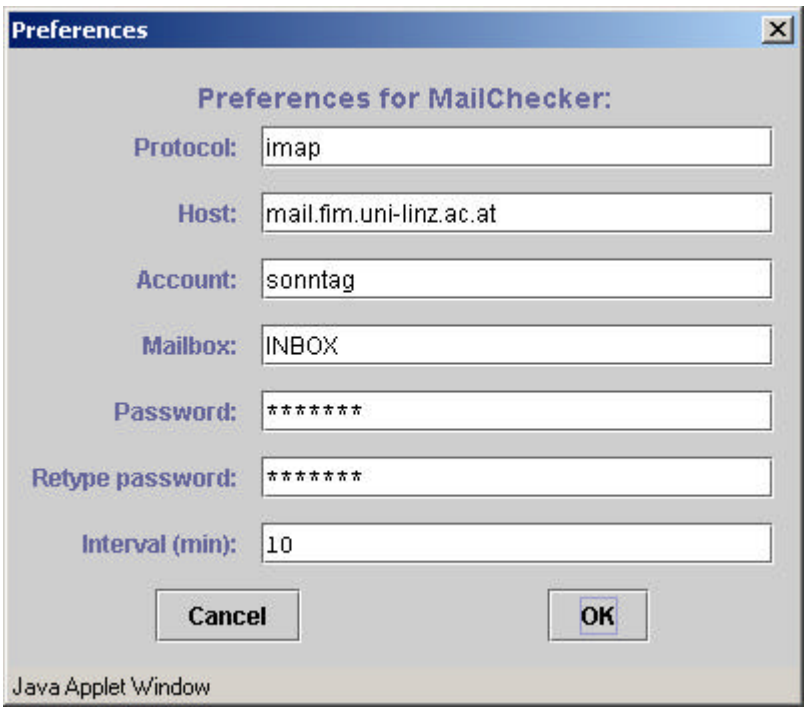

Abbildung 76: Konfigurations-Dialog: Agent zum Abholen von E-Mails

Dieser Agent stellt die Ergebnisse auf zwei verschiedene Arten zur Verfügung: Einerseits können alle neuen Mails subskribiert werden, andererseits auch lediglich deren Anzahl. Die Unterscheidung, ob es sich um einen neue Mail handelt oder nicht, wird anhand der Message-ID getroffen: Ist diese nicht gespeichert so handelt es sich um eine bis dato unbekannte Nachricht, welche anschließend weitergeleitet wird. Die ID's werden gespeichert und nicht mehr vorhandene (= Mail wurde vom Server gelöscht) entfernt.

6.4.3 Agent zur Anzeige der Anzahl der neuen E-Mails

Dieser Agent subskribiert sich bei Nachrichten-Quellen (z. B. 6.4.2) und zeigt die Anzahl der E-Mails (Abonnementtyp „SEND_MAIL_COUNT“) im Benutzerinterface an (siehe Abbildung 77). Es werden keine eigenen Tätigkeiten durchgeführt oder Dienste angeboten.

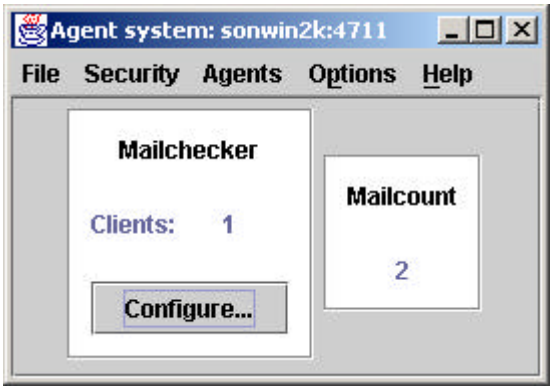

Abbildung 77: Benutzerinterface des Agent zur Anzeige der Anzahl neuer E-Mails

6.4.4 Basis E-Mail - Agent

Dieser Agent ist eine Basisklasse für verschiedenste Agenten zur Bearbeitung von Nachrichten. Implementiert ist folgende Funktionalität:

? Suche nach Agenten zum Versand von E-Mail sowie Übernahme aller notwendigen Konversationen. Bei Nicht-Vorhandensein entsprechender Agenten findet auch eine Verlagerung auf andere Rechner statt, auf denen bekanntermaßen derartige Agenten vorhanden sind (mit anschließender Rückkehr).
? Der Agent kann „verankert“ werden, d. h. er wird immobil : Es wird nicht versucht E-Mails auch über andere Rechner zu versenden.
? Unterstützung für Subskriptionen (sowohl aktiv als auch passiv)
? Implementierung des Rückruf zum Heimat-Server (siehe 6.1.2)

Ein Benutzerinterface wird nicht zur Verfügung gestellt. Da der Agent auch keine eigene Tätigkeit entfaltet, ist er als abstrakt definiert; es können keine Objekte davon erzeugt werden.

6.4.5 Agent für die Filterung von Nachrichten

Dieser Agent ist eine Subklasse des Basis-E-Mail – Agenten, welcher auch noch die Funktionalität zum Abholen von Nachrichten implementiert (Verbindung zu Agenten wie unter 6.4.2 beschrieben). Auch die Filterung ist bereits implementiert, sodaß in Subklassen lediglich noch Regeln eingefügt werden müssen, um einen vollständigen Agenten zur Filterung zu erhalten (wenn auch in diesem Fall ohne Benutzerinterface und mit fest programmierten Regeln).

6.4.6 Zeichenketten-Filter mit Benachrichtigung per SMS

Dieser häufig in der Praxis verwendete Agent ist eine einfache (insgesamt 4 neue Klassen) Verbindung aus einem allgemeinen Filter-Agenten mit der Klasse zum Versand von SMS aus dem Framework. Wird in einer empfangenen Mail (Subskription an alle Agenten, welche dies anbieten, daher z. B. auch aus Webseiten; siehe 6.3.3) eine bestimmte Phrase gefunden, so wird eine SMS mit vorbestimmtem Text an eine zu konfigurierende Nummer gesendet (siehe Abbildung 78; mit „Send-SMS“ kann der Versand der SMS zu Testzwecken manuell ausgelöst werden). Dieser Versand erfolgt über Webseiten, welche dieses Service im Internet anbieten. Hierzu werden die notwendigen Seiten geladen, Formulare ausgefüllt und abgesendet, sowie die Rückmeldungen überprüft. Die notwendige Berechtigung (Verbindung zu einem externen Webserver) wird vom Agenten erworben.

Die Verbindung des Filters mit einer besonderen Aktion erfolgt durch eine neue Subklasse von Action, welche als Modell für verschiedenste Aufgaben dienen kann:

```
public class SendSMSAction extends Action
{
    private StringFilterMailAgent myAgent=null;
    private boolean isTest=false;

    public SendSMSAction(StringFilterMailAgent myAgent,boolean test)
    {   this.myAgent=myAgent;
        isTest=test;
    }

    public Message doAction(Message msg) throws ActionException
    {   myAgent.doSend(false,isTest);
        return msg;
    }
}
```

Ein besonderes Problem beim Versand von SMS stellt dar, daß die Webserver Vorkehrungen gegen ein derartiges „Hijacking“ ihrer Dienste treffen. So müssen etwa Cookies akzeptiert und gesendet werden, obwohl diese keinerlei Funktion in Hinsicht auf das Senden übernehmen (bei Trennung von Provider und Nummer/Text wird etwa der Provider im *Formular* weitergegeben und *nicht* über das Cookie oder nur in internen Datenbanken). Da eine derartige Synchronisation von Funktionen auf dem Server auch eine gewisse Zeit benötigt, mußten explizite Verzögerungen (8 Sekunden nach Senden eines Formulars/Seite) eingebaut werden. Auf diese Weise wird sozusagen ein Benutzer imitiert, welcher auch eine gewisse Zeit zum Besichtigen bzw. Ausfüllen der Seite benötigt (Im Gegensatz zu einem normalen Benutzer wird jedoch nur die bloße Seite geladen, aber keine Bilder!).

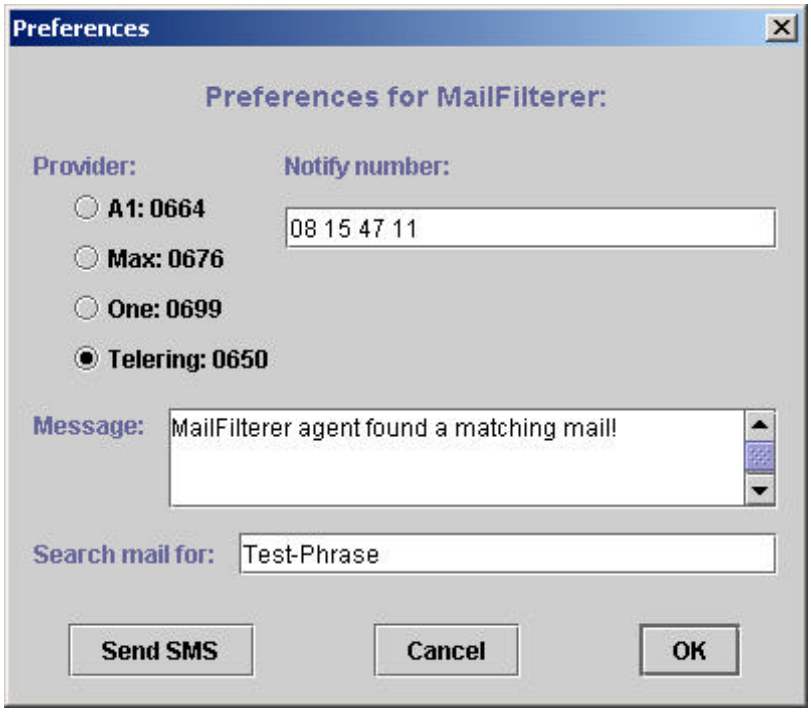

Abbildung 78: Konfigurations-Dialog: Benachrichtigung per SMS nach Phrasen-Suche

Von der rechtlichen Seite[8] her ist diese Verwendung unbedenklich, solange es sich um eine private Nutzung handelt. Gewerbliche oder betriebliche Nutzung des Dienstes ist grundsätzlich in den Nutzungsbedingungen verboten. Ein Ausschluß einer privaten Nutzung wäre aber sicherlich möglich, insbesondere da die Werbung nicht mehr gesehen wird, welche ansonsten zur Kostentragung dient. Da dies jedoch auch durch das Ausschalten des Ladens von Bildern erreicht werden kann, liegt auch in dieser Hinsicht kein Problem bei der automatischen Nutzung vor. Zu beachten ist jedoch, daß durch die automatische Benutzung keine übermäßige Belastung des Servers erfolgt, also der Dienstes nur in verträglichem Maß genutzt wird (Massen-SMS sind auch explizit verboten; dies könnte hier analog Anwendung finden).

6.4.7 Automatische Weiterleitung (Forward)

Dies ist ein besonders einfacher Agent als Beispiel für die Verwendung des Basis-Filter-Agenten. Es werden alle Mails, welche im Titel das Wort „Agent“ enthalten (case-insensitive), an eine feste Adresse weitergeleitet. Der *vollständige* Sourcecode (ohne Kommentare) lautet:

```
package FilterTest;

import MailAgents.FilterMailAgent;
import Filtering.Condition.*;
import Filtering.Action.*;
import Filtering.Rules.*;
import FIM.Util.Crypto.NamedKeyAndCertificate;

public class ForwardTest extends FilterMailAgent
{
     public ForwardTest() {}

     protected void initialize(Object param)
          throws PkgAgentSystem.InitializationException
     {
          NamedKeyAndCertificate nkc=new NamedKeyAndCertificate(
               "Signing keys",getKeyPair("Agents own keys").
               getKeyPair().getPrivate(),getIdentity());
          ruleList.addRule(
               new SimpleActionRule(
                    new Filtering.Condition.SubjectTerm("Agent",false),
                    new SendResponse(this,new ForwardMsg(
                         "sonntag@fim.uni-linz.ac.at"),
                         "mail.fim.uni-linz.ac.at",nkc)
               )
          );
     }
}
```

6.4.8 Übersicht über das Nachrichten-Filterungs Framework

Das kleine Framework zur Filterung von Nachrichten besteht aus Regeln, welche aufgrund von beliebig geschachtelten Bedingungen Aktionen durchführen. Um einen Transfer von Informa-

[8] Diese Ausführungen beziehen sich auf die Nutzungsbedingungen von http://sms.orf.at/ (28.11.2001).

tionen aus den Bedingungen zu den Aktionen zu ermöglichen, besteht zusätzlich eine „Umgebung“ (=Hash-Tabelle; genannt „Environment“), in welcher Daten abgelegt und ausgelesen werden können.

Für einfache Applikationen reichen die vorliegenden Klassen aus. Für komplexere Anwendungen sind Erweiterungen durch eigene Klassen problemlos möglich (siehe etwa 6.4.6 Zeichenketten-Filter mit Benachrichtigung per SMS für eine neue Aktion).

Alle Elemente werden hier nur kurz beschrieben, da es sich um einen Nebenzweck der Arbeit handelt, welcher nur der Demonstration der praktische Nutzbarkeit des Agentensystem dient.

6.4.8.1 Nachrichten-Klassen

Da die Klassen für Nachrichten im JavaMail Framework nicht serialisierbar sind und daher weder mit Agenten noch innerhalb des Agentensystem als Nachrichten transportiert werden können, mußten eigene Klassen geschaffen werden. Diese können aus JavaMail Objekten erstellt und wieder in solche zurückgewandelt werden.

? MessageElement: Basisklasse für Nachrichten. Enthält Header und einen Inhalt.

? Message: E-Mail Nachricht. Enthält die üblichen Felder sowie (auch rekursive) Attachments. Erlaubt zusätzlich tiefe Kopien zu erstellen sowie eine Nachricht aus einer HTML-Seite zu konstruieren bzw. in eine solche umzuwandeln.

? Attachment: Speichert zusätzlich (abgeleitet von MessageElement) noch einen optional enthaltenen Dateinamen.

6.4.8.2 Bedingungen

Die implementierten Bedingungen können in vier Gruppen eingeteilt werden: Vergleiche, logische Verknüpfungen, Konstanten, sowie inhaltliche Tests.

? Vergleiche: Es kann auf =, !=, <, <=, > und >= abgefragt werden. Als konkrete Datentypen existieren ein Datumsvergleich (optional kann die Zeitzone angegeben werden) sowie ein Vergleich von Integer-Werten.

? Logische Verknüpfungen: Bedingungen können mittels „und“, „oder“ sowie mit „nicht“ miteinander verknüpft werden. Andere Verknüpfungen sind denkbar, können jedoch hiermit nachgebildet werden.

? Konstanten: Es existieren Bedingungen welche immer wahr bzw. falsch zurückliefern.

? Inhalts-Tests: Hier existieren eine größere Menge: Anzahl der Attachments, Text-Prüfung des Haupttextes sowie der Überschrift oder allgemein, Länge der Nachricht in Bytes sowie in Zeilen, Absendedatum, Header-Strings, Absender und Addressat (als Text sowie als E-Mail Addresse), Überprüfung des Nachrichten-Umgebungsspeichers, sowie allgemein von Header-Elementen.

6.4.8.3 Aktionen

Aktionen können in zwei Gruppen unterteilt werden: Parser-Aktionen, welche auf den Nachrichten-Umgebungsspeicher arbeiten, sowie sonstige Aktionen (mit externen Resultaten oder besonderen Rückgabewerten).

? Parser: SetEnvironmentString (ein Umgebungswert wird gesetzt; unabhängig von der Nachricht), ParseDelimitedString (String extrahieren der von bestimmten Strings begrenzt wird), ParseAttachment (Typ und Inhalt werden gespeichert), sowie ParseGeneralInformation (allgemeine Daten werden gespeichert: Addressat(en), Absender, Reply-Addresse, CC-Address(en), Typ des Inhalts, Codierung, Absendedatum, Titel, Text).

? Aktionen: NoAction (keinerlei Tätigkeit), ComposeMail (eine feste Nachricht wird erstellt, welche von der Eingabe-Nachricht unabhängig ist; im Text markierte Elemente werden durch Daten aus der Umgebung ersetzt), SendReply (eine sehr einfache Nachricht wird erstellt; Titel, Text, Addressat), SendMail (eine fixe Nachricht wird erstellt; alle Felder konfigurierbar), SaveLocal (ein Umgebungs-Datum wird in eine lokale Datei gespeichert; z. B. ein Attachment), ForwardMessage (die Eingangs-Nachricht wird, ev. mit Zitierungs-Anmerkungen, an einen bestimmten Adressaten weitergeleitet), SendResponse(die aus vorigen Aktionen resultierende Nachricht wird tatsächlich abgesandt; direkt, nicht über einen weiteren Agenten), sowie SerialAction (beliebig viele Aktionen werden zu einer zusammengefaßt).

Besondere Bedeutung besitzt noch die Aktion CombineResponses, mit welcher die Ergebnis-Nachrichten mehrerer Aktionen zu einer einzigen Nachricht zusammengefaßt werden können. Diese wird auch verwendet, wenn mehrere Regeln abgearbeitet werden und Ergebnisse liefern (siehe unten). Als Bedingung wird vorausgesetzt, daß der Absender und alle Adressaten identisch sind: Ansonsten wird eine Ausnahme ausgeworfen. Der Inhalt der Nachricht setzt sich aus der Konkatenation der einzelnen Inhalte und der Signaturen (bei gleichem Typ, ansonsten wer-

den die Inhalte als getrennte Attachments verschickt) zusammen. Attachments und Header werden hintereinander hinzugefügt.

6.4.8.4 Regeln

Es existieren zwei konkrete Regeln: Eine einfache Regel, wonach die Aktion ausgeführt wird, wenn die Bedingung wahr ist, sowie eine Liste von Regeln, welche überprüft werden. Bei letzterer kann zusätzlich spezifiziert werden, ob nach der ersten erfolgreichen Regel die Überprüfung abgebrochen werden soll, oder immer alle Regeln überprüft werden. In letzterem Fall werden alle Einzelergebnisse mittels der Klasse CombineResponses zu einem einheitlichen Gesamtergebnis zusammengefaßt.

Regeln sind hier in Form von Programmcode zu spezifizieren. Es ist jedoch auch eine textuelle Spezifikation denkbar, z. B. in XML (siehe [Bonifati 2001] für ein System zur Abbildung von Regeln in XML).

6.5. Agent zur Kauf-Unterstützung

Im Hinblick auf E-Commerce wurde ein Agent erstellt, welcher bei verschiedenen Online-Buchshops (Definition siehe Abbildung 79; das Formular zur Suche auf der Webseite ist neben dem URL als einziges zu spezifizieren) nach bestimmten Titeln suchen soll. Hierzu wird jedoch nicht (wie sonst oft) eine genaue Definition der Seite verwendet, sondern versucht, die Ergebnisseite anhand von allgemeinen Regeln zu zergliedern und so die gesuchten Elemente zu identifizieren. So sind etwa in einer Liste der Suchergebnisse einzelne Titel regelmäßig durch horizontale Linien getrennt bzw. in Tabellen zusammengefaßt (bei letzteren ist noch zu berücksichtigen, daß es sich meist um mehrere Zeilen pro Titel handelt). Diese heuristische Suche besitzt natürlich den Nachteil, daß nicht immer alle Elemente (insbesondere. der Autor; siehe im Beispiel den Anbieter Barnes&Noble) genau identifiziert werden können. Die Ergebnisse werden anschließend dem Benutzer präsentiert (siehe Abbildung 80 für das Ergebnis einer Beispiels-Suche).

