Universität Leipzig
Fakultät für Mathematik und Informatik
Institut für Informatik

Agentensysteme –
aktueller Entwicklungsstand und Konzeption eines universellen News-Watcher-Agent

Diplomarbeit

vorgelegt von
Timo Böhme

Leipzig, März 1999

Inhalt

Abbildungsverzeichnis

1 Einleitung

Agent *[lat.-italien.], 1) allgemein: jeder im Auftrag oder Interesse eines anderen Tätige.* [Mey94]

Agent, Softwareagent, Intelligente Agenten – das sind Begriffe, die in den letzten Jahren für eine neue Kategorie von Software Verwendung finden[1]. Waren es am Anfang vor allem Forschergruppen im universitären Umfeld, die sich um neue Technologien zur Überwindung von abzeichnenden Engpässen in der Informationsverarbeitung bemühten, werden diese Ansätze jetzt auch intensiv von den Entwicklungsabteilungen der Industrie weiterentwickelt.

Der Grund für diese Entwicklung liegt sowohl in der steigenden Komplexität von Anwendungen wie auch in der dramatisch anwachsenden Datenflut, die von Programmen, aber auch von den Benutzern, bewältigt werden muß. Ein entscheidender Faktor für diesen Prozeß ist der weltweite Datenzugriff, der durch das Internet ermöglicht wird. Der Konflikt zwischen der Menge angebotener Information und den Möglichkeiten, diese zu verarbeiten, spitzt sich weiter zu, da derzeitige Soft- und Hardwarelösungen zur Informationsaufbereitung nicht mit der Expansion des Internet mithalten können.

Diese Probleme werfen eine Reihe von Fragen auf, die von den Forschern und Entwicklern beantwortet werden müssen:

- Wie kann der notwendige Datenaustausch in den Netzen reduziert werden?
- Wie können freie Ressourcen (z.B. CPU-Zeit, Speicher, Netzkapazität) effektiv erkannt und zur Überwindung von Engpässen genutzt werden?
- Wodurch kann eine last- und ressourcenabhängige optimale Verteilung der Datenverarbeitung realisiert werden?
- Wodurch lassen sich Skalierbarkeitsprobleme bei steigenden Benutzerzahlen oder Datenmengen umgehen?
- Welche Arbeitsabläufe des Benutzers lassen sich automatisieren oder vereinfachen?

[1] Obwohl der Begriff des Intelligenten Agenten innerhalb der künstlichen Intelligenz schon seit fast 40 Jahren Verwendung findet [Bra97], erfährt er erst seit Anfang der 90er Jahre seine große Popularität und wird weltweit zum wichtigen Forschungsschwerpunkt (z.B. Software Agents Group am MIT: http://agents.www.media.mit.edu/groups/agents, Intelligent Software Agents Abteilung der Carnegie Mellon University: http://www.cs.cmu.edu/~softagents, Intelligent Software Agents Forschung an der University of Aberdeen: http://www.csd.abdn.ac.uk/research/intelligent_agents.html)

- Wie läßt sich das Know How anderer Programme nutzen und wie können in diesem Fall die Aufgaben auf die kompetentesten Komponenten verteilt werden?
- Wie läßt sich eine aktive Hilfe für den Benutzer realisieren?
- Welche Nutzungspotentiale bieten die verfügbaren Informationsquellen und welcher Techniken bedarf es, diese zu erschließen?

Die ersten vier Fragen beziehen sich auf die effiziente Nutzung der zur Verfügung stehenden Hardwareressourcen durch entsprechend ‚intelligent' gestaltete Software. Die restlichen Fragen suchen nach Lösungen, dem Benutzer die Suche nach relevanten Informationen abzunehmen, ihn sozusagen vor dem ‚Ersticken im Informationsmüll' zu bewahren und den Mehrwert, den der Informationsreichtum bietet, auch nutzen zu können.

Die Antwort, die heute auf diese Fragen immer öfter gegeben wird – zumindest im Hinblick auf zukünftige Realisierungen – lautet: der Einsatz von Agenten.

Over its three decade history, software engineering has developed an increasingly powerful array of tools with which to tackle the complexity of software systems, of which the most recent additions are the notions of an intelligent agent and multi-agent system. [Woo98, S. 31]

Im ersten Teil meiner Arbeit werde ich deshalb in Kapitel 2 den bisher ohne genaue terminologische Erklärung gebrauchten Begriff des Agenten näher spezifizieren. Dabei sollen die einer exakten Spezifikation entgegenstehenden Schwierigkeiten aufgezeigt und durch eine Reihe von bisher aufgestellten Definitionen unterlegt werden. In der Zusammenfassung sollen die im Sinne der Arbeit wesentlichen Merkmale eines Agenten herausgestellt werden. Anschließend werden verschiedene Varianten der Klassifikation des breiten Spektrums von Agenten vorgestellt. Im nächsten Kapitel wird der aktuelle Stand von Forschung und Entwicklung anhand konkreter Projekte und Produkte vorgestellt.

Der zweite Teil der Arbeit ist der Entwicklung eines spezifischen Agentensystems gewidmet. Im Kapitel 4 werden dazu die erforderlichen Technologien vorgestellt und die Entwicklungsphasen bis zum Design ausgearbeitet. Das darauffolgende Kapitel ist der praktischen Realisation in dem Projekt Handiks gewidmet. Es beschreibt die notwendigen Anpassungen in der Architektur und dokumentiert die Implementierung.

Die Diplomarbeit entstand im Rahmen eines Praktikums am Fraunhofer Institut für Arbeitswirtschaft und Organisation (IAO) in Stuttgart.

2 Softwareagenten – Theoretische Betrachtungen

2.1 Terminologische Abgrenzung

2.1.1 Was sind Agenten? – Schwierigkeiten einer allgemeingültigen Definition

Für die nachfolgenden Ausführungen ist es sinnvoll, den Begriff des Softwareagenten in einer ersten Arbeitsdefinition abzugrenzen: Ein Softwareagent ist ein Programm (oder Programmodul), welches im Auftrag eines Benutzers bestimmte Aufgaben selbständig ausführen kann. Diese erste Definition soll nur eine ungefähre Vorstellung des Themenkreises vermitteln und muß später präzisiert werden.

Die Anzahl der Publikationen über Softwareagenten steigt derzeit nahezu exponentiell an. Waren es Anfang 1998 noch 10.350 Links, die von der Internet-Suchmaschine AltaVista auf die Anfrage „software agents“ gemeldet wurden [Kru98/1], so liefert dieselbe Anfrage im Juni 98 schon 15.914 Treffer. Selbst wenn es sich dabei um teilweise doppelte oder veraltete Links handelt, die Tendenz ist klar ersichtlich.

Obwohl der Begriff des Softwareagenten also in einer steigenden Zahl von Veröffentlichungen verwendet wird, gibt es bis heute keine allgemein akzeptierte Definition desselben. Als Konsequenz daraus ergibt sich die uneingeschränkte Verwendungsmöglichkeit des Begriffs, wodurch er Gefahr läuft, zu einer merkmalslosen Worthülse degradiert zu werden – ähnlich wie man es heute am Begriff Multimedia sehen kann. Neben Softwareagent findet man häufig auch den Begriff des intelligenten Agenten bzw. des intelligenten Softwareagenten. Diese sind nach meiner Ansicht bezüglich einer Definition noch schwerer zu fassen, da der Terminus Intelligenz weit entfernt von einer allgemein anerkannten Begriffsbestimmung ist. Weitere Kritikpunkte sind die undifferenzierte Verwendung von ‚Agent‘ und ‚intelligenter Agent‘ in den verschiedenen Veröffentlichungen sowie die unterbewußt suggerierte Vermenschlichung durch den Zusatz ‚intelligent‘ [Lan95]. Im weiteren werde ich die verschiedenen Begriffe unter Softwareagent zusammenfassen und nur bei Zitaten oder Verweisen auf andere Autoren deren Terminologie verwenden.

Woraus resultieren die Probleme einer eindeutigen Definition? Aus dem Wesen der Softwareagenten ergibt sich ihr interdisziplinärer Charakter. Abhängig von welchem Forschungsschwerpunkt oder praktischer Anforderung der Agent betrachtet wird, ergeben sich die verschiedenen Ansätze für Definitionen. Nach [Bre98, S. 39 ff.] haben folgende Forschungsrichtungen wesentlichen Einfluß auf die Entwicklung eines Softwareagenten:

- Künstliche Intelligenz (KI)
- verteilte KI
- Netzwerk- und Kommunikationssysteme (verteilte Systeme)
- Entscheidungstheorie
- Psychologie

In den nächsten beiden Abschnitten werde ich anhand verschiedener Definitionen die unterschiedlichen Blickwinkel auf Softwareagenten vorstellen und kommentieren. Dabei habe ich bewußt eine Unterteilung in Forschung und Industrie gewählt, um die Differenzen von wissenschaftlichem Anspruch an Agenten und aktuellem Einsatz zu illustrieren.

2.1.2 Definitionen aus Sicht der Forschung

Obwohl Agenten im Rahmen der KI schon lange Zeit einen Forschungsschwerpunkt darstellen [Nwa96], habe ich mich bei der Auswahl beispielhafter Definitionen auf Veröffentlichungen der letzten vier Jahre beschränkt, um das sich gerade jetzt ausbreitende Anwendungsfeld aufzuzeigen. Aufgrund des mannigfaltigen Materials, welches von universitären Forschungseinrichtungen zu diesem Thema vorhanden ist, stellen die nachfolgenden Definitionen keine repräsentative Auswahl dar, sie sollen vielmehr einen Eindruck der verschiedenen Herangehensweisen an das Thema Softwareagent vermitteln.

An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors. [Rus95, S. 33]

Dieses Zitat stammt aus dem Buch „Artificial Intelligence: a Modern Approach“ von Russel und Norvig, welches derzeit häufig als Standardlehrbuch zur KI verwendet wird und das Thema intelligente Agenten als Leitfaden durch die Themengebiete verwendet. Die zentralen Punkte der äußerst allgemein gehaltenen Definition sind beobachten und agieren innerhalb einer Umwelt. Entscheidend ist dabei die genaue Spezifizierung von Umwelt und die Bedeutung von Beobachtung und Handlung. Für den Fall, daß Umwelt definiert wird als etwas, das Eingaben produziert und Ausgaben entgegennimmt und Beobachtung die Aufnahme, Handlung die Ausgabe von Daten ist, dann kann jedes Programm als Agent angesehen werden. Es ist deshalb notwendig, Einschränkungen bzgl. dieser drei Komponenten vorzunehmen, um eine Abgrenzung von Softwareagenten zu anderen Programmen zu erhalten. Die folgenden Definitionen führen dazu weitere Präzisierungen auf.

Intelligent agents continuously perform three functions: perception of dynamic conditions in the environment; action to affect conditions in the environment; and reasoning to interpret perceptions, solve problems, draw inferences, and determine actions. [Hay95]

Hayes-Roth vom Stanford Knowledge Systems Laboratory fügt der vorangegangenen Definition eine Funktion hinzu, welche die Aufgaben Beobachtung und Handlung miteinander in Beziehung setzt und zwar derart, daß die Handlungen durch Ableitungen bzw. Schlußfolgerungen über den Beobachtungsdaten ausgewählt werden. Diese Beschreibung entspricht im wesentlichen der eines Systems aus dem Bereich der KI. Das hier formulierte Entscheidungskriterium für Agenten, das Vorhandensein eines Schlußfolgerungsmechanismus, ist daher eher für die Abgrenzung einer großen Klasse von KI-Software gegenüber anderen Programmen geeignet.

Autonomous agents are computational systems that inhabit some complex dynamic environment, sense and act autonomously in this environment, and by doing so realize a set of goals or tasks for which they are designed. [Mae95]

Maes vom MIT Media Lab ist einer der Pioniere auf dem Gebiet der Agentenforschung und Leiterin der Software Agents Group am MIT. Ihre Definition enthält wichtige Aussagen, die von vielen anderen Forschern ebenfalls als grundlegend für Agenten angesehen werden. Dazu gehört das autonome Verhalten, welches den Agenten befähigt, Entscheidungen auf Grundlage seines inneren Zustandes selbst zu fällen. Ein weiterer Punkt ist die Zielorientiertheit bei der Wahl der Handlungen und schließlich werden auch die möglichen Umgebungen mit den Attributen komplex und dynamisch eingeschränkt, wobei ‚komplex' ein sehr unscharfes Kriterium darstellt. Für eine bessere Unterscheidung zwischen Agenten und anderen Programmen findet sich auf der Homepage der von ihr geleiteten Software Agents Group folgende Aussage:

Software agents differ from conventional software in that they are long-lived, semi-autonomous, proactive and adaptive. [Mit98]

Diese Darstellung ist als Zusatz zu den vorangegangenen Definitionen zu sehen. Die danach einen Agenten gegenüber anderer Software am besten unterscheidenden Eigenschaften sind neben der Langlebigkeit vor allem seine Proaktivität und Anpassungsfähigkeit, die nur in wenigen anderen Programmen anzutreffen sind. Es gibt jedoch eine Gruppe von Software, die diese Eigenschaften in sich vereint und meiner Meinung nach nicht zu den Agenten gerechnet werden sollte: Computerviren. Aus diesem Grund möchte ich die folgenden Definitionen anführen.

We use the term "agents" for autonomous software processes that interact according to certain conventions. Agents may be solitary, as for instance personal information filtering agents, or they may be social and interact with each other on behalf of their clients. Agents for electronic markets, workflow agents, or information brokers are examples of social agents. [Vos97]

Voss arbeitet in der CSCW-Gruppe (Computer Supported Collaborative Work) der GMD – Forschungszentrum Informationstechnik GmbH[2] und ist derzeit in Forschungsprojekten zum Sozialverhalten von Agenten eingebunden. Dieser Aspekt des Sozialverhaltens, z. B. der Kommunikation der Agenten untereinander, wird auch in ihrer Definition hervorgehoben. Ein interessanter Punkt ist auch die Forderung nach Konventionen, welche die Interaktion des Agenten mit seiner Umwelt bestimmen. Dadurch scheiden die vorher angeführten Computerviren als Instanzen von Agenten aus.

Abschließend möchte ich ein von Foner ausgearbeitetes Kriterienbündel anführen [Fon94], in dem er vielen als Agent bezeichneten Programmen dieses Prädikat aberkennt und seine Vorstellungen über das Wesen von Agenten beschreibt. Zusammenfassend sollte, nach seiner Ansicht, ein Agent über die folgenden Eigenschaften verfügen:

- *autonomy*: selbständige Handlungen im Interesse des Benutzers,
- *personalizability*: Anpassung an den Benutzer,
- *discourse*: Dialogfähigkeit mit dem Benutzer,
- *risk and trust*: vorhersagbares Verhalten durch das ein Vertrauensverhältnis vom Benutzer zu dem Agenten bei der Delegation von Aufgaben aufgebaut werden kann
- *domain*: abgegrenzte Domäne um Risiken bei dem Aufbau des Vertrauensverhältnisses abzuwägen
- *graceful degradation*: bei Problemen (besonders bei Mißverständnissen im Dialog) wenigstens Teile einer Aufgabe erfüllen und nicht die gesamte Ausführung abbrechen
- *cooperation*: Interaktion zwischen Benutzer und Agent ist eher partnerschaftlich denn kommandoorientiert
- *anthropomorphism*: menschliches Verhalten nachahmen; nach Foner keine notwendige Eigenschaft
- *expectations*: Agent sollte die Erwartungen des Benutzers erfüllen

Foner stellt mit seinen Forderungen zum Kommunikationsakt sehr hohe Anforderungen an einen ‚wirklichen' Agenten. Dieser erhält damit fast humane Züge, was sich auch an dem von Foner vorgestellten Beispiel eines Agenten zeigt. Dieser ist ein MUD (*multi user dungeon game*[3])-Roboter (Julia), der sich wie ein menschlicher Spieler durch das Spiel bewegt.

Während die meisten der bisher vorgestellten Definitionen den Terminus Softwareagent sehr allgemein fassen und weitere Anmerkungen im speziellen Fall zur Abgrenzung benötigen, verwendet Foner diesen Begriff

[2] http://www.gmd.de/Welcome.de.html
[3] Internet-Rollenspiele, in denen Dialoge und Aktionen die Spielzüge darstellen

sehr restriktiv und schließt damit eine große Zahl der mit Agentenfunktionalität beworbenen Programme aus.

Im nächsten Kapitel gehe auf das Verständnis der Industrie über den Definitionsbereich von Agenten ein und zeige dabei die Parallelen und Unterschiede zu den oben aufgeführten Definitionen auf.

2.1.3 Definitionen aus Sicht der Industrie

Die hier vorgenommene Zusammenstellung von Agentendefinitionen im Kontext der Industrie ist nur eine kleine Auswahl um die teilweise differierenden Ansichten zu veranschaulichen. Vergleichende Betrachtungen mit den Definitionen aus dem universitären Bereich folgen im anschließenden Kapitel.

The term agent is used to represent two orthogonal concepts. The first is the agent's ability for autonomous execution. The second is the agent's ability to perform domain oriented reasoning. [Vir98]

Diese Definition stammt von Sankar Virdhagriswaran, einem Mitarbeiter der Software-Entwicklungsfirma Crystaliz. Sie enthält zwei wesentliche Agenteneigenschaften, die auch schon in den vorherigen Definitionen zu finden waren, nämlich die autonome Ausführung und die dafür notwendige Fähigkeit, Schlüsse zu ziehen. Obwohl beide Eigenschaften eine zentrale Rolle bei agentenartigen Programme besitzen, eignet sich diese Beschreibung jedoch höchstens als grobes Klassifikationskriterium, da weitere Merkmale zur Abgrenzung von ‚normaler' Software fehlen (siehe das Problem Computervirus weiter oben).

Intelligent agents are software entities that carry out some set of operations on behalf of a user or another program with some degree of independence or autonomy, and in so doing, employ some knowledge or representation of the user's goals or desires. [Gil95]

Die dem Strategiepapier der IBM „The Role of Intelligent Agents in the Information Infrastructure" entstammende Definition fügt der vorangegangenen als wichtiges Merkmal die Zielorientiertheit bzgl. der Wünsche des Benutzers hinzu, wobei der Agent dafür eine Wissensrepräsentation besitzen muß. Die Definition bleibt jedoch zu unscharf, da über den Komplexitätsgrad der einzelnen Eigenschaften keine Aussagen getroffen werden und es deshalb eine Frage der Interpretation ist, welche Anwendungen zu den „intelligenten Agenten" gerechnet werden können.

An agent is an object that searches or watches for information based on criteria you specify. In an Agent Server application an agent, when activated by the dedicated watcher application, monitors a collection and takes a specified action when it locates a document that meets its search criteria. [Ver96, S. 1-4]

Veritys Beschreibung eines Agenten unterscheidet sich, wie auch die folgenden Definitionen, von den bisherigen darin, daß sie mit einem sehr eingeschränkten Blickwinkel auf das jeweilige Produkt formuliert wurde. In diesem Beispiel wird ein Agent mit einem Objekt gleichgesetzt, welches mittels bestimmter Vorgaben von dem Benutzer eine Dokumentenkollektion überwacht und bei Übereinstimmung mit den Suchkriterien Handlungen ausführt. Agenten werden damit auf den Bereich des Information Retrieval festgelegt. Ein wesentlicher Unterschied zu den vorangegangenen Formulierungen besteht auch darin, daß die Definition auf Grundlage der Funktionen und nicht mittels technologischer Prinzipien erfolgt.

Many distributed applications can benefit from active, mobile objects. Voyager allows agents*, autonomous objects, to move themselves and continue executing as they move. In this way, agents can act independently on the behalf of a client, even if the client is disconnected or unavailable.* [Obj97]

Bei der Firma ObjectSpace und ihrem Produkt Voyager™ steht der Mobilitätsaspekt von Agenten im Mittelpunkt. Einziges Kriterium ist die Fähigkeit von Objekten, sich selbständig (autonom) über ein Netzwerk zu bewegen und dabei nicht in ihrer Ausführung unterbrochen zu werden. Mobilität wird auch von vielen anderen Firmen als grundlegendes Merkmal betrachtet. Das zeigt sich u. a. in der Zahl von Ankündigungen für Entwicklungsumgebungen, die speziell auf die Anforderungen mobiler Programme zugeschnitten sind (siehe dazu auch Kapitel 3.3.2. auf Seite 43).

2.1.4 Zusammenfassung

Die ausgewählten Definitionen zeigen deutlich die Problematik einer kurzen und prägnanten Darstellung von Softwareagenten, bei der keine wichtigen Bereiche ausgegrenzt aber auch keine Software, die nicht zu den Agenten gehören sollte, mit einbezogen wird. Keine der vorgebrachten Beschreibungen erfüllt auch nur annähernd die Kriterien einer exakten Definition.

Bei der vergleichenden Betrachtung der Definitionen aus Forschung und Industrie muß zuerst eine Unterteilung der industriellen Definitionen vorgenommen werden. Anhand der Beispiele lassen sich zwei Kategorien unterscheiden:

1. Die aus den Forschungslabors stammenden Arbeiten, die meistens noch nicht mit konkreten Produkten (allenfalls Alpha- oder Betaversionen) einhergehen und oft in engem Kontakt mit universitärer Forschung stehen. Zu diesen zählen die ersten beiden Definitionen der Industriebeispiele.

2. Artikel, die auf der Grundlage kommerziell angebotener Produkte entstanden sind. Die restlichen vier Definitionen gehören in diese Kategorie.

Die Definitionen der ersten Kategorie ähneln schon aufgrund ihrer Herkunft denen aus dem universitären Umfeld. Die Begriffsbestimmungen aus diesen Bereichen sind möglichst umfassend angelegt und verwenden konzeptionelle Kriterien und Eigenschaften zur Beschreibung. Es ist damit möglich, die verschiedenen Definitionen als aufeinander aufbauend bzw. sich ergänzend zu betrachten und so zu einer, im speziellen Fall möglicherweise besser angepaßten, erweiterten Beschreibung zu gelangen. Im Gegensatz dazu stehen die Definitionen der zweiten Kategorie. Sie fokussieren auf einen kleinen Ausschnitt von vorstellbaren Agentenanwendungen. Durch die funktionale Darstellung lassen sie sich auch kaum verknüpfen, da die Beschreibung immer spezieller ausfallen und damit nur noch einem multifunktionalen Agenten gerecht würde.

Um die oben genannten Probleme zu umgehen, haben Wooldridge und Jennings in [Woo95] einen Mittelweg gewählt, indem sie zwei Verwendungsformen für den Terminus Agent einführen. Der erste wird von ihnen als schwacher Agentenbegriff bezeichnet, der allgemein akzeptierte Vorstellungen von einem Agenten enthält, den zweiten bezeichnen sie als starken Agentenbegriff, der eher Anlaß zu kontroversen Diskussionen gibt.
Der schwache Agentenbegriff wird von ihnen über Merkmale definiert, die sie als Grundvoraussetzung eines Agentenprogramms ansehen. Danach bezeichnet der Begriff Agent ein Hardware- oder Softwaresystem mit folgenden Eigenschaften:

- *autonomy*: Agenten können ohne direkte Kontrolle des Benutzers operieren und haben dabei Kontrolle über ihren internen Zustand;
- *social ability*: Interaktion von Agenten mit anderen Agenten oder Benutzern über eine Agenten-Kommunikationssprache;
- *reactivity*: zeitlich bestimmte Reaktionen auf Änderungen der Umwelt;
- *pro-activeness*: selbstinitialisierte Ausführung von Handlungen aufgrund interner Zielvorgaben.

Je nach Auslegung der Anforderungen der einzelnen Eigenschaften lassen sich die meisten der in den vorangegangenen Definitionen beschriebenen Agentensysteme auch in die durch den schwachen Agentenbegriff definierte Domäne einordnen.
Der starke Agentenbegriff trägt dagegen dem wesentlich restriktiveren Verständnis von Forschern der künstlichen Intelligenz Rechnung, wann ein System als Agent bezeichnet werden kann. Dabei spielen vor allem menschenähnliche Merkmale eine Rolle wie mentale Zustände (z. B. Wissen, Glauben, Intentionen, Verpflichtungen) oder die Verwendung animierter Figuren für eine Mensch-Maschine-Schnittstelle.

Obwohl dieser Ansatz besser in der Lage ist, die differierenden Vorstellungen von Agentensystemen zu erfassen, kann auch er keine befriedigende Lösung bieten, da die schwache Auslegung durch ihre Universalität ein zu breites Spektrum erfaßt, während der starke Agentenbegriff mehrere Erweiterungen enthält, deren genaue Spezifizierung individuell (von den jeweiligen Forschern) erfolgt.

Trotzdem läßt sich der schwache Agentenbegriff als kleinster gemeinsamer Nenner für eine Diskussion über Agenten verwenden und soll auch hier die eingangs gegebene Definition ersetzen.

Die in diesem Kapitel gezeigten Probleme einer präzisen Abgrenzung des Diskursbereiches Softwareagent bedingen alternative Ansätze für die sinnvolle Auseinandersetzung mit diesem Thema. Eine Variante ist die Einteilung in Kategorien, welche die Diskussion bestimmter Eigenschaften ermöglichen. Dieses kann mit einer Klassifikation des zu untersuchenden Bereiches nach bestimmten Kriterien erreicht werden. Im nächsten Kapitel werden Beispiele dazu angeführt.