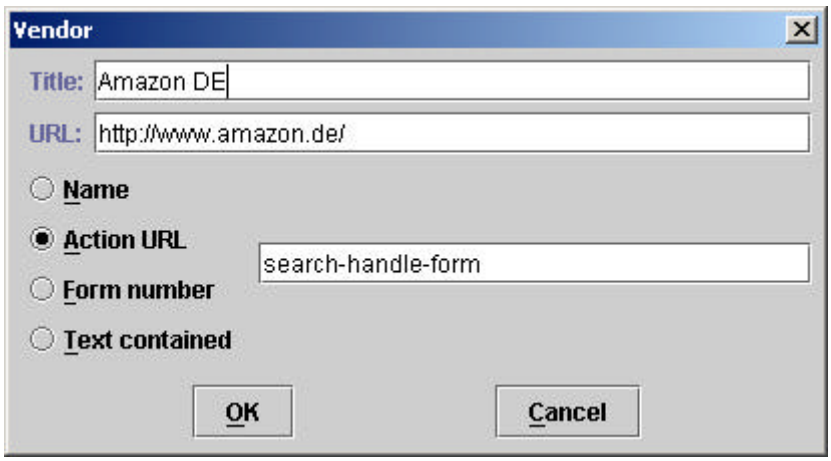

Abbildung 79: Definition eines Internet Buchshops

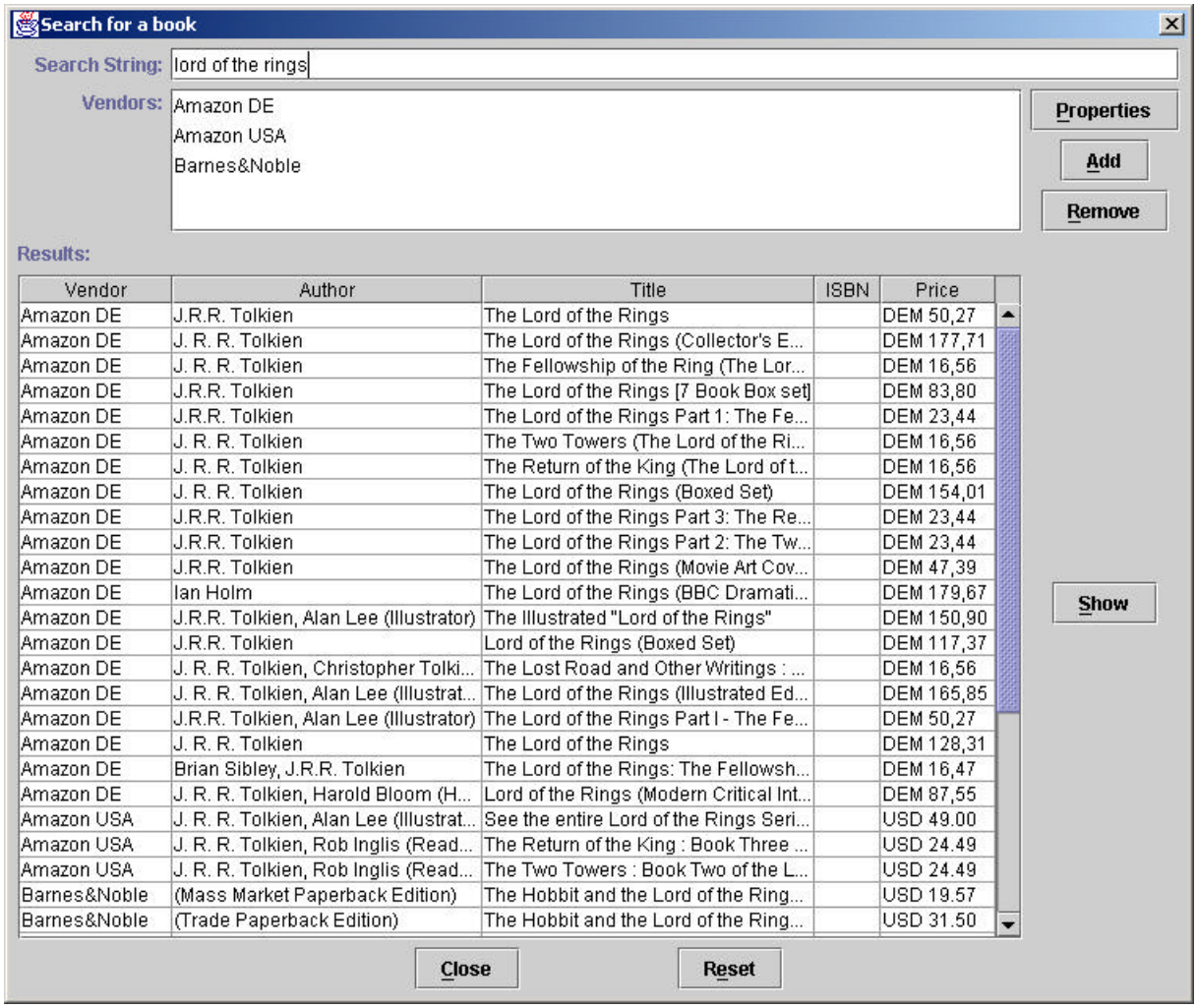

Abbildung 80: Ergebnis einer Beispiels-Suche nach einem Buch

6.6. Das Sport-Portal

Ein größeres Projekt (welches sich jedoch derzeit teilweise noch in Entwicklung befindet) ist die Realisierung eines Webportals für Sport-Informationen, welches auch für Agenten zugänglich ist. Hierüber erfolgt nur ein grober und kurzer Überblick, da dies das Thema einer geson-

derten Arbeit (Dissertation von Frau Dipl.-Ing. Susanne Reisinger; zur Zeit in Ausarbeitung; siehe ebenso [Mühlbacher et al. 2001]) ist.

Wichtige Elemente sind die Integration von XML (insbesondere für den Datenaustausch), rekonfigurierbare Protokolle (siehe 4.5.1.5), sowie die Bereitstellung von besonderen Diensten mittels Agenten.

Hierbei wird das Sicherheitssystem umfassend eingesetzt sowie reale Bezahlung für Leistungen verwendet. Agenten bezahlen jedoch nicht für eigene Ressourcen-Konsumation, sondern für die Übermittlung von Informationen oder anderen Leistungen, die für den Besitzer erbracht wurden (z. B. Bezahlung von Club-Mitgliedsbeiträgen).

6.6.1 Umfang

Folgende Elemente sollen im Portal dargestellt werden, welches für mehrere Sportarten/Clubs intendiert ist:

? Neuigkeiten: Bezüglich des Portals bzw. jedes einzelnen Clubs.

? Ereignisse: Online (Portal) sowie von jedem Club gesondert. Hierzu gehören auch etwaige „Ergebnisse“ wie Endstände von Turnieren, etc.

? Sonstige Daten: Einfache Verwaltung der Daten der Mitglieder und der Clublokalitäten (z. B. für Platzreservierung)

6.6.2 XML-Integration

Für Parsen bzw. Erstellen von XML-Dateien/-Strings wird ein fertiger XML-Parser verwendet, welcher eine Java-Schnittstelle besitzt. Auf diesem aufbauend und mittels weiteren Hilfsklassen werden Elemente des ebXML-Protokolls implementiert (siehe 4.5.1.6 ebXML Nachrichten). Dieses Format wird zum Austausch der Daten zwischen Agenten und dem Portal verwendet. Durch die genaue Definition der Struktur (DTD’s /Schemata) und des Inhalts können verschiedenste Implementierungen auf die Daten zugreifen und mit dem Portal interagieren.

6.6.3 Besondere Dienste für Agenten

Die Integration des Zugangs zum Portal über Agenten soll zusätzliche Dienste bzw. Möglichkeiten bereitstellen welche über die Webseiten selbst nur schwer oder gar nicht erreichbar sind. Beispiele hierfür sind:

? Bezahlung von Beiträgen/Leistungen ohne daß ein sicherer Webserver (SSL, Kreditkarten-Gateway, etc.) benötigt wird: Entweder über E-Cash oder per Kreditkarte.

? Buchung von Spiel-Plätzen wobei Präferenzen des Benutzers dem Agenten bekannt sind und dieser dann mit dem System nach Prioritäten verhandelt, um möglichst günstige/zeitlich besser gelegene/schönere Plätze automatisch zu reservieren.

? Filterung von Neuigkeiten bzw. Nachrichten nach Präferenzen des Besitzers und anschließende Benachrichtigung mit Exzerpten oder Hinweisen; sowohl über E-Mail als auch per SMS oder Nachrichtenfenstern.

? Sammlung von Daten im Web (z. B. Spielergebnisse) die anschließend im Portal dargestellt werden.

7. Diskussion und kritische Würdigung

In diesem Kapitel werden die Ergebnisse der Arbeit kurz diskutiert und ein Vergleich des implementierten Agentensystems mit bereits existierenden Systemen durchgeführt. Weiters wird erläutert, welche Probleme hiermit nicht gelöst wurden bzw. nicht gelöst werden können. Abschließend werden noch unklare Fragestellungen dargestellt, bei welchen daher weitere Forschung wünschenswert ist.

7.1. Vergleich von POND mit anderen Agentensystemen

Im Kontrast zu vielen anderen Agentensystemen wurde bei POND schon von Anfang an großer Wert auf ein umfassendes Sicherheitssystem gelegt, welches auch dynamisch modifizierbar sein sollte. Bei der Zuteilung von Basis-Berechtigungen wird sowohl vom Besitzer als auch vom Hersteller des Programmcodes ausgegangen. Zusätzliche Berechtigungen werden aufgrund einer Einschätzung der damit verbundenen Risiken mit einem Geldwert versehen und können im Austausch gegen diesen erworben werden. Ein weiterer Punkt war die Ausstattung mit Funktionen zur Beweissicherung, sodaß später einzelne Parteien nicht abstreiten können, bestimmte Handlungen vorgenommen zu haben. Dies insbesondere im Hinblick darauf, daß bei Einsatz z. B. im E-Commerce die genaue Nachvollziehbarkeit oftmals Streitigkeiten vermeiden hilft, weil der Ausgang von etwaigen Rechtsstreitigkeiten bereits determiniert ist. Auch wurde POND als offenes System entworfen: Grundsätzlich kann jeder Agent Zugang erlangen und vielfach wird dies auch tatsächlich erlaubt sein (z. B. bei Agentensystemen von Produkt-Anbietern).

Im Gegensatz dazu konzentrieren sich existierende Agentensysteme auf andere Aspekte von Agenten wie etwa der Kooperation, der Mobilität, oder der Integration mit anderen Elementen (bestehender Software, Hardware, anderen Agentensytemen, etc.). Meist wird zwar der Sicherheitsaspekt mitbedacht, doch handelt es sich oft nur um punktuelle Maßnahmen. Ein weiteres Problem bestehender Systeme ist, daß vielfach von einem (zumindest teilweise) geschlossenen System ausgegangen wird, in welches „fremde“ Agenten nicht oder nur unter besonderen Bedingungen zugelassen werden. Es ist daher nur ein Schutz hinsichtlich des Besitzers des Agenten enthalten: Bei Mißverhalten eines Agenten können jederzeit später auf externem Weg Konsequenzen gesetzt werden (z. B. bei Mitarbeitern des Unternehmens, welches das Agentensystem betreibt).

Auch erfolgt bei POND eine strikte Trennung zwischen dem Agenten und dessen Besitzer: Ein Agent ist nicht ein „Anhängsel“ seines Besitzers, dem dessen Berechtigungen zugeordnet werden, sondern eigenständig. Die ihm zugänglichen Aktionen können sogar über diejenigen hinausgehen, welche seinem Besitzer bei direktem Zugriff zugestanden würden (z. B. das Lesen bestimmter Dateien), wenn es sich um besonders vertrauenswürdigen Code handelt und die Daten nicht weitergegeben, sondern nur zur Verarbeitung herangezogen werden.

Für einen Vergleich von POND mit anderen existierenden Agentensystemen (welche bereits unter 2.6 kurz dargestellt wurden) siehe die Tabelle 4.

Kriterium	**Aglets**	**Grasshopper**	**Voyager**	**Hive**	**Jini**	**Mole**	**D‘ Agents**	**POND**
Portabilität des Systems	X	X	X	X	X	X	(X[2])	X
Portabilität von Agenten	X	X	X	X	X	X	(X[2])	X
Unterstützung von Mobilität	X	X	X	X	-	X	X[3]	X
Offenes System	X	X	X	-	X	-	(X[4])	X
Gruppierung von Agentensystemen möglich	X	X	-	X	-	-	X	-
Verzeichnisdienst (L=Lokal, G=Global)	L	G	G	L	G	G	-	-
Gesicherte Übertragung von Agenten (S=Signiert, E=Verschlüsselt, H=Hashwert)	H	S, E	S, E	-	-	-	S, E	S, E
Kommunikation zwischen Agenten (M=Methodenaufruf, N=Nachrichtenaustausch, S=Streams)	M, N	M, S	M	M	M	M, N	N, S	N
Identifikation von Agenten (A=Angaben des Agenten, Z=Zertifikate)	A	Z	A	A	A	A	Z	A, Z
Agenten haben einen Besitzer	-	X	X	-	X	-	X	X
Code-Signaturen	-	X	X	-	X	-	X	X
Dynamische Berechtigungen	-	-	-	-	-	(X[1])	-	X
Bezahlung für Berechtigungen	-	-	-	-	-	-	-	X
Beweissicherungs-Maßnahmen	-	-	-	-	-	-	-	X

Tabelle 4: Vergleich von POND mit anderen existierenden Agentensystemen

1. Entweder alle Berechtigungen oder keine; nur dies kann zur Laufzeit verändert werden
2. Ja nach der verwendeten Programmiersprache (Java, Scheme, TCL)

3. Unterstützt starke Mobilität
4. Bei Mobilität muß der Zielrechner dem Quellrechner vollständig vertrauen, da Sicherheitsprüfungen ausschließlich beim Start des Agenten erfolgen

7.2. Offene Probleme

Das implementierte Agentensystem POND ist zwar vollständig, doch bestehen Erweiterungsmöglichkeiten sowie verbesserungswürdige Stellen. So kann ein Agent derzeit beliebig viele Threads erzeugen, doch ist er selber dafür zuständig diese auch ordnungsgemäß zu beenden wenn er verlagert oder terminiert wird. Dies wird zur Zeit vom System *nicht* erzwungen. Zwar verlieren alle Threads in diesen Fällen jegliche Berechtigungen, doch hindert sie dies nicht daran, Speicherplatz und CPU-Zeit zu verbrauchen (jegliche andere Aktionen sind ihnen mangels Berechtigungen verwehrt). Zu diesem Zweck müßte für jeden Agenten eine eigene Thread-Gruppe angelegt werden und bei Erzeugung eines Threads dieser der Gruppe hinzugefügt werden. Über diese Gruppe könnten dann alle Threads eines Agenten beendet (oder zumindest ihre Priorität verändert) werden.

Aufgrund des Konzeptes, daß kein Agent Informationen über andere Agenten erlangen können sollte, ohne daß dieser die Daten freiwillig preisgibt, existiert kein Verzeichnisdienst. Dies hat sich in der Praxis jedoch als Hindernis erwiesen. Agenten sind aus diesem Grund auf eine explizite Konfiguration angewiesen oder müssen potentielle Partner über Broadcasts ausfindig machen. Dies führt einerseits zu einer Belastung des Agentensystems, andererseits müssen Agenten viele uninteressante Nachrichten zumindest kurz bearbeiten. Die Einrichtung eines Verzeichnisdienstes wäre daher sinnvoll, wobei auf die Wahrung der Anonymität besonderer Wert gelegt werden sollte: Einerseits indem Agenten freigestellt wird, ob sie registriert werden möchten, andererseits über einen Mapping-Service, mittels welchem Agenten nach bestimmten Eigenschaften gefunden werden könnten (u. U. erst nach Rückfrage beim betroffenen Agenten; oder Zustellung einer ersten Nachricht über diesen Service). Dies setzt jedoch voraus, daß Agenten diesem Mittler vertrauen. Eine Separation vom Agentensystem ist daher wünschenswert. Dieser Agent könnte dann beispielsweise bei Transaktionen auch als „trusted third party“ Einsatz finden.

Bei der Bezahlung für Berechtigungen ist kein zeitliches Element enthalten: Fragt ein Agent nach dem Preis für eine Berechtigung, so existiert keine festgelegte Zeitspanne in welcher der Preis fix bleibt. Dies wäre auch durch den Agenten nur äußerst schwer (wenn überhaupt) zu überprüfen. Im implementierten System bleibt der Preis daher grundsätzlich fest, sofern der

Agent nicht seinerseits diesbezügliche Aktionen setzt, wie etwa das Anbieten zusätzlicher Klassifikationen. Für eine Integration weiterer dynamischer Elemente wie etwa eine Rückstufung oder Preiserhöhung bei verdächtigen Aktionen müßte daher eine Transaktion oder ein zeitliches Limit integriert werden, um wechselseitige Fairneß zu garantieren.

Durch die Wahl der Programmiersprache Java entstanden einige Einschränkungen, welche suboptimal sind. So ist es nicht möglich, CPU-Zeit oder Speicher an Agenten zuzuteilen und diese auch tatsächlich darauf zu beschränken. Dies würde eine Modifikation der JVM erfordern, was aber sofort zu mangelnder Portabilität führt. Andere Abhilfsmöglichkeiten bestehen hier leider nicht.

7.3. Unbeantwortete Fragen

Nicht alle Probleme und Fragen konnten gelöst werden, wobei eine Schwierigkeit besondere hervorsticht: Wie kann ein Agent vor dem Rechner geschützt werden, auf dem er sich befindet (und zwar ohne die Verwendung von „tamper-proof“ Hardware in Verbindung mit Software, welcher vertraut wird)? Hierzu gehört auch die Geheimhaltung von Daten vor dem Rechner. Zur Zeit existieren lediglich allererste Ansätze. So können Daten z. B. auf einem sicheren Server verschlüsselt werden und der Schlüssel bleibt dort zurück. Jedoch können die Daten dann auf dem Weg des Agenten auch nicht mehr verwendet werden, sodaß gleich eine Teilung in mobile und immobile Daten erfolgen könnte. Ein anderer Ansatz ist zeitlich begrenzter Schutz. Hierbei werden Daten bzw. Code so verändert bzw. verschlüsselt, daß eine Entschlüsselung mindestens eine bestimmte Zeitspanne in Anspruch nimmt. In der Zwischenzeit wären die Daten dann vor unbefugtem Zugriff sicher. Dies hat jedoch auch einige Nachteile, da es sich eben nur um einen eng zeitlich begrenzten Schutz handelt (die Daten sind eine relativ kurze Zeit später jedenfalls entschlüsselbar) und zusätzliche Dienste wie sichere Zeitstempel erforderlich sind. Selbst dann steht es oft im Belieben eines Partners, ein mit Hilfe dieser Mittel geschlossenes Geschäft nachträglich zu vernichten, etwa indem der Agent verzögert wird: Der Schutz läuft ab und die gesamten Daten werden ungültig. Dies könnte z. B. einem Agentensystem einen verspäteten Rücktritt von einer Übereinkunft erlauben.

Ein weiteres ungelöstes Problem ist die Konfiguration größerer Systeme: Einem Benutzer kann eine komplexe Konfiguration und Parametrisierung nicht zugemutet werden. Genau dies ist jedoch erforderlich, wenn schwierigere (und damit höherwertige und für eine Arbeitsersparnis bedeutende) Aufgaben durchgeführt werden sollen. Ein Ansatz hierfür könnte die Entwicklung

von selbstlernenden Systemen sein, welche zuerst eine Lern-Phase durchmachen. Problematisch hierbei ist, daß nicht nur Faktenwissen und Regeln erworben werden müssen, sondern oft auch auf die dahinterstehenden Überlegungen geschlossen werden sollte. Existierende Expertensysteme sind jedoch für Agenten nicht geeignet, da sie große Wissenbasen voraussetzen (Problem bei Mobilität), lange Berechnungen durchführen (große Rechenleistung erforderlich), und meist nur langsame Anpassungen ermöglichen (lange Reaktionszeit).

Ein Gebiet für weitere Forschung ist auch die Kommunikation zwischen Agenten, wobei weniger auf Protokolle und Formate Wert gelegt werden muß (derartige existieren in vielen Variationen), als auf die Einbeziehung von Semantik (z. B. explizite Beschreibung derselben) und die automatische Erstellung von Adaptern. Bei der Anwendung von Agentensystemen bzw. Systemen von Agenten hat es eher weniger Sinn zu versuchen, ein allgemeingültiges universelles Protokoll zu entwerfen, als vielmehr die einfache Interoperabilität zu gewährleisten. Hierbei sollte auch besonders darauf geachtet werden, verschiedene Grundkonzepte zu integrieren: Methodenaufruf und Nachrichten-Systeme, synchrone und asynchrone Verarbeitung, Zeitlimits, etc. Ein Ansatz in diese Richtung könnten konfigurierbare Protokolle sein, wie sie im Ansatz mit den Haupt- und Sub-Protokollen in POND integriert wurden.