2.2 Klassifikation von Softwareagenten

2.2.1 Kriterien für Klassifikationen

Wie das vorige Kapitel zeigte, erscheint es wenig praktikabel, das breite Spektrum der Agententechnologie mittels einer knappen Definition zu erfassen. Trotzdem ist es wünschenswert, Basiskriterien anzugeben, anhand derer Aussagen zu bestimmten Agenten und Vergleiche untereinander gemacht werden können. Ein sich daraus ergebendes Klassifikationsschema bietet die Möglichkeit, durch die Auswahl bestimmter Kriterien Agenten in Gruppen zusammenzufassen und so zu einer selektiven Definition zu gelangen.

Die Wahl der Kriterien ist abhängig von dem Ziel, das mit der daraus resultierenden Einteilung verfolgt wird. Entwickler, die sich für den internen Aufbau eines Agenten interessieren, werden andere Schwerpunkte herausstellen als Benutzer, denen die von dem Agenten angebotene Funktionalität wichtig ist. Neben dieser grundlegenden Entscheidung wird auch die Anzahl der verwendeten Kriterien wesentlich durch den späteren Verwendungszweck bestimmt. Viele Kategorien ermöglichen eine genaue Einteilung der Agenten, lassen sich jedoch kaum in einem Schema veranschaulichen.

Den im folgenden präsentierten Klassifikationen liegen unterschiedliche Zielvorstellungen zu Grunde, auf die bei den entsprechenden Punkten eingegangen wird. Es handelt sich dabei um eine Zusammenfassung von in der Literatur häufig vorgenommenen Gliederungen.

2.2.2 Klassifikation nach Eigenschaften

Die Kategorien im ersten Beispiel beruhen auf charakteristischen Merkmalen von Agenten wie sie schon teilweise in den Definitionen zur Anwendung kamen. Im folgenden werden die wichtigsten[4] davon vorgestellt und erläutert.

- Autonomie
  Die Autonomie gilt als eines der bedeutendsten Kriterien bei der Unterscheidung zwischen Agenten und herkömmlicher Software. Autonom handelnde Agenten sind in der Lage, die ihnen vorgegebenen Ziele und dabei zu lösende Teilprobleme ohne regelnde Eingriffe des Benutzers zu verfolgen. Dadurch wird der Benutzer von Wartezeiten und Überwachungsaufgaben befreit, da der Agent nach Erhalt der Aufgabenbeschreibung diese selbständig abarbeitet und erst bei Beendigung das Resultat in einer vorher vereinbarten Form liefert. Diese Arbeitsweise ist unter anderem bei der Arbeit mit dem Internet interessant, da die Verbindung zum Netz nach Aktivierung des Agenten getrennt werden kann und erst wieder aufgebaut werden muß um die Ergebnisse abzufragen.
  Der Grad der Autonomie ist abhängig von dem Aufgabengebiet und dem Vertrauen, das der Benutzer dem Agenten entgegenbringt und sollte deshalb von diesem bestimmt werden können. Ein einfacher Informationsagent kann normalerweise seine Aufgabe komplett erfüllen, ohne einer Entscheidung des Nutzers zu bedürfen. Dagegen wird von Agenten, die zur Unterstützung von Kaufentscheidungen entwickelt werden, erwartet, daß sie vor Entscheidungen mit finanziellen Konsequenzen die Bestätigung des Benutzers einholen.

- Mobilität
  Dieses Merkmal kennzeichnet eine große und leistungsfähige Gruppe von Agenten und kann, jedenfalls was die ‚echte' Mobilität betrifft, als Herausstellungsmerkmal gegenüber bisheriger Software angesehen werden. Mobilität bezeichnet die Fähigkeit von Agenten, sich auf andere Plattformen zu bewegen und dort ausgeführt zu werden. Unter Mobilität i. e. S. verstehe ich dabei das Vermögen eines Agenten, zu einem beliebigen Zeitpunkt zu einem anderen Rechner zu wechseln und an der Stelle fortzusetzen, an der er vor dem Transfer angelangt war. Nach erfolgter Übertragung wird der Prozeß auf dem Ursprungsrechner beendet. Eine einfachere Form der Mobilität besteht in der Übertragung von Programmcode und dessen Ausführung auf dem Zielrechner, etwa analog der Verfahrensweise bei Java-Applets. Solange ein Agent das Netzwerk nur für kommunikative Zwecke verwendet, kann er nicht mit dem Attribut ‚mobil' versehen werden.
  Die derzeitige Entwicklung bei Agentensystemen, speziell im industriellen Bereich, ist stark von dem Mobilitätsgedanken geprägt. Es werden schon eine Reihe von Entwicklungsumgebungen angeboten,

[4] Als Grundlage dienten folgende Arbeiten: [Bre98], [Etz95] und [Fra97]

die die Implementierung, vor allem auch ‚echter', Mobilität ermöglichen (siehe dazu auch Kapitel 3.3.2 auf Seite 43). Unterschiede bei der Konzeption von mobilen Agenten liegen dabei in der Aufteilung der Funktionalität zum Zusammen- und Entpacken des Agenten. Entweder bringt der Agent selbst diese Fähigkeiten mit oder verläßt sich auf Dienste der Laufzeitumgebung. Der häufig realisierte letzte Fall hat den Nachteil, daß der Agent in Ermangelung von Standards für Schnittstellen zu solchen, nur zwischen Rechnern mit gleicher Programmumgebung wechseln kann. Der Vorteil dieser Lösung besteht in der geringeren Codegröße, da er im Idealfall selbst keine Funktionen für seine Verschickung mitbringen muß.
Die Eigenschaft der Mobilität wirft, neben der Frage der funktionellen Lösung, noch mehrere Fragen hinsichtlich Sicherheit und Verwaltung bzw. Abrechnung auf. Die Sicherheitsanforderungen sind wesentlich höher als heute bei der Ausführung von Programmen auf WWW-Seiten, da die Agenten normalerweise ohne Anforderungen eines Systemverwalters auf dessen Rechner übertragen werden und dort zur Ausführung kommen. Neben dieser Sicherheitsproblematik ist ein weiterer Punkt die Vergütung der aufgewendeten Rechenkapazität und benutzten Dienste bei der Ausführung des Agenten auf einem fremden Host, die der Eigentümer des Agenten leisten muß. Die Entwicklung derartiger Abrechnungsmodalitäten steckt selbst bei WWW-Angeboten noch in den Kinderschuhen, so daß eine Lösung erst auf längere Sicht zu erwarten ist. Bis dahin werden sich mobile Agenten eher auf Intranet-Anwendungen beschränken, wo derartige Anforderungen eine geringere Rolle spielen.
Mobilität ist eng mit Autonomie verknüpft, da es unpraktikabel wäre, wenn der Agent von fremden Rechnern Verbindungen zum Benutzer aufbauen müßte, um bestimmte Entscheidungen anzufordern. Es sind im Gegenteil Autonomie und Mobilität, die die Netzwerkbelastung reduzieren sollen, da sie sich direkt zu den Informationsquellen bewegen können, dort lokal recherchieren und nur das Endergebnis zum Benutzer übertragen werden muß.

- Kommunikation
  Unabhängig vom Grad seiner Autonomie bedarf ein Agent der Interaktion mit seiner Umwelt um seine Ziele zu erreichen. Zum einen sollte er eine benutzerfreundliche Schnittstelle zum Anwender beinhalten – idealerweise in quasi-natürlicher Sprache und eventuell mit einer animierten Versinnbildlichung seines internen Zustandes (siehe dazu auch [Kod96]) – vor allem aber benötigt er Schnittstellen zu Datenquellen und anderen Agenten. Letztere Fähigkeit ermöglicht den aktiven Einsatz in Multiagentensystemen. Dafür benötigen die Agenten eine Kommunikationssprache (ACL – Agent Communication Language), in welcher die Syntax und Semantik der Nachrichten standardisiert ist (siehe auch Kapitel 3.1.1 auf Seite 23). Einfache Kommunikationsprotokolle bieten eine fest vorgegebene Anzahl von Funktionen zum Austausch von Daten und Mitteilungen, wie sie für Multiagentensysteme mit von vornherein fest definierter Aufgabenteilung benötigt werden (z. B. Interfaceagenten zur Dateneingabe, Transaktionsagenten zur Datenbearbeitung und andere Interfaceagenten zur Ausgabe von Statusinformationen). Komplexere Kom-

munikationssprachen sind für eine kooperative Zusammenarbeit der Agenten notwendig.

- Kooperation
  Wenn Agenten aufgrund von Ressourcenbeschränkungen oder Zeitlimits nicht in der Lage sind, ein bestimmtes Ziel zu erreichen, können sie dieses Problem möglicherweise durch die Vergabe von Teilaufgaben an andere Agenten lösen. Agenten, die über derartige Fähigkeiten verfügen, werden als kooperativ bezeichnet. Dafür sind jedoch bestimmte Eigenschaften erforderlich wie eine symbolische Präsentation des Problembereichs und eine Schlußfolgerungskomponente, die in der Lage ist, Teilziele abzuleiten, die an andere Agenten weiterdelegiert werden können sowie eine mächtige Kommunikationssprache, die es erlaubt, Kommunikationsakte den Erfordernissen dynamisch anzupassen.
  Um kooperative Agenten zu entwickeln, bieten einige Systeme sogenannte Marktplätze an, auf denen sich Agenten treffen und über eine Art Ausschreibung ihre Dienste und Arbeitsangebote bekanntgeben können. Die für die Kooperation notwendige Kommunikation enthält viele Parallelen zur menschlichen Kommunikation, weshalb zur ihrer Beschreibung auch die Sprechakttheorie von Austin[5] Verwendung findet (z.B. in [Bre98, S. 103]).

- Reaktivität
  Die Reaktivität ist eine grundlegende Eigenschaft von Agenten und wird von jedem Agenten in bestimmtem Umfang unterstützt. Sie beinhaltet die Fähigkeit eines Agenten, auf Änderungen bzw. Informationen aus der Umwelt in geeigneter Art und Weise zu reagieren. Echte reaktive Agenten erhalten die Informationen über Sensoren (Schnittstellen zur Umwelt) während sogenannte deliberative Agenten über eine interne abstrakte Repräsentation ihrer Umwelt verfügen und daraus die notwendigen Schlußfolgerungen ziehen können (siehe dazu auch Kapitel 2.2.3 auf Seite 15).

- Proaktivität
  Gegenüber der Reaktivität, die auch die meisten herkömmlichen Programme aufweisen, stellt die Proaktivität eine qualitative Steigerung dar. Proaktive Agenten werden nicht nur aufgrund äußerlicher Einflüsse aktiv, sie können auch bei Erreichen bestimmter interner Zustände selbst die Initiative ergreifen. Eine wichtige Eigenschaft, die mit der Proaktivität einhergeht, ist die Zielorientiertheit. Nur die Agenten, die über intern repräsentierte Zielvorgaben verfügen können auch sinnvoll ihre Umwelt durch Aktionen beeinflussen, die diese in Richtung der Zielerfüllung verändern.

- Schlußfolgerung
  Mit der Fähigkeit, Schlußfolgerungen über seiner aktuellen Wissensbasis zu ziehen, kann ein Agent aktiv das Erreichen seiner Ziele beeinflussen. Dabei wird von der Schlußfolgerungskompo-

[5] Austin, J. L., How to do things with words, Harvard University Press, Cambridge 1962

nente verlangt, daß sie rational arbeitet, d. h., daß die sich daraus ableitenden Handlungen nicht dem Gesamtziel entgegen arbeiten. Die Technologien dafür kommen meistens aus der klassischen KI oder beruhen auf dem Einsatz von neuronalen Netzen. Derartige Agenten sind von ihrer Struktur her den deliberativen Agenten zuzuordnen und besitzen einen entsprechend komplexen Aufbau.

- Lernfähigkeit / Adaption
  Agenten mit der Fähigkeit, aus den bisherigen Daten zu lernen, d. h. sich an ihre Umwelt anzupassen (Adaption), setzen normalerweise Schlußfolgerungsmechanismen voraus. Die Konsequenzen aus den Schlußfolgerungen werden von diesen Agenten nicht nur in Aktionen mit der Umwelt umgesetzt, sondern auch zur Anpassung des internen Modells. Agenten für das Information Retrieval könnten z. B. ihr Profil anpassen, wenn der Benutzer durch Feedback die Relevanz der bisher vom Agenten gelieferten Dokumente einschätzt.
- Kontinuierliche Ausführung (engl. temporal continuity)
  Ein Agent ist normalerweise über einen längeren Zeitraum aktiv, arbeitet dabei im Hintergrund und kann bei Bedarf den Benutzer über den aktuellen Zustand informieren. Damit hebt er sich von einfachen Programmen ab, die auf eine bestimmte Eingabe mit einer entsprechenden Ausgabe reagieren und dann ihre Arbeit beenden.

Mit Hilfe der o. g. Kriterien lassen sich Agenten bzgl. ihrer Eigenschaften sehr genau einordnen und nach jeder einzelnen Komponente vergleichen bzw. zusammenfassen. Problematisch wird jedoch eine Übersicht nach dieser Einteilung, da für jedes Merkmal eine Dimension erforderlich ist. Aus diesem Grund haben z. B. Gilbert et al. in [Gil95], Nwana in [Nwa96] und Brenner et al. in [Bre98, S. 33 ff.] jeweils drei der ihnen am wichtigsten erscheinenden Eigenschaften ausgewählt und die anderen Merkmale diesen untergeordnet. Die Variante des letztgenannten Autors möchte ich stellvertretend für die anderen Lösungen vorstellen.

Die ausgewählten Kategorien entsprechen in etwa denen von Gilbert et al. Der einzige Unterschied besteht darin, daß Gilbert et al. anstelle der Anzahl von Agenten das Merkmal Agency verwendet, welches den Grad der Interaktion angibt und damit auch in etwa mit der Anzahl von Agenten gleichgesetzt werden kann. Im Gegensatz zu Gilbert et al. ist die Achseneinteilung mit jeweils zwei Werten recht grob angegeben und eignet sich daher eher für die Zuordnung von Agenten zu bestimmten Oberklassen. Für feinere Unterscheidungen zwischen einzelnen Agenten sollte daher eine Einteilung wie in Gilbert et al . herangezogen werden. Von den restlichen Eigenschaften von Agenten können die Reaktivität, Proaktivität, Schlußfolgerung und Lernfähigkeit auf der Intelligenzachse abgebildet werden, Kommunikation und Kooperation hängen eng mit der Anzahl der Agenten im System zusammen und die Autonomie korreliert mit der Mobilität.

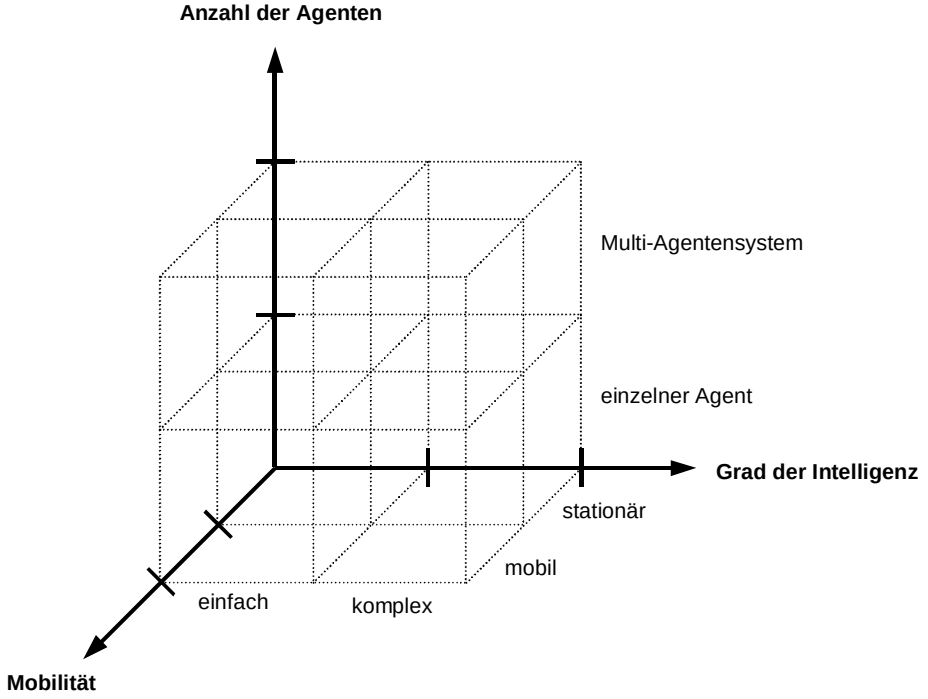

Abbildung 1: Klassifikationsmatrix für Agentensysteme nach Brenner et al.

2.2.3 Klassifikation nach Architektur

Von besonderem Interesse für Entwickler von Softwareagenten sowie Forschern aus dem Bereich der künstlichen Intelligenz ist die verwendete Architektur und die ihr zugrundeliegende theoretische Basis. Mit der in den letzten Jahren verstärkt anwendungsorientierten Sicht auf Agenten und der daraus resultierenden Entwicklung alternativer Architekturen lassen sich die heute verfügbaren Agenten in die nachfolgenden drei Kategorien einteilen.

- Deliberative Agenten
  Der Ansatz dieser Architektur geht auf das klassische Paradigma der Symbolverarbeitung aus der künstlichen Intelligenz zurück. Ein deliberativer Agent besitzt demzufolge eine explizite symbolische Repräsentation der Welt, d. h. von dem für ihn relevanten Teil der Welt, und trifft Entscheidungen aufgrund logischer Schlußfolgerungen, die auf Symbolverarbeitung basieren. Für den Schlußfolgerungsprozeß benötigt der Agent neben dem Weltmodell auch eine Repräsentation seines inneren Zustandes.
  Ein verbreiteter Ansatz zu dessen Modellierung stammt von Rao und Georgeff [Rao91]. Bei dem von ihnen entworfenen BDI-Agenten setzt sich der innere Zustand aus den Komponenten für Überzeugungen (***b**elief*), Wünschen (***d**esire*) und Intentionen (***i**ntention*) zusammen. Eine bei anderen Autoren extra aufgeführte Komponente für Ziele (*goal*) geht bei ihnen in den Wünschen mit auf. Zu den Überzeugungen gehören Fakten über die Umwelt sowie Erwartungen an zukünftige Umweltzustände. In den Wünschen werden mög-

liche Umweltzustände danach kategorisiert, ob sie erstrebenswert sind oder verhindert werden sollten, wodurch implizit die Ziele des Agenten definiert werden. Die Intentionen stellen wiederum die Wünsche dar, die der Agent gerade verfolgt, da er, bedingt durch die Komplexität dieses Ansatzes, nicht in der Lage ist, alle Wünsche bzw. Ziele gleichzeitig zu berücksichtigen.
Deliberative Agenten haben den Vorteil, daß sie sich durch Änderung des Weltmodells und ihres inneren Zustandes an nahezu beliebige Aufgaben anpassen lassen. Auch sind die ihnen zugrundeliegenden Theorien innerhalb der KI weitgehend erforscht; jedoch zeigen gerade diese Arbeiten auch gravierende Probleme symbolverarbeitender Systeme auf, wie z. B.:

- Übertragung der realen Welt in eine akkurate und adäquate symbolische Repräsentation mit begrenztem Umfang innerhalb einer bestimmten Zeit
- Schlußfolgerungskomponente muß in einer ressourcenbegrenzten Umgebung in vorgegebener Zeit nützliche Resultate liefern (siehe z. B. die Ausführungen in [Woo95] zur logischen Allwissenheit)
- Reaktion auf Änderungen der Umwelt in einem zeitlich vertretbaren Rahmen (in Zusammenhang dazu stehen Fragen der Anpassen der internen Weltrepräsentation sowie Entscheidungen über Abbruch oder Fortsetzung der aktuellen Ableitungsketten)

Wie aus der Aufzählung ersichtlich, liegt die Hauptschwierigkeit deliberativer Agenten in ihrer zeitlichen Reaktionsfähigkeit. Um diesem Problem zu begegnen, wurde die im folgenden vorgestellte Architektur entwickelt.

- Reaktive Agenten
  Diese Art von Agenten läßt sich am besten im Vergleich zu den deliberativen Agenten beschreiben. Sie besitzen kein explizites symbolisches Modell ihrer Umwelt und haben auch keine Komponente für Schlußfolgerungsprozesse. Im Gegensatz dazu arbeiten sie nach dem Reiz-Reaktion Schema, ähnlich den Verhaltensmodellen des Behaviourismus (Vertreter: B. F. Skinner). Schema. Der Agent nimmt über Sensoren Änderungen seiner Umwelt war, verarbeitet diese in dafür spezifisch entwickelten Modulen und vollzieht Aktionen entsprechend der Resultate. Dabei brauchen die Verarbeitungskomponenten nicht notwendigerweise eine komplexe Struktur aufzuweisen, um angemessene Reaktionen hervorzurufen. Dadurch lassen sich mit dieser Architektur schnelle, kompakte und, bzgl. einer Aufgabe, flexible Softwareagenten erstellen. Auch zeigen diese Agenten oft ein robusteres Verhalten, da selbst bei Ausfall einer Komponente die anderen, meist unabhängig ausgelegten, Komponenten weiterhin funktionieren.
  Einer der stärksten Verfechter dieses Ansatzes ist R. Brooks, der formuliert hat, daß sich intelligentes Verhalten eines Agenten aus seiner Interaktion mit der Umwelt ergibt und keine symbolische Verarbeitung voraussetzt. Brooks hat selber eine auf diesem Konzept

basierende ‚Subsumptions-Architektur' entwickelt, die aus mehreren Ebenen besteht, wobei die höheren Ebenen aufbauend auf den unteren Ebenen immer komplexere Verarbeitungen durchführen können. Mit dieser Architektur gelang es ihm, Roboter Aufgaben ausführen zu lassen, die für symbolische Systeme als sehr beachtlich gelten würden.
Trotz dieser augenscheinlichen Vorteile gibt es auch hier Nachteile wie die Festlegung der Agenten auf jeweils einen bestimmten Aufgabenbereich und die damit verbundene nahezu komplette Neuentwicklung von Agenten für andere Einsatzgebiete oder die Frage, ob sich die Zielorientiertheit deliberativer Agenten durch die Interaktionen reaktiver Agenten ausreichend nachbilden läßt. Aus diesem Grund wurden Systeme mit nachfolgender Architektur entwickelt.

- Hybride Agenten
  Nach Ansicht vieler Forscher sind weder komplett deliberative noch reaktive Architekturen für die Entwicklung von Agenten geeignet. Alternativ wird von ihnen ein hybrides System vorgeschlagen, das die Vorteile beider Ansätze vereint und damit den jeweiligen Nachteilen begegnet. Daraus ergibt sich, daß solche Agenten aus wenigstens zwei Subsystemen bestehen: ein reaktives, welches auf Ereignisse aus der Umwelt ohne den Umweg komplexer Schlußfolgerungen reagieren kann und ein deliberatives, das ein symbolisches Weltmodell besitzt, Pläne ausarbeitet und entsprechende Handlungen, um die Ziele zu erreichen, veranlaßt. Die reaktive Komponente wird dabei oft vorrangig behandelt, um vorgegebene Antwortzeiten zu gewährleisten.
  Die Struktur hybrider Agenten legt eine Schichtenarchitektur nahe, bei der höhere Schichten auch einen größeren Abstraktionsgrad besitzen. Während die unterste Schicht Daten von den Sensoren entgegennimmt und direkt Effektoren ansprechen kann (reaktive Komponente), befassen sich die oberen Schichten mit langfristigen Zielen (deliberative Komponente). Ein Hauptproblem derartiger Systeme liegt in der Kontrolle der Interaktionen der einzelnen Schichten.

Für alle drei Architekturen existieren Beispiele von Agentensytemen (siehe z. B. Übersicht in [Bre98, S. 78]). Eine vergleichende Betrachtung von Agenten aus Industrie und Forschung zeigt eine Tendenz bei Agenten in kommerziellen Produkten in Richtung reaktiver Architektur während im Forschungsumfeld, dominiert durch den Einfluß der künstlichen Intelligenz, die Agenten eher eine deliberative Architektur aufweisen. Anspruchsvollere Agentensysteme (wie z. B. [Oas98]) werden oft in hybrider Architektur konzipiert.

2.2.4 Klassifikation nach Anwendung

Vom Standpunkt des Anwenders aus gesehen, sind die bisherigen Klassifikationen wenig hilfreich für eine vergleichende Übersicht von Softwareagenten. Für ihn sind weniger die internen Details als die von dem Agenten nach außen gezeigte Funktionalität, sein Einsatzgebiet sowie das darin gezeigte Verhalten von Interesse.

Die nachfolgende Gliederung entstand durch Abstraktion von bestimmten Agentenanwendungen. Aufgrund dessen ist es möglich, daß neue Anwendungsgebiete nur durch Erweiterung dieses Schemas aufgenommen werden können. Außerdem können Agenten eine komplexere Funktionalität aufweisen, so daß sie mehreren Kategorien zuzuordnen sind.

2.2.4.1 Information Management

Eine große Gruppe von Softwareagenten dient der Informationsrecherche, -aufbereitung und –verwaltung. Der Bedarf hierfür ist direkt mit der Entwicklung des Internet verknüpft. Einfache Informationsagenten waren deshalb auch mit die ersten im größeren Maßstab eingesetzten Agenten. Obwohl der Bereich der Informationsverarbeitung vielfältige Aufgabengebiete besitzt, kristallisieren sich die folgenden Schwerpunkte heraus.

- Information Retrieval (IR)
  Ingwersen beschreibt das Gebiet des IR wie folgt: „*Information retrieval is concerned with the process involved in the representation, storage, searching and finding of information which is relevant to a requirement for information desired by a human user*. [Ing92, S. 49]“
  Für den Bereich der Agenten ist davon besonders die Aufgabe der gezielten Informationssuche interessant. Wesentliche Kriterien, welche die Qualität der Suchergebnisse bestimmen, sind *precision* (die Menge an relevanten Informationen im Ergebnis) und r*ecall* (die Menge der gefundenen relevanten Informationen aus der theoretischen Menge aller relevanten Daten).
  Im Internet als einem der größten Anwendungsgebiete für IR werden Informationsagenten von den Suchmaschinen[6] eingesetzt, die dem Benutzer erlauben, das WWW nach bestimmten Kriterien zu durchsuchen. Den Agenten kommen dabei bislang vor allem die Aufgabe der Informationsbeschaffung zu, was sich auf den r*ecall* auswirkt. Es handelt sich dabei um stationäre Agenten, die autonom nach einem vorgegebenen Algorithmus WWW-Seiten laden, aufbereiten und dem Suchindex hinzufügen. Informationen über noch zu untersuchende Seiten erhalten die Agenten durch die Analyse der auf einer Seite enthaltenen Links, wobei die Reihenfolge ähnlich wie bei einer Breitensuche bestimmt wird, um flächendeckende Informationen bieten zu können.
  Dieser Ansatz hat den Nachteil, daß auch eine größere Anzahl derartiger Agenten nicht in der Lage ist, mit der Dynamik des WWW (Ändern, Löschen oder Erstellung von Inhalten) Schritt zu halten [Law98]. Einen Lösungsansatz hierfür könnten mobile Agenten liefern, die auf die http-Server migrieren, dort lokal die Änderungen protokollieren und die gesammelten Daten zur Suchmaschine transferieren.

[6] Beispiele für Suchmaschinen: AltaVista™ (http://altavista.digital.com), HotBot (http://www.hotbot.com)

Zukünftig ist auch der Einsatz von Agenten zur Erhöhung der *precision*-Werte denkbar. Diese bestimmen mittels eines nutzerspezifischen Profils Filterkriterien für eine weitere Selektion der Daten. Das Profil bzw. der Agent bleibt über die einmalige Anwendung hinaus erhalten und steht so für jede weitere Anfrage zu Verfügung.

- News Watcher
  Mittels News-Watcher-Agenten kann der Benutzer bestimmte Nachrichtenquellen abonnieren und die Informationen aufgrund persönlicher Vorgaben filtern lassen. Dem Agenten kommen dabei die Aufgaben zu, die Daten zusammenzuführen, automatisch zu aktualisieren sowie ihre Relevanz bzgl. des Nutzerprofils zu bestimmen. Der Vorteil dieser Vorgehensweise gegenüber einer wiederholten manuellen Suche besteht in der zeitlichen Entkopplung von Suche und Aufbereitung der Daten auf der einen und Abruf durch den Nutzer auf der anderen Seite. Damit entfallen die sonst notwendigen Wartezeiten und der Benutzer kann unmittelbar auf aktuelle Informationen zurückgreifen.
  Im WWW existieren bisher einfache Formen von News-Watcher-Agenten, die es erlauben, eine persönliche Web-Seite zusammenzustellen, indem der Benutzer aus dem Angebot die für ihn interessanten Informationsgebiete auswählt.
- GroupWare
  Die Automatisierung von Routineaufgaben innerhalb von Arbeitsgruppen bietet ein breites Feld für den Einsatz von Agententechnologie. Mit ihrer Hilfe kann der notwendige Kommunikationsaufwand zwischen den Anwendern reduziert, sowie Entscheidungen beschleunigt werden. Zu den Aufgabengebieten gehören:
  - Filtern, Weiterleiten und Verteilen von Email
  - globale Terminplanung, abgestimmt auf die Vorgaben und bestehenden Termine aller Beteiligten
  - Bereitstellung kontextbezogener Information aus den verfügbaren Datenquellen zur Unterstützung von Entscheidungen

Die aktuellen Groupwarelösungen sind inzwischen mit einer Anzahl agentenartiger Module ausgestattet, die dem Benutzer die eine oder andere Aufgabe abnehmen können – speziell bei der Behandlung von Email. Sie sind heute jedoch noch weit davon entfernt, autonom komplexere Vorgänge zu bearbeiten.

2.2.4.2 Schnittstellenagenten

Obwohl sie sich eher durch die Funktionsweise der Agenten definiert, kann man sie auch durch eine abstraktere Anwendungssicht beschreiben. Schnittstellenagenten können überall dort eingesetzt werden, wo der Benutzer mit dem Computer interagiert. Sie haben die Aufgabe, Hilfestellungen im entsprechenden Kontext anzubieten, sowie weiterführende Ratschläge zu erteilen. Aufgrund dieser Zielstellung werden sie

auch in der Literatur unter dem Begriff ‚Advising and Focusing' geführt [Bre98, S. 224].

Im Gegensatz zu den meisten anderen Agenten benötigen sie keine Profilvorgaben durch den Benutzer, sondern können ein solches selbständig während einer Trainingsphase erstellen und auch später noch anpassen. Dieses erreichen sie durch Protokollieren der Nutzeraktivitäten und dem Erkennen von Mustern und Regularitäten darin. Ausgestattet mit einer entsprechenden Trainingsmenge können sie bestimmte Tätigkeiten des Benutzers vorwegnehmen oder weiterführende Informationen in der entsprechenden Situation anbieten.

Der Erkennungsprozeß erfordert die Verwendung spezieller Techniken des maschinellen Lernens. Deshalb besitzen Vertreter dieser Kategorie von Agenten einen höheren Komplexitätsgrad[7] als vergleichbare Agenten aus anderen Bereichen. Je besser ein Schnittstellenagent in der Lage ist, das Verhalten seines Benutzers vorauszusagen und/oder zu imitieren, desto höher ist die Wahrscheinlichkeit, daß er von dem Benutzer als Hilfe akzeptiert und genutzt wird.

Abhängig davon, wie benutzerspezifisch die zu erkennenden Verhaltensmuster sind, benötigen diese Agenten eine relativ lange Trainingsphase, um Datenmaterial sammeln zu können, d.h. ihre Lernkurve steigt nur langsam an. Dieses stellt ein großes Hindernis für die Akzeptanz von Schnittstellenagenten dar. In [Las94] wird versucht, diesem Problem zu begegnen, indem anstelle eines einzelnen Agenten ein Multiagentensystem eingeführt wird. Jeder Benutzer besitzt einen persönlichen Agenten mit einem bestimmten Erfahrungsgrad. Wenn ein neuer Agent ohne eigene Trainingsdaten erstellt wird, kann er trotzdem von Anfang an bestimmte Voraussagen treffen, da er Zugriff auf die Erfahrungsdaten der anderen Agenten hat. Obwohl sich das Verhalten der Benutzer normalerweise unterscheidet, liefert dieser Ansatz Ergebnisse, die einen Einsatz des Agenten ohne eine lange Lernphase zulassen.

2.2.4.3 Prozeßüberwachung und –steuerung

Systeme, die sich mit einem weitgehend abgeschlossenen und beschränkten Weltmodell darstellen lassen, sind ein idealer Einsatzort für Multiagentenanwendungen. Mit ihren Sensoren können sie den Zustand des Systems überwachen, diesen mit den in den Profilen vorgegebenen Zielen vergleichen, notwendige Aktionen in Kooperation mit den anderen Agenten beschließen und ausführen.

Derartige Systeme finden sich u.a. in industriellen Fertigungsanlagen, bei denen es auf Prozeßoptimierung ankommt. Einzelne Arbeitsplätze lassen sich durch Agenten modellieren, welche die Parameter und Ka-

[7] Bei Vergleichen wird dieser auch als erhöhte ‚Intelligenz' des Agenten bewertet

pazitäten widerspiegeln. Durch Zusammenschluß zu Arbeitsgruppen und Verwendung eines geeigneten Kommunikationsprotokolls können Fertigungsaufgaben effizient erledigt werden.

Ein weiteres Anwendungsgebiet dieser Agenten ist die Überwachung von Patienten auf der Intensivstation [Lar96]. Um schnellstmöglich auf Schwankungen des Gesundheitszustand eingehen zu können, bedarf es des Spezialwissens eines Facharztes verbunden mit der ständigen Kontrolle des Patienten. Diese beiden Punkte lassen sich in einem deliberativen Multiagentensystem integrieren in dem das Fachwissen des Arztes in dem Weltmodell integriert ist. Aufgrund der hohen Verantwortung, die mit Entscheidungen in diesem Bereich einhergeht, sind bisher Prototypen nur anhand von Simulationsdaten getestet worden. Es zeigte sich, daß sie eine wichtige Unterstützung bei zeitkritischen Entscheidungen des Arztes sein können [Lar96, S. 1].

2.2.4.4 Unterhaltung

Im Unterhaltungssektor lassen sich zwei Hauptrichtungen von Agentenanwendungen unterscheiden. Auf der einen Seite existieren Computerspiele mit vielfältigen computeranimierten Charakteren, die sich ganz natürlich als Agenten implementieren lassen (autonomes Verfolgen bestimmter Ziele, Kooperation, Kommunikation). Diese haben schon eine längere Entwicklungsphase hinter sich und profitieren von der steigenden Leistungsfähigkeit der Rechner, die immer komplexere Verhaltensweisen ermöglicht.

Auf der anderen Seite entwickelt sich ein Markt für Agenten, die das Auffinden anderer Internetteilnehmer mit ähnlichen Interessen zum Ziel haben. Diese Gruppenbildung dient einerseits dazu, Kontakte mit anderen Menschen herzustellen, andererseits aber auch für die Erstellung von Katalogen mit Verweisen auf für den Benutzer interessante Angebote.

Beiden Arten von Unterhaltungsagenten ist gemein, daß es sich um Multiagentensysteme handelt. Im Falle der Computerspiele ist weniger die Kommunikation als die schnelle Reaktionsfähigkeit und Erstellung von Handlungsplänen gefragt, während die für die Gruppenbildung zuständigen Agenten im wesentlichen bestimmte Profildaten der Benutzer kommunizieren, um Übereinstimmungen zu erkennen.

2.2.4.5 Electronic Commerce

Obwohl der vom Menschen initiierte elektronische Handel erst jetzt langsam Bedeutung erlangt, gibt es schon einige Ansätze und Überlegungen, zum Einsatz von Softwareagenten auf diesem Gebiet.

Dabei lassen sich zwei verschiedene Ansätze unterscheiden. In einer ersten und einfacheren Stufe dienen sie lediglich der Unterstützung des

Einkäufers, indem sie ausgerüstet mit Angaben zu dem gewünschten Produkt verschiedene Angebotsseiten besuchen und eine Liste mit den Preisangaben der Anbieter erstellen. Bessere Agenten dieser Art lassen dann die direkte Bestellung von dieser Übersicht aus zu und sind in der Lage, mehr als eine Produktkategorie zu verarbeiten.

In der nächsten wesentlich anspruchsvolleren Stufe agieren Kauf- und Verkaufsagenten selbständig in einem virtuellen Marktplatz und treffen Entscheidungen bzgl. Geldausgaben und Preisen. Der Benutzer gibt nur noch Eckwerte zu Minimal- und Maximalpreisen vor, sowie der Art der Verhandlungsstrategie. Weitere Agenten zur Unterstützung des Handels (z.B. Broker) sind denkbar.

Für solche komplexen Handlungsszenarien bedarf es standardisierter Multiagentenumgebungen sowie Mechanismen für Sicherheit, Authentifikation und Verschlüsselung. Weiterhin bestimmen schnelle und ‚intelligente' Entscheidungsverfahren über den Erfolg der Agenten. [Gut98] gibt einen Überblick über den aktuellen Stand von agentenbasierten Electronic Commerce.

3 Entwicklungsstand

Dieses Kapitel gibt anhand ausgewählter Beispiele einen Eindruck vom derzeitigen Entwicklungsstand aktueller Agentensystemen. Eine entscheidende Rolle für deren Weiterentwicklung tragen dabei die Standardisierungsbemühungen. Sie werden vorrangig betrachtet.

3.1 Standards

Agenten, die in abgeschlossenen Welten unabhängig voneinander für sich existieren, können keinen besonderen Vorteil aus Normierungen ziehen. Im Gegensatz dazu sind mobile Agenten sowie Agenten in offenen Multiagentenumgebungen auf Standards angewiesen, wenn sie die Möglichkeiten des internetweiten Austauschs nutzen sollen.

Es überrascht daher kaum, daß die wesentlichen Bemühungen zur Einreichung und Verabschiedung von Standards sich gerade auf diese zuletzt genannten Bereiche konzentrieren. Im folgenden werden die wichtigsten davon mit den hinter ihnen stehenden Organisationen vorgestellt.

3.1.1 Knowledge Sharing Effort

Der Anfang der 90er Jahre gegründete *Knowledge Sharing Effort* (KSE) ist eine Initiative von Firmen und Universitäten mit dem Ziel der Entwicklung von Konventionen und Techniken zum Aufbau großer Wissensbasen, deren Inhalte untereinander austauschbar und wiederverwendbar sind. Neben der Veröffentlichung von Spezifikationen und technischen Abhandlungen soll auch eine öffentlich zugängliche Programmbibliothek mit generischen Funktionen entstehen, welche die Umsetzung der Konzepte demonstriert.

Nach einer Analyse der größten Hindernisse beim systemübergreifenden Wissensaustausch bildeten sich 4 Arbeitsgruppen, die für die entsprechenden Problemfelder Technologien für die Überwindung der Hemmnisse entwickeln sollten. Im einzelnen sind das die folgenden Gruppen:

1. *Interlingua Group*
   Die *Interlingua Gro*up Entwickelt eine allgemeine Sprache, mit der sich der Inhalt von Wissensbasen ausdrücken läßt und als Übertragungsmedium geeignet ist. Sie soll als Vermittler zwischen verschiedensprachigen Systemen und als Standardschnittstelle für Programmbibliotheken eingesetzt werden.

2. *Knowledge Representation System Specification Group* (KRSS)
   Da sich selbst innerhalb einer Repräsentationsfamilie der Daten-

austausch durch kleine syntaktische oder semantische Unterschiede erschwert wird, besteht das Ziel dieser Gruppe in der Definition einheitlicher Konstrukte auf der Ebene einer solchen Familien.

3. *Shared, Reusable Knowledge Based Group* (SRKB)
   Beschäftigt sich mit der Angleichung der Inhalte von Wissensbasen, so daß ein Austausch darüber möglich wird. Im Wesentlichen geht es dabei um die Übersetzung verschiedener Ontologien und Modelle für deren Austausch.

4. *External Interface Group*
   Definiert das Interaktionsverhalten von wissensbasierten Systemen (WBS) mit anderen Modulen zur Laufzeit. Besondere Beachtung findet dabei die Kommunikation zwischen zwei WBS sowie zwischen einem WBS und einem konventionellen DBMS.

Für den Bereich Softwareagenten sind insbesondere die Teile interessant, die als Agenten-Kommunikationssprache (ACL - *Agent Communication Language*) für die standardisierte Kommunikation zwischen verschiedenen Agenten verwendet werden können. Speziell die Ergebnisse aus drei Arbeitsgruppen führen gemeinsam zu einem dreischichtigen Sprachmodell, wobei sich jede einzelne Ebene unabhängig von den anderen weiterentwickeln läßt und auch – abhängig von der Anwendung – alleine einsetzbar ist.

Im folgenden gehe ich genauer auf die Bestandteile dieser ACL ein, da sie, abgesehen von ihrem Einsatz in einer Anzahl von Agentensystemen, auch als Grundlage für Standardisierungen Verwendung finden.

1. KIF

   KIF steht für *Knowledge Interchange Format*, entwickelt von der Interlingua Arbeitsgruppe, um Wissen zu kodieren. In erster Linie wurde es als eine Interlingua entworfen, d.h. die Anzahl notwendiger Übersetzungsmodule für den Austausch zwischen verschiedenen WBS kann auf ein Mindestmaß von zwei pro WBS reduziert werden, da nur noch nach und von KIF zu übersetzen ist[8].

   Unabhängig davon eignet sich KIF jedoch auch als externe Repräsentationssprache für Wissen sowie – jedoch eingeschränkt – für die interne Darstellung. Selbst die Interaktion mit dem Benutzer über KIF ist denkbar, wenn auch etwas abstrakt[9] und deshalb höchstens als Testmöglichkeit für Programmierer interessant. Folgende Eigenschaften zeichnen KIF aus:

[8] Der Aufwand wird von quadratisch zu linear verringert

[9] Nach [Gen92] kann man die Verwendung von KIF als Benutzerschnittstelle mit dem Schreiben eines Dokumentes in Postscript vergleichen

- es besitzt eine formal definierte deklarative Semantik, d.h. die Bedeutung eines Ausdrucks ist ohne die Verwendung eines Interpreters verständlich
- es verwendet eine Präfix-Variante des Prädikatenkalküls erster Stufe mit verschiedenen Erweiterungen und ist logisch abgeschlossen
- KIF ist in der Lage, nichtmonotone Schlußfolgerungsregeln zu repräsentieren, was z.B. für die Übermittlung des Weltmodells eines Agenten in einer dynamischen Umgebung gebraucht werden könnte
- die Repräsentation von Wissen über die Repräsentation von Wissen wird unterstützt, wodurch neue Konstrukte für die Wissensrepräsentation ohne Änderungen an der Sprache eingeführt werden können

Das Format von KIF orientiert sich stark an der Programmiersprache LISP und verwendet als Alphabet den ASCII-Zeichensatz. Ein logischer Ausdruck der Form $\forall x : P(x) - Q(x)$ würde als Satz in KIF folgende Form besitzen: `(forall ?x (=> (P ?x) (Q ?x)))`.

Die in [Gen92] vorgestellte Definition von KIF ist als erweiterbares Grundgerüst gedacht, das durch die in der Sprache gebotenen Elemente an die spezifischen Bedürfnisse der Anwendung angepaßt werden kann, z.B. an eine *frame*-basierte Wissensdarstellung.

Ob sich KIF für die interne Repräsentation von Wünschen und Zielen von Agenten eignet, hängt wahrscheinlich von der Komplexität der benötigten Schlußfolgerungsmechanismen über diesen Strukturen und den Umfang der Daten ab. Für den Austausch solcher Informationen zwischen Agenten läßt es sich jedoch aufgrund seiner umfassenden Funktionalität sinnvoll einsetzen.

2. Ontolingua

Die SRKB-Arbeitsgruppe stellte als ein Hauptproblem für den Datenaustausch zwischen WBS die Notwendigkeit der Verwendung einer gemeinsamen Ontologie heraus. Eine Ontologie in diesem Sinne läßt sich wie folgt definieren: „*A specification of a representational vocabulary for a shared domain of discourse — definitions of classes, relations, functions, and other objects — is called an ontology.“* [Gru93]

Der Einsatz einer Ontologie in einem WBS erfordert eine Repräsentationssprache, die diese Konzepte im Programm formal kodiert. Die einfachste Art, eine zwischen den Systemen austauschbare Ontologie zu erhalten, ist, die gleiche Repräsentationssprache zu benutzen. Da jedoch verschiedene Anwendungen unterschiedliche Anforderungen an Schlußfolgerungsmechanismen stellen und die existierenden Repräsentationssysteme jeweils für andere Aufgaben optimiert sind, ist die Verwendung einer einheitlichen Repräsentati-

onssprache nicht zweckmäßig. Daraus ergibt sich das Problem, daß eine gemeinsame Ontologie in vielen Repräsentationssystemen kodiert werden muß.

Die Entwicklung von Ontolingua versucht, diese Schwierigkeiten zu minimieren. Es ist ein System zur Beschreibung von Ontologien in einer zu bestehenden Repräsentationssprachen kompatiblen Form. Erreicht wird dies durch Übersetzungsmodule, die gegebene Definitionen, die in einer standardisierten, deklarativen Sprache vorliegen, in die Formate der einzelnen Repräsentationssysteme übersetzt (siehe Abbildung 2).

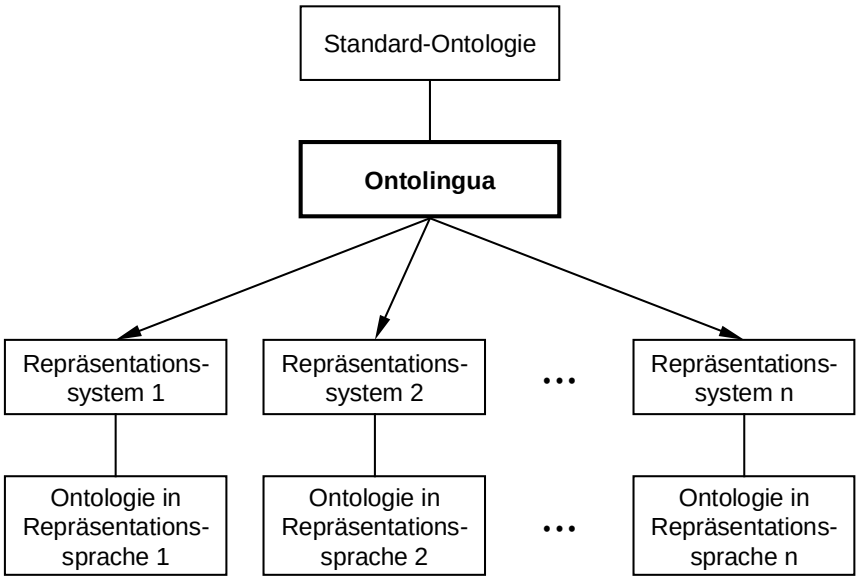

Abbildung 2: Übersetzungskonzept von Ontolingua

Die Syntax und Semantik von Ontolingua basiert auf KIF. Damit ist es einem menschlichen Leser direkt möglich, den Inhalt zu erfassen, andererseits unterstützt es in dieser Form jedoch nicht automatische Schlußfolgerungsoperationen. Seine Ausdrucksmächtigkeit wurde so gewählt, daß es dem aktuellen Stand der Wissensrepräsentation entspricht, es stellt jedoch kein Repräsentationssystem dar.

Die Definitionen von Ontolingua besitzen neben der formalen Notation in KIF auch eine englische Beschreibung, die zwar nicht geparsed wird, aber das Verständnis der Ontologie erleichtern soll. Ein Beispieleintrag, entnommen aus [Gru93], sieht wie folgt aus:

```
(define-class AUTHOR (?author)
```
„An author is a person who writes things. An author must have created at least one document. In this ontology, an author is known by his or her real name.“
```
:def (and (person ?author)
     (= ( value-cardinality ?author
          AUTHOR.NAME) 1)
     (value-type ?author AUTHOR.NAME
      biblio-name)
     (>= (value-cardinality ?author
          AUTHOR.DOCUMENTS) 1)
     (⇔   (author.name ?author ?name)
          (person.name ?author ?name))))
```

Die Sprachen, in die Ontolingua übersetzt, sind je nach ihrem Fokus in ihrer Ausdrucksstärke eingeschränkt, um ein gutes Laufzeitverhalten zu erreichen. Aus dem Ansatz von Ontolingua, eine gegebene Ontologiebeschreibung nach allen ausgewählten Repräsentationssystemen übersetzen zu können, folgt, daß es sich auf die kleinste gemeinsame Menge der von jedem System unterstützten Konstrukte beschränken muß. Es ist damit hinsichtlich der KIF Sprache schon vom Ansatz her unvollständig.

An der Stanford University wird ein Ontolingua Server[10] betrieben, der die oben beschriebenen Übersetzungsdienste anbietet, gleichzeitig aber auch netzwerkweiten Zugriff für die Erstellung und Abfrage von Ontologien bietet. Er verfügt gegenüber der ‚Standardontolingua' über einige Erweiterungen, insbesondere für die Verknüpfung ontologischer Module. Eine ausführliche Beschreibung des Servers wird in [Far96] gegeben.

3. KQML

Obwohl die *External Interface*-Arbeitsgruppe des KSE mit der Standardisierung von ‚high-level' Schnittstellen zwischen WBS betraut ist, hat sich ihre Aktivität bald auf den Entwurf eines gemeinsamen Sprachstandards, mit dem Informationen und Wissen zur Laufzeit zwischen Systemen ausgetauscht werden kann, zusammen mit einem Protokoll für dessen Übertragung, verlagert.

Diese als Knowledge Query and Manipulation Language (KQML) bezeichnete Sprache ist sowohl Nachrichtenformat, wie auch –protokoll. Sie kann sowohl zur Interaktion zwischen Anwendungen und intelligenten Systemen, als auch zur Kooperation zwischen zwei oder mehreren WBS (z.B. Agenten) für kooperatives Problemlösen eingesetzt werden (in der weiteren Beschreibung werden Agenten als Handlungspartner eingesetzt).

[10] Der Server ist unter http://ontolingua.stanford.edu/ erreichbar

KQML besteht aus einer erweiterbaren Menge von Performativen, d.h. bestimmte Sprechakttypen[11], welche die möglichen Operationen definieren, die ein Agent auf das Wissen, Wünsche und Ziele eines anderen anwenden kann. Mit den gegebenen Performativen als Basis lassen sich höhersprachliche Interaktionsmodelle entwickeln, angepaßt an die Anforderungen der entsprechenden Anwendung.