8. Zusammenfassung und Ausblick

Intelligente und mobile Agenten sind ein wichtiges Element der Zukunft, insbesondere im E-Commerce: Mit fortschreitender Technik und Leistungsfähigkeit der Computer sollen diese auch immer mehr Aufgaben durchführen. Hierbei stellt sich jedoch eine natürliche Grenze, da Aufgaben im traditionellen Sinn sehr genau beschrieben werden müssen. Ab einer gewissen Komplexität ist dann der Beschreibungsaufwand so hoch, daß sich die anschließende automatische Durchführung kaum mehr auszahlt. Dies insbesondere bei Aufgaben, welche sich von Mal zu Mal leicht verändern (wenn auch ihr Grundmuster gleich bleibt). Ein wichtiges Element ist daher die Integration von mehr Autonomie und Intelligenz in Programme, sodaß ähnliche bzw. komplexe Aufgaben selbständig durchgeführt werden können. Mobilität hingegen beruht nicht auf Überlegungen hinsichtlich der Nutzbarkeit, sondern auf technischen Grundlagen: Optimale Ressourcenauslastung bzw. einfachere Programmierung. Beides sind unerläßliche Elemente, auch wenn sie nicht unbedingt immer in Form von mobilen intelligenten Agenten vorkommen werden, sondern z. B. auch in „normalen“ Programmen derartige Aspekte enthalten sein werden.

Sowohl verstärkte Autonomie als auch ein auf kommerzielle Elemente erweiterter Anwendungsbereich bedingen, ein größeres Augenmerk auf Sicherheitsaspekte zu legen, was ein Hauptaspekt dieser Arbeit ist. Agenten (insbesondere mobile) müssen vor vielen potentiellen Gefahren geschützt werden. Auch hier lassen sich manche Elemente verallgemeinern und so für einen noch breiteren Anwendungsbereich nutzbar machen, wie etwa dynamische Berechtigungen, welche gegen Leistungen (welcher Art auch immer) erworben werden können. Logische Konsequenz der Existenz von Berechtigungen ist, daß für deren Mißbrauch oder Verletzungen auch Konsequenzen bestehen müssen: Nicht alle potentiellen Schädigungen können verhindert werden. Für diesen Fall wurden auch rechtliche Aspekte von mobilen Softwareagenten erläutert, ein Bereich des bisher weitgehend vernachlässigt wurde.

In der Zukunft wird das vorgestellte Agentensystem POND weiterentwickelt werden, wobei nun der Schwerpunkt mehr auf konkreten Anwendungen liegt, wie etwa dem Sport-Portal. Doch insbesondere die Verwendung zur Nachrichten-Verarbeitung soll ebenfalls erweitert werden, um eine bessere Integration mit Mail-Programmen zu bieten und mehr Funktionen standardmäßig zur Verfügung zu stellen.

In anderer Hinsicht wurde bereits eine Erweiterung abgeschlossen: Ein Zugriff auf das Agentensystem ist nun auch über Webseiten möglich. Hierzu wurde von einem griechischen Austauschstudenten einer englischen Universität (Marios Kalnis) ein Java Servlet implementiert ([Kalnis 2001], [Web-POND]), welches einfaches Management von Agenten über Webseiten erlaubt (keine Benutzer-Identifizierung). Auch hier soll weitere Aktivität erfolgen, sodaß eine stärkere Integration erreicht wird und auch hier Sicherheitsaspekte einfließen.

Ein zweites Forschungsgebiet ist die Verbindung von rechtlichen Aspekten mit Agenten. Vieles ist hier noch im Fluße und nicht genau erforscht bzw. geklärt. Beispiele sind die Jurisdiktion (welche Rechtsordnung ist auf mobile Agenten anwendbar), die Verantwortlichkeit für Handlungen (insbesondere wenn der eigentliche Urheber nicht zur Rechenschaft gezogen werden kann; existieren weitere haftbare Personen?), ist ein Agent eine eigene Person (in Analogie zu einer juristischen; mit allen daraus entstehenden Problemen: Wie führt man etwa Exekution gegen ihn?), sowie praktische Aspekte insbesondere zur Verfahrensvereinfachung (z. B. automatische Streitschlichter als erste Instanz bei Problemen). Interessant sind ebenso Datenschutz-Aspekte: Ist die Zustimmung eines Agenten für eine Datenverwendung gültig, welche Daten sind wie geschützt, etc.

Anhang

1. Klassenhierarchien

1.1. Klassenhierarchie (Agentensystem + Utility-Klassen)

class java.lang.Object
- *class java.util.AbstractCollection (implements java.util.Collection)*
  - *class java.util.AbstractSet (implements java.util.Set)*
    - *class java.util.HashSet (implements java.lang.Cloneable, java.io.Serializable, java.util.Set)*
      - class PkgAgentSystem.Classification.ClassificationSet
- *class javax.swing.AbstractListModel (implements javax.swing.ListModel, java.io.Serializable)*
  - class FIM.Util.GUI.NamedListModel
    - class FIM.Util.GUI.CertificateModel
    - class FIM.Util.GUI.KeyPairModel
- class PkgAgentSystem.AgentData
- class PkgAgentSystem.AgentIDStore
- class PkgAgentSystem.Experiences.AgentModel (implements java.io.Serializable)
- class PkgAgentSystem.AMPStreamHandlerFactory (implements java.net.URLStreamHandlerFactory)
- class FIM.payment.AnonymousInvoice (implements FIM.payment.Invoice)
  - class FIM.payment.NamedInvoice
- class FIM.Util.Coder.Base64.Base64Coder
- class PkgAgentSystem.Messaging.BroadcastMessage (implements PkgAgentSystem.Messaging.MessageType)
  - class PkgAgentSystem.Experiences.SendExperienceMessage
- class FIM.Util.WWW.BrowserControl
- *class java.security.cert.Certificate (implements java.io.Serializable)*
  - class PkgAgentSystem.Crypto.AgentIdentity (implements PkgAgentSystem.Classification.CertificateClassification, java.lang.Cloneable, PkgAgentSystem.Classification.RegularCustomerClassification)
    - class PkgAgentSystem.Crypto.AgentIdentityExtension
  - class PkgAgentSystem.Crypto.PublicKeyCertificate
- *class java.security.cert.CertificateFactorySpi*
  - class PkgAgentSystem.Crypto.AgentIdentityFactory
  - class PkgAgentSystem.Crypto.PublicKeyCertificateFactory
- class PkgAgentSystem.Classification.CertificateWrapper (implements PkgAgentSystem.Classification.CertificateClassification, java.io.Serializable)
  - class PkgAgentSystem.Classification.RevocationCheckCertificateWrapper (implements PkgAgentSystem.Classification.RevocationCheckCertificateClassification)
- *class java.lang.ClassLoader*
  - *class java.security.SecureClassLoader*
    - class PkgAgentSystem.AgentClassLoader
- *class java.awt.Component (implements java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable)*
  - *class java.awt.Container*
    - *class javax.swing.JComponent (implements java.io.Serializable)*
      - *class javax.swing.JPanel (implements javax.accessibility.Accessible)*
        - class FIM.Util.GUI.VerticalExtendingPanel (implements javax.swing.Scrollable)
  - *class java.awt.Window (implements javax.accessibility.Accessible)*
    - *class java.awt.Dialog*
      - *class javax.swing.JDialog (implements javax.accessibility.Accessible, javax.swing.RootPaneContainer, javax.swing.WindowConstants)*
        - class PkgAgentSystem.GUI.CreateAgentDialog (implements java.awt.event.ActionListener, javax.swing.event.ListSelectionListener)
        - class PkgAgentSystem.GUI.DestinationDialog (implements java.awt.event.ActionListener)
        - class PkgAgentSystem.GUI.EditIdentityDialog (implements java.awt.event.ActionListener, javax.swing.event.ListSelectionListener)
        - class PkgAgentSystem.GUI.LogDialog (implements java.awt.event.ActionListener)
        - class PkgAgentSystem.GUI.SelectIdentityDialog (implements java.awt.event.ActionListener, javax.swing.event.ListSelectionListener)

- *class java.awt.Frame (implements java.awt.MenuContainer)*
  - *class javax.swing.JFrame (implements javax.accessibility.Accessible, javax.swing.RootPaneContainer, javax.swing.WindowConstants)*
    - class PkgAgentSystem.GUI.MainFrame (implements java.awt.event.ActionListener)
- class PkgAgentSystem.Conversation (implements java.lang.Cloneable, java.io.Serializable)
  - class PkgAgentSystem.Experiences.ExperienceExchangeConversation
  - class PkgAgentSystem.TerminateAgentConversation
- class FIM.Util.WWW.Cookies.Cookie (implements java.io.Serializable)
- class FIM.Util.WWW.Cookies.CookiePathComparator (implements java.util.Comparator)
- class FIM.Util.WWW.Cookies.CookieStore (implements java.io.Serializable)
- class FIM.payment.Currency (implements java.io.Serializable, FIM.Util.XML.XMLEncodeable)
- class FIM.payment.DataWrapper (implements java.io.Serializable)
- *class javax.swing.DefaultListSelectionModel (implements java.lang.Cloneable, javax.swing.ListSelectionModel, java.io.Serializable)*
  - class FIM.Util.GUI.ReadOnlySelectionModel
- *class java.util.Dictionary*
  - *class java.util.Hashtable (implements java.lang.Cloneable, java.util.Map, java.io.Serializable)*
    - *class java.util.Properties*
      - *class java.security.Provider*
        - class PkgAgentSystem.Crypto.AgentCryptographyProvider
  - class FIM.Util.ReadOnlyStore (implements java.io.Serializable)
- *class java.util.EventObject (implements java.io.Serializable)*
  - class PkgAgentSystem.PermissionEvent
- class PkgAgentSystem.Experiences.Experiences (implements java.io.Serializable)
- *class javax.swing.filechooser.FileFilter*
  - class FIM.Util.GUI.ExtensionFileFilter
- class FIM.Util.FilterEnumeration (implements java.util.Enumeration)
  - class FIM.Util.WWW.Cookies.CookieEnumeration
  - class FIM.Util.Crypto.KeyOrCertificateEnumeration
  - class PkgAgentSystem.Experiences.StatementEnumeration
- class FIM.Util.FilterIterator (implements java.util.Iterator)
- class java.awt.FlowLayout (implements java.awt.LayoutManager, java.io.Serializable)
  - class FIM.Util.GUI.FlowingLayout
- class FIM.Util.WWW.Form.FormElement (implements javax.swing.text.Element, java.io.Serializable)
  - class FIM.Util.WWW.Form.ActionElement
    - class FIM.Util.WWW.Form.ButtonElement
      - class FIM.Util.WWW.Form.ResetButtonElement
      - class FIM.Util.WWW.Form.SubmitButtonElement
    - class FIM.Util.WWW.Form.InputResetElement
    - class FIM.Util.WWW.Form.InputSubmitElement
      - class FIM.Util.WWW.Form.InputImageElement
  - class FIM.Util.WWW.Form.ElementGroup
    - class FIM.Util.WWW.Form.SingleSelectionGroup
  - class FIM.Util.WWW.Form.SelectableElement
    - class FIM.Util.WWW.Form.InputCheckboxElement
    - class FIM.Util.WWW.Form.InputRadioElement
  - class FIM.Util.WWW.Form.TextElement
    - class FIM.Util.WWW.Form.InputHiddenElement
    - class FIM.Util.WWW.Form.InputTextElement
      - class FIM.Util.WWW.Form.InputPasswordElement
  - class FIM.Util.WWW.Form.TextareaElement
- class FIM.Util.WWW.Form.HTMLForm (implements javax.swing.text.Element, java.io.Serializable)
- *class java.io.InputStream*
  - *class java.io.FilterInputStream*
    - class FIM.Util.Coder.Base64.Base64InputStream
    - class FIM.Util.CopyFilterInputStream
    - class FIM.Util.Coder.WSStripInputStream
  - *class java.io.ObjectInputStream (implements java.io.ObjectInput, java.io.ObjectStreamConstants)*
    - class FIM.Util.ClassLoaderObjectInputStream
- class FIM.payment.InvoiceItem (implements java.io.Serializable, FIM.Util.XML.XMLEncodeable)
- class FIM.Util.Crypto.JarUtils
- *class java.awt.event.KeyAdapter (implements java.awt.event.KeyListener)*
  - class PkgAgentSystem.GUI.CreateAgentDialog.MyKeyListener
- class PkgAgentSystem.GUI.MainFrame.ToggleUIListener (implements java.awt.event.ItemListener)

- class PkgAgentSystem.Messaging.Message (implements PkgAgentSystem.Messaging.MessageType)
  - class PkgAgentSystem.Messaging.AcceptMessage
  - class PkgAgentSystem.Messaging.ChallengeIdentityMessage
    - class PkgAgentSystem.Messaging.AcceptAndReplyChallengeMessage
    - class PkgAgentSystem.Messaging.ProveIdentityMessage
  - class PkgAgentSystem.Messaging.DeclineMessage
  - class PkgAgentSystem.Messaging.MessageReply
    - class PkgAgentSystem.Messaging.ReplyToLateMessage
    - class PkgAgentSystem.Messaging.UnknownConversationMessage
    - class PkgAgentSystem.Messaging.UnknownConversationTypeMessage
  - class PkgAgentSystem.Messaging.RejectMessage
  - class PkgAgentSystem.Messaging.ResultMessage
  - class PkgAgentSystem.Experiences.SendStatementsMessage
  - class PkgAgentSystem.Messaging.SignedMessage
  - class PkgAgentSystem.Messaging.TerminateAgentMessage
- *class java.awt.event.MouseAdapter (implements java.awt.event.MouseListener)*
  - class PkgAgentSystem.GUI.CreateAgentDialog.MyMouseListener
  - class PkgAgentSystem.GUI.MainFrame.PopupListener
- class FIM.synchronisation.Mutex
- class FIM.Util.Crypto.NamedKeyAndCertificate (implements java.io.Serializable)
- class FIM.Util.Crypto.NamedKeyPair (implements java.io.Serializable)
- *class java.io.OutputStream*
  - *class java.io.FilterOutputStream*
    - class FIM.Util.Coder.Base64.Base64OutputStream
    - class FIM.Util.CopyFilterOutputStream
  - class PkgAgentSystem.LimitedFileOutputStream
- class FIM.payment.PaymentBase (implements FIM.payment.Payment)
  - class FIM.payment.CreditCardPayment
  - class FIM.payment.DataPayment
  - class FIM.payment.EmptyPayment
  - class FIM.payment.VoucherPayment
- *class java.security.Permission (implements java.security.Guard, java.io.Serializable)*
  - *class java.security.BasicPermission (implements java.io.Serializable)*
    - class PkgAgentSystem.CreateAgentPermission
    - class PkgAgentSystem.FileLimitPermission
  - class PkgAgentSystem.UnresolvedPermission (implements java.io.Serializable)
- *class java.security.PermissionCollection (implements java.io.Serializable)*
  - class PkgAgentSystem.DynamicPermissions (implements java.io.Serializable)
    - class PkgAgentSystem.ValuedPermissions
  - class PkgAgentSystem.PermissionsHash (implements java.io.Serializable)
  - class PkgAgentSystem.UnresolvedPermissionCollection (implements java.io.Serializable)
- class PkgAgentSystem.PermissionsEnumerator (implements java.util.Enumeration)
- class PkgAgentSystem.Crypto.PersonalSecurityStore
  - class PkgAgentSystem.Crypto.IAIKSecurityStore
- class PkgAgentSystem.Crypto.PersonalSecurityStoreFactory
- *class java.security.Policy*
  - class PkgAgentSystem.PolicyByValue
    - class PkgAgentSystem.AgentPolicy
- class FIM.payment.Price (implements java.lang.Comparable, java.io.Serializable, FIM.Util.XML.XMLEncodeable)
- *class java.io.Reader*
  - *class java.io.FilterReader*
    - class FIM.Util.WWW.CanonicalFilterReader
    - class FIM.Util.WWW.CanonicalHTMLFilterReader
    - class FIM.Util.WWW.ConvertHTMLCharFilterReader
    - class FIM.Util.WWW.HTMLBreakFilterReader
    - class FIM.Util.WWW.StripTagFilterReader
- class PkgAgentSystem.ResourcePackage
- class FIM.Util.Crypto.SealedAndSignedObject (implements java.io.Serializable)
- class FIM.synchronisation.Semaphor
- class FIM.Util.SMS.SMSPost
- class PkgAgentSystem.Experiences.Statement (implements java.io.Serializable)

- class PkgAgentSystem.Experiences.TrustStatement
- *class java.lang.Thread (implements java.lang.Runnable)*
  - class FIM.Util.Threads.CancellableThread
    - class PkgAgentSystem.AgentBase (implements java.io.Serializable)
      - class PkgAgentSystem.GUI.GUIAgentBase
    - class PkgAgentSystem.AgentSystem
      - class PkgAgentSystem.GUI.GUIAgentSystem
    - class PkgAgentSystem.ServingThread
- *class java.lang.Throwable (implements java.io.Serializable)*
  - *class java.lang.Exception*
    - class PkgAgentSystem.AgentException
      - class PkgAgentSystem.AgentNotRegisteredException
      - class PkgAgentSystem.CannotSendException
      - class PkgAgentSystem.CreateAgentFailureException
      - class PkgAgentSystem.DelayException
        - class PkgAgentSystem.MovingDeniedException
        - class PkgAgentSystem.PersistingDeniedException
      - class PkgAgentSystem.InitializationException
      - class PkgAgentSystem.Messaging.MessageException
        - class PkgAgentSystem.Messaging.IllegalMessageException
        - class PkgAgentSystem.Messaging.MessageDeliveryException
        - class PkgAgentSystem.Messaging.UnknownAgentException
      - class PkgAgentSystem.ReceiveAgentDeniedException
    - class FIM.Util.SMS.CannotSendException
    - *class java.io.IOException*
      - class PkgAgentSystem.AuthenticationException
      - class FIM.Util.Coder.WrongFormatException
- class FIM.Util.Threads.Timer (implements java.io.Serializable)
- class FIM.Util.Threads.TimerQueue
  - class PkgAgentSystem.AgentTimerQueue (implements FIM.Util.Threads.TimerAction)
- *class java.net.URLConnection*
  - class PkgAgentSystem.AMPURLConnection
- *class java.net.URLStreamHandler*
  - class PkgAgentSystem.AMPStreamHandler
- class PkgAgentSystem.Experiences.ValueComparator (implements java.util.Comparator)
- class FIM.payment.VoucherWrapper (implements java.io.Serializable)
- *class java.awt.event.WindowAdapter (implements java.awt.event.WindowListener)*
  - class PkgAgentSystem.GUI.CreateAgentDialog.CloseWindowListener
  - class PkgAgentSystem.GUI.MainFrame.CloseWindowListener
- class FIM.Util.WWW.WWWPageUtils
- class FIM.Util.XML.XMLByteArray (implements java.io.Serializable, FIM.Util.XML.XMLEncodeable)
- class FIM.Util.XML.XMLString (implements java.io.Serializable, FIM.Util.XML.XMLEncodeable)
- class FIM.Util.XML.XMLUtils

1.2. Interface-Hierarchie

- interface PkgAgentSystem.Classification.AgentClassification
  - interface PkgAgentSystem.Classification.CertificateClassification
    - interface PkgAgentSystem.Classification.RevocationCheckCertificateClassification
  - interface PkgAgentSystem.Classification.RegularCustomerClassification
- *interface java.lang.Cloneable*
  - interface PkgAgentSystem.Messaging.MessageType (also extends java.io.Serializable)
- interface FIM.Util.HasReadOnlyStore
  - interface PkgAgentSystem.HasVetoableReadOnlyStore
- interface PkgAgentSystem.PermissionChangeListener
- *interface java.io.Serializable*
  - interface FIM.payment.Invoice (also extends FIM.Util.XML.XMLEncodeable)
  - interface PkgAgentSystem.Messaging.MessageType (also extends java.lang.Cloneable)
  - interface FIM.payment.Payment (also extends FIM.Util.XML.XMLEncodeable)
- interface FIM.Util.Threads.TimerAction
- interface FIM.Util.XML.XMLEncodeable