Neben der direkten Kommunikation zweier Agenten bietet KQML eine Architektur für spezielle Agenten, so genannte Vermittler (*communication facilitators*), die, als eine Art Broker, den Nachrichtenaustausch koordinieren können. Sie bieten über spezielle Performative u.a. folgende Funktionen an:

- Identifizierung anderer Agenten explizit durch Name, Adresse oder implizit durch beworbene Fähigkeiten, die bei der Anmeldung bei einem Brokeragenten übermittelt wurden
- Verwaltung einer Registrierungsdatenbank, in der die von den Agenten offerierten und gesuchten Dienste abgelegt sind
- Kommunikationsservice, der die Informationen von einem Agenten an interessierte Agenten weiterleitet
- Übersetzungsservice, durch den semantische und/oder ontologische Unterschiede zwischen Agenten ausgeglichen werden

Die Ausdrücke in KQML besitzen eine Ebenenstruktur. Sie bestehen aus einer Inhaltsschicht (*content layer*), die von einer Nachrichtenschicht gekapselt wird (*message wrapper*), welche wiederum von einer Kommunikationsschicht eingeschlossen ist (*communication wrapper*). Diese Schichtenfolge ist in Abbildung 3 schematisch dargestellt.

Die **Inhaltsebene** enthält in einer beliebigen Sprache, die einigen allgemeinen syntaktischen Regeln folgen muß, einen Ausdruck, der das zu übermittelnde Wissen kodiert. Für KQML ist das Format der Nachricht unerheblich, der Empfänger sollte aber einen Parser dafür enthalten.

Auf der **Nachrichtenebene** wird die Art des Sprechaktes bzw. Performativs festgelegt, der mit dem Inhalt verbunden ist, wie z.B. eine Frage, Anweisung o.ä. Weiterhin können optional Angaben hinzugefügt werden, die eine Analyse und entsprechende Weiterleitung des Nachrichtenpaketes ermöglichen, ohne das der Inhalt bekannt sein muß. Dazu gehören unter anderem Hinweise zum Sprachformat des Inhalts und die benutzte Ontologie.

[11] Die Sprechakttheorie, die von J. L. Austin entwickelt und von J. R. Searle fortgeführt wurde, untersucht den Handlungsaspekt sprachlicher Äußerungen. Ein Sprechakt ist gekennzeichnet durch eine Äußerung (*lokutionärer* Akt) zusammen mit der damit vollzogenen Handlung (*illokutionärer* Akt). Die Sprechakte können über die sie bestimmenden Verben klassifiziert werden. Auf dieser Grundlage werden in KQML bestimmte Typen von Sprechakten definiert, die für die Kommunikation zwischen Agenten benötigt werden (z.B. mitteilen, anfragen, befehlen).

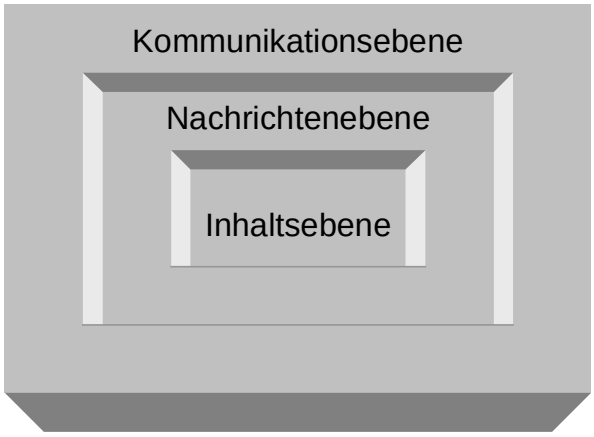

Abbildung 3: Ebenenstruktur von KQML

Die äußere **Kommunikationsebene** fügt der Nachricht in einer zweiten Schicht weitere Angaben hinzu, welche die untere Ebene der Kommunikationsparameter betreffen, wie z.B. die eindeutigen Bezeichner von Sender und Empfänger, eine Nachrichten-ID usw. Diese Angaben werden von der KQML-Anwendung an die Protokolle der Transportschicht[12] weitergegeben, die für die Weiterleitung über das Netzwerk verantwortlich sind.

Die Kodierung einer KQML-Nachricht erfolgt in einer LISP-ähnlichen Klammernotation. Parameter werden mit einem Doppelpunkt begonnen und sind, da sie einen eindeutigen Namen tragen, nicht von der Reihenfolge abhängig. Ein Beispiel für einen Nachrichtenaustausch zwischen zwei Agenten A und B könnte wie folgt aussehen (in Anlehnung an [Fin94]):

```
(ask-one
    :content (PRICE IBM ?price)
    :receiver B
    :sender A
    :language LPROLOG
    :ontology NYSE-TICKS
    :reply-with Q1)
```

und B antwortet mit

```
(reply
    :content (1234)
    :receiver A
    :language LPROLOG
    :ontology NYSE-TICKS
    :in-reply-to Q1)
```

[12] Aktuelle Implementierungen benutzen beispielsweise TCP/IP (Transmission Control Protocol / Internet Protocol), Email, HTTP (HyperText Transfer Protocol) und CORBA (Common Object Request Broker Architecture)

Aufgrund der großen Anzahl von Performativen in KQML wird von einem KQML-konformen System nicht verlangt, alle zu unterstützen. Dagegen ist es sogar erlaubt, je nach Bedarf weitere Performative einzuführen. Die einzige Bedingung ist, daß in KQML spezifizierte Performative nicht anders als angegeben behandelt werden dürfen.

Das Semantikmodell, das KQML zugrunde liegt, geht davon aus, daß jeder Teilnehmer über eine Wissensbasis (WB) verfügt, auf die sich die Kommunikationsakte beziehen. Zu diesen gehören Aussagen über und Fragen nach dem Inhalt der WB, Anforderungen für Hinzufügen oder Löschen von Einträgen und Anforderungen, bestimmte Einträge der WB (das Wissen) für die Lösung von Aufgaben zu benutzen. Durch dieses Modell braucht Kontextinformation nicht bei jeder Nachricht ausgetauscht zu werden und ermöglicht so die Reduktion der Sätze auf die wesentliche Aussage.

Die Spezifikation zu KQML von der *External Interface*-Arbeitsgruppe [Fin93] stammt von 1993 und wurde seit dem in mehreren Projekten getestet und Implementiert. Seit 1997 liegt ein Vorschlag von Labrou und Finin vor [Lab97], der eine Überarbeitung speziell der Menge der reservierten Performative sowie deren Bedeutung und Verwendung beinhaltet. Er nimmt verstärkt Bezug auf den Einsatz in Agentenumgebungen, indem er die dort interessanten Kommunikationsszenarien beschreibt.

Jeder der vorgestellten Standards des *Knowledge Sharing Effort* leistet einen Beitrag für die Vereinheitlichung der Interaktion von Agenten. Die Zusammenfassung aller drei Spezifikationen führt zu einer leistungsfähigen ACL, die den Anforderungen derzeitiger Agentensysteme genügt. KQML stellt als Protokoll und Nachrichtenrahmen die Syntax der Sprache dar, KIF, genutzt für das Kodieren der Nachricht, definiert, zusammen mit Objekten und Relationen aus Ontolingua, die Semantik.

3.1.2 FIPA

Im Gegensatz zum KSE, der sich nur teilweise mit agentenspezifischen Fragen beschäftigt, wurde die ‚Foundation for Intelligent Physical Agents' (FIPA) ausschließlich mit dem Ziel des Erarbeitens von Richtlinien und Standards für die Entwicklung von Agenten gegründet[13]. Sie ist eine gemeinnützige Vereinigung, der 47 Firmen, Organisationen und Universitäten weltweit angehören[14].

Auslöser für die Gründung war die Erkenntnis, das bestimmte Agententechnologien ein fortgeschrittenes Stadium erreicht haben, jedoch nur durch Standards nutzbringend eingesetzt werden können. Deshalb sieht die FIPA ihre Aufgabe in der Definition von Schnittstellen der verschie-

[13] Gründungstag ist der 5.9.1996
[14] Stand 30.10.1998

denen Komponenten in der Umwelt eines Agenten, mit denen er interagiert. Dazu gehören Menschen, andere Agenten, Software allgemein, sowie die physische Welt.

Die FIPA veröffentlicht in bestimmten zeitlichen Abständen Spezifikationen, welche die Interoperabilität zwischen agentenbasierten Anwendungen maximieren soll. Diese Dokumente sind für jeden Interessierten frei zugänglich und sollen in ihren endgültigen Versionen bei den entsprechenden Standardisierungsgremien eingereicht werden.

Da es sich bei der FIPA um eine recht junge Vereinigung handelt, existieren bereits einige, teilweise proprietäre, Arbeiten anderer Firmen und Organisationen zur Definition einheitlicher Schnittstellen für Agentensysteme, wie z.B. vom bereits vorgestellten KSE. In ihrem eigenen Entwurf bemüht sich die FIPA deshalb, so weit wie möglich diese Vorarbeiten zu integrieren. Aus diesem Grund kann es bei einer konformen Implementierung zur Erhebung von Lizenzgebühren kommen.

Die Spezifikationen werden in einzelne thematische Einheiten gegliedert, die jeweils eine eindeutige Nummer zugeordnet bekommen. Nachfolgende Versionen brauchen damit nur die Einheiten aufzuführen, in denen Änderungen vorgenommen wurden oder die neu hinzugekommen sind. Die Einheiten werden je nach ihrer intendierten Rolle in Normative und Informative unterschieden. Normative Spezifikationen legen fest, wie sich ein Agent nach außen verhalten muß, während informative Spezifikationen bestimmter Anwendungsfälle Richtlinien für den Einsatz der FIPA-Technologien geben.

In der folgenden Tabelle sind die derzeit veröffentlichten Spezifikationen mit den behandelten Themengebieten zusammengestellt. Einheiten, die mit ‚N' gekennzeichnet sind, tragen normativen Charakter, die mit ‚I' informativen und Einheiten mit zusätzlichem ‚M' beinhalten methodische Vorgaben für die Agentenentwicklung.

		veröffentlicht Oktober 1997	veröffentlicht Oktober 1998	
Einheit		FIPA 97 Version 1.0	FIPA 97 Version 2.0	FIPA 98 Version 1.0[15]
1	N	Agent Management	Agent Management	Agent Management Extensions
2	N	ACL	ACL	
3	N	Agent Software Integration		
4	I	Personal Travel Assistant		

[15] In [Fip98] wird diese Ausgabe als Version 0.2 bezeichnet, was dem Stand einzelner Artikel eher entspricht; entsprechend dem Inhalt ist die endgültige Version im Juli 1999 zu erwarten

		veröffentlicht Oktober 1997	veröffentlicht Oktober 1998	
Einheit		FIPA 97 Version 1.0	FIPA 97 Version 2.0	FIPA 98 Version 1.0[15]
5	I	Personal Assistant		
6	I	Audio Visual Entertainment & Broadcasting		
7	I	Network Management & Provision		
8	N			Human-Agent Interaction
9[16]	I/M			Product Design and Manufacturing Agents
10	N			Agent Security Management
11	N			Agent Management Support for Mobility
12	N			Ontology Service
13	I/M			Developer's Guide

Tabelle 1 : Inhalte der FIPA-Spezifikationen in Anlehnung an [Mam98]

Die einzelnen Einheiten befassen sich mit folgenden Themen:

- *Agent Management*
  In diesem Abschnitt wird ein Referenzmodell für eine Agentenplattform beschrieben, in dem Agenten nach dem FIPA-Standard operieren und Verwaltungsaufgaben an ihnen vorgenommen werden können. Die Plattform stellt den Agenten verschiedene Dienste zur Verfügung, wie z.B. Nachrichtenverteilung, die wiederum als Agenten implementiert sind. Diese können über formale Kommunikationsakte basierend auf definierten Nachrichtenmengen angesprochen und genutzt werden. Festlegungen zur verwendeten Ontologie und Semantik ermöglichen es den Agenten sich gegenseitig über ihre Möglichkeiten zu unterrichten. Als Basisprotokoll, welches eine FIPA-konforme Agentenumgebung unterstützen muß, ist *Internet Inter-ORB*[17] *Protocol* (IIOP) vorgesehen. Empfohlen wird weiterhin die Verwendung von *Common Object Request Broker Architecture* (CORBA) zur Kapselung der grundlegenden Mechanismen der Kommunikation (Spezifikation von CORBA und IIOP in [Omg98]).
- *Agent Communication Language* (ACL)
  Das Problem des standardisierten Austauschs von Nachrichten zwischen verschiedenen Agenten hat die FIPA durch die Entwicklung

[16] In [Mam98] ist diese Einheit nicht aufgeführt, wohl aber in [Fip98]
[17] *Object Request Broker*

einer eigenen Agenten-Kommunikationssprache angegangen. Ähnlich wie KQML basiert sie auf der Sprechakttheorie und definiert eine Menge von Nachrichtentypen. Für die Vergabe von Aufträgen an Agenten beinhaltet die Spezifikation Vorgaben für eine Menge von Protokollen, wie z.B. Kontraktnetz und verschiedene Arten von Auktionen.

- *Agent Software Integration*
  Aufgrund der relativ kurzen Entwicklungsgeschichte von Agenten besteht der größte Teil derzeitiger Anwendungen aus nicht für Agenten aufbereiteter Software. Zur Erfüllung ihrer Aufgaben müssen Agenten jedoch mit dieser Software in Verbindung treten. Teil 3 der Spezifikation beinhaltet deshalb ausschließlich Normierungen für generische Agent-Software-Kopplungen. Wesentlich dabei ist die Verwendung von Wrappern, die über einen dynamischen Registriermechanismus die Funktionen der Software in der Agentenumgebung anbieten. Als Vermittler wird ein Agent Resource Broker (ARB) definiert, der die Angebote koordiniert und die Benutzung durch die Agenten verwaltet, wie z.B. Parameterübermittlung, Authentifizierung, Rechtevergabe.
- *Personal Travel Assistance*
  Unter Verwendung der aufgestellten Normen aus den anderen Teilen der Spezifikation demonstriert dieser Abschnitt die Verwendung der Agententechnologie für das Gebiet der Reiseplanung und –durchführung. Beschrieben wird ein sogenanntes Personal Travel Assistance (PTA) System, welches aus einer Anzahl Agenten besteht, die den Benutzer, sowie die Dienstleistungen der Reiseanbieter und –agenturen repräsentieren, als auch die Vermittlung zwischen diesen durchführen. Dabei ist das System verantwortlich für die zeitgerechte Zustellung der Informationen, Kostenabrechnung, Qualitätssicherung und angemessenen Schutz privater Daten.
- *Personal Assistant*
  Unter einem Personal Assistant (PA) versteht die FIPA einen Agenten, der halbautomatisch im Auftrag des Benutzers agiert, über ein Modell der Nutzerinteressen vefügt und Dienstleistungen gegenüber dem Benutzer, anderen Menschen bzw. deren PA's anbietet. Beispielgebend für ein solches System wird die Verwaltung eines persönlichen Terminplaners beschrieben [Fip97], die in der Lage ist, Zeit und Ort für Treffen mit mehreren Teilnehmern festzulegen.
- *Audio Video Entertainment & Broadcasting*
  Als ein weiters Anwendungsgebiet für Agententechnologie enthält Teil 6 der FIPA Spezifikation die nutzerspezifische Bereitstellung von audio-visuellen Informationen über digitale Nachrichtennetzwerke. Verantwortlich für die Programmzusammenstellung sind auf der einen Seite die Agenten der Inhalteanbieter, auf der anderen Seite Agenten des Benutzers mit Vorgaben zu dessen bevorzugten Programmen. Ein solches System soll die notwendigen Aktionen des Benutzers selbst bei einer steigenden Anzahl von Anbietern so einfach und gering wie möglich halten.

- *Network management & provisioning*
  Die teilweise recht chaotische Struktur des Internet auf Leitungs- bzw. Verbindungsebene mit seinen vielen Netzwerkanbietern und deren verschiedenen Dienstleistungen zum einen und den unterschiedlichen Kundenwünschen zum anderen ist Anlaß für die Vorschläge zu diesem Teilgebiet. Im Mittelpunkt steht die Bereitstellung von dynamischen Virtual Private Networks (VPN) zwischen bestimmten Nutzern mit vorgegebenen Qualitätsmerkmalen. Automatisierte Verhandlungen zwischen Agenten, welche die Interessen der Benutzer vertreten und Agenten der Netzwerkbetreiber, die nach erfolgtem Vertragsabschluß die entsprechenden Netzwerkkonfigurationen veranlassen, sollen sowohl eine schnelle und unkomplizierte Bereitstellung der Netzwerkkapazitäten ermöglichen, als auch deren profitable Auslastung gewährleisten.

- *Human Agent Interaction*
  Dieser erstaunlicherweise erst im FIPA'98-Entwurf aufgenommene normative Abschnitt definiert grundlegende Aspekte der Mensch-Agent Schnittstelle. Zwei Agentendiensten kommt dabei eine zentrale Rolle zu. Der User Dialog Management Service (UDMS) stellt einen Wrapper für verschiedene Benutzerschnittstellen (BS) dar, der eine Abbildung zwischen der BS und der ACL vornimmt und so die Interaktion auf ACL-Ebene ermöglicht. Neben diesem für die Kommunikation notwendigen Dienst dient der User Personalization Service (UPS) dem Aufbau eines Benutzermodells, das die Interessen und Wünsche des Benutzers widerspiegelt und Grundlage für die Entscheidungen des Agenten ist. Dabei bietet dieser Dienst die Flexibilität, die notwendigen Informationen explizit oder durch Beobachtung des Benutzerverhaltens zu erhalten.

- *Product Design and Manufacturing Agents*
  Der Bereich vom Produktdesign über die Herstellung bis hin zur Auslieferung liegt oft nicht in einer Hand, sondern bedarf der Zusammenarbeit vieler Firmen auf internationaler Ebene. Dabei ist die Verwendung von Standards unabdingbar. Dieser Abschnitt demonstriert die Relevanz der Integration von Agentenstandards in den Produktionsprozeß anhand von Beispielen (z.B. die gemeinsame Nutzung von Ontologien mit Einbeziehung von STEP[18], Sicherheit von Agenten innerhalb von *virtual private networks*) und weist dabei auf aktuelle Trends in der industriellen Fertigung hin. Außerdem soll anhand von Testfällen die Vollständigkeit der derzeitigen Spezifikation untersucht und notwendige Weiterentwicklungen aufgezeigt werden.

- *Agent Security Management*
  Bei der Interaktion von Agenten mit anderen Partnern sowie bei mobilen Agenten treten erhöhte Sicherheitsrisiken auf. Der Teil 10 der FIPA Spezifikation benennt deshalb die wesentlichen Sicherheitsschwachstellen bei der Agentenverwaltung und spezifiziert die Mög-

[18] Standard die Repräsentation und den Austausch von Produktdaten (ISO 10303, http://www.nist.gov/sc4/www/stepdocs.htm)

lichkeiten für eine sichere Kommunikation unter Agenten durch die FIPA Agentenplattform. Die hier vorgestellte Spezifikation beschränkt sich auf die kleinste Menge der notwendigen Technologien und ergänzt die Angaben in Teil 1.

- *Agent Management Support for Mobility*
  Dieser Teil der Spezifikation normiert die technologischen Voraussetzungen, die für die Unterstützung mobiler Softwareagenten notwendig sind. Er bezieht sich auf die Agentenplattform der FIPA und stellt eine Erweiterung zur Agentenverwaltung aus Teil 1 dar. Die Normierung bezieht soweit als möglich existierende Standards ein und enthält auch bestimmte Verwaltungsaufgaben[19] für nicht mobile Agenten. Die Existenz dieses Teils der Spezifikation bedeutet nicht, das FIPA-konforme Agentenumgebungen mobile Agenten unterstützen müssen.
- *Ontology Service*
  Notwendige Voraussetzung für den Austausch von Informationen zwischen verschiedenen Agenten ist die Definition der verwendeten Ontologie. In diesem Teil werden deshalb die Technologien beschrieben, die einem Agenten den Umgang mit explizit deklarativ repräsentierten Ontologien ermöglicht. Ein Ontologieagent bietet dafür einen Ontologiedienst an, der es anderen Agenten ermöglicht, die verfügbaren Ontologien zu erkunden und zu verwenden, Ausdrücke zwischen verschiedenen Ontologien zu übersetzen, Beziehungen zwischen Termen zu klären, sowie eine gemeinsame Ontologie aufzustellen, die es zwei Agenten erlaubt, miteinander zu kommunizieren.
  Zu diesem Service sind die Schnittstelle spezifizert, die Interaktionsprotokolle, Kommunikationsakte und ein allgemeines Vokabular. Interne Implementierungen sowie das Format der Ontologie bleiben dem Entwickler vorbehalten. Jedoch ist eine Meta-Ontologie definiert, die den Austausch von Wissen zwischen Agenten erlaubt.
- *Developer's Guide*
  Der letzte Teil der 98er Spezifikation der FIPA enthält keine weiteren Anwendungsgebiete oder Normierungen, sondern ist der Erläuterung möglicher unklarer Bereiche der 97er Spezifikation vorbehalten. Darunter befinden sich auch Themen, die mehr als einen Teil der Normierungen betreffen.

Die FIPA ist bemüht, mit ihrem teilweise hierarchischen Gerüst von Spezifikationen alle relevanten Einsatzgebiete für Agenten und das daran beteiligte Umfeld zu standardisieren. Durch die Beachtung bereits existierender Standards sollte es nicht schwer fallen, bereits vorhandenen Agentensystemen FIPA-Konformität zu verleihen. Aufgrund der kurzen Zeitspanne seit die ersten Spezifikationen veröffentlicht wurden, gibt es derzeit jedoch noch kaum Anwendungen, die sich an ihnen orientieren.

[19] Z.B. die Konfiguration von Agenten

3.1.3 Agent Society

Neben der FIPA existiert ein weiterer Zusammenschluß von Firmen und Universitäten zur Unterstützung von Agententechnologien – die *Agent Society*[20]. Ihr Ziel ist die Förderung und Koordination neuer Entwicklungen auf dem Gebiet der Agentensysteme sowie damit verbundener Technologien. Das sich daraus ergebende Tätigkeitsfeld beinhaltet z.B. Sammeln und Publizieren von Informationen zu Agententechnologien, -anwendungen und –märkten, Vermarkten der Produkte, Fördern von Standards sowie von Technologietransfer zwischen Entwicklung und Industrie.

Noch während der Gründungsphase wurde schon von den Initiatoren der *Agent Society* über ein Protokoll und Framework interoperabler Agentenanwendungen diskutiert. Das Ergebnis waren Vorschläge zu einem *Simple Agent Transfer Protocol* (SATP, [Whi96]), mit dem Nachrichten oder komplette Agenten versendet werden können und zu einer *Common Agent Platform* (CAP, [Tha96]), einer Architektur, die die benötigten Dienste für mobile Agenten bereit stellt.

Mit der Bildung einer von der *Agent Society* gesponserten *Agent Interop Working Group*[21] verschob sich der Fokus weg von der Erarbeitung eigener Spezifikationen hin zur Diskussion und Unterstützung anderer Standardisierungsbemühungen, wie z.B. des FIPA-Entwurfs.

3.1.4 Object Management Group (OMG)

Die OMG[22] liefert mit ihren Spezifikationen für verteilte Objekte wie CORBA und IIOP (siehe [Omg98]) eine wichtige Grundlage, um agentenspezifische Protokolle und Anwendungen zu entwickeln, die eine netzwerkweite Kommunikation und Mobilität ermöglichen.

Darüber hinaus enthält der 1995 verabschiedete CORBA-Standard 2.0 in der Spezifikation zur *Common Facilities Architecture* [Omg95] einen die Belange von Agenten betreffenden Abschnitt. Die *Common Facilities* stellen mit Hilfe der Basisdienste (*Object Services*) spezialisierte, höherwertige Dienstleistungen für Objekte zur Verfügung. Sie werden grob unterteilt in *Horizontal Common Facilities*, welche diejenigen Dienste beinhalten, die von den meisten CORBA-Systemen verwendet werden und in *Vertical Market Facilities*, die jeweils auf ein bestimmtes Anwendungsgebiet zugeschnitten sind, wie z.B. Finanz- oder Fertigungssysteme.

Der *Horizontal Common Facilities* untergeordnet sind die *Task Management Common Facilites*, die Anwendungen eine Infrastruktur zum

[20] Im Internet erreichbar unter http://www.agent.org/
[21] Berichte über Meetings sind unter http://www.agent.org/society/meetings/workshop9702/index.html einzusehen
[22] Homepage http://www.omg.org

Modellieren von Prozessen und Benutzeraufgaben zur Verfügung stellen. Zu dieser Gruppe gehören die *Agent Facilities* mit Angaben zu den Schnittstellen und unterstützten Funktionen von Agenten. Wie aus der Spezifikation ersichtlich, basieren sie weitestgehend auf der KQML, die in 3.1.1 vorgestellt wurde. Die Anpassung an das Objektkonzept erfolgt durch die Definition zwei verschiedener Arten von Agenten: statische Agenten und mobile Agenten. Während statische Agenten für die Verarbeitung von Informationen und den Kontakt zu Anwendungsprogrammen ausgelegt sind, besteht die Hauptaufgabe mobiler Agenten in dem Transport der Informationen. Für die Umsetzung von KQML bedeuted dies, daß die mobilen Agenten im wesentlichen für die beiden äußeren Schichten (Kommunikations- und Nachrichtenebene) und die statischen Agenten für die innere Schicht (Inhaltsebene) zuständig sind.