- interface FIM.payment.Invoice (also extends java.io.Serializable)
- interface FIM.payment.Payment (also extends java.io.Serializable)

1.3. Package-Hierarchie

1.3.1 Agentensystem

- PkgAgentSystem
  - Classification
  - Crypto
  - Experiences
  - GUI
  - Messaging

1.3.2 Utility-Klassen

- FIM
  - payment
  - synchronisation
  - test
  - Util
    - Coder
      - Base64
    - Crypto
    - GUI
    - SMS
    - Threads
    - WWW
      - Cookies
      - Forms
    - XML

- Protocol
  - Contact
  - Identification
    - None
    - Passphrase
    - SignData
  - Inquiry
  - Sell
  - Test

1.3.3 Basis- und Beispiels-Agenten

- Demo
  - Messages

- BaseAgents
  - Beeper
  - CommandReturn

- MessageboxMessenger
- SelfTask
- Subscription

- Download

- Reminder

- ComparisonShopper
  - Webpages

1.3.4 Agenten zur Nachrichten-Verarbeitung

- MailAgents
  - EmailSender
  - MailChecker
  - WebsiteChecker

- Filtering
  - Action
    - Parser
  - Condition
  - Rules
- FilterTest

- SMSMailFilterer

- Stocks

2. Code Beispiele

2.1. Sicherheits-System Test-Agenten

2.1.1 Test-Agent

```
/*
 * @(#)Test.java
 *
 * Part of the AgentSystem "POND"
 * Developed at the Institute for Information Processing and Microprocessor
Technology (FIM)
 * Johannes Kepler University Linz
 * Altenbergerstr. 69, A-4040 Linz, Austria
 */

import PkgAgentSystem.*;
import java.io.*;
import java.security.*;
import java.security.cert.Certificate;
import PkgAgentSystem.Classification.*;

/**
The agent-class for testing security. Gives general information and tries
attacks.

@author  Michael Sonntag
@version 1.0, 1.7.2000
@see     ProtectionTest
*/
public class Test extends AgentBase implements PermissionCha ngeListener
{
     // The certificate to use as additional classification
     private Certificate cert=null;

     /**
     Main method of agent. First prints the classification-status and
     offers a certificate for additional classification (if provided)
     and prints all optional permissions.
     Then tries to do restricted actions and some attacks in
     different conditions:
     <ul>
     <li>Directly after starting the agent
     <li>After trying to buy AllPermission
     <li>After returning AllPermission (if got in previous step)
     <li>In a doPrivileged block
     <li>In a new Thread
     </ul>
     Afterwards the agent terminates itself.
     */
    public void agentMain()
    {
        System.out.println("Agent "+getAgentName()+" sta rted.");
        if(cert!=null)
        {
            System.out.println("Before additional classific ations:");
            getAgentSystem().dumpClassification(this);
            ClassificationSet set=new ClassificationSet();
            CertificateWrapper wrap=new CertificateWra pper(cert);
            set.add(wrap);
            set=getAgentSystem().offerAdditionalClassification(this,set);
            System.out.println("Number of accepted classifications: "+
          (set!=null?set.size():0));
            System.out.println("After additional classific ations:");
```

```
    getAgentSystem().dumpClassification(this);
}
else
{
    System.out.println("Classification info:");
    getAgentSystem().dumpClassification(this);
}
// Print all optional permissions
System.out.println("  Optional permissions: ");
java.util.Enumeration enum=getAgentSystem().
  getOptionalPermissions(this).elements();
while(enum.hasMoreElements())
{
    Permission p=(Permission)enum.nextElement();
    System.out.println("  "+p.toString()+" = "+
  getAgentSystem().getPrice(this,p));
}
// Try several attacks
System.out.println("  Trying attacks...");
tryIt();
double price=getAgentSystem().getPrice(this,new AllPe rmission());
System.out.println("  Price for AllPermission: "+price);
Permission perm=getAgentSystem().buyPermission(this,
  new AllPermission(),price);
if(perm!=null)
{
    System.out.println("    Promote successfull (Permission: "+
  perm.toString()+")");
    tryIt();
}
else
    System.out.println("    Promote failed");
if(perm!=null)
{
    System.out.println("  Trying after returning pe rmission...");
    getAgentSystem().returnPermission(this,perm);
    tryIt();
}
AccessController.doPrivileged(new PrivilegedAction()
{
    public Object run()
    {
        System.out.println("  Trying when privil eged...");
        tryIt();
        return null;
    }
});
Thread t=(new Thread()
{
    public void run()
    {
        System.out.println("  Trying in a different thread...");
        tryIt();
    }
});
t.start();
try
{
    t.join();
}
catch(InterruptedException e)
{
}
System.out.println("Terminating agent "+getAgentName()+"...");
stopAgent();
  // Should not be reached!
System.out.println("Agent "+getAgentName()+" termin ated.");
```

```
}


/**
The method for trying several attacks and restricted a ctions.
*/
private void tryIt()
{
 Class myClass=this.getClass();
    ClassLoader cl=myClass.getClassLoader();
    try
    {
        ClassLoader cls=cl.getSystemClassLoader();
        cls.getResource("autoexec.bat");
        System.out.println("!!!Got system classloader!!!");
    }
    catch(Exception e)
    {
        System.out.println("   Couldn´t get system classloader ("+
      e.toString()+")");
    }
    try
    {
        ClassLoader cls=cl.getClass().getSuperclass().
      getClassLoader().getSystemClassLoader();
        cls.getResource("autoexec.bat");
        System.out.println("!!!Got system classloader!!!");
    }
    catch(Exception e)
    {
        System.out.println("   Couldn´t get system classloader of
      superclass ("+e.toString()+")");
    }
    try
    {
        Class[] formparams=new Class[1];
        formparams[0]=Class.forName("java.security.Permission");
        java.lang.reflect.Method addM=cl.getClass().
      getMethod("grantAdditionalPermission",formparams);
        Object[] actparams=new Object[1];
        actparams[0]=new AllPermission();
        addM.invoke(cl,actparams);
        System.out.println("Added AllPermission via classloader");
    }
    catch(Exception e)
    {
        System.out.println("   Couldn´t add permission through
      classloader ("+e.toString()+")");
    }
    try
    {
        ClassLoader ascl=getAgentSystem().getClass().getClassLoader();
        System.out.println("!!!Got classloader of AgentSystem!!!");
        ascl.loadClass("Test");
    }
    catch(Exception e)
    {
        System.out.println("   Couldn´t get classloader of agentsystem
      ("+e.toString()+")");
    }
    ProtectionDomain protDom=null;
    try
    {
        protDom=myClass.getProtectionDomain();
        System.out.println("!!!Got ProtectionDomain...");
        PermissionCollection coll=protDom.getPermissions();
        coll.add(new AllPermission());
```

```
            System.out.println("!!!Added new AllPermiss ion...");
        }

       catch(Exception e)
        {
            System.out.println("   Couldn´t get ProtectionDomain ("+
          e.toString()+")");
        }
        try
        {
            Class[] formparams=new Class[0];
            java.lang.reflect.Method getM=myClass.getClass().
          getMethod("getProtectionDomain",formparams);
            Object[] actparams=new Object[0];
            protDom=(ProtectionDomain)getM.invoke(myClass,actparams);
            System.out.println("!!!Got ProtectionDomain through
          reflection...");
        }
        catch(Exception e)
        {
            System.out.println("   Couldn´t get ProtectionDomain through
          reflection API ("+e.toString()+")");
        }
        if(protDom!=null)
        {
            try
            {
                PermissionCollection coll=protDom.getPermissions();
                System.out.println("!!!Got PermissionColle ction...");
                Class collClass=coll.getClass();
                Class[] formparams=new Class[1];
                formparams[0]=Class.forName("java.security.Permission");
                java.lang.reflect.Method addM=
              collClass.getMethod("add",formparams);
                Object[] actparams=new Object[1];
                actparams[0]=new AllPermission();
                addM.invoke(coll,actparams);
                System.out.println("!!!Added permission by
              reflection API...");
            }
            catch(java.lang.reflect.InvocationTargetException e)
            {
                System.out.println("   Couldn´t add permission through
              reflection-call ("+e.toString( )+"-> "+
              e.getTargetException().toString()+")");
            }
            catch(Exception e)
            {
                System.out.println("   Could´t get PermissionCollection ("+
              e.toString()+")");
            }
        }
        try
        {
            FileInputStream in=new FileInputStr eam("c:\\Autoexec.bat");
            System.out.println("!!!Read c:\\Autoexec.bat");
            in.read();
            in.close();
        }
        catch(Exception e)
        {
            System.out.println("   Couldn´t read c:\\autoexec.bat ("+
          e.toString()+")");
        }
        try {
            OutputStreamWriter out=new OutputStreamWriter(
          new LimitedFileOutputStream("c:\\Test.out.txt"));
```

```
            System.out.println("!!!Wrote to c:\\Test.out.txt");
            out.write("Testtext");
            out.close();
        }
        catch(Exception e)
        {
            System.out.println("   Couldn't write to c:\\test.out.txt ("+
          e.toString()+")");
        }
     try
     {
         Runtime.getRuntime().exec("cmd.exe /c dir");
            System.out.println("!!!Started process");
     }
     catch(Exception e)
     {
            System.out.println("   Couldn´t spawn child process ("+
          e.toString()+")");
     }
    }

    public void permissionsChanged(PermissionEvent event)
    {
        if(event.getType()==PermissionEvent.PERMISSION_GRANTED)
            System.out.println("Permission granted: "+
          event.getPermission().toString());
    }

    protected void initialize(Object param) throws Initializ ationException
    {
        addPermissionListener(this);
        if(param!=null && param instanceof Certificate)
            cert=(Certificate)param;
    }

}
```

2.1.2 Test-Agent: Beenden des Agentensystems

```
/*
 * @(#)Destroy.java
 *
 * Part of the AgentSystem "POND"
 * Developed at the Institute for Information Processing and Microprocessor
Technology (FIM)
 * Johannes Kepler University Linz
 * Altenbergerstr. 69, A-4040 Linz, Austria
 */

import PkgAgentSystem.*;
import java.io.*;
import java.security.*;
import java.security.cert.Certificate;
import PkgAgentSystem.Classification.*;

/**
The agent-class for testing security. This agent tries to terminate the
agent-system.

@author  Michael Sonntag
@version 1.0, 1.7.2000
@see     Test
*/
public class Destroy extends AgentBase
{
    public void agentMain()
    {
```

```
        System.out.println("Agent "+getAgentName()+" started.");
        final AgentSystem sys=getAgentSystem();
        System.out.println("Trying to cancel the agent system...");


         try
         {
              sys.terminate(10);
              System.out.println("!!!Agent system cancelled!!!");
         }
         catch(Exception e)
         {
              System.out.println("Couldn´t cancel agent system ("+
                   e.toString()+")");
         }
         AccessController.doPrivileged(new PrivilegedAction()
         {
              public Object run()
              {
                   System.out.println("Trying to cancel the agent system
                        when privileged...");
                   try
                   {
                        sys.terminate(10);
                        System.out.println("!!!Agent system cancelled!!!");
                   }
                   catch(Exception e)
                   {
                        System.out.println("Couldn´t cancel agent system
                             when privileged ("+e.toString()+")");
                   }
                   return null;
              }
        });
        System.out.println("Terminating agent "+getAgentName()+"...");
        stopAgent();
        // Should not be reached!
        System.out.println("Agent "+getAgentName()+" terminated.");
    }
}
```

2.1.3 Hauptprogramm

```
/*
 * @(#)ProtectionTest.java
 *
 * Part of the AgentSystem "POND"
 * Developed at the Institute for Information Processing and Microprocessor
Technology (FIM)
 * Johannes Kepler University Linz
 * Altenbergerstr. 69, A-4040 Linz, Austria
 */

import PkgAgentSystem.*;
import PkgAgentSystem.Crypto.AgentIdentity;
import java.net.URL;
import java.net.InetAddress;
import java.security.cert.Certificate;
import java.security.KeyPair;
import FIM.Util.Crypto.NamedKeyAndCertificate;

/**
The agentsystem-class for testing security. Tries all possible combinations
of code and owner certificates with the Test-Agent.

@author  Michael Sonntag
@version 1.0, 1.7.2000
```

```
@see     Test
*/
public class ProtectionTest extends AgentSystem
{


    public ProtectionTest(String homedir,String librarydir)
    {
        super(4711,librarydir,homedir,false,null);
    }

    public static void main(String args[])
    {
        if(args.length!=2)
        {
            System.out.println("Usage: java ProtectionTest home-directory
           library-directory");
            System.exit(1);
        }
        String homedir=args[0];
        System.out.println("ProtectionTest\n\nRequires a security store
          with the following elements:");
        System.out.println("    Trusted certificate with name 'ca'");
        System.out.println("    Untrusted identity with the name
          'Untrusted Identity'");
        System.out.println("    Keypair with name 'Untrusted Keypair'
          (Used in untrusted identity)");
        System.out.println();
        ProtectionTest as=new ProtectionTest(homedir,args[1]);
        System.out.println("Starting ProtectionTest");
        as.start();
        try
        {
            int id=1;
         InetAddress adr=InetAddress.getLocalHost();
            URL someLoc=new URL("amp",adr.getHostName(),"");
            NamedKeyAndCertificate nkc=as.securityStore.
          getAgentPrivateKeyAndIdentity("Untrusted Identity");
            if(nkc==null)
                throw new RuntimeException("Could not retrieve agent
              identity: 'Untrusted Identity'!");
            Certificate trusted=as.securityStore.
          getTrustedCertificate("ca");
            if(trusted==null)
                throw new RuntimeException("Could not retrieve trusted
              certificate: 'ca'!");
            KeyPair kp=as.securityStore.getKeyPair("Untrusted Ke ypair").
          getKeyPair();
            if(kp==null)
                throw new RuntimeException("Could not retrieve untrusted
              keypair: 'Untrusted Keypair'!");
            AgentIdentity identity=null;
            AgentBase main=null;
            // 1: No owner certificate; no code certificate
            System.out.println();
         System.out.println("No owner certificate; no code ce rtificate:
          Permission group 0");
         identity=new AgentIdentity("Test-Agent "+id,""+id,someLoc,
          "Michael Sonntag (Without certificate)");
            main=as.startAgent("Test",homedir+"\\code\\test.jar",
          false,identity,null,null,true);
            id++;
            main.join();
            // 2: No owner certificate; code certificate
            System.out.println();
         System.out.println("No owner certificate; code certif icate:
          Permission group 6");
```

```
identity=new AgentIdentity("Test-Agent "+id,""+id,someLoc,
 "Michael Sonntag (Without certificate)");
   main=as.startAgent("Test",homedir+"\\scode\\sTest.jar",
 false,identity,null,null,true);
   id++;
   main.join();
   // 3: No owner certificate; code certificate with ca-check
   System.out.println();
System.out.println("No owner certificate; code certificate with
 ca-check: Permission group 9");
identity=new AgentIdentity("Test-Agent "+id,""+id,someLoc,
 "Michael Sonntag (Without certificate)");
   main=as.startAgent("Test",homedir+"\\sCodeCA\\sTestca.jar",
 false,identity,null,null,true);
   id++;
   main.join();
   // 4: No owner certificate; code from local host
   System.out.println();
System.out.println("No owner certificate; code from local host:
 Permission group 1");
identity=new AgentIdentity("Test-Agent "+id,""+id,someLoc,
 "Michael Sonntag (Without certificate)");
   main=as.startAgent("Test",homedir+"\\code\\test.jar",
 true,identity,null,null,true);
   id++;
   main.join();
   // 5: Owner certificate; no code certificate
   System.out.println();
System.out.println("Owner certificate; no code certificate:
 Permission group 0");
   identity=new AgentIdentity("Test-Agent "+id,""+id,someLoc,
 "Untrusted Keypair",kp.getPublic(),"Michael Sonntag, DSA",
 nkc.getCertificate());
   main=as.startAgent("Test",homedir+"\\code\\test.jar",
 false,identity,null,null,true);
   id++;
   main.join();
   // 6: Owner certificate; code certificate
   System.out.println();
System.out.println("Owner certificate; code certificate:
 Permission group 7");
   identity=new AgentIdentity("Test-Agent "+id,""+id,someLoc,
 "Untrusted Keypair",kp.getPublic(),"Michael Sonntag, DSA",
 nkc.getCertificate());
   main=as.startAgent("Test",homedir+"\\scode\\sTest.jar",
 false,identity,null,null,true);
   id++;
   main.join();
   // 7: Owner certificate; code certificate with ca
   System.out.println();
System.out.println("Owner certificate; code certificate
 with ca: Permission group 10");
   identity=new AgentIdentity("Test-Agent "+id,""+id,someLoc,
 "Untrusted Keypair",kp.getPublic(),"Michael Sonntag, DSA",
 nkc.getCertificate());
   main=as.startAgent("Test",homedir+"\\scodeCA\\sTestca.jar",
 false,identity,null,null,true);
   id++;
   main.join();
   // 8: Owner certificate; code from local host
   System.out.println();
System.out.println("Owner certificate; code from local host:
 Permission group 1");
   identity=new AgentIdentity("Test-Agent "+id,""+id,someLoc,
 "Untrusted Keypair",kp.getPublic(),"Michael Sonntag, DSA",
 nkc.getCertificate());
   main=as.startAgent("Test",homedir+"\\code\\test.jar",
```

```
 true,identity,null,null,true);
    id++;
    main.join();
    // 9: Owner certificate with ca; no code certificate
    System.out.println();
System.out.println("Owner certificate with ca; no code
 certificate: Permission group 0");
    identity=new AgentIdentity("Test-Agent "+id,""+id,someLoc,
 "Untrusted Keypair",kp.getPublic(),"ca",trusted);
    main=as.startAgent("Test",homedir+"\\code\\test.jar",
 false,identity,null,null,true);
    id++;
    main.join();
    // 10: Owner certificate with ca; code certificate
    System.out.println();
System.out.println("Owner certificate with ca; code
 certificate: Permission group 8");
    identity=new AgentIdentity("Test-Agent "+id,""+id,someLoc,
 "Untrusted Keypair",kp.getPublic(),"ca",trusted);
    main=as.startAgent("Test",homedir+"\\scode\\sTest.jar",
 false,identity,null,null,true);
    id++;
    main.join();
    // 11: Owner certificate with ca; code certificate with ca
    System.out.println();
System.out.println("Owner certificate with ca; code certificate
 with ca: Permission group 11");
    identity=new AgentIdentity("Test-Agent "+id,""+id,someLoc,
 "Untrusted Keypair",kp.getPublic(),"ca",trusted);
    main=as.startAgent("Test",homedir+"\\scodeCA\\sTestca.jar",
 false,identity,null,null,true);
    id++;
    main.join();
    // 12: Owner certificate with ca; code from local host
    System.out.println();
System.out.println("Owner certificate with ca; code from local
 host: Permission group 1");
    identity=new AgentIdentity("Test-Agent "+id,""+id,someLoc,
 "Untrusted Keypair",kp.getPublic(),"ca",trusted);
    main=as.startAgent("Test",homedir+"\\code\\test.jar",
 true,identity,null,null,true);
    id++;
    main.join();
    // 13: Owner certificate from system-owner; no code certificate
    nkc=as.securityStore.getOwnerPrivateKeyAndCertificate(
 "Server Certificate");
    System.out.println();
System.out.println("Owner certificate from system-owner;
 no code certificate: Permission group 3");
    identity=new AgentIdentity("Test-Agent "+id,""+id,someLoc,
 "System Keypair",nkc.getCertificate().getPublicKey(),
 "System Certificate",nkc.getCertificate());
    main=as.startAgent("Test",homedir+"\\code\\test.jar",
 false,identity,nkc.getPrivateKey(),null,true);
    id++;
    main.join();
    // 14: Owner certificate from system-owner; code certificate
    System.out.println();
System.out.println("Owner certificate from system-owner;
 code certificate: Permission group 4");
    identity=new AgentIdentity("Test-Agent "+id,""+id,someLoc,
 "System Keypair",nkc.getCertificate().getPublicKey(),
 "System Certificate",nkc.getCertificate());
    main=as.startAgent("Test",homedir+"\\sCode\\sTest.jar",
 false,identity,nkc.getPrivateKey(),null,true);
    id++;
    main.join();
```

```
            // 15: Owner certificate from system-owner; code
           certificate with ca
            System.out.println();
           System.out.println("Owner certificate from system-owner;
           code certificate with ca: Permission group 5");
            identity=new AgentIdentity("Test-Agent "+id,""+id,someLoc,
           "System Keypair",nkc.getCertificate().getPublicKey(),
           "System Certificate",nkc.getCertificate());
            main=as.startAgent("Test",homedir+"\\scodeCA\\sTestca.jar",
           false,identity,nkc.getPrivateKey(),null,true);
            id++;
            main.join();
            // 16: Owner certificate from system-owner; code from
           local host
            System.out.println();
           System.out.println("Owner certificate from system-owner;
           code from local host: Permission group 2");
            identity=new AgentIdentity("Test-Agent "+id,""+id,someLoc,
           "System Keypair",nkc.getCertificate().getPublicKey(),
           "System Certificate",nkc.getCertificate());
            main=as.startAgent("Test",homedir+"\\code\\test.jar",
           true,identity,nkc.getPrivateKey(),null,true);
            id++;
            main.join();
            // 20: No owner certificate; no code certificate
            id=20;
            System.out.println();
           System.out.println("No owner certificate; no code certificate
           (Simple identity): Permission group 0");
           identity=new AgentIdentity("Test-Agent "+id,""+id,someLoc,
           "Michael Sonntag (Without certificate)");
            main=as.startAgent("Test",homedir+"\\code\\test.jar",
           false,identity,null,trusted,true);
            id++;
            main.join();
            // 333: No owner certificate; no code certificate; Tries to
           destroy the agent system
            System.out.println();
           System.out.println("Destroyer: No owner certificate;
           no code certificate: Permission group 0");
           identity=new AgentIdentity("Destroyer-Agent 333","333",
           someLoc,"The devil");
            main=as.startAgent("Destroy",homedir+"\\code\\test.jar",
           false,identity,null,null,true);
            id++;
            main.join();
        }
        catch(Exception e)
        {
            System.err.println(e.toString());
        }
        as.cancel();
        try
        {
            as.join();
        }
        catch(InterruptedException e)
        { }
        System.out.println("ProtectionTest ends!");
        System.exit(0);
    }

}
```

2.2. Einfacher Beispiels-Agent: Beeper

2.2.1 BeeperAgent

```
/*
 * @(#)Beeper.java
 *
 * Basic agents
 * Developed at the Institute for Information Processing and Microprocessor
Technology (FIM)
 * Johannes Kepler University Linz
 * Altenbergerstr. 69, A-4040 Linz, Austria
 */
package BaseAgents.Beeper;

import java.util.*;
import java.io.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

import PkgAgentSystem.*;
import PkgAgentSystem.GUI.*;
import PkgAgentSystem.Messaging.Message;
import PkgAgentSystem.Messaging.SignedMessage;
import PkgAgentSystem.Crypto.AgentIdentity;
import PkgAgentSystem.Crypto.AgentIdentityExtension;


/**
Example agent: Has a button and makes "beep" when clicked on it. This can
also be
issued as a command through <code>BeepConversation</code>.

@author  Michael Sonntag
@version 1.0, 1.4.2001
*/
public class BeeperAgent extends GUIAgentBase implements ActionListener
{
     protected transient JButton beep=new JButton("Beep!");

     public BeeperAgent()
     {
          registerConversation(new BeepConversation(this));
     }

     protected void showDialog()
     {
          doBeep();
     }

     protected void doBeep()
     {
          Toolkit.getDefaultToolkit().beep();
          logMessage("Beep!");
     }

    protected javax.swing.JPanel createVisualization()
    {
        javax.swing.JPanel visualization=new javax.swing.JPanel();
        visualization.setLayout(new BorderLayout());
        visualization.setBorder(new javax.swing.border.EtchedBorder());
        javax.swing.JPanel pane=new javax.swing.JPanel();
        pane.setLayout(new GridBagLayout());
        pane.setBackground(Color.white);
        pane.setOpaque(true);
        javax.swing.JLabel label=new javax.swing.JLabel("Beeper ("+
```

```
          getIdentity().getAgentName()+")");
        GridBagConstraints cons=new GridBagConstraints();
        cons.insets=new Insets(10,10,10,10);
        cons.fill=GridBagConstraints.HORIZONTAL;
        cons.anchor=GridBagConstraints.CENTER;
        cons.weightx=1.0;
        cons.weighty=0.0;
        cons.gridx=0;
        cons.gridy=0;
        pane.add(label,cons);
        cons.fill=GridBagConstraints.NONE;
        cons.ipadx=10;
        cons.ipady=10;
        cons.gridy=1;
        if(beep==null)
            beep=new JButton("Beep!");
        beep.setHorizontalAlignment(SwingConstants.CENTER);
        beep.addActionListener(this);
        pane.add(beep,cons);
        visualization.add(pane);
        return visualization;
    }

    public void actionPerformed(ActionEvent e)
    {
        showDialog();
    }

}
```

2.2.2 Testprogramm

```
/**
System for testing the beeper. Creates a beeper (with an extended identity)
and a beep sender.

@author  Michael Sonntag
@version 1.0, 1.4.2001
*/
class BeeperTestSystem extends GUIAgentSystem
{
    public BeeperTestSystem(int port,String libraryDir,String agen tDir,
     String systemDir,boolean exitOnLast,String savedFile,
     boolean useSecurity)
    {
        super(port,libraryDir,agentDir,systemDir,exitOnLast,
          savedFile,useSecurity);
    }

    public static void main(String[] args)
    {
        String home=System.getProperty("user.home");
        String librarydir=System.getProperty("user.dir");
        boolean security=true;
        if(args.length<3)
        {
            System.err.println("Usage: java BaseAgents.Beeper.BeeperAgent
               agentDir libraryDir systemDir [NOSEC]\n");
            System.exit(1);
        }
        home=args[0];
        librarydir=args[1];
        String systemDir=args[2];
        if(args.length>3)
        {
            if(args[3].equalsIgnoreCase("NOSEC"))
                security=false;
```

```
        }
        BeeperTestSystem as=new BeeperTestSystem(-1,librarydir,home,
               systemDir,false,null,security);
        as.start();
        AgentIdentityExtension beeperIdentity=new AgentIdentityExtens ion(
          "Beeper","0",as.getLocation(),"Michael Sonntag");
        AgentIdentity senderIdentity=new AgentIdentity("Beep se nder"
          ,"1",as.getLocation(),"Michael Sonntag");
        try
        {
            beeperIdentity.addInformation(
          AgentIdentityExtension.FIRST_NAME,"Michael");
            beeperIdentity.addInformation(
          AgentIdentityExtension.LAST_NAME,"Sonntag");
            beeperIdentity.addInformation(
          AgentIdentityExtension.TITLE,"Dipl.-Ing.");
            beeperIdentity.addInformation(
          AgentIdentityExtension.EMAIL,"sonntag@fim.uni-linz.ac.at");
            beeperIdentity.addInformation("Magic word","James POND");
            as.startAgent("BaseAgents.Beeper.BeeperAgent",null,
          false,beeperIdentity,null,null,true);
            as.startAgent("BaseAgents.Beeper.SendBeepAgent",null,
          false,senderIdentity,null,null,true);
        }
        catch(Exception e)
        {
            System.err.println(e.toString());
        }
        try
        {
            if(as.isAlive())
                as.join();
        }
        catch(InterruptedException e)
        {   // Do nothing, just end (Shouldn't happen anyway)
        }
        System.exit(0);
    }
}
```

2.2.3 BeepMessage

```
/*
 * @(#)BeepMessage.java
 *
 * Basic agents
 * Developed at the Institute for Information Processing and Microprocessor
Technology (FIM)
 * Johannes Kepler University Linz
 * Altenbergerstr. 69, A-4040 Linz, Austria
 */

package BaseAgents.Beeper;

import PkgAgentSystem.Messaging.BroadcastMessage;

/**
Message transporting the command to "beep". Has no content; the command is
transeferred through the class.
Is a broadcast as the beep-agent will be usually unknown.

@author  Michael Sonntag
@version 1.0, 1.4.2001
*/
public class BeepMessage extends BroadcastMessage
{
```

```
    public BeepMessage()
    {
    }

    public String toString()
    {
        return "Beep!";
    }

}
```

2.2.4 BeepConversation

```
/*
 * @(#)BeepConversation.java
 *
 * Basic agents
 * Developed at the Institute for Information Processing and Microprocessor
Technology (FIM)
 * Johannes Kepler University Linz
 * Altenbergerstr. 69, A-4040 Linz, Austria
 */

package BaseAgents.Beeper;

import PkgAgentSystem.AgentBase;
import PkgAgentSystem.Conversation;
import PkgAgentSystem.Messaging.MessageType;
import PkgAgentSystem.Messaging.MessageException;
import PkgAgentSystem.Crypto.AgentIdentity;


/**
Simple conversation: Allows sending a broadcast and when receiving an a p-
propriate message
calls the beep-method in the agent.

@author  Michael Sonntag
@version 1.0, 1.4.2001
*/
public class BeepConversation extends Conversation
{

    /**
    The ID for this type of conversation (4737).
    */
    public final static long conversationTypeID=4737;

    public BeepConversation(AgentBase myAgent)
    {
        super(myAgent);
    }

    protected boolean handleMessage(MessageType msg)
    {
        if(super.handleMessage(msg))
            return true;
        if(isInitiator())
        {
            // Send a broadcast
            try
            {
                broadcastMessage(new BeepMessage());
```

```
            }
            catch(MessageException e)
            {
                getAgent().logMessage("Error sending beep-broadcast: "+
              e.toString());
            }
            endConversation();
            return true;
        }
        else
        {
            // Broadcast received
            if((msg instanceof BeepMessage) &&
              (getAgent() instanceof BeeperAgent))
            {
                // It is a BeepMessage and we are a beeper, so beep!
                getAgent().logMessage("Received beep command");
                ((BeeperAgent)getAgent()).doBeep();
                endConversation();
                return true;
            }
        }
        endConversation();
        return false;
    }

}
```

2.2.5 SendBeepAgent

```
/*
 * @(#)SendBeepAgent.java
 *
 * Basic agents
 * Developed at the Institute for Information Processing and Microprocessor
Technology (FIM)
 * Johannes Kepler University Linz
 * Altenbergerstr. 69, A-4040 Linz, Austria
 */

package BaseAgents.Beeper;

import java.util.*;
import java.io.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

import PkgAgentSystem.*;
import PkgAgentSystem.GUI.*;
import PkgAgentSystem.Messaging.Message;
import PkgAgentSystem.Messaging.SignedMessage;

/**
An example agent for sending a beep. If clicked on the button, a broadcast
containing a
<code>BeepMessage</code> is sent. If a <code>BeeperAgent</code> is present,
it will beep.

@author  Michael Sonntag
@version 1.0, 1.4.2001
*/
public class SendBeepAgent extends GUIAgentBase implements ActionListener
{
     protected transient JButton beep=new JButton("Send Beep");

     public SendBeepAgent()
```

```
    {
        registerConversation(new BeepConversation(this));
    }

    protected void showDialog()
    {
        sendBeep();
    }
        protected void sendBeep()
    {
        logMessage("Sending beep");
        Conversation conv=newConversation(
            BeepConversation.conversationTypeID,null);
        startConversation(conv);
        logMessage("Beep sent");
    }


    protected javax.swing.JPanel createVisualization()
    {
        javax.swing.JPanel visualization=new javax.swing.JPanel();
        visualization.setLayout(new BorderLayout());
        visualization.setBorder(
            new javax.swing.border.EtchedBorder());
        javax.swing.JPanel pane=new javax.swing.JPanel();
        pane.setLayout(new GridBagLayout());
        pane.setBackground(Color.white);
        pane.setOpaque(true);
        javax.swing.JLabel label=new javax.swing.JLabel(
            "Beep sender ("+getIdentity().getAgentName()+")");
        GridBagConstraints cons=new GridBagConstraints();
        cons.insets=new Insets(10,10,10,10);
        cons.fill=GridBagConstraints.HORIZONTAL;
        cons.anchor=GridBagConstraints.CENTER;
        cons.weightx=1.0;
        cons.weighty=0.0;
        cons.gridx=0;
        cons.gridy=0;
        pane.add(label,cons);
        cons.insets=new Insets(10,10,10,10);
        cons.fill=GridBagConstraints.NONE;
        cons.anchor=GridBagConstraints.CENTER;
        cons.ipadx=10;
        cons.ipady=10;
        cons.weightx=1.0;
        cons.weighty=0.0;
        cons.gridx=0;
        cons.gridy=1;
        if(beep==null)
            beep=new JButton("Send Beep");
        beep.setHorizontalAlignment( SwingConstants.CENTER);
        beep.addActionListener(this);
        pane.add(beep,cons);
        visualization.add(pane);
        return visualization;
    }

    public void actionPerformed(ActionEvent e)
    {
        showDialog();
    }
}
```

2.3. Kommando zur Rückkehr

2.3.1 GoHomeCommand

```
/* @(#)GoHomeCommand.java
 * Basic agents
 * Developed at the Institute for Information Processing and Microprocessor
Technology (FIM)
 * Johannes Kepler University Linz
 * Altenbergerstr. 69, A-4040 Linz, Austria
 */

package BaseAgents.CommandReturn;

/** An interface for marking an agent to be able to accept commands to go
to another host. How the command is verified (signed message, etc.) is not
defined.<br>
<strong>May only be used on subclasses of BaseAgent!!!</strong>

@author Michael Sonntag
@version 1.0
*/
public interface GoHomeCommand
{
    /**
    Called when the agent should go to another host.
    @param whereTo The host to go to. If <code>null</code>,
           go to the home host.
    @return <code>false</code> if an error occured (if successful, will
           not exit as terminated from within)
    */
    public boolean goHome(java.net.URL whereTo);
}
```

2.3.2 RetractableAgent

```
/* @(#)RetractableAgent.java
 *
 * Basic agents
 * Developed at the Institute for Information Processing and Microprocessor
Technology (FIM)
 * Johannes Kepler University Linz
 * Altenbergerstr. 69, A-4040 Linz, Austria
 */

package BaseAgents.CommandReturn;

import java.awt.*;
import java.awt.event.*;
import PkgAgentSystem.*;
import PkgAgentSystem.GUI.*;
import PkgAgentSystem.Messaging.*;

/**
Base class for an agent, which can be "retrieved" from another host.
Uses the <code>SendCommandConversation</code> for receiving the command,
which verifies the command according to the signature, which must be from
the sender of the message.
<br>Just implements the <code>GoHomeCommand</code> interface.
@author Michael Sonntag
@version 1.0
*/
public class RetractableAgent extends GUIAgentBase implements GoHomeCommand
{

    public RetractableAgent()
    {
```

```
        registerConversation(new SendCommandConversation(this));
    }

    public boolean goHome(java.net.URL whereTo)
    {
        try
        {
            if(whereTo==null)
                whereTo=getIdentity().getHomeAddress();
            move(whereTo);
        }
        catch(Exception e)
        {
            logMessage("Couldn't go home to "+whereTo.toString()+".");
            return false;
        }
        return true;
    }

}
```

2.4. Programmabschnitte zum Erwerb von Berechtigungen

```
// Private key (and AgentIdentity) of the agent to be filled in on creation
private NamedKeyAndCertificate nkc=null;
...
{
...
     Permission perm=null;
     if(!myAgent.hasPermission(
          new java.util.PropertyPermission("*","read")))
     {
          perm=buyPermission(
               new java.util.PropertyPermission("*","read"));
              if(perm==null)
          {
               myAgent.logMessage("Couldn't get necessary property
                     permission!");
                    return null;
          }
     }
...
}

...

private Permission buyPermission(Permission perm)
{
     Invoice inv=myAgent.getAgentSystem().getPrice(myAgent,perm);
     Calendar cal=Calendar.getInstance();
     cal.setTime(new Date());
     cal.add(Calendar.YEAR,1);
     byte[] number={4,0,0,0,0,8,1,5,4,7,1,1,0,3,3,3,3};
     Payment payment=new CreditCardPayment(inv,"ACME.Card",
          "John Doe",number,cal.getTime());
     if(nkc==null)   // Don't have info -> cannot buy permissions
          return null;
     PrivateKey pk=nkc.getPrivateKey();
     String algo=pk.getAlgorithm();
     if(algo.equalsIgnoreCase("RSA"))
          algo="MD5/RSA";
     try
     {
          payment.sign(algo,pk,nkc.getCertificate());
     }
     catch(Exception e)
```

```
     {
          myAgent.logMessage("Error signing payment for the necessary
               permission ("+perm.toString()+"): "+e.toString());
          return null;
     }
     return myAgent.getAgentSystem().
          buyPermission(myAgent,perm,payment);
}
```

2.5. InquiryProtocol

```
package Protocol.Inquiry;

import PkgAgentSystem.AgentBase;
import PkgAgentSystem.Crypto.AgentIdentity;
import PkgAgentSystem.Messaging.UnknownAgentException;
import PkgAgentSystem.Messaging.MessageException;
import PkgAgentSystem.Messaging.IllegalMessageException;
import PkgAgentSystem.Messaging.Message;

import Protocol.Protocol;
import Protocol.Contact.ContactProtocol;

/**
We inquiry about something. The actual inquiry is defined by the data
contained in the inquiry message. The other agent may react on the inquiry
either with an answer, a rejection, ill-formed request or information-not-
available message.
Ending states:
3: Successful termination
14: The inquiry could not be sent to the other agent
15: The answer could not be sent to the inquiring agent
16: The other agent sent a rejection message
17: No inquiry provided
18: Contact-suprotocol failed
@author Michael Sonntag
@version 1.0
*/
public abstract class InquiryProtocol extends Protocol
{

    /**
    The ID for this type of conversation.
    */
    public final static long conversationTypeID=1003;

    /**
    The message containing the inquiry we send to the other party.
    */
    private InquiryMessage inquiry=null;

    /**
    The message containing the answer from the other party.
    */
    private AnswerMessage answer=null;

    /**
    The message containing the reject from the other party (if accepted,
    stays <code>null</code>).
    */
    private RejectMessage reject=null;

    /**
    Start the inquiry protocol. We are set as the initator.
    @param agent ourself
    @param the other partner to ask for something
    */
```

```
public InquiryProtocol(AgentBase agent,AgentIdentity other,
          boolean asInitiator)
{
    super(agent,other,asInitiator);
}

/**
Constructor for the recipient of the initial message. The other
(initiators) identity is later retrieved from the message.
@param agent ourself
*/
public InquiryProtocol(AgentBase agent)
{
    super(agent);
}

/**
Sets the inquiry we want to send to the other agent
@param inquiry the inquiry to send to the other agent
*/
public void setInquiry(org.w3c.dom.Document inquiry)
{
    if(this.inquiry!=null)
        throw new IllegalStateException("Inquiry already set!");
    if(inquiry==null)
        throw new IllegalArgumentException("Inquiry required!");
    this.inquiry=new InquiryMessage(inquiry);
}

/**
Retrieve the inquiry.
@return the inquiry
*/
public org.w3c.dom.Document getInquiry()
{
    return inquiry==null?null:inquiry.getInquiry();
}

/**
Retrieve the answer.
@return the answer
*/
public org.w3c.dom.Document getAnswer()
{
    return answer==null?null:answer.getAnswer();
}

/**
Retrieve the rejection message.
@return the rejection message
*/
public RejectMessage getRejectMsg()
{
    return reject;
}

/**
Send our inquiry to the other agent
*/
public void sendInquiry()
{
    state.curState=2;
    try
    {
        sendMessage(inquiry);
    }
    catch(MessageException e)
```

```
        {
            getAgent().logMessage("Could not send inquiry message ("+
                   e.toString()+"). Ending protocol.");
            state.curState=14;
            endProtocol();
        }
    }

    public void answerInquiry(InquiryMessage msg)
    {
        state.curState=3;
        try
        {
            sendMessage(doAnswerInquiry(msg));
        }
        catch(MessageException e)
        {
            getAgent().logMessage("Could not send answer message ("+
                   e.toString()+"). Ending protocol.");
            state.curState=15;
        }
        endProtocol();
    }


    /**
    We answer the inquiry of the other agent.
    @param msg the message we received containing the inquiry
    @return the message containing the answer or rejection
    */
    public abstract Message doAnswerInquiry(InquiryMessage msg);

    /**
    Receive the answer to our inquiry.
    @param msg the message we received containing the other parties answer
    */
    public void receiveAnswer(AnswerMessage msg)
    {
        state.curState=3;
        answer=msg;
        endProtocol();
    }

    /**
    Receive the rejection of our inquiry.
    @param msg the message we received containing the others reje ction
    */
    public void receiveRejection(RejectMessage msg)
    {
        state.curState=16;
        reject=msg;
        endProtocol();
    }

    /**
    Adjust our state according to how the contact protocol ended
    */
    public void adjustStateAfterSubprotocolTerminated(Protocol subprotocol)
    {
        if(subprotocol.getState().curState!=2)
        {
            state.curState=18;
            endProtocol();
        }
    }

    /**
```

```
    Performs the inquiry protocol by handling the message.
    @param msg the message we received
    @return the state of this conversation (whether it has ended or not
         and in which state: error, success, etc.)
    */
    public ProtocolState doProtocol(Message msg)
    {
        if(isInitiator())
        {
            switch(state.curState)
            {
                case 0:
                    if(inquiry==null)
                    {
                        state.curState=17;
                        endProtocol();
                    }
                    else
                    {
                        state.curState=1;
                        startSubProtocol(new ContactProt ocol(getAgent(),
                       getCounterpartIdentity(),true),null);
                    }
                break;


                case 1:
                    sendInquiry();
                break;
                case 2:
                    if(msg instanceof AnswerMessage)
                    {
                        receiveAnswer((AnswerMessage)msg);
                    }
                    else if(msg instanceof RejectMessage)
                    {
                        receiveRejection((RejectMessage)msg);
                    }
                    else
                        return null;
                break;
                default:
                    endProtocol();
                break;
            }
        }
        else
        {
            switch(state.curState)
            {
                case 0:
                    state.curState=1;
                    startSubProtocol(new ContactProt ocol(getAgent(),
                   getCounterpartIdentity(),false),msg);
                break;
                case 1:
                    state.curState=2;
                break;
                case 2:
                    if(msg instanceof InquiryMessage)
                        answerInquiry((InquiryMessage)msg);
                    else
                        return null;
                break;
                default:
                    endProtocol();
                break;
```

```
            }
        }
        return state;
    }

}
```