Die in den *Agent Facilities* vorgegebenen Richtlinien waren vielen Anwendern zu spezifisch auf eine bestimmte Art von Agenten festgelegt, insbesondere läßt die wenig Spielraum für Agentenmodell bzw. –architektur. Da sich diese aufgrund der noch jungen Geschichte von Agenten kontinuierlich weiterentwickeln und derzeit existierende Agentenentwicklungssysteme auf verschiedenen Ansätzen beruhen, dauerte es nicht lange, bis die Änderungswünsche Beachtung fanden.

Auf der Grundlage einer Ausschreibung[23] der OMG, für eine dort *Mobile Agent Facility* (MAF) genannte Spezifikation, wurden verschiedene Vorschläge eingereicht, von denen die gemeinsame Einreichung der GMD FOKUS und IBM [Gmd97] im Februar 1998 als OMG Technologie akzeptiert wurde.

Diese, inzwischen als *Mobile Agent System Interoperability Facilities Specification* (MASIF) bezeichnet, hat die Interoperabilität von Agentensystemen mit Unterstützung mobiler Agenten zum Ziel. Definiert werden die für die Mobilität notwendigen Schnittstellen mit besonderer Beachtung von Sicherheitskonzepten. Zum jetzigen Zeitpunkt gehören dazu:

- Agentenmanagement
  Verschiedene Agentensysteme können mittels standardisierter Operationen über ein und dieselbe Anwendung administriert werden.
- Agentenübertragung
  Hier werden die Schrittfolgen und Zuständigkeiten für das Initiieren, Vorbereiten und Ausführen der Übertragung, sowie die Aufnahme durch das Zielagentensystem festgelegt.
- Namenskonventionen für Agenten und –systeme
  Als Voraussetzung für die Interoperabilität ist es notwendig, daß die jeweiligen Bezeichner eindeutig sind und Konformität über die Syntax und Semantik von Parametern herrscht.

[23] Common Facilities RFP 3 (Request for Proposals), OMG TC Document 95-11-3, November 3, 1995

- Typisierung von Agentensystemen und Abfragesyntax
  Das Ziel der MASIF-Spezifikation ist vorerst nicht, Austauschbarkeit zwischen allen Agentensystemen zu erreichen, sondern ‚nur' zwischen den in der selben Sprache entwickelten Systemen – diese können jedoch von verschiedenen Herstellern stammen. Damit ein Agent erkennen kann, welche anderen Systeme für eine Interaktion zur Verfügung stehen, wird eine Typisierung eingeführt, die alle erforderlichen Informationen enthält und einheitlich abfragbar ist.

Die MASIF zugrundeliegende Architektur geht, aufbauend auf die CORBA-Dienste, von drei Komponenten aus:

- Agenten, die in verschiedenen Zuständen vorliegen können,
- Agentensystemen, in welchen die Agenten ausgeführt werden und die in Regionen und Plätze organisiert sind (siehe Abbildung 4), sowie
- MAFFinder, eine Art Nameserver, der den Agenten und –systemen das gegenseitige Auffinden ermöglicht.

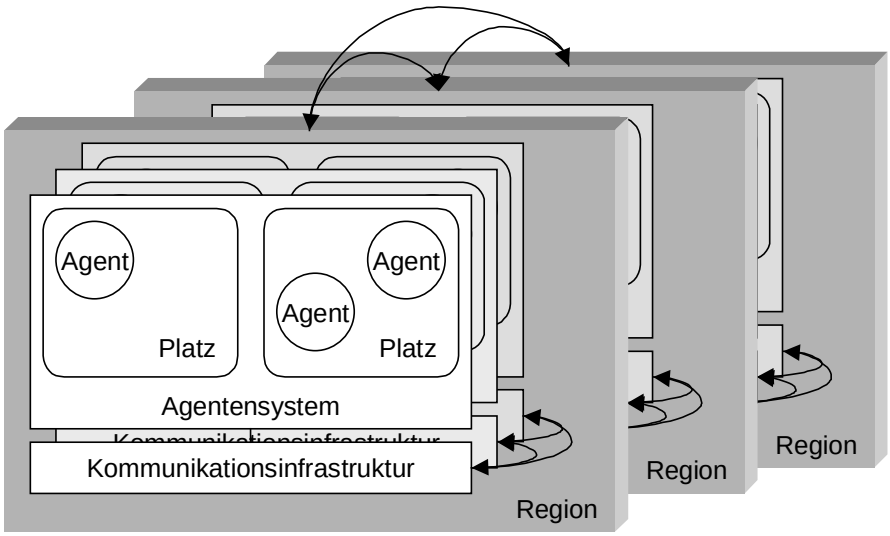

Abbildung 4: MASIF – hierarchische Organisationsstruktur der Agentenumgebung

Die MASIF-Spezifikation muß als eine Art Kompromiß angesehen werden, da sie auf der einen Seite den Herstellern von Agentensystemen entgegenkommt, ihre Produkte ohne zu großen Aufwand und ohne Paradigmenwechsel an den Standard anzupassen und dadurch auf breiter Ebene akzeptiert wird, auf der anderen Seite leistet sie aber nicht die erwünschte Zusammenarbeit möglichst aller Systeme. Ein weiterer Kritikpunkt ist das völlige Ausklammern der Inhaltsebene ausgetauschter Nachrichten, was auf längere Sicht die Zusammenarbeit verschiedener Agenten behindern wird. Dieses ist jedoch dem aktuellen Entwicklungsstand möglicher Protokolle und Sprachen geschuldet. Es bleibt zu hoffen, daß in einer späteren Version der sich dynamisch weiterentwickelnden OMG-Standards auch dieser Punkt Beachtung findet.

3.1.5 World Wide Web Consortium (W3C)

Die vom W3C[24] erarbeiteten Protokolle lassen sich teilweise für den Datenaustausch zwischen und von Agenten einsetzen (z.B. *Extensible Markup Language*[25] (XML) zur Kodierung eines Nachrichtenformats, *Resource Description Framework*[26] (RDF) für die Beschreibung von Metadaten). Mobile Agenten erfordern jedoch weitergehende Spezifikationen, insbesondere für strikte Sicherheitsmechanismen.

Im Rahmen der Arbeiten zu verteilten hypermedialen Anwendungen[27] wurden existierende Programmiersprachen auf ihre Fähigkeiten zur Unterstützung mobiler Programme (Agenten) untersucht, um sie in die Empfehlungen des W3C aufzunehmen[28]. Neben Fragen der Sicherheit und Plattformneutralität wurde deshalb auch das jeweilige Lizenzmodell betrachtet.

Diese Untersuchung wurde jedoch schon Ende 1996 durchgeführt und bis heute liegen keine abschließenden Ergebnisse vor. Es hat den Anschein, als ob das W3C die Ausarbeitung der Richtlinien für Agentensysteme anderen Organisationen (z.B. FIPA) überläßt und sich auf die Spezifikation der allgemeineren Übertragungsprotokolle beschränkt.

3.2 Beispiele für Softwareagenten in der Forschung

Wie auch auf anderen Gebieten sind die Prototypen von Agenten aus Forschungseinrichtungen denen kommerzieller Produkte in Hinsicht der Komplexität bzw. sogenanntem ‚intelligenten' Verhalten meist überlegen. An ihnen werden zuerst die Theorien und Algorithmen aus der künstlichen Intelligenz und anderen relevanten Wissenschaftsbereichen getestet und im Falle positiver Ergebnisse von der Industrie übernommen.

Die folgende Liste enthält eine Auswahl aktueller Projekte, in denen die Entwicklung und Untersuchung von Softwareagenten den Schwerpunkt bilden:

- *Oz Project*[29] – Entwicklung von *believable agents* für die Erstellung interaktiver Dramen
- *MASL Survivability Simulator*[30] – Multi-Agenten-Simulation
- *Guardian*[31] – Überwachung von Patienten auf der Intensivstation

[24] Homepage http://www.w3.org
[25] Spezifikationen unter http://www.w3.org/TR/REC-xml ; weitere Informationen unter http://www.w3.org/XML
[26] Spezifikation unter http://www.w3.org/TR/PR-rdf-syntax
[27] Weitere Informationen unter http://www.w3.org/DesignIssues/HyperMediaDev.html
[28] Der Stand der Evaluierung ist unter http://www.w3.org/MobileCode/index.html abrufbar
[29] http://www.cs.cmu.edu/afs/cs.cmu.edu/project/oz/web/oz.html
[30] http://dis.cs.umass.edu/research/mass/index.html
[31] http://ksl-web.stanford.edu/projects/guardian/index.html

- *Market Maker*[32] – Infrastruktur zur Entwicklung dynamischer Märkte für e-Commerce
- *Yenta*[33] – gruppenbildendes System
- *Mobile Agents for Routing Discovery*[34] – Untersuchung und Simulation des Einsatzes mobiler Agenten zur Erfassung dynamischer Netzwerktopologien
- *The Virtual Theater project*[35] – Entwicklung künstlicher Charaktere

Stellvertretend für die verschiedenen Projekte wird als nächstes ein Agentensystem ausführlich beschrieben, das viele der in Kapitel 2.2.2 aufgeführten Eigenschaften aufweist und sich durch interessante Ansätze für die Profilerstellung und Multiagentenfunktionalität von anderen Systemen unterscheidet.

3.2.1 Yenta

Das von Foner am MIT entwickelte Multiagentensystem mit Namen Yenta ([Fon97] und [Fon98]) läßt sich nach der Klassifikation aus 2.2.4 in den Unterhaltungsbereich einordnen. Dort gehört es als ein fortschrittliches Mitglied dem Bereich der gruppenbildenden Systeme an.

Im Yenta besitzt jeder Teilnehmer einen Agenten mit einem persönlichen Profil (Interessen des Benutzers). Dieses tauschen die Agenten untereinander aus und analysieren es auf Interessenübereinstimmungen. Werden solche gefunden, kommt es zur Bildung von sogenannten Clustern[36], in denen Menschen, bzw. deren Agenten, mit potentiell ähnlichen Interessengebieten zusammengefaßt sind. Innerhalb eines Clusters ist es möglich, Nachrichten auszutauschen, Mitglieder können sich vorstellen und Fragen an Experten richten, deren Existenz wahrscheinlich ist, wenn das Themengebiet den Interessen gleicht, die dem Cluster zugrundeliegen.

Diese Idee ist nicht ganz neu. Als ‚automated collaborative filtering'[37] [Fon97] lag sie schon Projekten wie Webdoggie[38], HOMR[39] oder Firefly[40] zugrunde. Bei diesen dienten jedoch Angaben zu ganz bestimmten Bereichen (z.B. Musik oder Film) für die Ermittlung von Übereinstimmun-

[32] http://ecommerce.media.mit.edu/maker/maker.htm
[33] http://foner.www.media.mit.edu/people/foner/yenta-brief.html
[34] http://nelson.www.media.mit.edu/people/nelson/research/routes-coopagents
[35] http://ksl-web.stanford.edu/projects/cait/index.html
[36] Entspricht dem von Foner benutzten Terminus
[37] Übersetzt: automatisches gemeinschaftliches Filtern (gemeint ist die Zusammenarbeit von Agenten beim Vergleich ihrer Profile)
[38] Projekt von Lashkari und Maes am MIT Media Lab (ehemals unter http://homr.www.media.mit.edu/projects/homr erreichbar, ist der Server derzeit – 01/1999 – nicht verfügbar)
[39] Projekt von Metral und Maes am MIT Media Lab (ehemals unter http://webhound.www.media.mit.edu/projects/homr erreichbar, ist der Server derzeit – 01/1999 – nicht verfügbar)
[40] Inzwischen ein kommerzielles Produkt (1998 von Microsoft aufgekauft); zu sehen unter http://www.firefly.com

gen. Dabei benötigten die Agenten nicht einmal eine Wissensrepräsentation dieser Domäne, sondern konnten praktisch auf Formulardaten arbeiten. Aus diesem Grund war es auch nicht notwendig, diese Agenten mit ausgefeilten ‚intelligenten' Algorithmen auszustatten.

Yenta unterscheidet sich in diesen und weiteren Punkten wesentlich von den erwähnten Programmen. Ausgangspunkt für die Entwicklung war die Beobachtung, daß es viele passive Nutzer des Internet gibt, die, da sie nicht selbst aktiv für Inhalte sorgen, nichts voneinander erfahren, aber möglicherweise einander weiterhelfen könnten. Ein weiterer Aspekt ist die Schwierigkeit, Experten für ein bestimmtes Thema im Internet zu finden. Die Yenta-Agenten haben also die Aufgabe, ohne Hilfe durch den Benutzer, dessen Interessenschwerpunkte herauszufinden, mit anderen Agenten Kontakt aufzunehmen und Cluster zu bilden.

Der Ansatz von Yenta, diese Funktionen adäquat umzusetzen sieht wie folgt aus:

- Interessen des Benutzers ermitteln
  Da sich das System nicht auf ein bestimmtes Thema beschränkt, ist es wenig sinnvoll, die Interessen durch die Angabe von Stichwörtern durch den Benutzer zu erfassen (abgesehen davon, daß dieser nicht mit einbezogen werden sollte). Ein späterer Vergleich dieser individuellen Angaben wird nur selten zu Übereinstimmungen führen. Statt dessen verarbeitet der Agent alles in elektronischer Textform vorliegende Material, das der Benutzer lokal gespeichert hat. Dazu gehören Email, News-Texte, Textdateien im Dateisystem u.ä[41]. Für diese Dokumente werden, nach Stemming und Bildung der inversen Wortfrequenz, Dokumentenvektoren berechnet. Mit diesen ist es dann möglich, lokal Minicluster[42] zu bilden, indem das Skalarprodukt zwischen jeweils zwei Dokumentvektoren ermittelt wird und bei Überschreitung eines bestimmten Schwellwert die zugehörigen Dokumente zusammenfaßt werden[43]. Dieser Schritt wird von Foner als ‚Preclustering' bezeichnet.
- Kontaktaufnahme
  Im Gegensatz zu anderen Ansätzen verwendet Yenta eine strikt dezentralisierte *peer-to-peer* Architektur, d.h. es gibt keine zentrale Anlaufstelle, die Informationen über die Existenz und/oder den Inhalt anderer Agenten besitzt. Der Vorteil einer solchen Struktur ist, daß sie sich leichter skalieren läßt (der Aufwand eines zentralen Servers würde in Abhängigkeit von der Zahl der Agenten quadratisch steigen), das System ist weniger fehleranfällig und private Informationen, die in den Agenten gespeichert sind, sind besser vor Eindring-

[41] Foner verwendet des Begriff ‚Grain' (Korn, Samen) für diese Dokumente, da in einer späteren Version die Einbeziehung nicht-textuellen Materials vorgesehen ist

[42] Foner bezeichnet diese als ‚Granule' (Körnchen)

[43] Der tatsächliche Algorithmus ist etwas komplexer, um eine sinnvolle Zusammenfassung zu erhalten; er kann in [Fon97] nachgelesen werden; im wesentlichen basiert dieser Algorithmus auf dem *vector space model*, das z.B. in [Sal89] beschrieben ist

lingen geschützt. Auf der anderen Seite muß jedoch das Problem gelöst werden, wie ein Agent weitere Agenten finden kann. Da jeder Agent eine Liste besitzt, in der weitere ihm bekannte Agenten verzeichnet sind (ein sogenannter *rumor cache*), wird davon ausgegangen, daß es reicht, eine einmalige Bootstrapping-Phase zu durchlaufen, in der mindestens ein anderer Agent gefunden werden muß und später nur das Abarbeiten der *rumor caches* bekannter Agenten notwendig ist. Für das Bootstrapping kommen sowohl Broadcasts in lokalen Netzen in Frage, als auch die Abfrage einer zentralen Registratur, die über eine Liste bekannter Agentenadressen verfügt.

- Clusterbildung
  Cluster werden von jedem Agenten selbst verwaltet. Sie bestehen aus einander ähnlichen Minicluster verschiedener Agenten. Jeder Agent besitzt dafür einen Clustercache, in dem alle Verbindungen zwischen seinen Miniclustern und ähnlichen anderer Agenten enthalten sind. Weiterhin besitzt ein Agent einen *rumor cache*, da er Informationen über eine beschränkte Zahl anderer zuletzt kontaktierter Agenten verwaltet. Wenn ein Agent A mit einen anderen Agent B zum ersten Mal in Verbindung tritt, prüft A jeden seiner Minicluster auf Ähnlichkeit mit jedem Minicluster von B. Liegt der Wert über einem bestimmten Schwellwert, trägt der A die übereinstimmenden Minicluster in seinen Clustercache ein. Nach diesem Schritt testet A, ob Übereinstimmungen mit dem Inhalt von B's Rumor Cache bestehen. Bei positiven Ergebnisse kontaktiert A die entsprechenden Agenten anschließend in der selben Weise, wie er mit B verfahren hat[44]. B wiederum verfährt analog mit A.

Mit diesem Ansatz konnten besonders die Vorteile einer dezentralen Struktur nachgewiesen werden. Es wurden nicht ungewöhnlich viele Nachrichten zwischen den Agenten versendet, die Cluster wuchsen stetig und zu jedem Zeitpunkt war der Vernetzungsstand eines Agenten abrufbar, da es nicht notwendig ist, daß ein Agent über die Minicluster aller anderen Agenten informiert sein muß. Auch das Ausscheiden eines Agenten hat keine negativen Folgen auf die Clusterbildung. Er wird einfach als Mitglied eines Cluster gestrichen. Während eines Feldversuchs mit 1000 Yentas konnte dem Algorithmus eine gute Leistungsfähigkeit und Konvergenz in angemessener Zeit bescheinigt werden.

3.3 Beispiele für kommerzielle Softwareagenten

Der größte Bereich kommerzieller Produkte, in denen Agenten eine wesentliche Rolle spielen, sind Agentenprogrammiersprachen bzw. –umgebungen. Andere Einsatzgebiete sind Anwendungen, die mit einer Agentenfunktionalität erweitert wurden. Mangels detailierter Informationen kann man in diesen Fällen kaum entscheiden, ob daß Attribut Agent auch durch entsprechende Eigenschaften der Software gerechtfertigt ist.

[44] Diese Vorgehensweise wird auch als Entsprechung zur Mund-zu-Mund Propaganda verstanden

Die Auswahl der Beispiele erfolgte deshalb maßgeblich unter Berücksichtigung der verfügbaren Dokumentation.

3.3.1 Informationsagenten

Mit Search´97 von Verity [Ver98] sowie Fulcrum Knowledge Network [Ful98] werden zwei leistungsfähige *Information Retrieval*-Produkte angeboten, die jeweils mit Agenten für das automatische wiederholte Ausführen von Anfragen erweitert wurden. Da diese Systeme weiter unten in Kapitel 4.4 evaluiert werden, wird hier auf eine genauere Beschreibung verzichtet.

3.3.2 Agenten-Entwicklungsumgebungen / -sprachen

Die Popularität von Agententechnologie und der damit verbundene Druck auf Softwarehersteller, ihre Produkte entsprechend zu erweitern, hat zur Entwicklung einer Vielzahl von Agenten-Entwicklungswerkzeugen geführt[45]. Sie lassen sich grob in Bibliotheken (z.B. Intelligent Agent Library[46], Kafka[47]) und Entwicklungsumgebungen (z.B. JACK Intelligent Agents[48], Agentx[49]) einteilen. Die meisten Werkzeuge legen den Schwerpunkt auf die Unterstützung mobiler Agenten, d.h. Übertragung von Programmcode und die Ausführung auf anderen Rechnern verbunden mit der Fähigkeit zur Kommunikation. Weitere Eigenschaften von Agenten werden dagegen nur in geringem Maße unterstützt. Eine Ausnahme davon bildet das im folgenden beschriebene System.

AgentBuilder[50]

Der von Reticular Systems entwickelte AgentBuilder besteht aus zwei Hauptkomponenten: dem Toolkit zur Entwicklung von Agenten und dem Laufzeitsystem zur Ausführung der Agenten. Die Software ist komplett in Java geschrieben und erzeugt wiederum Java-Programme. Diese Eigenschaft teilt sie mit fast allen anderen Agenten-Entwicklungswerkzeugen. Anders als einige dieser Produkte basiert sie auf universitären Forschungsarbeiten (Shoham, Thomas [Ret99b]). Diese wird bei der Definition von Agenten, ihrem Interaktionsverhalten und der Umgebung deutlich.

Agenten werden über ein mentales Modell, welches Überzeugungen (*beliefs*), Fähigkeiten (*capabilities*), Vereinbarungen (*commitments*), Verhaltensregeln (*behavioral rules*) und Intentionen (*intention*) beinhaltet, beschrieben. Diese Konzepte haben folgende Bedeutung:

[45] Eine aktuelle Liste kommerzieller und akademischer Produkte bietet [Ret99a]
[46] http://www.bitpix.com/business/main/bitpix.htm
[47] http://www.fujitsu.co.jp/hypertext/free/kafka
[48] http://www.agent-software.com.au/jack.html
[49] http://www.iks.com
[50] http://www.agentbuilder.com/index.html

- *belief*
  Beinhaltet die Überzeugungen des Agenten über den aktuellen Zustand der internen und externen Welt und kann durch neue Informationen geändert werden. Ein Agent kann Überzeugungen über die Welt, über Überzeugungen anderer Agenten, über Interaktion mit anderen Agenten und über seine eigenen Überzeugungen haben.
- *capability*
  Verbindet Aktionen mit den notwendigen Vorbedingungen. Nur die in den *capabilities* spezifizierten Aktionen können vom Agenten ausgeführt werden. Zur Ausführung kann es nur kommen, wenn alle Vorbedingungen erfüllt sind.
- *commitment*
  Ein *commitment* stellt eine Vereinbarung dar, eine bestimmte Aktion zu einem bestimmten Zeitpunkt auszuführen. Normalerweise wird ein *c.* von einem Agenten zu einem anderen kommuniziert. Wenn dieser aufgrund seiner Fähigkeiten in der Lage ist, die Aktion auszuführen bestätigt er dem anfragenden Agenten den Auftrag. Trotzdem muß die Aktion nicht unbedingt ausgeführt werden. Dies kann eintreten, wenn der Systemzustand die Aktion verhindert (z.B. keine Verbindung zur Datenbank).
- *behavioral rules*
  Über die Verhaltensregeln wird festgelegt, wie ein Agent beim Eintreten einer bestimmten Situation reagiert. Die Regeln haben die Form von *when-if-then*-Ausdrücken. Im *when*-Teil werden die Ereignisse aufgezählt, auf die reagiert werden soll (z.B. Antworten anderer Agenten). Falls diese eintreffen werden Bedingungen aus dem *if*-Teil mit dem aktuellen Zustand des Agenten verglichen. Sind alle Bedingungen erfüllt, werden die Aktionen des *then*-Teils ausgeführt.
- *intention*
  Mit Intentionen kann ein Agent mit einem anderen vereinbaren, daß dieser einen bestimmten Weltzustand zu einer bestimmten Zeit herbeiführt. Dieses ist mit dem *commitment* vergleichbar. Jedoch können zum Erreichen des intendierten Zustands mehrere Aktionen notwendig sein. Welche das sind wird nicht vorgegeben. Der beauftragte Agent muß selbst die notwendigen Aktionen aussuchen. Dazu benötigt er Fähigkeiten zur Planung.

Zur Kommunikation zwischen Agenten wird KQML eingesetzt (siehe auch 3.1.1), da es eine gewisse Verbreitung erfahren hat, flexibel gegenüber den zu übertragenen Inhalten ist und von vornherein Erweiterungen des Sprachumfangs zuläßt.

Die Programmierung von Agenten und ihrer Umgebung kann komplett in einer grafischen Umgebung auf einem hohen Abstraktionsniveau erfolgen. Die Werkzeuge setzen die Eingaben in eine objekt-orientierte Sprache um (RADL - *Reticular Agent Definition Language*), die bei Bedarf auch von Hand editiert werden kann.

Der Entwicklungszyklus einer Agentenanwendung umfaßt die folgenden Schritte:

- Problembereich analysieren
  In diesem Schritt können über objekt-orientierte Modellierungswerkzeuge die Objekte des Problembereichs und ihre Beziehungen untereinander abgebildet werden. Als Ergebnis erhält man ein Weltmodell, dessen Methoden von den Agenten genutzt werden können sowie eine formale Definition in Form einer Ontologie, die für die Kommunikation der Agenten untereinander eingesetzt werden kann.
- Architektur der Agentengemeinschaft (*agency*)
  Hier wird die Problemstellung in einzelne Aufgaben zerlegt und diese als eine Rolle den einzelnen Agenten zugewiesen. Daraufhin wird für jeden Agenten ein Gerüst erzeugt und die Interaktion mit anderen Agenten sowie die verwendeten Protokolle festgelegt.
- Verhalten der Agenten festlegen
  Die Programmierung des Verhaltens eines Agenten stellt die eigentliche Entwicklung des Agenten dar. Hier müssen die Verhaltensregeln, Fähigkeiten und die anfänglichen Zustände der Überzeugungen, Vereinbarungen und Intentionen bestimmt werden. Dabei werden eine Benutzerschnittstellen-Bibliothek, eine Agentenaktions-Bibliothek sowie eine Agentendefinitionsdatei erzeugt, die zusammen in einer Projektklasse (PAC - *project accessory class*) abgelegt werden.
- Agent ausführen und debuggen
  Während der Ausführung des Agenten bietet AgentBuilder die Möglichkeit, die Anwendung zu debuggen, d.h. der Zustand eines Agenten und seine Operationen können überwacht werden.

Wenn eine Anwendung als RADL-Datei vorliegt, kann die in der Laufzeitumgebung ausgeführt werden. Diese stellt einen Interpreter für jeden Agenten bereit, der die eintreffenden Nachrichten überwacht und Aktionen entsprechend des Agentenzustands auslöst. Dieser Ausführungsprozeß ist in Abbildung 5 dargestellt.

Da AgentBuilder ein noch junges Produkt ist, steht noch nicht der komplette Funktionsumfang bereit. Speziell die Enterprise-Version, die den Umgang mit verteilten Objekten über CORBA bzw. DCOM (*distributed common object model*) ermöglichen soll und das Entwickeln mobiler Agenten unterstützt, ist noch nicht verfügbar. Jedoch bietet die jetzige Version alle notwendigen Werkzeuge für die Erstellung einer Multiagentenumgebung und bietet eine leistungsfähige hybride Agentenarchitektur (siehe Kapitel 2.2.3).

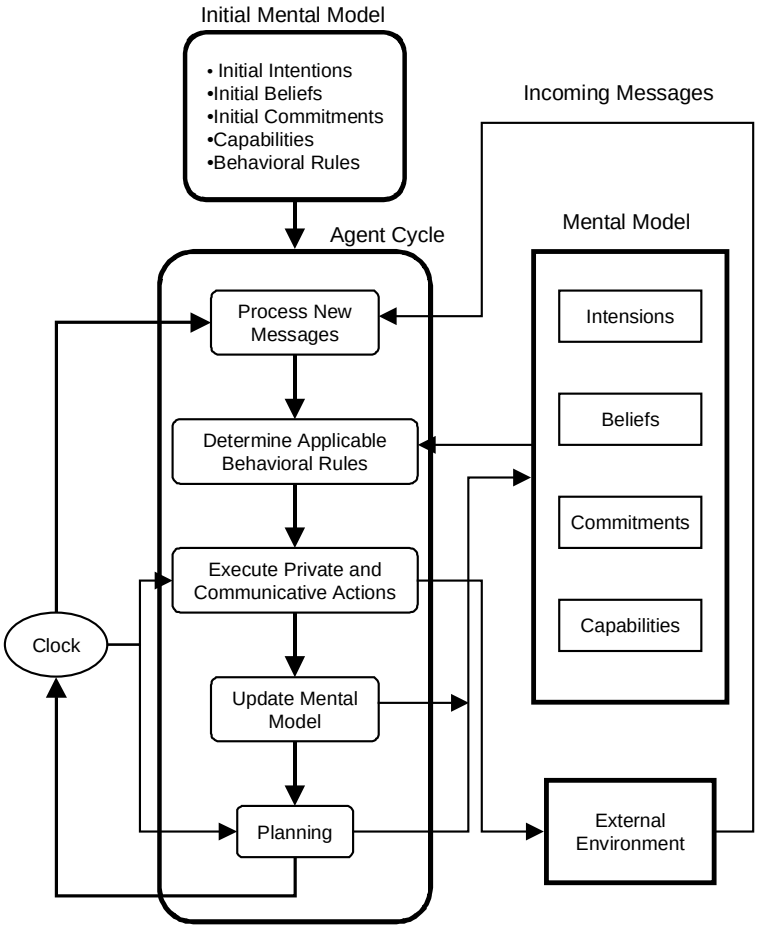

Abbildung 5: AgentBuilder – Ausführen eines Agenten

3.4 Zusammenfassung

Nach der ersten Euphorie über Agententechnologien Anfang der 90er Jahre hat sich inzwischen die Erkenntnis durchgesetzt, daß dem breiten Einsatz kooperativer und mobiler Agenten der Mangel an allgemein akzeptierten Standards auf diesem Gebiet entgegensteht. Der erste Ansatz in dieser Richtung kam von der KSE, die eine vollwertige ACL spezifizierten, von der vor allem KQML in Projekten implementiert und als Grundlage für nachfolgende Standardisierungen verwendet wurde. So basierte unter anderem der erste Versuch der OMG, Agenten in ihre Objektarchitektur zu integrieren, auf diesem Protokoll- und Nachrichtenformat.

Die geringe Anwendung dieser ersten Spezifikationen offenbart jedoch bestimmte Schwächen. Entweder behandeln sie nur einen kleinen Ausschnitt, wie z.B. ein Austauschformat, das für eine echte Kooperation nicht ausreicht, oder die Definition der Agentenumgebung ist zu restriktiv und auf einen begrenzten Aufgabenbereich zugeschnitten. Die neueren Arbeiten von FIPA und OMG adressieren diese Probleme, indem sie zum einen alle relevanten Komponenten und Schnittstellen beschreiben, zum anderen aber bei inhaltliche Details den Entwicklern freie Hand las-

sen. Es zeichnet sich ab, daß OMG und FIPA sich weiter einander annähern werden und auch Unterstützung von anderen Organisationen (z.B. Agent Society, W3C) erhalten.

4 Ein Agentensystem für das *Information Management*

Im zweiten Teil meiner Arbeit geht es um die praktische Anwendung und Umsetzung der in den Kapiteln davor gewonnenen Erkenntnisse. Im Mittelpunkt steht dabei die Entwicklung eines Agentensystems zur Lösung einer Problemstellung aus dem Bereich des *Information Management*.

In diesem Kapitel werde ich die der Aufgabenstellung zugrundeliegenden Projekte vorstellen, die sich daraus ergebenden Anforderungen ableiten, den Einsatz existierender Software diskutieren und schließlich eine Konzeption für ein entsprechendes System präsentieren.

4.1 Motivation

Das Fraunhofer Institut für Arbeitswirtschaft und Organisation (IAO)[51], an dem diese Diplomarbeit entstand, beschäftigt sich unter anderem mit der elektronischen Erschließung von Märkten, die bisher vorwiegend mit papierbasiertem Workflow operierten. Dazu gehören Produktpräsentation und –vermarktung, Vermittlungsagenturen und Ausschreibungsdienste.

Das dafür angebotene Dienstleistungsspektrum reicht von der Beratung zu Netzwerkfragen, Projektmanagement, Electronic Commerce und Softwareerstellungsprozessen, über die Konzeption mit Anforderungsspezifikationen, Entwurf und Wahl der Softwaretechnologie, bis zur Realisierung des Projektes als Prototyp oder betriebsfertiges System.

Eine wichtige Komponente vieler Projekte bildet die integrierte Suchfunktion, über die der Benutzer zielgerichtete Suchanfragen nach Informationen über einfache Suchmasken, Formulare oder die Verwendung einer proprietären Anfragesprache stellen kann (letzteres, obwohl die flexibelste Methode, bleibt, wegen der erforderlichen Einarbeitung, Experten vorbehalten).

Die Schwierigkeiten und Probleme, die sich mit dieser zum Bereich *Information Retrieval* gehörenden Aufgabenstellung verbinden sind vielfach diskutiert worden. In den als nächstes vorgestellten aktuellen Projekten kommt zu den bekannten Anforderungen an eine Suchkomponente noch ein zeitlicher Aspekt, da hier das Interesse auf einer wiederholten Suche mit gleichen oder ähnlichen Parametern liegt. Wie ich in Kapitel 4.3 zeigen werde, gibt es einen eleganten Weg zur Lösung dieser erweiterten Suchproblematik durch den Einsatz von Agententech-

[51] Homepage http://www.iao.fhg.de

nologie. Im folgenden werden aber zuerst die Projekte vorgestellt, die mit zur Entwicklung des später beschriebenen Agentensystems führten.

4.1.1 Handwerks-Centrum

„Handwerks-Centrum, DER elektronische Marktplatz für Handwerk und Mittelstand“ wurde vom Fraunhofer IAO im Auftrag der Deutschen Telekom Berkom GmbH entwickelt. Ausgangspunkt war die Erkenntnis, daß Electronic Commerce, d.h. die Anbahnung und Abwicklung von Geschäften auf elektronischen Marktplätzen, zunehmen wird. Besonders das bundesdeutsche Handwerk bildet mit seinen rund 560.000 Betrieben und der anstehenden Einführung neuer Informations- und Kommunikationstechnologien in Zukunft einen der größten Märkte.

Dem Benutzer bietet sich ein umfangreiches System, welches auf die Anforderungen eines Handwerksbetriebes zugeschnitten ist. Er kann Leistungen anbieten, Leistungen in Anspruch nehmen und Geschäftsprozesse abwickeln. Das „Handwerks-Centrum“ ist eine integrierte Plattform, welche von geschlossenen und auch von offenen Benutzergruppen verwendet werden kann. Die anvisierten Leistungsanbieter sind in Abbildung 6 dargestellt.

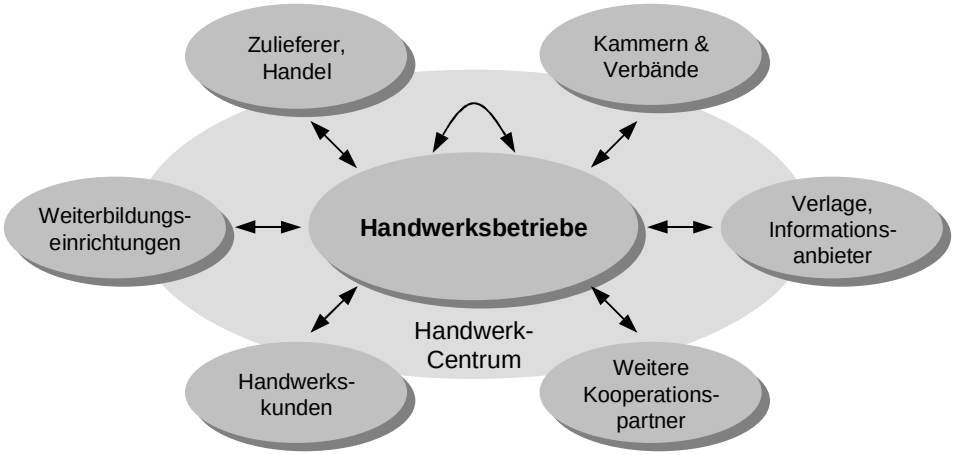

Abbildung 6: Handwerk-Centrum – Marktplatzstruktur (mit leichten Modifikationen aus der Projektbeschreibung übenommen)

Neben der Erschließung neuer Vermarktungskanäle hat dieses System zum Ziel:

- die Senkung wesentlicher Prozeßkosten,
- Einführung und Förderung neuer Arten der Zusammenarbeit zwischen den Benutzern,
- eine schnelle und effiziente Informationsbeschaffung bzw. Informationsversorgung.

Um diesem Anspruch gerecht zu werden, müssen abgesehen von Funktionen wie persönliche Präsentation, Zugriff auf Produktkataloge,

elektronische Angebotseinholung und Bestellung, Ausschreibungsdienste, Adressverzeicnnisse vor allem leistungsfähige Suchmechanismen angeboten werden. Speziell die Suche bzgl. bestimmter Produktgruppen und Ausschreibungsgebiete muß oft und regelmäßig wiederholt werden, um auf neue Einträge schnell reagieren zu können. Eine Automatisierung, die eine Art änderungsabhängiges Abonnement darstellt, kann dem Benutzer die Arbeit abnehmen und dadurch Zeit und Kosten einsparen. Das später vorgestellte Agentensystem soll diese Aufgabe übernehmen.

4.1.2 ELPRO

Im Auftrag der Europäischen Kommission und in Zusammenarbeit mit Partnern aus ganz Europa entwickelt das IAO in Verbindung mit dem Institut für Arbeitswissenschaft und Technologiemanagement der Universität Stuttgart einen Prototypen für ein elektronisches Ausschreibungssystem, genannt ELPRO (**El**ectronic Public **Pro**curement System for Europe). Es bietet eine Plattform, auf der Ausschreibungen im europäischen Maßstab veröffentlicht und verarbeitet werden.

Ziel ist es, einen benutzerfreundlichen Online-Dienst zu entwickeln, der Beschaffer und Anbieter aus Behörden und Firmen jeder Größe durch alle Phasen des Ausschreibungsprozesses begleitet. Dabei sind die nationalen Besonderheiten, der multilinguale Charakter des Systems und die notwendigen Sicherheitsmechanismen zum Schutz der Daten zu beachten.

Der elektronisch unterstützte Prozeß umfaßt den gesamten Ausschreibungszyklus:

auf Seiten des Ausschreibenden	auf Seiten des Leistungsanbieters
• Bekanntmachung der Vorinformation	• Suche nach Vorankündigungen
• Bekanntmachung der Ausschreibung	• Suche nach Ausschreibungen
• Erstellung, Spezifikation und Versand der Unterlagen	• Anforderung der Unterlagen
• Beantwortung von Zusatzanfragen	• Anfragen von Zusatzinformationen
• Verwaltung der eingehenden Angebote und Selektion der Anbieter	• Erstellung und Versand des Angebots
• Benachrichtigung der Gewinner	• Erhalt der Auftragsbestätigung

Theoretisch läßt sich der gesamte Prozeß mittels eines Multiagentensystems modellieren, indem den einzelnen Schritten spezielle Agenten zugeordnet werden, die über Vermittler ihre Daten austauschen, auf neue oder geänderte Ausschreibungen reagieren und selbst Angebote unter

Berücksichtigung bestimmter Rahmenbedingungen erstellen können. Der im vorangegangenen Kapitel beschriebene Stand der Entwicklung von Agentensystemen und die wirtschaftliche Bedeutung von Entscheidungen zu Ausschreibungen lassen jedoch eine solche enge Integration und Verantwortung der Agenten noch nicht zu.

In der Spezifikation des Prototyps ist aber die Bereitstellung von Softwareagenten für eine schnelle, individuelle und automatische Benachrichtigung und Verteilung von Informationen vorgesehen. Wie im Projekt „Handwerk-Centrum“ geht es auch hier unter anderem um die kontinuierliche Abfrage des Datenbestandes nach einem einmalig definierten Suchmuster. Das zu entwickelnde Agentensystem soll auch in diesem Projekt eingesetzt werden.

4.2 Anforderungen

Die in 4.1.1 und 4.1.2 vorgestellten Projekte stehen stellvertretend für eine größere Anzahl von Anwendungen mit vergleichbaren Funktionsblöcken für die erweiterte Suche. In diesem Abschnitt sollen die Gemeinsamkeiten herausgearbeitet und daraus, sowie allgemein, Anforderungen an ein derartiges Suchmodul – im weiteren als Komponente bezeichnet – abgeleitet werden, das sich in die entsprechenden Programme integrieren läßt.

Die für die zu entwickelnde Komponente ins Auge gefaßten Zielanwendungen bestehen aus einer mehrschichtigen Architektur (z.B. „*Three Tier*-Modell[52]“). Basierend auf einer Datenschicht (Datenbanken, Fileserver u.ä.) arbeitet die eigentliche Anwendungslogik in einer oder mehreren mittleren Schichten und die oberste Schicht wird durch das grafische Front-end (Benutzerschnittstelle) gebildet.

Der Anknüpfungspunkt der Komponente an die Anwendung sollte in den mittleren Schichten liegen, da zum einen die Integration so einfach wie möglich, d.h. ohne großen programmtechnischen Aufwand, erfolgen soll (die Komponente enthält alle benötigten Funktionen zur Manipulation und Verwaltung – keine speziellen zusätzlichen Routinen innerhalb der Anwendung), zum anderen bedeutet eine nahtlose Integration aber auch, daß die Funktionen innerhalb der Anwendungsoberfläche erscheinen, d.h. das Front-end muß entsprechend angepaßt werden (siehe Abbildung 7 auf Seite 56).

Eine Folgerung aus der vorangegangenen Entscheidung des minimalen Aufwandes seitens der Anwendung ist der direkte Zugriff auf die zu durchsuchenden Daten mit Routinen der Komponente. Dafür ist es jedoch sinnvoll, von der Anwendung genutzte externe Software der unteren bzw. auch mittleren Schicht ebenfalls einzubeziehen.

[52] Siehe Schichtenarchitektur in [Bal96, S. 640 ff.]

Von den Anwendungen werden bzgl. des *Information Retrieval* neben der reinen Stichwortsuche Funktionen zur Kombination von Suchbegriffen, Beachtung der Besonderheiten von Datenstrukturen bei der Suchraumbeschränkung und Möglichkeiten für Rankingoperationen benötigt. Anspruchslosere Programme sollten aber auch über einfache Aufrufe, welche die dahinterliegende Komplexität verbergen, die einzelnen Aktionen anstoßen können – eine Aufgabe für das Schnittstellendesign.

Die Funktion, durch die sich diese Anwendungen qualitativ von anderen Programmen mit Suchmöglichkeiten unterscheiden, ist jedoch die wiederholte automatisierte Ausführung von Suchausdrücken, die in einem Profil vorgegebenen sind. Die Notwendigkeit dafür liegt in den teilweise hochgradig dynamischen Daten, mit denen die Anwendungen operieren, d.h. die Datenbasis für die Suche ändert sich kontinuierlich und der Anwender ist an der schnellen Benachrichtigung über bestimmte Änderungen interessiert.

Da sich die Realisierung dieser Funktion in allen Anwendungen ähnelt und viele davon als Web-Programme implementiert sind, die nur auf Anfragen reagieren können, selbst aber nicht aktiv werden, bietet sich die Auslagerung in eine eigene Komponente an. Sie muß die Speicherung und nach individuellen Vorgaben gesteuerte Ausführung der einzelnen Anfragen übernehmen. Um eine eindeutige Zuordnung von Anfragen und Ergebnissen zu erhalten, kann auf die Benutzerverwaltung der Anwendungen zurückgegriffen werden.

In der folgenden Liste führe ich allgemeine Anforderungen an die Komponente auf sowie davon abgeleitete Folgerungen.

- Die Komponente muß sich in verschiedene, vorher nicht bekannte, Anwendungen integrieren lassen.

  ⇒ Es müssen verschiedene Betriebsysteme und Rechnerarchitekturen unterstützt werden, d.h. die Komponente sollte ohne spezielle Anpassungen oder extra Compilieren auf allen wichtigen Plattformen laufen.

  ⇒ Sie muß skalierbar sein, da die Belastung durch die Anzahl der Benutzer und das Datenaufkommen ebenfalls unbekannt sind.

    → Der Ausführungsort sollte unabhängig von der integrierenden Anwendung sein, d.h. die Komponente kann auf einen dedizierten Rechner mit entsprechender Leistungsfähigkeit ausgelagert werden (netzwerkweiter Zugriff).

    → Die Funktionseinheiten der Komponente können im Netzwerk verteilt werden, d.h. die Komponente sollte eine modulare Architektur aufweisen und die Kommunikation zwischen den Modulen kann über das Netzwerk erfolgen.

→ Mehrere Komponenten können parallel auf verschiedenen Rechnern eingesetzt werden, d.h. sowohl die Anbindung mehrerer unabhängiger Komponenten an eine Anwendung als auch die von einer Komponente verteilte Ausführung auf mehreren Komponenten sollte möglich sein.

- Die Komponente basiert auf einem generischen Konzept mit leichter Integrationsmöglichkeit in bestehende Umgebungen.

  ⇒ Sie muß abgeschlossenen sein, d.h. sie ist für sich lauffähig (entspricht dem *Componentware*-Ansatz).

  ⇒ Sie besitzt normierte, einfach zu implementierende Schnittstellen für Funktionsaufruf und Parameterübergabe, d.h. die Daten aus dem Suchformular bzw. die Benutzerangaben vom Front-end können möglichst direkt übergeben werden und die Komponentenverwaltung kann über wenige Funktionen bzw. extern erfolgen.

  ⇒ Es werden verschiedene Arten von Schnittstellen zur Verfügung gestellt, die speziell für die Kopplung mit der jeweiligen Anwendung optimiert sind (z.B. lokal über Bibliothek mit Funktionsaufrufen oder entfernter Aufruf über Socketschnittstelle).

  ⇒ Verwendung von standardisierten Protokollen, die von verschiedenen Systemen und Programmbibliotheken unterstützt werden. Dadurch läßt sich die Implementierungszeit verkürzen, da Aufgaben wie Empfangen und Parsen der Informationen von vorgefertigten Modulen übernommen werden können.

  ⇒ Vorhandene Datenverwaltungssysteme sollten einbezogen werden (z.B. DBMS). Ziel ist dabei die bestmögliche Ausnutzung von Funktionen externer Software bei gleichzeitiger Wahrung der Unabhängigkeit von bestimmten Softwareprodukten

- Die Komponente muß die Suchfunktionalität bereitstellen.

  ⇒ Es werden verschiedene Datenquellen unterstützt, d.h. es kann über mehreren Datenbeständen mittels einer Anfrage gleichzeitig gesucht werden, dabei kann die Art der Daten unterschiedlich sein.

  ⇒ Es müssen ebenso verschiedene Datentypen[53] unterstützt werden, d.h. die Art und Weise des Zugriffs auf die Daten, speziell die Protokolle, sowie Strukturen in denen sie vorliegen sind unterschiedlich (z.B. Datenbank, Internetseiten über HTTP, Email, Dateisystem usw.).

[53] Datentypen sind nicht identisch mit Protokollen, Art der Speicherung oder dem Format von Daten; sie sind vielmehr eine Klassifikation von Zugriffsmethoden auf verschiedene Datenquellen

⇒ Die Komponente unterstützt verschiedene Datenformate, d.h. die Daten können innerhalb eines bestimmten Datentyps in verschiedenen Formaten auftreten (z.B. ANSI-Text-, Word-, PDF-, HTML-Dokument usw.).

⇒ Die Einbindung neuer Datenquellen zur Laufzeit ist möglich, d.h. neue Datenbestände können durch einen Befehl an die Komponente hinzugefügt oder entfernt werden, wodurch das System verfügbar bleibt und gleichzeitig auf Änderungen der Umgebung reagieren kann.

⇒ Eine Erweiterung auf neue Datentypen und –formate kann möglichst ohne Änderungen an der Komponente erfolgen. Da sich im Laufe der Nutzung der Komponente herausstellen kann, daß zu durchsuchende Daten in einem neuen Format vorliegen, welches von ihr noch nicht unterstützt wird und der Quelltext der Komponente normalerweise nicht verfügbar ist, sollte sich die Komponente in dieser Richtung erweitern lassen ohne das eine Neucompilierung notwendig wird

⇒ Suchparameter sollten über boolesche Operatoren miteinander verknüpft werden können (z.B. ‚AND', ‚OR').

⇒ Um die verschiedenen hierarchischen Strukturen von Datentypen im Suchausdruck berücksichtigen zu können, muß ein einheitliches Repräsentationsmodell entwickelt werden (z.B. Datenbanken: Datenbank ➔ Tabelle ➔ Datensatz ➔ Attribut; Dateisystem: Datenträger ➔ Verzeichnisname ➔ Dateiname).

- Anfragen müssen automatisiert ausgeführt werden.

  ⇒ Die Komponente soll eine individuelle (anwenderbezogene) Speicherung der Anfragedefinition erlauben, d.h. Anfragen müssen Kategorien – meistens einzelnen Benutzern – zugeordnet werden, um die Zuordnung der Ergebnisse zu ermöglichen und Beschränkungen bzw. Berechtigungen bei der Ausführung berücksichtigen zu können.

  ⇒ Die Ausführungshäufigkeit gespeicherter Anfragen kann vorgegeben werden. Da Anforderungen an die Aktualität (Reaktionszeit bei Änderung der Daten) unterschiedlich sind und die einzelnen Datenquellen verschiedene Updatezyklen haben, muß das Ausführungsintervall einer Anfrage durch bestimmte Kriterien angegeben werden können (z.B. Zeitangaben, minimale Updatezyklen).

  ⇒ Es sollten mehrere Varianten der Ergebniszusammenstellung und –übermittlung angeboten werden, d.h. für die einzelnen Benutzer (oder Kategorien) kann festgelegt werden, wann über neue Ergebnisse informiert wird (z.B. nach x neuen Ergebnissen,

einmal am Tag) und auf welche Weise (z.B. Email, persönliche Newsseite im Internet).

- Die Datensicherheit muß gewährleistet werden.

  ⇒ Der Zugriff auf die Komponente wird nur autorisierten Programmen erlaubt. Falls die Aufrufe über das Intranet geleitet werden, muß die Herkunft und Berechtigung überprüft werden.

  ⇒ Direktzugriffe eines Benutzers werden nur nach Autorisierung und nur auf eigene Daten gestatten, d.h. ein Benutzer darf keine Möglichkeit haben, Anfragen und Ergebnisse anderer einzusehen oder zu manipulieren.

- Die Komponente kann einfach verwaltet werden. Es sollte sowohl eine Schnittstelle vorhanden sein, über die die Anwendung die Funktionen zur Verwaltung der Komponente erreicht, wie auch ein allgemeines Verwaltungswerkzeug, damit in einem ersten Integrationsschritt diese Aufgabe nicht von der Anwendung realisiert werden muß (so wenig wie möglich Änderungen an der Anwendung).