2.6. ContactProtocol

```
package Protocol.Contact;

import PkgAgentSystem.AgentBase;
import PkgAgentSystem.Crypto.AgentIdentity;
import PkgAgentSystem.Messaging.UnknownAgentException;
import PkgAgentSystem.Messaging.MessageException;
import PkgAgentSystem.Messaging.IllegalMessageException;
import PkgAgentSystem.Messaging.Message;
import PkgAgentSystem.Messaging.MessageType;

import Protocol.Protocol;

/**
Basic identification of the other agent and termination of the conversat i-
on.
Termination is independent of who was the initiator of the conversation.
Both sides must always
be prepared to receive a <code>TerminationMessage</code> at any time and
always have to react
appropriately by sending a confirmation and terminating the conversation.
Ending states:
2: Identity matches connection
3: Identity does NOT match the connection
4: Could not successfully exchange identities
@author Michael Sonntag
@version 1.0
*/
public class ContactProtocol extends Protocol
{
    /**
    The ID for this type of conversation.
    */
    public final static long conversationTypeID=1001;

    /**
    Partner of the communication (this is the exchanged identity;
    Conversation#getCounterpartIdentity()returns the one set by the user
    / retrieved from the first message).
    */
    private AgentIdentity other=null;

    /**
    Start the contact protocol. We are set as the initator.
    @param agent ourself
    @param other partner of the communication
    */
    public ContactProtocol(AgentBase agent,AgentIdentity other,
                   boolean asInitiator)
    {
        super(agent,other,asInitiator);
        if(other==null)
            throw new IllegalArgumentException("Identity of counterpart
                              must be provided");
        this.other=other;
    }

    /**
    Constructor for the recipient of the initial message. The other
    (initiators) identity is later retrieved from the message.
    @param agent ourself
    */
    public ContactProtocol(AgentBase agent)
    {
        super(agent);
```

```
}
/**
Retrieve the other agent.
@return the remote agent
*/
public AgentIdentity getOtherAgent()
{
    return other;
}

/**
Starts the communication with the other agent by exchanging the
identities. First we send our identity and the other agent should
answer by replying with his identity.
*/
public void sendIdentification()
{
    state.curState=1;
    try
    {
        sendMessage(new SendIdentityMessage(getAgent().getIdentity()));
    }
    catch(UnknownAgentException e)
    {
        getAgent().logMessage("Other agent unknown (Could not deliver
                message). Ending protocol.");
        state.curState=4;
        endProtocol();
    }
    catch(MessageException e)
    {
        getAgent().logMessage("Could not send identity message.
               Ending protocol.");
        state.curState=4;
        endProtocol();
    }
}

/**
We answer the request of the other party to start the transaction. The
others identity is retrieved from the message, compared to the identity
in the message and stored. In reply we send our own identity.
@param msg the message we received containing the initiators identity
*/
public void answerIdentification(SendIdentityMessage msg)
{
    if(!msg.getIdentity().equals(getCounterpartIdentity()))
        state.curState=3;
    other=msg.getIdentity();
    state.curState=2;
    try
    {
        Message nmsg=new SendIdentityMessage(getAgent().getIdentity());
        sendMessage(nmsg);
    }
    catch(MessageException e)
    {
        getAgent().logMessage("Could not send identity message as an
               answer. Ending protocol.");
        state.curState=4;
    }
    endProtocol();
}
```

```
/**
Receive the identity of the other party. This must match first the
parameters of the message and second the identity we provided for
initiating the conversation.
@param msg the message we received containing the other parties
       identity
*/
public void receiveIdentification(SendIdentityMessage msg)
{
    state.terminated=true;
    if(msg.getIdentity().equals(getCounterpartIdentity()))
    {
        state.curState=2;
        other=getCounterpartIdentity();
    }
    else
    {
        getAgent().logMessage("Identity provided does not match the
               conversation. Ending protocol.");
        state.curState=3;
    }
    endProtocol();
}

/**
Send the termination message to the other agent. We still have to wait
till the other agent confirms the termination. If a timeout is desired,
this has to be done externally.
@param msg the <code>TerminationMessage</code> to send, which might
          include a reason for termination
@throws MessageException if an error occured during sending the
          termination message
*/
public void sendTermination(TerminationMessage msg)
                    throws MessageException
{
    sendMessage(msg);
}

/**
Sends the confirmation that the transaction is to be ended. The
conversation should terminate even if an error occured during sending
(on problems, the other side should use a timeout).
@throws MessageException if an error occured during sending the
          confirmation message
@param msg the termination message we received
*/
public void confirmTermination(TerminationMessage msg)
                    throws MessageException
{
    state.curState=9999;
    sendMessage(new TerminationConfirmationMessage());
    endProtocol();
}

/**
Receives the termination message from the other party. This indicates,
that it already has abandoned the conversation, and so should we now.
@param msg the confirmation for the termination we initiated
*/
public void receiveTerminationConfirmation(
                    TerminationConfirmationMessage msg)
{
    endProtocol();
```

```
}



/**
Performs the startup protocol by handling the message.
@param msg the message we received
@return the state of this conversation (whether it has ended or not and
        in which state: error, success, etc.)
*/
public ProtocolState doProtocol(MessageType msg)
{
 if(state.curState==9998 && msg instanceof TerminationMess age)
    {
        try {
            sendTermination((TerminationMessage)msg);
            state.curState=9999;
            return state;
        }
        catch(MessageException e) {
            getAgent().logMessage("Could not send termination message
          ("+e.toString()+"). Ending protocol anyway.");
            endProtocol();
        }
    }
    if(state.curState==9999 || (msg instanceof TerminationMessage) ||
          (msg instanceof TerminationConfirmationMessage))
    {
        try {
      return doTerminationProtocol(msg);
        }
        catch(MessageException e) {
      getAgent().logMessage("Could not send termination message
          ("+e.toString()+"). Ending protocol anyway.");
            endProtocol();
        }
    }
    if(isInitiator())
    {
        switch(state.curState)
        {
            case 0:
                sendIdentification();
            break;
            case 1:
                if(msg instanceof SendIdentityMessage)
                    receiveIdentification((SendIdentityMessage)msg);
                else
                    return null;
            break;
            default:
                endProtocol();
            break;
        }
    }
    else {
        switch(state.curState)
        {
            case 0:
                if(msg instanceof SendIdentityMessage)
                    answerIdentification((SendIdentityMessage)msg);
                else
                    return null;
            break;
            default:
                endProtocol();
            break;
```

```
            }
        }
        return state;
    }
    /** Performs the termination protocol by sending a termination message.
    The recipient must always check any message received if it is a
    <code>TerminationMessage</code> and react on it by ca lling
    {@link doTerminationProtocol(Message msg)
    #doTerminationProtocol(Message)}. All messages other
    than <code>TerminationConfirmationMessage</code> from the other agent
    should be ignored.
    @param reason the (optional) reason for ending the protocol
    @return the state of this conversation (curState=9999)
    @throws MessageException if an error occured during the processing of
            the message
    */
    public ProtocolState terminateProtocol(String reason)
                            throws MessageException
    {
        sendTermination(new TerminationMessage(reason));
        state.curState=9999;
        return state;
    }

    /** Performs the termination protocol. The recipient must always check
    any message received if it is a <code>TerminationMessage</code> and
    react on it by calling {@link doTerminationPr otocol(Message msg)
    #doTerminationProtocol(Message)}. All messages other than
    <code>TerminationConfirmationMessage</code> from the other agent should
    be ignored after this.
    @param msg the message we received or the
               <code>TerminationMessage</code> including a reason if
               starting the termin ation
    @return the state of this conversation (whether it has ended or not and
               in which state: error, success, etc.)
    @throws MessageException if an error occured during the processing of
               the message
    @throws IllegalMessageException if the message passed as parameter is
               not a <code>TerminationConfirmationMessage</code>
    */
    public ProtocolState doTerminationProtocol(MessageType msg)
              throws MessageException,IllegalMessageException
    {
        if(msg instanceof TerminationMessage)
        {
            confirmTermination((TerminationMessage)msg);
        }
        else if(msg instanceof TerminationConfirmationMessage)
        {
            receiveTerminationConfirmat ion(
                   (TerminationConfirmationMessage)msg);
        }
        else
            return null;
        return state;
    }

    /** Check whether the contact was successfully established or not.
    @return <code>true</code> if the state is 2, which means the protocol
            terminated succes sfully
    */
    public boolean successfullyIdentified()
    {
        return state.curState==2;
    }

    public void setStartTermination()
```

```
    {
        state.curState=9998;
    }

}
```

Abkürzungsverzeichnis

AA	Attribute Authority (Zertifizierungsstelle für Attribute)
ACL	Agent Communication Language (Agenten-Kommunikationssprache)
AI	Artificial Intelligence (Künstliche Intelligenz)
CA	Certificate Authority (Zertifizierungsstelle)
CORBA	Common Object Request Broker Architecture
DAI	Distributed Artificial Intelligence (Verteilte künstliche Intelligenz)
DOM	Document Object Model (Dokumenten-Objektmodell)
FAQ	Frequently Asked Questions (Oft gestellte Fragen)
IA	Intelligent Agent (Intelligenter Agent)
IT	Informations Technologie
JDK	Java Development Kit (Java Entwicklungsumgebung)
JVM	Java Virtual Machine (Virtuelle Java Maschine; Laufzeitumgebung)
MAS	Multi-Agentensysteme
MP	Message Passing (Nachrichten Transfer)
ORB	Object Request Broker (Teil von CORBA)
OOD	Object-Oriented Design (Objekt-orientierter Entwurf)
OOM	Object-Oriented Modeling (Objekt-orientierte Modellierung)
OOP	Object-Oriented Programming (Objekt-orientierte Programmierung)
OS	Operating System (Betriebssystem)
PDA	Personal Digital Assistant (Persönlicher Digitaler Assistent)
PKI	Public-Key Infrastructure (Public-Key Infrastruktur)
RPC	Remote Procedure Call (Entfernter Prozedur-Aufruf)
RMI	Remote Method Invocation (Entfernter Methoden-Aufruf)
UUID	Universal Unique Identifier (Weltweit eindeutiger Bezeichner)
VM	Virtual Machine (Virtuelle Maschine)
XML	eXtensible Markup Language (Erweiterbare Beschreibungssprache)
XSL	eXtensible Stylesheet Language (Erweiterbare Formatvorlagensprache)

Abbildungsverzeichnis

Literaturverzeichnis

[Acharya/Edjlali 1997] Anurag Acharya, Guy Edjlali: History-bases Access Control for Mobile Code. Technical report TRCS97-25 (ACM-CCCS-98)

[Active-X] Microsoft: Active-X Controls http://www.microsoft.com/com/tech/activex.asp (21.12.2001)

[Aglets] IBM Aglets Software Development Kit Home http://www.trl.ibm.co.jp/aglets/ (21.12.2001)

[Altavista] altavista. The search company: http://www.altavista.com/ (21.12.2001)

[Anthony et al. 2001] Patricia Anthony, Wendy Hall, Viet D. Dang and Nicholas R. Jennings: Autonomous agents for participating in multiple on-line auctions. In: Proceedings IJCAI Workshop on E-Business and the Intelligent Web. Seattle 2001 54-64

[Asokan et al. 1996] N. Asokan, Matthias Schunter, Michael Waidner: Optimistic Protocols for Fair Exchange. IBM Research Report RZ 2858 (#90806) 09/02/96

[Tai/Kosaka 1999] Hideki Tai, Kazuya Kosaka: The Aglets Project. Communications of the ACM March 1999. New York: acm Press 1999. 100f (Volume 42, Number 3)

[Aglets.org] aglets.org http://www.aglets.org/ (24.10.2000)

[Applets] Sun: Applets Applets http://java.sun.com/applets/ (21.12.2001)

[Atrada] Atrada: Angebots-Agent http://www.atradapro.de/at/PartnerSite/Support/Features/AngebotsAgent.asp (21.12.2001),
Ausschreibungs-Agent http://www.atradapro.de/at/PartnerSite/Support/Features/GesuchsAgent.asp (21.12.2001)

[Auction-Bot] Auction-Bot http://auction.eecs.umich.edu (2.8.2000)

[Bates 1994] Joseph Bates: The Role of Emotion in Believable Agents. Communications of the ACM July 1994. New York: acm Press 1994. 122-125 (Volume 37, Number 7).

[Blaschek 2000] Günther Blaschek: Modes in User Interfaces of Information Systems. 81-98 In: Hofer Susanne, Beneder Manfred (Ed.): IDIMT'00. 8th Interdisciplinary Information Management Talks. Linz: Universitätsverlag Rudolf Trauner 2000

[Bonifati 2001] Angela Bonifati, Stefano Ceri, Stefano Paraboschi: Pushing Reactive Services to XML Repositories using Active Rules. In Proc. 10th World-Wide-Web Conference, 2001.

[Bredin et al. 1998] Jonathan Bredin, David Kotz, Daniela Rus: Market-based Resource Control for Mobile Agents. In: Proceedings of the Second International Conference on Autonomous Agents. ACM Press 1998

[Brenner et al. 1998] Walter Brenner, Rüdiger Zarnekow, Hartmut Wittig: Intelligente Softwareagenten. Grundlagen und Anwendungen. Berlin et al.: Springer 1998. ISBN 3-540-63431-2

[Brewington et al. 1999] Brian Brewington, Robert Gray, Katsuhiro Moizumi, David Kotz, Georg Cybenko, Daniela Rus: Mobile Agents for Distributed Information Retrieval. 355-395 In: Matthias Klusch (Ed.): Intelligent Information Agents. Agent-Based Information Discovery and Management on the Internet. Berlin: Springer 1999

[BSCW] GMD: BSCW (Basic Support for Cooperative Work) http://bscw.gmd.de/ (21.12.2001)

[BSI-A1] Schnittstellenspezifikation zur Entwicklung interoperabler Verfahren und Komponenten nach SigG/SigV. Signatur-Interoperabilitätsspezifikation SigI. Abschnitt A1: Zertifikate. http://www.bsi.bund.de/esig/basics/techbas/interop/bsi/sigi-a1.pdf (21.12.2001)

[BSI-A2] Schnittstellenspezifikation zur Entwicklung interoperabler Verfahren und Komponenten nach SigG/SigV. Signatur-Interoperabilitätsspezifikation SigI. Abschnitt A2: Signatur. http://www.bsi.bund.de/esig/basics/techbas/interop/bsi/sigi-a2.pdf (21.12.2001)

[Burmeister et al. 1998] B. Burmeister, S. Bussmann, A. Haddadi, K. Sundermeyer: Agent-Oriented Techniques for Traffic and Manufacturing Applications: Progress Report. 161-174 In: Nicholas R.

Jennings, Michael J. Wooldridge (Ed.): Agent Technology. Foundations, Applications, and Markets. Berlin: Springer 1998

[CBT-LVA] CBT Seminar http://www.fim.uni-linz.ac.at/Research/Applaud/CBT-Seminar/index.htm (21.12.2001)

[Charlton et al. 2001] Patricia Charlton, David Bonnefoy, Nicolas Lhuillier: Dealing with Interoperability for Agent-Based Services. http://leap.crm-paris.com/agentcities/Resources/resources.html (21.12.2001)

[Chess 1998] David M. Chess: Security Issues in Mobile Code Systems. 1-14 In: Giovanni Vigna (Ed.): Mobile Agents and Security. Berlin: Springer 1998. (Lecture Notes in Computer Science 1419)

[CNETShoppper] CNET Shopper http://shopper.cnet.com (21.12.2001)

[CookieCentral] cookiecentral.com: http://www.cookiecentral.com/ (21.12.2001)

[Cookie-Spec] Netscape: Persistent Client State HTTP Cookies http://home.netscape.com/newsref/std/cookie_spec.html (21.12.2001)

[Cugola et al. 1996] Gianpaolo Cugola, Carlo Ghezzi, Gian Pietro Picco, Giovanni Vigna: A Characterization of Mobility and State Distribution in Mobile Code Languages. In: Proceedings of the 2nd ECOOP Workshop on Mobile Object Systems. Linz 1996, 10-19

[DAgents] D'Agents Homepage http://agent.cs.dartmouth.edu/ (21.12.2001)

[DCE] The Open Group Portal to the World of DCE: http://www.opengroup.org/dce/ (21.12.2001)

[DCE-RPC] dcerpc.net: http://dcerpc.net/ (21.12.2001)

[Decker/Lesser 1992] K. Decker, V. Lesser: Generalizing the Partial Global Planning Algorithm. In International Journal on Intelligent Cooperative Information Systems, Vol. 1, Nr. 2, 319-346, 1992.