Das hier vorgestellte Anforderungspaket stellt die Rahmenbedingungen für die zu entwickelnde Komponente. In den folgenden Kapiteln geht es um die Wahl einer geeigneten Technologie zur Realisierung.

4.3 Bestimmung des Einsatzgebietes: *News Watcher Agent*

Viele der in 4.2 genannten Anforderungen werden von existierenden Suchmaschinen erfüllt. Sie sind in der Lage, Dokumente über verschiedene Protokolle zu erreichen, diverse Formate zu lesen, alles in einem Index aufzubereiten und suchbar zu machen. Dieses ist jedoch nur die Basis der beschriebenen Komponente.

Aufbauend auf der Suchfunktionalität arbeitet der Teil, welcher der Suche eine höhere Qualität verleiht, indem er das zeitversetzte und wiederholte Ausführen von Anfragen ermöglicht. Mit den Eigenschaften der Benutzerbezogenheit von Vorgaben, der Ausführung von Aktionen im Interesse eines Benutzers, der Reaktion auf externe Ereignisse (Datenänderung) und der über einen längeren Zeitraum fortdauernden Aktivität bietet er viele wesentliche Merkmale eines Agenten.

Ein Vergleich mit der in 2.2.4 aufgestellten Klassifikation zeigt, daß die geforderte Funktionalität ziemlich genau mit dem in der Rubrik Information Management beschriebenen News-Watcher-Agenten (Seite 19) übereinstimmt. Die dort aufgeführte Abonnementfunktion einer Nachrichtenquelle mit Filterung der Information entsprechend persönlicher Vorgaben läßt sich äquivalent auf die Spezifikation eines Suchausdrucks (Filterung), der auf eine Datenquelle zyklisch angewendet wird (Abonnement), abbilden.

Nachdem gezeigt wurde, daß die zentrale Funktionalität der zu entwikkelnden Komponente ‚klassische' Agentenmerkmale aufweist und es sich dabei prinzipiell um eine Art von News-Watcher-Agenten handelt, werden im folgenden zuerst die Aufgaben an diese Agenten zusammengefaßt, gefolgt von einer Untersuchung, welche agentenspezifischen Techniken dabei verwendet werden können.

4.3.1 Aufgabenschwerpunkte an die Agenten

Die Agenten operieren auf einer von der konkreten Anwendungslogik abstrahierenden Schicht zwischen der Schnittstelle zur Anwendung auf der einen Seite und der Suchfunktionalität auf der anderen. Abbildung 7 zeigt die Einbettung der Agentenfunktionalität sowie die Integration der gesamten Komponente in die Anwendung.

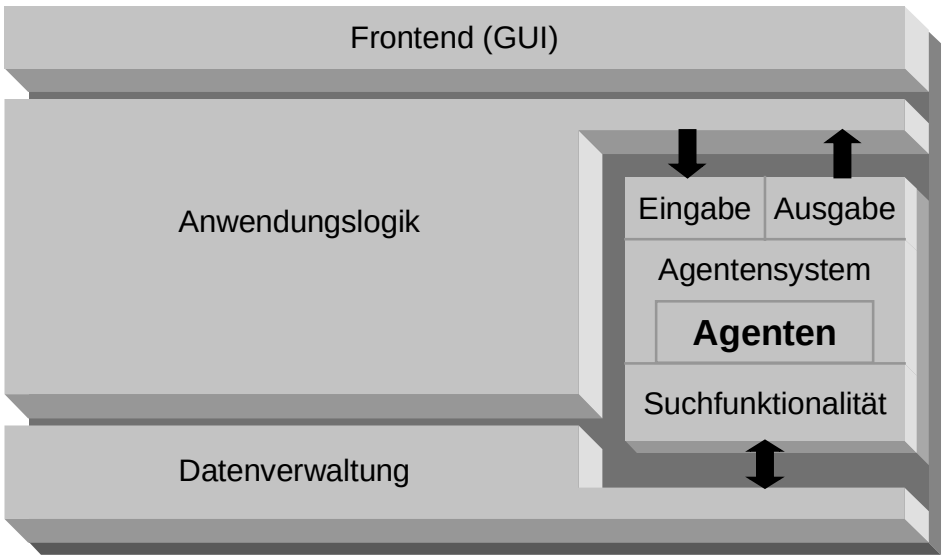

Abbildung 7: Einbettung der Agentenkomponente in eine Anwendung

Der Zuständigkeitsbereich der Agenten erstreckt sich von der Übernahme der Parameter über deren Aufbereitung, das Auslösen entsprechender Aktionen bis zur Zusammenstellung der Ergebnisse und deren Versendung oder Benachrichtigung der Anwendung.

Das Agentensystem muß für jeden Suchausdruck einen neuen Agenten erzeugen. Dieser erhält in seinem Profil die Parameter des Suchauftrags sowie weitere Vorgaben des und Angaben zum Benutzer, wie Aktivitätsniveau (gibt an, wie oft der Agent einen Suchvorgang über neue Daten durchführt), Art der Ergebnisaufbereitung, Art der Benachrichtigung, Zugriffsberechtigungen des Benutzers.

Anhand der Vorgaben beginnt der Agent mit der Beobachtung der betreffenden Datenquellen. Dafür sind verschiedene Modelle denkbar. Er kann sich für bestimmte Datenquellen registrieren und wird über jede Datenänderung informiert, worauf er einen Suchdurchlauf startet oder er besitzt einen Schedulingmechanismus, der eine Suche in einstellbaren

Intervallen durchführt, eventuell dynamisch an die Updatefrequenz der Datenquelle und die Auslastung des Systems angepaßt.

Der Agent muß einen Zwischenspeicher besitzen, um Ergebnisse zusammenzuführen, die im Laufe der Überwachung eintreffen, diese nach Rankingwerten zu sortieren und gegebenenfalls mit weiterer Information anzureichern (z.B. Hinzufügen einer Inhaltszusammenfassung des gefundenen Dokuments). Entsprechend seines Profils generiert er daraus eine Zusammenfassung in einem Ausgabeformat (z.B. Email, HTML-Seite) oder benachrichtigt die Anwendung, die daraufhin die Resultate über eine Schnittstelle abfragen kann.

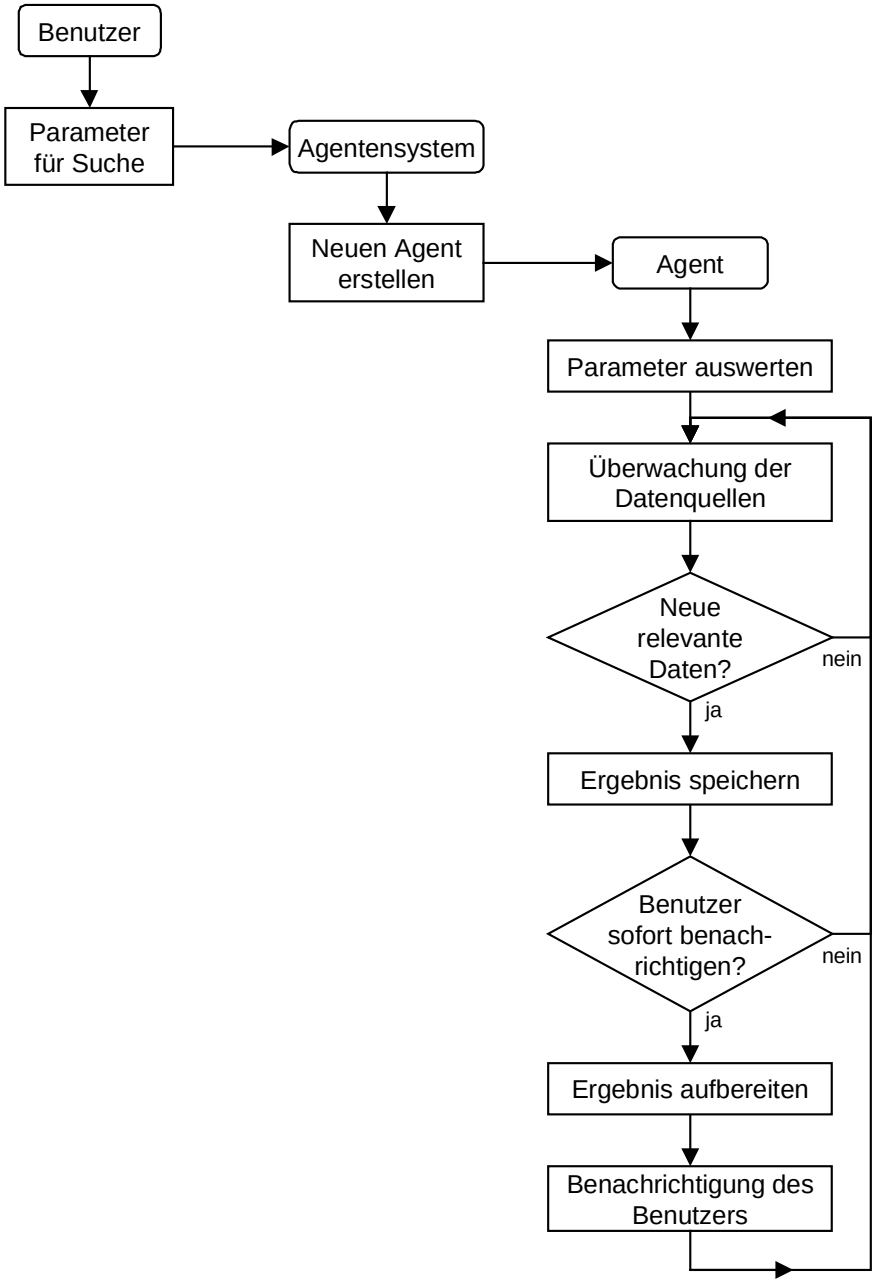

Abbildung 8: Allgemeiner Arbeitsablauf eines News-Watcher-Agenten

Neben den Agenten, die für die in Abbildung 8 beschriebenen Aufgaben zuständig sind, wird für die Verwaltung der Agenten sowie die Überwachung ihres Lebenszyklus ein Agentensystem benötigt. Dieses kann auch Dienste bereitstellen, über die die Agenten mit ihrer Umwelt kommunizieren können (z.B. Vermittler beim Zugriff auf Datenquellen).

4.3.2 Diskussion anwendbarer Eigenschaften von Agenten

In den vorangegangenen Kapiteln wurden zentrale Funktionen der zu entwickelnden Komponente als charakteristisch für Agenten eingestuft, der genaue Agententyp und seine Aufgaben herausgearbeitet. Ziel dieses Abschnitts ist es, zu untersuchen, welche der grundlegenden Eigenschaften von Agenten (siehe Kapitel 2.2.2), für unseren Aufgabenbereich interessant sind.

- Autonomie
  Eine der wesentlichen Eigenschaften der hier einzusetzenden Agenten ist es, daß sie nach einmaliger Übergabe der Parameter ohne eine weitere Interaktion mit dem Benutzer ihr Ziel, d.h. Auffinden relevanter Informationen, verfolgen. Gerade darin liegt die qualitative Verbesserung gegenüber bisherigen Suchfunktionen. Der Benutzer bemerkt die Aktivität des Agenten erst durch die Zustellung von oder Benachrichtigung über Ergebnisse. Autonomie bedeutet aber nicht, daß der Benutzer nicht mehr auf den Agenten Einfluß nehmen kann. Anpassungen der Vorgaben oder temporäres Anhalten müssen jederzeit möglich sein.
- Mobilität
  Im Rahmen eines News-Watcher-Agenten kann Mobilität sinnvoll sein, wenn es um die Überwachung nicht lokaler, weit entfernter Datenquellen geht. In diesem Fall kann der Agent, ausgerüstet mit dem Suchprofil und weiteren Vorgaben, sich zum Rechner der Datenquelle begeben, dort lokal die Anfragen stellen, die Daten entsprechend der Vorgaben filtern und nur die Resultate zu seinem Ausgangsort zurückschicken. Damit können die Netzbelastung und Reaktionszeit verringert werden. Dieses ist jedoch nur bei WAN[54]-Verbindungen empfehlenswert, bei denen aus Kosten- , Zeit-, oder Sicherheitsgründen[55] das Datenaufkommen gering gehalten werden soll. Hinderlich an diesem Konzept ist jedoch die Notwendigkeit einer Agentenplattform auf dem Zielrechner mit der Datenquelle, die dem Agenten die Übertragung und Ausführung erlaubt. Solange noch keine allgemein akzeptierten Standards dafür existieren (vgl. Kapitel 3.1.2 und 3.1.4), kann dieses Merkmal von Agenten in einem allgemeinen Ansatz noch nicht verwirklicht werden.

[54] Wide Area Network – Bezeichnung für Netzwerkstrukturen über große Entfernungen mit Teilnetzen verschiedener Anbieter

[55] Müssen alle Daten zwischen Agent und Datenquelle über das Netzwerk ausgetauscht werden, können sie unkontrolliert von dritten abgehört oder sogar verändert werden, soweit nicht kryptographisch sichere Protokolle verwendet werden (z.B. SSL – *secure socket layer*)

- Kommunikation
  Als eingebettete Komponente muß das Agentensystem Fähigkeiten zur Kommunikation mit der übergeordneten Anwendung (und damit indirekt mit dem Benutzer) sowie mit den zu observierenden Datenquellen besitzen. Dabei können aber einfache Schnittstellen und standardisierte Protokolle verwendet werden, da es sich hierbei um eine feste Menge von Befehlen und entsprechender Antworten handelt. Ein für Entscheidungsprozesse oder Zielerreichung notwendiger Austausch zwischen Agenten wird bei diesem Aufgabengebiet nicht gebraucht (siehe dazu auch den nächsten Punkt).
- Kooperation
  Da die von den Agenten zu realisierende Funktion nicht übermäßig viele Teilaufgaben besitzt, ist es nicht zweckmäßig, diese auf mehrere Agenten aufzuteilen (z.B. Kommunikationsagent zur Anwendung ➔ Suchagent ➔ Informationsagent). Somit besitzt jeder Agent die gleiche Funktionalität und kann die ihm übertragene Aufgabe alleine bewältigen. Es wäre zwar denkbar, daß sich Agenten mit ähnlichen Vorgaben über ihre Resultate austauschen und damit den Aufwand zum Erreichen des Ziels minimieren, jedoch dürfte der Erwartungswert für die Übereinstimmung der Parameter von verschiedenen Benutzern gering sein und damit die Kosten der Zusammenarbeit nicht rechtfertigen. Ein Anwendungsfall für eine konzertierte Aktion von Agenten zum Erreichen eines Ziels wäre gegeben, wenn sich die Suche über mehrere Datenquellen erstreckt und die Agenten zum Rechner mit den Daten übertragen werden. Aufgrund der Aussagen zur Mobilität (siehe Seite 58) wird dieser Fall in dem aktuellen System keine Rolle spielen. Zusammenfassend kann gesagt werden, daß die hier benötigten Agenten über keine kooperativen Eigenschaften verfügen müssen.
- Reaktivität
  Die Umwelt, die die Agenten beobachten, besteht aus den verschiedenen Datenquellen. Die Sensoren sind demnach die Schnittstellen, über die sie von Änderungen der Daten informiert werden. Entsprechend dieser Informationen müssen sie reagieren, d.h. Anfragen ausführen.
- Proaktivität
  Normalerweise wird unter Proaktivität die Ergreifung der Initiative durch den Agenten aufgrund bestimmter Schlußfolgerungen über seinem internen Weltmodell verstanden. Eine sehr schwache Form der Proaktivität ist in unserer Anwendung gegeben, wenn der Agent in zeitlichen Intervallen aktiv wird, die er durch seine Vorgaben und dem Zustand des Systems festlegt. Wird dieser Schedulingmechanismus jedoch als unabhängiges Modul betrachtet, sind die davon angestoßenen Aktionen des Agenten lediglich als Reaktion, nicht als Proaktivität zu werten.
- Schlußfolgerung
  Der Ablauf der Handlungen des Agenten erfolgt nach einem fest definierten Muster. Er benötigt deshalb keine spezielle Wissensbasis als Weltmodell, durch die er Schlußfolgerungen im Sinne der KI ab-

leiten könnte. Für die Funktion des Agenten sind derartige Verfahren nicht notwendig.

- Lernfähigkeit / Adaption
  Mit der fehlenden Fähigkeit für Schlußfolgerungen gibt es auch keine Möglichkeit und keinen Bedarf, aus dem bisherigen Ablauf zu lernen und Handlungen entsprechend anzupassen. Eine wiederum sehr schwache Adaption an den unmittelbaren Zustand des Systems kann in der Beachtung der Auslastung bei der zeitlichen Bestimmung der nächsten Aktion gesehen werden.
- Kontinuierliche Ausführung
  Diese Eigenschaft ist genauso wie die Autonomie grundlegend für das gewünschte Verhalten der Komponente, da es nicht um einmalige Suchanfragen geht, sondern diese über einen längeren Zeitraum wiederholt ausgeführt werden.

Diese Untersuchung zeigt, daß für die Realisierung der Komponente viele Merkmale der Agententechnologie benötigt werden. Wiederum stellt sich aber auch heraus, daß es sich dabei um die allgemeineren und weniger komplexen Eigenschaften handelt. Leistungsfähige Konzepte wie Mobilität, Kooperation und Lernfähigkeit sind hier nicht oder nur in einfachsten Formen von Bedeutung, da das Aufgabengebiet keine Anpassungen des Agenten an seine Umgebung verlangt (feststehender Algorithmus), ein Agent seine Aufgabe ohne Probleme alleine lösen kann und die benötigten Funktionen von der lokalen Agentenumgebung bereitgestellt werden können. Als positives Resultat dieser Erkenntnis kann man die Möglichkeit betrachten, daß sich die Komponente auch ohne hochspezialisierte Agentenbibliotheken und Programmierumgebungen mit vertretbarem Aufwand implementieren läßt.

4.4 Evaluation existierender Systeme

Mit der in 4.3.2 vorgenommenen Untersuchung zur Anwendung der Agententechnologie in der zu realisierenden Komponente wird klar, daß spezielle Agentenentwicklungssysteme, wie Aglets[56] oder Odyssey[57], die im wesentlichen die Mobilität unterstützen, nicht als Werkzeug in Frage kommen. Ebenso verhält es sich mit Bibliotheken für kooperative Agentensysteme, wie dMARS[58] oder JAFMAS[59].

Bevor jedoch die Suchkomponente mittels einer eigenen Konzeption umgesetzt wird, werden als Alternative fertige Programmpakete, die Lösungen für diesen Aufgabenbereich anbieten, evaluiert. Es handelt sich um zwei kommerzielle Produkte, die als Suchmaschinen auf Intranetebene um agentenartige Funktionen ergänzt wurden. Die Programme

[56] http://www.trl.ibm.co.jp/aglets
[57] http://www.genmagic.com/technology/odyssey.html
[58] http://www.aaii.oz.au/proj/dMARS-prod-brief.html
[59] http://www.ececs.uc.edu/~abaker/JAFMAS

werden zunächst einzeln vorgestellt und danach bzgl. einer Merkmalsliste, die sich an den Anforderungen aus 4.2 orientiert, miteinander verglichen. Am Schluß folgt daraus eine Einschätzung ihrer Tauglichkeit für den Einsatz als universelles News-Watcher Agentensystem.

4.4.1 DOCSFulcrum

Das als Fulcrum Knowledge Network von Fulcrum entwickelte Information Retrieval System, welches jetzt nach dem Zusammenschluß mit PC DOCS als DOCSFulcrum vermarktet wird, dient dem zentralisierten Zugriff auf die verschiedenen Informationsquellen eines Unternehmens. Die Quellen lassen sich in einer selbst definierten logischen Hierarchie anordnen und über eine einheitliche Schnittstelle durchsuchen.

DOCSFulcrum besteht aus einer vierschichtigen Architektur wie sie in Abbildung 9 dargestellt ist. Das Kernstück ist der *Knowledge Server*, der für die Indizierung der Datenquellen, die Suche, die Agenten und die Systemverwaltung zuständig ist. Im *Knowledge Center* werden die Metadaten wie Dokumentattribute oder auch die Agentenprofile verwaltet.

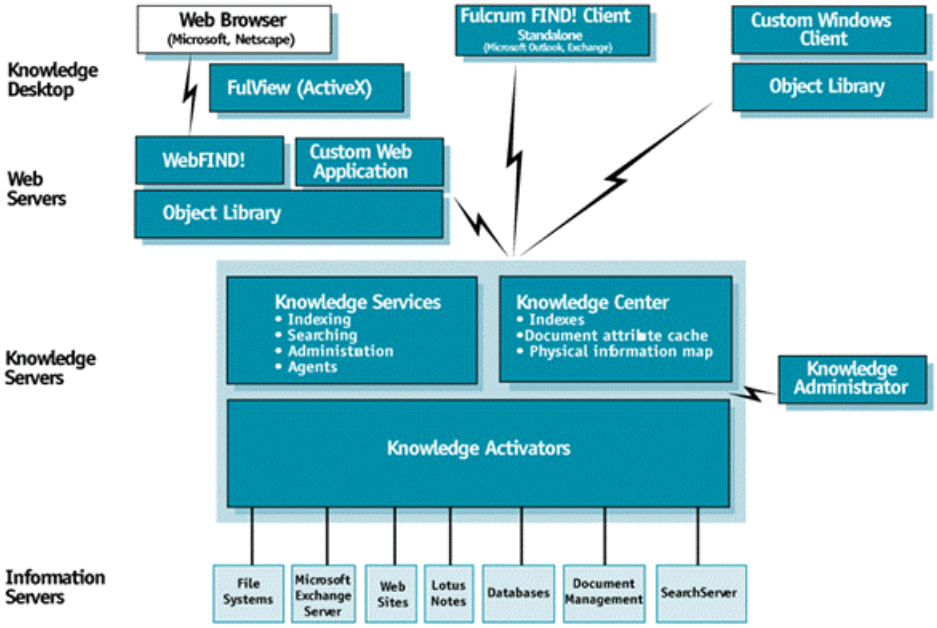

Abbildung 9: Architektur von DOCSFulcrum (aus [Pcd98])

Die Anbindung der Datenquellen erfolgt über sogenannte *Knowledge Activators*. Diese sind für jeweils einen bestimmten Datentyp zuständig und stellen die Verbindung zu den jeweiligen Quellen – in der Abbildung als *Information Server* bezeichnet – her.

Während die Administration der Datenquellen nur über ein mitgeliefertes Programm (*Knowledge Administrator*) erfolgen kann, bietet der *Knowledge Server* auf Benutzerseite verschiedene Schnittstellen an. Es gibt sowohl vorgefertigte anpaßbare Suchmasken für den WWW-Browser oder als einzelne Anwendung wie auch die Möglichkeit, über eine als

COM-Schnittstelle (*Component Object Model*) ausgelegte Bibliothek auf die Suchfunktionen zuzugreifen.

Die als ‚ProActive' bezeichneten Agenten können alle Abfrageoptionen in ihrem Profil verwenden und liefern die Fundstellen als WWW-Seite oder per Email an den Benutzer. Außer einem einfachen Verweis auf die relevanten Daten kann auch Dokumentzusammenfassungen aufgelistet werden.

DOCSFulcrum ist speziell auf die NT-Plattform von Microsoft zugeschnitten und hat seine Stärken in der Indizierung von Dokumenten aller gebräuchlichen Formate im Dateisystem, Daten von Lotus Notes und Microsoft Exchange Server sowie von WWW-Seiten. Datenbanken lassen sich nur über den separat zu erwerbenden *SearchServer* und nur nach Anpassung des Datenbankschemas und größerem Implementierungsaufwand integrieren.

4.4.2 Verity

Verity bietet mit seiner SEARCH'97 Produktpalette ein umfangreiches Angebot zur Lösung von *Information Retrieval*-Aufgaben über heterogenen Datenbeständen. Grundlage aller Anwendungen ist *der Information Server*, dessen komponentenartige Architektur (siehe Abbildung 10) entsprechende Erweiterungen zuläßt. Er ist für die Administration und Benutzerkommunikation auf die Zusammenarbeit mit einem WWW-Server angewiesen, kann aber auch durch extra zu erwerbende Entwicklungswerkzeuge in andere Anwendungen integriert werden.

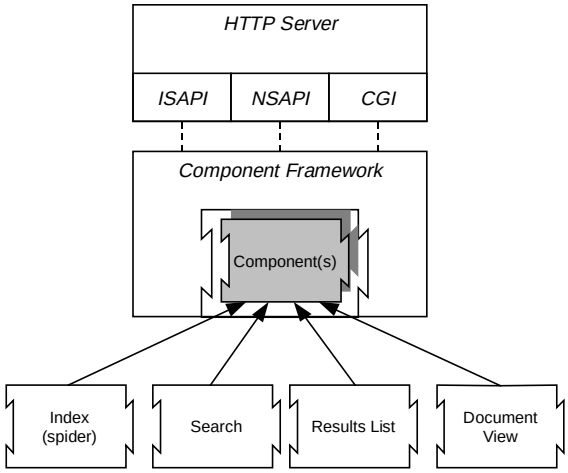

Abbildung 10: Architektur des SEARCH'97 Information Server von Verity (aus [Ver99])

Der *Information Server* ist für den gesamten Retrievalprozeß zuständig. Er beinhaltet einen Indexierer für verschiedene Dokumentformate, die diesem über Filter zugeführt werden, weiterhin besitzt er Komponenten zum Ausführen von Suchanfragen und zum Anzeigen der gefundenen Daten. Der Index wird in einem proprietären Format abgelegt. Zur Auswahl bestimmter Dokumente für eine Suche, werden diese beim Indexie-

ren in Dokumentenkollektionen zusammengefaßt. Weiterhin kann durch sogenannte *Topics* ein thematischer hierarchischer Katalog erstellt werden, der in den einzelnen Knoten Verweise auf die Dokumente enthält und damit eine frei definierbare logische Sicht auf die Daten bietet. Diese *Topics* können ebenfalls für eine Suchraumbeschreibung genutzt werden.