[Delgado et al. 1998] Joaquin Delgado, Naohiro Ishii, Tomoki Ura: Content-Based Collaborative Information Filtering: Actively Learning to Classify and Recommend Documents. In: Matthias Klusch, Gerhard Weiß (Ed.): Cooperative Information Agents II. Learning, Mobility and Electronic Commerce for Information Discovery on the Internet. Berlin: Springer 1998 (Lecture Notes in Artificial Intelligence 1435)

[Diemucha] Diemucha: Die erste Webseite für den Kundenfrust. http://www.diemucha.at/ (21.12.2001)

[DSG] Datenschutzgesetz – 2000, BGBl. I Nr. 165/1999

[DsigG] (Deutsches) Gesetz zur digitalen Signatur (Signaturgesetz - SigG): Art. 3 Gesetz zur Regelung der Rahmenbedingungen für Informations- und Kommunikationsdienste (Informations- und Kommunikationsdienste-Gesetz - IuKDG)

[DSRL] Richtlinie 95/46/EG des Europäischen Parlamentes und des Rates vom 24. Oktober 1995 zum Schutz natürlicher Personen bei der Verarbeitung personenbezogener Daten und zum freien Datenverkehr. Abl L 281, 23.11.1995, 31ff

[Doberkat 2000] Ernst-Erich Doberkat: Agenten. Ein kleiner Überblick. LOG IN 20 (2000) 10-16

[DOM] W3C: Document Object Model Specification http://www.w3.org/DOM/ (21.12.2001)

[Doran et al. 1997] J. E. Doran, S. Franklin, N. R. Jennings and T. J. Norman: On cooperation in multi-agent systems. The Knowledge Engineering Review, 12(3), 309-314, 1997 http://www.csd.abdn.ac.uk/~tnorman/publications/fomas.html (21.12.2001)

[Duden 1984] Ingrid Adam, et al.: Das neue Duden-Lexikon. Mannheim: Bibliographisches Institut 1984

[E-Bay] E-Bay: Proxy-Bidding http://pages.ebay.com/help/buyerguide/bidding-prxy.html (21.12.2001)

[ebXML] ebXML: http://www.ebxml.org/ (21.12.2001)

[eCash] Deutsche Bank AG http://www.ecash.de/ (13.3.2001)

[ECG] Ministerialentwurf betreffend ein Bundesgesetz, mit dem bestimmte rechtliche Aspekte des elektronischen Geschäfts- und Rechtsverkehrs geregelt werden (E-Commerce-Gesetz - ECG) http://www.parlinkom.gv.at/pd/pm/XXI/ME/his/002/ME00232_.html (21.12.2001)

[EC-RL] Verordnung 2000/31/EC des Europäischen Parlamentes und des Rates vom 8 Juni 2000 über bestimmte rechtliche Aspekte der Dienste der Informationsgesellschaft, insbesondere des elektronischen

Geschäftsverkehrs, im Binnenmarkt (' Richtlinie über den elektronischen Geschäftsverkehr'), Abl. 17.7.2000 L 178/1 http://europa.eu.int/eur-lex/de/lif/dat/2000/de_300L0031.html (21.12.2001)

[Edikte] Justizministerium: Ediktsdatei: http://www.edikte.justiz.gv.at/ (21.12.2001)

[Europay-Quick] Europay Austria: Quick – die elektronische Geldbörse. http://www.europay.at/quick/quick1.htm (19.4.2001)

[EVÜ] Europäisches Vertragsstatutübereinkommen. Übereinkommen über das auf vertragliche Schuldverhältnisse anzuwendende Recht BGBl III 1998/208

[Farhoodi/Fingar 1997] Faramarz Farhoodi, Peter Fingar: Competing for the Future with Intelligent Agents. http://home1.gte.net/pfingar/agents_doc_rev4.htm (21.12.2001)

[Fernabsatz-RL] Richtlinie 97/7/EG des Europäischen Parlamentes und des Rates vom 20. Mai 1997 über den Verbraucherschutz bei Vertragsabschlüssen im Fernabsatz. Abl L 144, 4.6.1997, 19ff

[Finin et al. 1993] Tim Finin, Jay Weber, Gio Wiederhold, Michael Genesereth, Richard Fritzson, Donald McKay, James McGuire, Richard Pelavin, Stuart Shapiro: Specification of the KQML Agent-Communication Language, http://www.cs.umbc.edu/kqml/papers/kqmlspec.pdf (21.12.2001)

[Finin et al. 1994] Tim Finin, Richard Fritzson, Don McKay, Robin McEntire: KQML as an Agent Communication Lnaguage. The Proceedings of the Third International Conference on Information and Knowledge Management (CIKM'94), ACM Press, November 1994.

[FIPA] Foundation for Intelligent Physical Agents. http://www.fipa.org (21.12.2001)

[FIPA-Comm-Act] FIPA Communicative Act Library Specification http://www.fipa.org/specs/fipa00037/ (21.12.2001)

[FIPA-Msg-Structure] FIPA ACL Message Structure Specification http://www.fipa.org/specs/fipa00061/ (21.12.2001)

[Foner 1993] Lenny Foner: What's an Agent, Anyway? A Sociological Case Study. http://foner.www.media.mit.edu/people/foner/Julia/Julia.html (21.12.2001)

[Franklin/Graesser 1996] Stan Franklin, Art Graesser: Is in an Agent, or just a program?: A Taxnomy for Autnonomous Agents. Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages. Springer 1996. http://www.msci.memphis.edu/~franklin/AgentProg.html (12.12.2001)

[Fricke et al. 2001] Stefan Fricke, Karsten Bsufka, Jan Keiser, Torge Schmidt, Ralf Sesseler, Sahin Albayrak: Agent-Based Telematic Services and Telecom Applications. IEEE Internet Computing March/April 2001 New York: IEEE Computer Society 2001. 43-48

[Fuchs 2000] Thomas Fuchs: In „geheimer" Mission. (Software-) Agenten im Einsatz. LOG IN 20 (2000) 43-46

[Furbach et al. 2000] Ulrich Furbach, Oliver Obst, Frider Stolzenburg: Intelligente Agenten und KI. LOG IN 20 (2000) 17-21

[Gehmeyr et al. 1998] A. Gehmeyr, J. Müller, A. Schappert: Mobile Information Agents on the Web. 262-277. In: Matthias Klusch, Gerhard Weiß (Ed.): Cooperative Information Agents II. Learning, Mobility and Electronic Commerce for Information Discovery on the Internet. Berlin: Springer 1998 (Lecture Notes in Artificial Intelligence 1435)

[Geizhals] Geizhals Preisvergleich: http://www.geizhals.at/ (21.12.2001)

[Genesereth/Ketchpel 1994] Michael R. Genesereth, Steven P. Ketchpel: Software Agents. Communications of the ACM July 1994. New York: acm Press 1994. 48-53 (Volume 37, Number 7)

[Glushko et al. 1999] Robert J. Glushko, Jay M. Tenenbaum, Bart Meltzer: An XML Framework for Agent-based E-Commerce. Communications of the ACM March 1999. New York: acm Press 1999. 106-114 (Volume 42, Number 3)

[Gomez] Gomez.com your guide to buying online: http://www.gomez.com/ (21.12.2001)

[Gong] Li Gong: Java 2 Platform Security Architecture. http://java.sun.com/j2se/1.3/docs/guide/security/spec/security-spec.doc.html (21.12.2001)

[Grasshopper] The Grasshopper agent development platform http://www.grasshopper.de/ (21.12.2001)

[Gray et al. 1998] Robert S. Gray, David Kotz, Georg Cybenko, Daniela Rus: D'Agents: Security in a Multiple-Language, Mobile-Agent System. 154-187 In: Giovanni Vigna (Ed.): Mobile Agents and Security. Berlin: Springer 1998. (Lecture Notes in Computer Science 1419)

[Green et al. 1998] Shaw Green, Padraig Cunningham, Fergal Somers: Agent Mediated Collaborative Web Page Filtering. In: Matthias Klusch, Gerhard Weiß (Ed.): Cooperative Information Agents II. Learning, Mobility and Electronic Commerce for Information Discovery on the Internet. Berlin: Springer 1998 (Lecture Notes in Artificial Intelligence 1435)

[Greenwald/Stone 2001] Amy Greenwald, Peter Stone: Autonomous Bidding Agents in the Trading Agent Competition. IEEE Internet Computing March/April 2001 New York: IEEE Computer Society 2001. 52-60

[Greif 1994] Irene Greif: Desktop Agents in Group-Enabled Products. Communications of the ACM July 1994. New York: acm Press 1994. 100-105 (Volume 37, Number 7)

[Guilfoyle 1998] C. Guilfoyle: Vendors of Intelligent Agent Technologies: A Market Overview. 91-101 In: Nicholas R. Jennings, Michael J. Wooldridge (Ed.): Agent Technology. Foundations, Applications, and Markets. Berlin: Springer 1998

[Guttman/Maes 1998] Robert H. Guttman, Pattie Maes: Cooperative vs. Competitive Multi-Agent Negotiations in Retail Electronic Commerce. 135-147 In: Matthias Klusch, Gerhard Weiß (Ed.): Cooperative Information Agents II. Learning, Mobility and Electronic Commerce for Information Discovery on the Internet. Berlin: Springer 1998 (Lecture Notes in Artificial Intelligence 1435)

[Guttman et al. 1999] Robert Guttman, Alexandros Moukas, Pattie Maes: Agents as Mediators in Electronic Commerce. 131-152 In: Matthias Klusch (Ed.): Intelligent Information Agents. Agent-Based Information Discovery and Management on the Internet. Berlin: Springer 1999

[Hadad/Kraus 1999] Merav Hadad, Sarit Kraus: SahredPlans in Electronic Commerce. 204-231 In: Matthias Klusch (Ed.): Intelligent Information Agents. Agent-Based Information Discovery and Management on the Internet. Berlin: Springer 1999

[Hammer 2000] V. Hammer: Signaturprüfungen nach SigI. Datenschutz und Datensicherheit 2/2000, 96

[Hattori et al. 1999] Fumio Hattori, Takeshi Ohguro, Makoto Yokoo, Shigeo Matsubara, Sen Yoshida: Socialware: Multiagent Systems for Supporting Network Communities. Communications of the ACM March 1999. New York: acm Press 1999. 55-61 (Volume 42, Number 3)

[Hayes-Roth et al. 1999] Barbara Hayes-Roth, Vaughan Johnson, Robert van Gent, Keith Wescourt: Staffing the Web with Interactive Characters. Communications of the ACM March 1999. New York: acm Press 1999. 103-105 (Volume 42, Number 3)

[Hayzelden et al. 1999] Alex Hayzelden, John Bigham, Michael Wooldridge, Laurie Cuthbert: Future Communication Networks Using Software Agents. 1-57. In: Alex L. G. Hayzelden, John Bigham (Eds.): Software Agents for Future Communication Systems. Berlin: Springer 1999

[Hive] MIT Media Lab http://www.hivecell.net/ (21.12.2001)

[Hohl 1998] Fritz Hohl: Time Limited Blackbox Security: Protecting Mobile Agents from Malicious Hosts. 92-113 In: Giovanni Vigna (Ed.): Mobile Agents and Security. Berlin: Springer 1998. (Lecture Notes in Computer Science 1419)

[Hohl 2000] Fritz Hohl: A Framework to Protect Mobile Agents by Using Reference States. In: Proceedings of the 20th International Conference on Distributed Computing Systems (ICDCS 2000), Los Alamitos: IEEE Computer Society 2000; http://mole.informatik.uni-stuttgart.de/papers/icdcs2000hohl.pdf (21.12.2001)

[Holzner 2001] Steven Holzner: Inside XML. Indianapolis: New Riders 2001

[HTML-Spec] W3C: Hyper Text Markup Language: http://www.w3.org/MarkUp/ (21.12.2001)

[Huemer 2001] Christian Huemer: <<DIR>>-XML^2 – Unambiguous Access to XML-based Business Documents in B2B E-Commerce. ACM Conference on Electronic Commerce (EC'01) Tampa (Florida, USA), October 2001

[Huhns 1999] Michael N. Huhns: Networking Embedded Agents. IEEE Internet Computing January/February 1999 New York: IEEE Computer Society 1999. 91-93

[Huhns/Singh 1998] Michael N. Huhns, Munindar P. Singh: Cognitive Agents. IEEE Internet Computing November/December 1998 New York: IEEE Computer Society 1998. 87-89

[Huhns/Mohamed 1999] Michael N. Huhns, Abdulla Mohamed: Benevolent Agents. IEEE Internet Computing March/April 1999 New York: IEEE Computer Society 1999. 96-98

[Humbert 2000] Ludger Humbert: Automatisierte Aktualisierung von Webseiten. LOG IN 20 (2000) 65-70

[Hsiang/Hsieh-Chang 1999] Jieh Hsiang, Tu Hsieh-Chang: Personalized Web Retrieval: Three Agents for Retrieving Web Information. 118-132 In: Toru Ishida (Ed.): Multiagent Platforms. Berlin: Springer 1999 (Lecture notes in artificial intelligence 1599)

[IAIK] IAIK TU Graz: jce-toolkit: http://jcewww.iaik.at/ (21.12.2001)

[Illinois EC Security Act] Illinois Electronic Commerce Security Act: http://www.legis.state.il.us/ilcs/ch5/ch5act175articles/ch5act175Sub2.htm (21.12.2000)

[I-LVA] Internet Seminar http://www.fim.uni-linz.ac.at/Research/Applaud/Internet-Seminar/index.htm (21.12.2001)

[Janca/Gilbert 1998] P. C. Janca, D. Gilbert: Practical Design of Intelligent Agent Systems. 73-89 In: Nicholas R. Jennings, Michael J. Wooldridge (Ed.): Agent Technology. Foundations, Applications, and Markets. Berlin: Springer 1998

[Java-Code] Tim Lindholm, Frank Yellin: The Java Virtual Machine Specification. Second Edition. Addison-Wesley 1999 http://java.sun.com/docs/books/vmspec/2nd-edition/html/VMSpecTOC.doc.html (21.12.2001) (Kapitel 4.8 und 4.9)

[JavaMailProviders] Third-Party Products for JavaMail: http://java.sun.com/products/javamail/Third_Party.html (21.12.2001)

[Jennings/Wooldridge 1996] Nicholas R. Jennings and Michael J. Wooldridge: Software Agents. IEE Review, January 1996, 17-20

[Jennings/Wooldridge 1998] Nicholas R. Jennings, Michael J. Wooldridge: Applications of Intelligent Agents. 3-28 In: Nicholas R. Jennings, Michael J. Wooldridge (Ed.): Agent Technology. Foundations, Applications, and Markets. Berlin: Springer 1998

[Jennings 1999] Nicholas R. Jennings: Agent-Oriented Software Engineering. Proc. 12th Int Conf on Industrial and Engineering Applications of AI, Cairo, Egypt, 4-10. http://www.ecs.soton.ac.uk/~nrj/download-files/cairo.pdf (21.12.2001)

[Jennings et al. 2001] N. R. Jennings, P. Faratin, A. R. Lomuscio, S. Parsons, C. Sierra and M. Wooldridge: Automated negotiation: Prospects, methods and challenges. International Journal of Group Decision and Negotiation 10 (2) 199-215 http://www.ecs.soton.ac.uk/~nrj/download-files/gdn01.ps (21.12.2001)

[Jennings 2001] Nicholas R. Jennings: An agent-based approach for building complex software systems. Communications of the ACM April 2001. New York: acm Press 2001. 35-41 (Volume 44, Number 4)

[Jini] Sun: http://www.sun.com/jini/ (21.12.2001)

[JNDI] Java Naming and Directory Interface http://java.sun.com/products/jndi/index.html (21.12.2001)

[Kaliski] Burton S. Kaliski Jr.: A Layman's Guide to a subset of ASN.1, BER, and DER. ftp://ftp.rsasecurity.com/pub/pkcs/ascii/layman.asc (21.12.2001)

[Kalnis 2001] Marios Kalnis: Intelligent Mobile Agents (Extended Version). Master Thesis an der University of Reading. 2001 http://www.fim.uni-linz.ac.at/research/agenten/WebInterface_Final_Report.pdf (21.12.2001)

[Kasbah] Kasbah: http://ecommerce.media.mit.edu/Kasbah/ (No longer available)

[Kearney 1998] P. Kearney: Personal Agents: A Walk on the Client Side! 125-136 In: Nicholas R. Jennings, Michael J. Wooldridge (Ed.): Agent Technology. Foundations, Applications, and Markets. Berlin: Springer 1998

[Kendall 2000] Elizabeth A. Kendall: Role Modeling for Agent System Analysis, Design and Implementation. IEEE Concurrency April-June 2000. New York: IEEE Computer Society 2000. 34-41 (Volume 8, Number 2)

[Klusch 1998] Matthias Klusch: Kooperative Informationsagenten im Internet. Hamburg: Verlag Dr. Kovac 1998

[Kotz/Gray 1999] David Kotz, Robert S. Gray: Mobile Agents and the Future of the Internet. Operating Systems Review Juli 1999. New York: acm Press 1999. 7-13 (Volume 33, Number 3)

[KQML] UMBC KQML Web http://www.cs.umbc.edu/kqml/ (21.12.2001)

[KSchG] Konsumentenschutzgesetz: Bundesgesetz vom 8. März 1979, mit dem Bestimmungen zum Schutz der Verbraucher getroffen werden (Konsumentenschutzgesetz - KSchG), BGBl 1979/140 idF BGBl I 1999/185

[Labrou/Finin 1997a] Yannis Labrou, Tim Finin: A Proposal for a new KQML Specification. TR CS-97-03 http://www.cs.umbc.edu/kqml/papers/kqml97.pdf (21.12.2001)

[Labrou/Finin 1997b] Yannis Labrou, Tim Finin: Comments on the Specification for FIPA ´97 Agent Communication Language. http://www.cs.umbc.edu/kqml/papers/fipa/comments.shtml (21.12.2001)

[Lange/Oshima 1999] Danny B. Lange, Mitsuru Oshima: Seven Good Reasons for Mobile Agents. Communications of the ACM March 1999. New York: acm Press 1999. 88f (Volume 42, Number 3)

[Letizia] Henry Liebermann: Letizia: An Agent that Assists Web Browsing. http://lieber.www.media.mit.edu/people/lieber/Lieberary/Letizia/Letizia.html (21.12.2001)

[Liebermann 1999] Henry Liebermann: Personal Assistants for the Web: An MIT Perspective. 279-292 In: Matthias Klusch (Ed.): Intelligent Information Agents. Agent-Based Information Discovery and Management on the Internet. Berlin: Springer 1999

[Lomuscio et al. 2000] Alessio R. Lomuscio, Michael Wooldridge, Nicholas R. Jennings: A Classification Scheme for Negotiation in Electronic Commerce. LNAI 1991: "Agent Mediated Electronic Commerce; A European Perspective". Springer: Berlin 2000, 19-34

[Ma 1999] Moses Ma: Agents in E-Commerce. Communications of the ACM March 1999. New York: acm Press 1999. 78-80 (Volume 42, Number 3)

[Maes 1994] Pattie Maes: Agents that Reduce Work and Information Overload. Communications of the ACM July 1994. New York: acm Press 1994. 31-40 (Volume 37, Number 7)

[Maes et al. 1999] Pattie Maes, Robert H. Guttmann, Alexandros G. Moukas: Agents That Buy and Sell. Communications of the ACM March 1999. New York: acm Press 1999. 81-91 (Volume 42, Number 3)

[MASIF-Spec] Mobile Agent System Interoperability Facilities Specification ftp://ftp.omg.org/pub/docs/orbos/1997/97-10-05.pdf (15.11.2000)

[Mastercard-AGB] Allgemeine Geschäftsbedingungen für die Mastercard: http://www.europay.at/htdocs/allesueber/agb_set.htm (13.3.2001)

[Mastercard-SET] Mastercard SET-Info http://www.europay.at/htdocs/masterpage/masterpage_sicherzahlen.htm (13.3.2001)

[Milojicic 2000] Dejan Milojicic: Agent Systems and Applications. IEEE Concurrency April-June 2000. New York: IEEE Computer Society 2000. 22f (Volume 8, Number 2)

[Minar et al. 2000] Nelson Minar, Matthew Gray, Oliver Roup, Raffi Krikorian, Pattie Maes: Hive: Distributed Agents for Networking Things. IEEE Concurrency April-June 2000. New York: IEEE Computer Society 2000. 24-33 (Volume 8, Number 2)

[Mitchell et al. 1994] Tom Mitchell, Rich Caruana, Dayne Freitag, John McDermott David Zabowski: Experience with a Learning Personal Assistant. Communications of the ACM July 1994. New York: acm Press 1994. 81-91 (Volume 37, Number 7)

[Mole] The Home of the Mole: http://mole.informatik.uni-stuttgart.de/ (24.10.2001)

[Mühlbacher/Sonntag 1999] Jörg R. Mühlbacher, Michael Sonntag: Teaching Software Engineering and Encouraging Entrepreneurship through E-Commerce. In: Proceedings 2nd International Conference on Innovation Through E-Commerce - IeC99. Manchester 135ff

[Mühlbacher et al. 2001] Jörg R. Mühlbacher, Susanne Reisinger, Michael Sonntag: Intelligent Agents and XML - A method for accessing webportals in both B2C and B2B E-Commerce. WOOBS'01 - Workshop on Object-Oriented Business Solutions; To appear

[Müller 1996] Jörg P. Müller: The Design of Intelligent Agents. A Layered Approach. Berlin: Springer 1996 (Lecture notes in artificial intelligence 1177)

[Necula/Lee 1998] Georg C. Necula, Peter Lee: Safe, Untrusted Agents Using Proof-Carrying Code. 61-91 In: Giovanni Vigna (Ed.): Mobile Agents and Security. Berlin: Springer 1998. (Lecture Notes in Computer Science 1419)

[Nick/Themis 2001] Zacharias Z. Nick, Panayiotopoulos Themis: Web Search Using a Genetic Algorithm. IEEE Internet Computing March/April 2001 New York: IEEE Computer Society 2001. 18-26

[Noriega/Sierra 1998] Pablo Noriega, Cales Sierra: Auctions and Multi-Agent Systems. 153-175 In: Matthias Klusch (Ed.): Intelligent Information Agents. Agent-Based Information Discovery and Management on the Internet. Berlin: Springer 1999

[Norman 1994] Donald A. Norman: How Might People Interact with Agents. Communications of the ACM July 1994. New York: acm Press 1994. 68-71 (Volume 37, Number 7)

[Nwana/Ndumu 1998] H. S. Nwana, D. T. Ndumu: A Brief Introduction to Software Agent Technology. 29-47 In: Nicholas R. Jennings, Michael J. Wooldridge (Ed.): Agent Technology. Foundations, Applications, and Markets. Berlin: Springer 1998

[Nwana/Wooldridge1996] Hyacinth Nwana and Michael Wooldridge: Software Agent Technologies, BT Technology Journal 14(4), 1996, 68-78. http://www.labs.bt.com/projects/agents/publish/papers/sat_report.htm (24.10.2001)

[Office-Agents] Microsoft: Microsoft Agent http://msdn.microsoft.com/msagent (2.8.2000)

[O'Meara/Patel 2001] Tadhg O'Meare, Ahmed Patel: A Topic-Specific Web Robot Model Based on Restless Bandits. IEEE Internet Computing March/April 2001 New York: IEEE Computer Society 2001. 27-35

[OMG/MASIF] Object Management Group http://www.omg.org / Mobile Agent System Interoperability Facilities Specification http://cgi.omg.org/cgi-bin/doc?orbos/97-10-05 (6.11.2000)

[ONTOLOGY.ORG] Ontology.org: http://www.ontology.org/ (21.12.2001)

[Oshima et al. 1998] Mitsuru Oshima, Guenther Karjoth, Kouichi Ono: Aglets Specification 1.1 Draft. http://www.trl.ibm.com/aglets/spec11.html (27.6.2001)

[Parunak 1998] H. Van Dyke Parunak: What Can Agents Do in Industry, and Why? An Overview of Industrially-Oriented R&D at CEC. 1-18 In: Matthias Klusch, Gerhard Weiß (Ed.): Cooperative Information Agents II. Learning, Mobility and Electronic Commerce for Information Discovery on the Internet. Berlin: Springer 1998 (Lecture Notes in Artificial Intelligence 1435)

[Pazandak 1998] Paul Pazandak: Agent System Comparison. http://www.objs.com/agility/tech-reports/9810-agent-comparison.html (23.11.2000)

[PDAComparison] PDA Comparison.com: http://www.pdacomparison.com/ (11.4.2001)

[PGP] PGP Security: http://www.pgp.com (19.4.2001)

[Primanestor] Primanestor!: http://www.primanestor.de/ (11.4.2001)

[RFC 2109] D. Kristoll, L. Montulli: HTTP State Manegment Mechanism: http://www.ietf.org/rfc/rfc2109.txt (11.5.2001) (= RFC 2109)

[Ricardo] Ricardo: Bietagent http://www.ricardo.de/sogehts/content_pr_bietagent.phtml (2.8.2000)

[Riecken 1994] Doug Riecken: Intelligent Agents. Introduction to the Special Issue on Intelligent Agents. Communications of the ACM July 1994. New York: acm Press 1994. 18-21 (Volume 37, Number 7)

[Rott 1998] E-Cash: Bestandsaufnahme. http://stud1.tuwien.ac.at/~e8525020/preecash.html (13.3.2001)

[Salton 1983] Gerard Salton, Michael J. McGill: Introduction to modern information retrieval. New York: McGraw 1983

[Sanders/Tschudin 1998] Tomas Sanders, Christian F. Tschudin: Protecting Mobile Agents Against Malicious Hosts. 44-60 In: Giovanni Vigna (Ed.): Mobile Agents and Security. Berlin: Springer 1998. (Lecture Notes in Computer Science 1419)

[Sandholm 1993] Tuomas Sandholm: An Implementation of the Contract Net Protocol Based on Marginal Cost Calculations. 252-262 Eleventh National Conference on Artificial Intelligence (AAAI-93), Washington

[Schoonderwoerd/Holland 1999] Ruud Schoonderwoerd, Owen Holland: Minimal Agents for Communications Network Routing: The Social Insect Paradigm. 305-325. In: Alex L. G. Hayzelden, John Bigham (Eds.): Software Agents for Future Communication Systems. Berlin: Springer 1999

[Schwehm 1998] Markus Schwehm: Mobile Softwareagenten. In: OBJEKTSpektrum 11-12/1998. Bergisch Gladbach: SIGS Conferences 1998. 19-23

[Seitz 2000] Jochen Seitz: Intelligente Agenten in der Telematik. LOG IN 20 (2000) 22-26

[SET] SET-Info http://www.goset.org/goset/Setinfo/setinfo_de.htm (13.1.2001)

[SETCO] STECO: SET Software Compliance Testing http://www.setco.org/testing.html (30.3.2001)

[SigG] Bundesgesetz über elektronische Signaturen (Signaturgesetz – SigG) BGBl 190/1999

[SigRL] Richtlinie 1999/93/EG des Europäischen Parlamentes und des Rats vom 13. Dezember 1999 über gemeinschaftliche Rahmenbedingungen für elektronische Signaturen. Abl. 19.1.2000 L 13/12

[SigVO] Signatur-Verordnung: Verordnung des Bundeskanzlers über elektronische Signaturen (Signaturverordnung – SigV) vom 2.2.2000. BGBl II 30/2000

[Singh 1998] Munindar P. Singh: Agent Communication Languages: Rethinking the Principles. Computer. Innovative Technology for computer professionals. New York: IEEE Computer Society 1998 (Volume 31, Number 12)

[Singh/Huhns 1999] Munindar P. Singh, Michael N. Huhns: Social Abstractions for Information Agents. 37-52 In: Matthias Klusch (Ed.): Intelligent Information Agents. Agent-Based Information Discovery and Management on the Internet. Berlin: Springer 1999

[Sloman/Logan 1999] Aaron Sloman, Brian Logan: Building Cognitively Rich Agents. Using the SIM_Agent Toolkit. Communications of the ACM March 1999. New York: acm Press 1999. 71-77 (Volume 42, Number 3)

[Smith/Poulter 1999] Howard Smith, Kevil Poulter: Sahe the Onotology in XML-based Trading Architectures. Communications of the ACM March 1999. New York: acm Press 1999. 110-111 (Volume 42, Number 3)

[Sonntag 1998] Michael Sonntag: Intelligent Agents - Inevitable Tools for Teleworkers. In: Online Collaboration Berlin. Second International Conference on Teleworking, Knowledge Management and Electronic Commerce. Bonn: I.W.H. 1998, 32-34

[Sonntag 2000] Michael Sonntag: Electronic Signatures for Legal Persons. In: Susanne Hofer, Manfred Beneder (Ed.): IDIMT'00. 8th Interdisciplinary Information Management Talks. Linz: Universitätsverlag Rudolf Trauner 2000, 233-256

[Sonntag/Hörmanseder 2000] Michael Sonntag, Rudolf Hörmanseder: Mobile Agent Security Based on Payment. In: ACM SIG Operating Systems, Operating Systems Review, New York: October 2000, 48-55 (Vol. 34, No. 4)

[Sonntag 2001] Michael Sonntag: Improving Communication to Citizens and within Public Administration by Attribute Certificates. In: Maria A. Wimmer (Ed.): Knowledge Management in e-Government. KMGov-2001. 2nd International Workshop jointly organised by IFIP WG 8.3 & 8.5, University of Linz and University of Siena. Linz: Universitätsverlag Rudolf Trauner 2001, 207-217

[Sonntag 2001a] Michael Sonntag, Susanne Reisinger: Important Factors for E-Commerce. In: Cristian Hofer, Gerhard Chroust (Ed.): IDIMT-2001. 9th Interdisciplinary Information Management Talks. Linz: Universitätsverlag Rudolf Trauner 2001, 301-312

[Sonntag 2002a] Michael Sonntag: Legal Apsects of Mobile Agents. With special consideration of the proposed Austrian E-Commerce Law. Accepted for publication in Session D (Systemic Aspects of Component-Based System Development) at 16th European Meeting on Cybernetics and Systems Research (EMCSR 2002). Wien: 2002

[Sonntag 2002b] Michael Sonntag: Elektronische Signaturen. Rechtswirkungen, Haftung von Zertifizierungsdiensteanbietern, sowie Sonderprobleme. In: Oliver Plöckinger, Dieter Duursma, Günther Helm

(Hrsg.): Aktuelle Entwicklungen im Internetrecht. Beiträge zur zivil-, straf- und verwaltungsrechtlichen Diskussion. Wien: nwV 2002, 143-163 *Erscheint Anfang 2002*

[Spalink et al. 2000] Tammo Spalink, John H. Hartman, Garth A. Gibson: A Mobile Agent's Effect on File Service. IEEE Concurrency April-June 2000. New York: IEEE Computer Society 2000. 62-69 (Volume 8, Number 2)

[Straßer et al. 1997] Markus Straßer, Joachim Baumann, Fritz Hohl: Mole. A Java Based Mobile Agent System. In: M. Mühlhäuser: (ed.), Special Issues in Object Oriented Programming. dpunkt Verlag 1997, 301-308

[Sturm 2000] Jake Sturm: Developing XML Solutions. Redmond: Microsoft Press 2000

[Swarm] Swarm Development Group http://www.swarm.org/ (23.11.2000)

[Sycara et al. 1998] K. P. Sycara, D. Zeng, K. Decker: Intelligent Agents in Portfolio Management. 267-281 In: Nicholas R. Jennings, Michael J. Wooldridge (Ed.): Agent Technology. Foundations, Applications, and Markets. Berlin: Springer 1998

[Tanenbaum 1995] Andrew S. Tanenbaum: Verteilte Betriebssysteme. München: Prentice Hall 1995

[Theilmann/Rothermel 2000] Wolfgang Theilmann, Kurt Rothermel: Optimizing the Dissemination of Mobile Agents for Distributed Information Filtering. IEEE Concurrency April-June 2000. New York: IEEE Computer Society 2000. 53-61 (Volume 8, Number 2)

[Thirunavukkarasu et al. 1995] Chelliah Thirunavukkarasu, Tim Finin, James Mayfield: Secret Agents -- A Security Architecture for the KQML Agent Communication Language. October 1995. CIKM'95 Intelligent Information Agents Workshop, Baltimore, December 1995 http://www.cs.umbc.edu//kqml/papers/secret.ps (16.11.2000)

[Tolksdorf 1999] Robert Tolksdorf: On Coordinating Information Agents and Mobility. 396-411 In: Matthias Klusch (Ed.): Intelligent Information Agents. Agent-Based Information Discovery and Management on the Internet. Berlin: Springer 1999

[Tsakiridou 2001] Evdoxia Tsakiridou: Unsichtbare Helfer in der Welt der vernetzten Computer. In: Siemens AG: Pictures of the Future Oktober 2001

[TW-LVA] Teleworking – LVA über das Internet: http://www.fim.uni-linz.ac.at/lva/ws98/Teleworking/index.htm (12.4.2001)

[US Signature Act] Electronic Signatures in Global and National Commerce Act, June 8, 2000. In: DuD. Datenschutz und Datensicherheit 24/2000

[UUID] UUID (Universal Unique IDentifier): http://www.opengroup.org/onlinepubs/9629399/apdxa.htm (27.10.2000)

[Vally/Courdier 1999] Jean-Dany Vally, Rémy Courdier: A Conceptual „Role-Centered" Model for Design of Multi-Agents System. 33-46 In: Toru Ishida (Ed.): Multiagent Platforms. Berlin: Springer 1999 (Lecture notes in artificial intelligence 1599)

[Vigna 1998] Giovanni Vigna: Cryptographic Traces for Mobile Agents. 137-153 In: Giovanni Vigna (Ed.): Mobile Agents and Security. Berlin: Springer 1998. (Lecture Notes in Computer Science 1419)

[Vogler et al. 1998] Hartmut Vogler, Marie-Luise Moschgath, Thomas Kunkelmann: Enhancing Mobile Agents with Electronic Commerce Capabilities. 148-159 In: Matthias Klusch, Gerhard Weiß (Ed.): Cooperative Information Agents II. Learning, Mobility and Electronic Commerce for Information Discovery on the Internet. Berlin: Springer 1998 (Lecture Notes in Artificial Intelligence 1435)

[Voyager] ObjectSpace: http://www.objectspace.com/products/voyager/ (24.10.2001)

[Web-POND] Web Interface for POND http://www.fim.uni-linz.ac.at/research/agenten/WebInterface.htm (20.12.2001)

[WebWasher] WebWasher: http://www.webwasher.com/ (9.1.2001)

[Weigend 2000] Michael Weigend: Intelligente Agenten. Einige praktische Unterrichtsideen. LOG IN 20 (2000) 47-53

[Weihmayer/Velthuijsen 1998] R. Weihmayer, H. Velthuijsen: Intelligent Agents in Telecommunications. 203-215 In: Nicholas R. Jennings, Michael J. Wooldridge (Ed.): Agent Technology. Foundations, Applications, and Markets. Berlin: Springer 1998

[Wenger/Probst 1998] D. Wenger, A. R. Probst: Adding Value with Intelligent Agents in Financial Services. 303-325 In: Nicholas R. Jennings, Michael J. Wooldridge (Ed.): Agent Technology. Foundations, Applications, and Markets. Berlin: Springer 1998

[Wong et al. 1999] David Wong, Noemi Paciorek, Dana Moore: Java-based Mobile Agents. Communications of the ACM March 1999. New York: acm Press 1999. 92-102 (Volume 42, Number 3)

[Wooldridge/Ciancarini 2001] Michael Wooldridge, Paolo Ciancarini: Agent-Oriented Software Engineering: The State of the Art. In: P. Ciancarini and M. Wooldridge (Eds.): Agent-Oriented Software Engineering. Berlin: Springer 2001. LNAI Vol. 1957

[Wooldridge/Jennings 1998] Michael Wooldridge, Nicholas R. Jennings: Pitfalls of Agent-Oriented Development. In K. P. Sycara and M. Wooldridge, editors*:* Agents '98: Proceedings of the Second International Conference on Autonomous Agents ACM Press, May 1998. http://www.csc.liv.ac.uk/~mjw/pubs/agents98.ps.gz (17.11.2000)

[Wooldridge/Jennings 1999] Michael J. Wooldridge, Nicholas R. Jennings: Software Engineering with Agents: Pitfalls and Pratfalls. IEEE Internet Computing 3 (3) 1999 New York: IEEE Computer Society 1999. 20-27

[X.509] ITU-T X.509: Information Technology - Open Systems Interconnection – The Directory: Authentication framework, 1997 (Clause 13: Obtaining Certified Attributes)

[X.680] ITU-T X.680: Abstract Syntax Notation One (ASN.1) - Specification of Basic Notation, 1994

[XML-Spec] W3C: XML: eXtensible Markup Language: http://www.w3.org/XML/ (21.12.2001)

[Yahoo-F] The New York Times; Lisa Guernsey: Welcome to the World Wide Web. Passport, Please? http://www.nytimes.com/2001/03/15/technology/15BORD.html (9.4.2001)

[Yemini et al. 1998] Y. Yemini, A. Dailianas, D. Florissi, G. Huberman: MarketNet: Market-Based Protection of Information Systems. In: Proceedings of the First International Conference on Information and Computation Economies. ACM Press 1998 (Also http://www.cs.columbia.edu/dcc/marketnet/Publications/ice98.ps; 28.6.2001)

[Zambonelli et al. 2000] Franco Zambonelli, Nicholas R. Jennings, Michael Wooldridge: Organisational Abstractions for the Analysis and Design of Multi-Agent Systems. Proc. 1st Int. Workshop on Agent-Oriented Software Engineering, Limerick, Ireland, 127-141. http://www.ecs.soton.ac.uk/~nrj/download-files/aose00.ps (21.12.2001)

[Zankl 2001] W. Zankl: Rechtsqualität und Zugang von Erklärungen im Internet. ecolex 2001, 344

[Zhou] Jianying Zhou: Achieving Fair Non-Repudiation in Electronic Transactions. Journal of Organizational Computing and Electronic Commerce, Lawrence Erlbaum Associates, 2001. To appear

Lebenslauf

Schulbildung:

1979 – 1983 4 Klassen Übungsvolksschule der PÄDAK der Diözese Linz

1983 – 1987 4 Klassen Übungshauptschule der PÄDAK der Diözese Linz

1987 – 1992 5 Klassen HTBLA Leonding Elektronik/Nachrichtentechnik (Abschluß mit ausgezeichnetem Erfolg)

Präsenzdienst:

1.7.1992 - 28.2.1993 (8 Monate, entlassen als Gefreiter)

Studium:

25.2.1993 - 15.4.1997, 9 Semester Informatik (Verkürzung der Mindeststudiendauer um ein Semester genehmigt, Abschluß mit Auszeichnung, Titel der Diplomarbeit: Telearbeit – Eine Untersuchung von Rahmenbedingungen unter besonderer Berücksichtigung der Telekommunikationsanbindung)

Seit 25.2.1993 Rechtswissenschaften als Zweitstudium (Abschluß des ersten Studienabschnitts: 10.6.1997; Studienabschluß Mitte Februar 2002)

Berufspraxis:

11.7.1988 - 10.8.1988 Ferialpraxis in der Lehrwerkstätte der ESG

9.7.1990 - 3.8.1990 Ferialpraxis im technischen Büro von Sprecher Energie

1.8.1991 - 31.8.1991 Ferialpraxis beim technischen Dienst von IBM Österreich

Juli 1996 bis November 1996: Aufbau des Netzwerkes und Erstellung der Kunden-, Artikel- und Bestellungs-Datenbank der Firma MSV-Versand (Versandhandel)

Seit Dezember 1996: EDV-Leiter von MSV-Versand (nunmehr MSV Handels- & Dienstleistungs GmbH) und zugehörigen Firmen

1.10.1997 - 30.9.1999 Vertragsassistent am Forschungsinstitut für Mikroprozessortechnik (FIM) (halbtags beschäftigt)

Associate coordinator des FIM im APPLAUD Projekt (EU SOCRATES Programm / ODL Projekt: 25097-CP-1-96-1-FI-ODL)

Seit 1.10.1999 Wissenschaftlicher Mitarbeiter am Institut für Informationsverarbeitung und Mikroprozessortechnik (FIM) (halbtags) für das Projekt über intelligente mobile Agenten (gefördert vom Jubiläumsfonds der Österreichischen Nationalbank, Projekt Nr. 7742; und dem Land Oberösterreich, Förderungs-Nr. Wi/Ge 201.515/1-2000/Wwin)

Session Organizer der Session A (E-commerce and E-Government - Experiences and Projects) der IDIMT-2002: 10th Interdisciplinary Information Management Talks

Auszeichnungen:

SS 1994 und WS 1994/95 Leistungsstipendium der Universität Linz

SS 1995 und WS 1995/96 Leistungsstipendium der Universität Linz

9.12.1997 Anerkennungspreis des Ministers für Wissenschaft, Verkehr und Kunst für herausragende Studienleistungen

Lehraufträge:

SS 1994 - SS 1997 Acht Beauftragungen als Tutor oder Instruktor (Algorithmen, Programmierpraktika, Computergrafik)

Vorlesung und Übung Algorithmen und Datenstrukturen 2, Praktikum 2 für Informatiktrainer (Internet Seminar), CBT (Computer Based Training) Seminar, Seminar Teleworking: LVA über das Internet, Programmierpraktikum 2 (Java), Rechtliche und technische Aspekte von E-Commerce, Anwendungen in Computernetzen

Eidesstattliche Erklärung

Ich erkläre an Eides statt, daß ich die vorliegende Dissertation selbständig und ohne fremde Hilfe verfaßt, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt, bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Linz, Jänner 2002 _____