Um Datenbanken in die Dokumentenkollektion zu integrieren, muß der *Information Server* um die *Database Access* Komponente erweitert werden. Mit ihr kann über ODBC (*Open Database Connectivity*) auf einige Datenbanksysteme zugegriffen werden. Der dahinter stehende Ansatz geht jedoch davon aus, daß jeweils eine Tabelle existiert, die pro Datensatz in ein oder mehreren Spalten ein Dokument beinhaltet, welches durch weitere Spalten mit Metadaten ergänzt wird. Zugriff auf mehrere Tabellen ist nur bedingt über die Erweiterung des Datenbankschemas durch Views realisierbar.

Die Agentenfunktionalität erhält man durch das Hinzufügen des Agent Server. Dieser leistet die Verwaltung und Ausführung der Agenten. Aus den Profilen der verschiedenen Agenten wird ein Profilnetzwerkes erzeugt, in dem gleiche Suchkriterien in einem Knoten enthalten sind. Dadurch kann die Anzahl der notwendigen Suchanfragen für eine Aktualisierung aller Agenten reduziert werden. Ein *Profiler* ist für die Überwachung der Dokumentenkollektion zuständig und benachrichtigt die Agenten, wenn ein neues relevantes Dokument eingetroffen ist. Die Agenten wiederum benachrichtigen den Benutzer über ein vorgegebenes Medium (WWW-Seite, Email, Pager). Eine Einstellung zur Häufigkeit der Abfragen ist nur durch die Vorgabe des Ressourcenverbrauchs des *Profilers* möglich.

Da der *Agent Server* nur über eine HTTP-Verwaltungsschnittstelle verfügt, ist für die Integration der Agentenfunktionen in eine Anwendung eine Erweiterung mit dem *Agent Server Toolkit* notwendig. Mit dessen Hilfe können die Verwaltung der Agenten und die Zustellungsmethoden sowie die Konfiguration des *Profilers* über API-Schnittstellen angesprochen bzw. durch eigene Treiber erweitert werden. Eine einfache Methode, einer Anwendung den Zugriff auf die Agentendaten zu ermöglichen, ist die Verwendung eines externen RDBMS für die Verwaltung des Agentenprofils und der Ergebnislisten. Über Datenbanktrigger können so Aktionen beim Eintreffen neuer relevanter Dokumente ausgelöst werden.

4.4.3 Merkmalsvergleich

	DOCSFulcrum	Verity
verglichene Produkte	DOCSFulcrum	Information Server V3.6, Agent Server V1.1, Agent Server Toolkit V2.1, Database Access V2.0
Suchfunktionalität		
Anfragesyntax	erweiterte boolesche Abfrage mit Wildcards; natürlichsprachige Anfragen (automatische Erstellung eines äquivalenten Booleschen Ausdrucks mit Eliminierung von Doppeldeutigkeiten und Erweiterung mit ähnlichen Ausdrükken)	erweiterte boolesche Abfrage mit Wildcards (Wildcards im *Agent Server* aufgrund eines Bugs nicht möglich); Syntax der WWW-Suchmaschinen einstellbar; Angabe eines Textabschnitts als Suchausdruck für ähnliche Dokumente (*query by example*)
linguistische Verfahren zur Verbesserung des Recall	Anreicherung der Anfrage durch Synonyme ist möglich; durch Vorgabe eines Dokumentes kann nach weiteren ähnlichen gesucht werden	über *Soundex* kann nach ähnlich klingenden Wörtern gesucht werden; *Stemming* sucht nach gleichen Wortstämmen; *Thesaurus* erweitert die Suchbegriffe um Synonyme; *Typo/N* sucht nach, bzgl. der Zeichenkette, ähnlichen Begriffen
Mehrsprachigkeit	✓ (europäische und asiatische Sprachen)	✓, 12 Sprachen, erweiterbar
Datentypen	▪ Dateisystem (auch UNIX- und Novell-Rechner), ▪ Microsoft Exchange Server, ▪ Lotus Notes, ▪ WWW-Seiten, ▪ Datenquellen, die mit *Fulcrum SearchServer* indiziert sind	▪ Dateisystem (auch UNIX- und Novell-Rechner), ▪ WWW-Seiten weitere Datentypen lassen sich über extra zu beziehende *Information Gateways* erschließen
Datenformate	ca. 150	ca. 40
Unterstützung von RDBMS	nicht direkt, nur bedingt über *Fulcrum SearchServer* (siehe Text)	nur bedingt über *Database Access* (siehe Text)

	DOCSFulcrum	**Verity**
Agentensystem		
Agentenverwaltung durch Benutzer	über mitgelieferten Client oder durch Funktionen der COM-Schnittstelle möglich; Änderungen des Suchausdrucks eines Agenten sind nicht vorgesehen	mit *Toolkit* über API oder Datenbank, sonst nur per HTTP-Server
Beeinflussung der Aktivität	Zustellungshäufigkeit einstellbar	nur für alle Agenten gemeinsam über *Profiler* möglich (siehe Text)
Arten der Ergebniszustellung	▪ WWW-Seite, ▪ Email	▪ WWW-Seite, ▪ Email, ▪ Pager weitere mit *Toolkit* selbst programmierbar
Systemintegration		
unterstützte Plattformen	Windows NT	*Information Server*: Windows NT, Sun Solaris *Agent Server*: Windows NT, Sun Solaris *Agent Server Toolkit* mit Datenbankunterstützung (Agenten und Ergebnislisten): Windows NT mit SQL Server, Sun Solaris mit Oracle oder Sybase, AIX 4.x mit Oracle *Database Access*: Windows NT mit Oracle oder Sybase, Sun Solaris mit Oracle oder Sybase
Schnittstellen zur Agentenverwaltung	COM-Schnittstelle	API des *Toolkit*
Schnittstellen zur Datenverwaltung	nur über mitgelieferte Anwendung	nur über weitere Entwicklungswerkzeuge (*Developer's Kit*)
Schnittstellen zur Systemverwaltung	nur über mitgelieferte Anwendung	nur über weitere Entwicklungswerkzeuge
unterstützte Datenverwaltungsprogramme	keine, Index- und Verwaltungsdaten werden selbst verwaltet	Oracle und Sybase durch *Toolkit* für Agentendaten und Ergebnislisten

Für die unterstützten Plattformen und Datenformate bieten beide Programme eine sehr umfangreiche Suchfunktionalität. Die Agentenmodule setzen darauf auf und bieten die wesentlichen Funktionen. Daß die Programme trotzdem für den Einsatz als **universelle** News-Watcher Agentensysteme schlecht geeignet sind, liegt an folgenden Punkten:

- Die Integration von Datenbanken mit verknüpften Tabellen als Datenquelle wird nicht oder nur mangelhaft unterstützt.
- Um die Systemverwaltung der Suchmaschine innerhalb der Anwendung durchzuführen ist ein hoher programmiertechnischer Aufwand erforderlich. Im Falle von DOCSFulcrum ist dieses überhaupt nicht vorgesehen.
- Spezielle Datenquellen können nur durch aufwendig programmierte eigene Treiber eingebunden werden. Die dafür notwendigen Entwicklungswerkzeuge müssen extra erworben werden.
- Die Systemunabhängigkeit ist stark eingeschränkt, da nur bestimmte Kombinationen von Plattformen und Datenbanken möglich sind.
- Der WWW-Server zentrierte Ansatz schränkt die universelle Einsetzbarkeit ein, insbesondere die vollständige Integration in eine Anwendung mit eigener Benutzeroberfläche.
- Die Produkte sind für kleinere Datenbestände und geringe Anzahl von Benutzern überdimensioniert.
- Die Lizenzgebühren der Produkte sind für kleinere Projekte nicht vertretbar und bei unbekannten Benutzerzahlen nicht kalkulierbar (z.B. bei Verity fünfstellige Lizenzgebühren laut [Bag96]).

4.5 Konzeption eines Framework

Im vorangegangenen Abschnitt wurde anhand von zwei Beispielen erläutert, daß existierende Agentensysteme nur partiell die in 4.2 genannten Anforderungen erfüllen und deshalb keine allgemeine Lösung für die beschriebenen Anwendungen darstellen. Aus diesem Grund soll in dieser Arbeit ein agentenbasiertes Framework erarbeitet werden, daß dieses Problem angeht. In Kapitel 5 wird dann der auf dieser Konzeption aufbauende realisierte Prototyp eines Agentensystems vorgestellt.

Die Spezifikation mit den minimalen Anforderungen, sozusagen das Pflichtenheft, an das zu erstellende Framework wurde schon weiter oben aufgestellt. Um eine klare qualitative Unterscheidung zu den existierenden spezialisierten Agentensystemen zu erreichen, sollen die Punkte universelle Einsetzbarkeit sowie einfache nahtlose Integration in andere Anwendungen besonders unterstützt werden. Da diese beiden Eigenschaften in einem bestimmten Verhältnis zueinander stehen, das für das Design des Framework entscheidend ist, werden sie zunächst einmal genauer untersucht, bevor es zum eigentlichen Entwurf kommt.

4.5.1 Universalität vs. einfache Integration

Eine Konzeption, die wie diese von wenig restriktiven Randbedingungen begleitet ist, verführt zu einem möglichst allgemeinen Entwurf, der in vielen unterschiedlichen Projekten verwendet werden kann. In unserem Fall könnte das bedeuten, daß nicht die Entwicklung einer News-Watcher Komponente, sondern ein allgemein einsetzbares Agentensystem angestrebt wird. Verfolgt man diesen Ansatz genauer, stellt sich heraus, daß dies nichts anderes als die Entwicklung einer Agentenumgebung ist, welche die Verwaltung und Kommunikation der Agenten regelt. Die Agenten müßten ebenfalls einen allgemeinen Aufbau besitzen, der Mechanismen zur Aktionsauswahl und Schnittstellen für Sensoren und Effektoren bereitstellt. Hier treten jedoch Schwierigkeiten auf, da entschieden werden muß, ob es sich um reaktive oder deliberative Agenten handelt, weil sich diese in ihrer Architektur grundlegend unterscheiden.

Um ein solches Agentensystem für den Einsatz anzupassen, müssen vielfältige Parameter gesetzt und spezifische Programmteile implementiert werden, z.B. Verhaltensweisen der Agenten, Datenfluß, Schnittstellen, benötigte Dienste. Dieses bedarf einer eigenen Agentenprogrammiersprache, wie z.B. Grasshopper[60], Odyssey[61], dMars[62].

Das Streben nach Universalität kollidiert jedoch mit der Forderung nach einfacher Integration. Jede für die Anpassung an die Anwendung erforderliche Programmierleistung, die sich auf interne Prozesse des Agentensystems bezieht, stellt einen unnötigen Aufwand dar und erschwert die Akzeptanz.

Der Idealfall einer einfachen Integration stellt ein Agentensystem dar, das unverändert in die Anwendung übernommen werden kann. Nur auf Seiten der Anwendung müssen Funktionen nachgerüstet werden, die auf die Schnittstellen der Komponente zugreifen und den Daten- bzw. Parameteraustausch ermöglichen. Da eine solche Komponente fest für eine bestimmte Aufgabe ausgelegt sein muß, kann sie nur noch sehr beschränkt einen universellen Charakter aufweisen.

Um einen Kompromiß zwischen diesen gegenläufigen Eigenschaften zu erhalten, muß das Aufgabengebiet der Komponente soweit eingeschränkt werden, daß alle notwendigen Routinen implementiert werden können und immer noch eine größere Zahl von Projekten dafür eine Verwendung hat. Für den vorliegenden Fall bedeutet dies die Beschränkung auf die News-Watcher Funktionalität, da sie, wie in 4.1 gezeigt, in verschiedenen Anwendungen gebraucht wird.

[60] http://www.ikv.de/products/grasshopper
[61] http://www.genmagic.com/technology/odyssey.html
[62] http://www.aaii.oz.au/proj/dMARS-prod-brief.html

Auch mit der vorgenommenen Einschränkung des Aufgabenbereiches bleibt das Thema Universalität als Leitgedanke für den Entwurf bestehen. Wenn nämlich als ein Ziel die Integrationsfähigkeit in möglichst alle Anwendungen, welche eine News-Watcher Komponente benötigen, formuliert wurde, bedeutet dies den Schwerpunkt auf Flexibilität bei Schnittstellen, Datenformaten, Plattformen, Skalierbarkeit usw. zu legen.

4.5.2 Bemerkungen zur Modellierung

Wie in [Bre98, S. 178 ff.] ausgeführt, müssen bei der agentenorientierten Analyse- und Designphase einige Besonderheiten beachtet werden, die neue Methoden und Modelle gegenüber der klassischen Softwareentwicklung erfordern.

Am ehesten lassen sich Agentensysteme mit objekt-orientierten Methoden beschreiben, da Agenten vereinfacht als Objekte mit einem bestimmten Zustand und definierten Schnittstellen und Methoden betrachtet werden können. Hierfür bietet sich die von Booch, Rumbaugh und Jacobson entwickelte Modellierungssprache UML[63] (*Unified Modelling Language*) an.

Agenten können jedoch im Vergleich zu Objekten über weitergehende Eigenschaften verfügen, die in den bisherigen objektorientierten Ansätzen nicht adäquat beschrieben werden. Dazu gehört z.B. die komplexere Struktur von deliberativen Agenten, die über Wünsche, Ziele, Intentionen sowie entsprechende Schlußfolgerungsmechanismen verfügen als auch die Kommunikationsfähigkeit. Gerade die zuletzt genannte Eigenschaft, die durch den Austausch semantischer Information weit über den einfachen Nachrichtenaustausch von Objekten hinausgeht und eine aufgabenabhängige Zusammenarbeit der Agenten ermöglicht, bedarf eines eigenen Kommunikations- und Kooperationsmodells sowie neuer Methoden für deren Entwurf.

Für den aktuellen Entwurf wurde in 4.3.2 argumentiert, daß ein kooperatives Verhalten der Agenten zwar denkbar und zu einem gewissen Grade hilfreich sein kann, der dazu notwendige Aufwand aber in keinem Verhältnis zum Nutzen steht und deshalb einzeln arbeitende Agenten ohne besonderen Kommunikationsfähigkeiten ausreichen. Weiterhin kann das Verhalten auf bestimmte reaktive Muster entsprechend des Zustands der Umgebung beschränkt werden, wodurch der Komplexitätsgrad gering gehalten werden kann.

Daraus wird ersichtlich, daß in diesem Fall keine besonderen Techniken für die Modellierung herangezogen werden müssen. Die bisherigen Modelle verknüpft mit einer abstrakteren Sichtweise für die Beschreibung der Aktionen von Agenten sind ausreichend. Da es sich in diesem Ka-

[63] Eine kurze Einführung in UML gibt [Fow97]; eine ausführliche Darstellung findet sich in [Boo99]

pitel vorerst um eine Designstudie handelt, werden keine Klassen- oder Objektmodelle erstellt, sondern die einzelnen Funktionsblöcke sowie die funktionalen Abhängigkeiten und Datenpfade zwischen ihnen wiedergegeben.

4.5.3 Design

Die Beschreibung des Framework für ein agentenbasiertes Suchwerkzeug, d.h. eine *News Watcher Agent*-Funktionalität (siehe dazu Kapitel 4.3), zur Integration in andere Anwendungen erfolgt in einem Top-Down Verfahren. Ausgehend von der Gesamtarchitektur und ihren Funktionseinheiten bzw. Modulen werden letztere schrittweise näher betrachtet, in die enthaltenen Komponenten zerlegt und erläutert.

Die diesem Entwurf zugrundeliegenden Bedingungen wurden in 4.2 vorgestellt, die von dem integrierten Agentensystem zu leistenden Funktionen in 4.3.1 und 4.3.2 diskutiert. Außerdem wurden in 4.5.1 zwei Anforderungen genauer untersucht, die wesentlich den hier gewählten Ansatz beeinflußt haben.

Die grundlegende Struktur des News Watcher-Systems ist in Abbildung 11 gezeigt[64]. Es besteht im wesentlichen aus zwei Modulen, die sich wiederum in mehrere Komponenten gliedern. Durch diese Aufteilung wird es möglich, das gesamte System auf verschiedene Rechner und Plattformen zu verteilen. Dabei ist nicht nur eine 1:1 Beziehung zwischen den Modulen sondern auch eine n:m Beziehung vorgesehen. Durch diesen Ansatz soll die Forderung nach Skalierbarkeit des Systems umgesetzt werden.

Die beiden Hauptmodule des Systems sind die vereinheitlichten indizierten Datenquellen und die Agentenumgebung. Neben der Möglichkeit der Skalierbarkeit gibt es noch einen weiteren Grund für diese Aufteilung: das Modul für die Datenhaltung kann unter Umständen komplett durch ein externes Softwareprodukt ersetzt werden, was die Ausnutzung spezifischer Funktionen (Filter, spezielle *Information Retrieval* Eigenschaften) ermöglicht. Zu beachten ist jedoch, daß in diesem Fall ein Wrapper für diese Anwendung geschrieben werden muß, der die notwendigen Schnittstellen für die Agentenumgebung bereitstellt.

[64] Eine Darstellung des kompletten Frameworks zeigt Abbildung 15

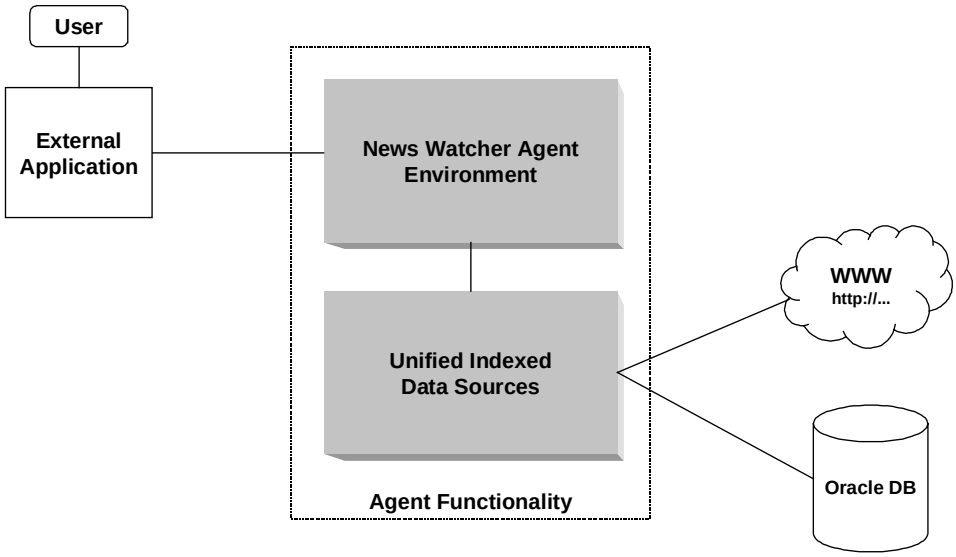

Abbildung 11: das Framework im Überblick

Neben der Betrachtung des Framework als modularisiertes System kann man es auch als eine 2-schichtige Architektur auffassen. Die untere Schicht (*Unified Indexed Data Sources*) stellt die Verbindungen zu den Datenquellen her, verbirgt deren unterschiedliche Struktur und stellt der nächsten Ebene Funktionen zum Zugriff zur Verfügung. Diese obere Schicht wird von der Agentenumgebung (*News Watcher Agent Environment*) gebildet und beinhaltet die Logik für die vom Benutzer bzw. der externen Anwendung, die das System integriert, beauftragte autonome Überwachung der Datenquellen.

Im folgenden werden die Module mit den enthaltenen Komponenten, angefangen bei der unteren Schicht, beschrieben und danach die Verbindung zwischen ihnen.

4.5.3.1 Modul 1: Unified Indexed Data Sources (Vereinheitlichte Datenquellen)

Dieses Modul dient der Aufbereitung der verschiedenen Datenquellen mit dem Ziel des einheitlichen Zugriffs und der Bereitstellung der Suchfunktionalität. In Abbildung 12 sind die enthaltenen Komponenten schematisch dargestellt.

Der *Data Source Manager* beauftragt *die Source Access and Converter Unit*, eine bestimmte Datenquelle zu observieren. Daraufhin werden die enthaltenen Daten und später Änderungen ausgelesen, in ein einheitliches Format gebracht und an den *Index Manager* übergeben. Dieser fügt sie in einen globalen Index aller überwachten Datenquellen ein und benachrichtigt den *Notifier* im Modul Agentenumgebung über das Vorliegen neuer Daten.

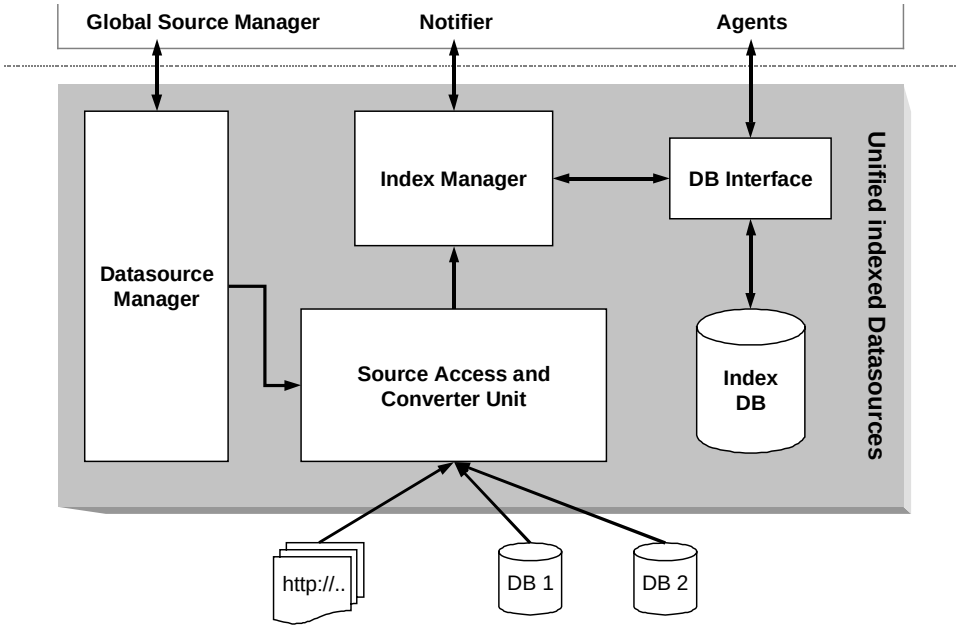

Abbildung 12: Framework Modul Vereinheitlichte Datenquellen

Data source Manager (Datenquellenverwaltung)

Der *Data source Manager* erfüllt die Protokoll- und Überwachungsfunktion. Er gibt Auskunft darüber, welche Schnittstellen und Filter vorhanden sind, welche Datenquellen derzeit zur Verfügung stehen und wie die Aktualisierung erfolgt. Weiterhin bearbeitet er Anfragen bzgl. der Aufnahme neuer Datenquellen und überwacht die bestehenden Verbindungen. Die Datenquellenverwaltung ist die Auskunftskomponente für die angeschlossenen Agentenmodule, von der diese aktuelle Statusinformationen erhalten.

Source Access and Converter Unit (Datenquellenzugriffs- und Kontvertereinheit)

Um für einen einheitlichen Zugriff auf verschiedene Datenquellen das notwendige Abstraktionsniveau zu erreichen, erfolgt dieser über eine 3-schichtige Architektur, wobei für den Datenbankzugriff eine weitere Ebene eingeschoben werden muß (siehe Abbildung 13). Ausgehend von einem spezifischen Interface zur Kommunikation mit der Datenquelle wird der Datenstrom durch ein allgemeines Interface für eine bestimmte Sorte von Daten (DB, WWW, Text) an ein Filter für den jeweiligen Dateninhalt geleitet. Die Ausgabe letzterer erfolgt in einer einheitlichen Form, die dem *Index Manager* zugeführt wird.

Access Unit (Zugriffseinheit)
Die Komponenten der Zugriffseinheit bilden die Schnittstellen zu den externen Datenquellen. In ihnen müssen alle low-level Funktionen für den Zugriff auf die entsprechenden Ressourcen sowie die Standardausgabefunktionen für die Datenübergabe an die Filter implementiert werden. Um diese spezifischen Komponenten möglichst einfach und damit leicht adaptierbar an weitere Datenbestände zu gestalten, werden von ihnen nur spezifische Funktionen zur Verfügung gestellt, die allgemeine-

ren Routinen befinden sich in den generischen Schnittstellenkomponenten.

DB Filter Unit (Datenbankfiltereinheit)
Diese Komponenten bilden eine Zwischenschicht, um dem vergleichsweise komplexeren Aufbau von Datenbanken Rechnung zu tragen. Sie enthalten die notwendigen Informationen über den Aufbau der Datenbank. Weiterhin wird in ihnen festgelegt, welche Felder für eine Indizierung in Frage kommen, in welchem Format dies vorliegen und welche Wertigkeiten den einzelnen Feldern bei der Indizierung zuzuordnen sind.

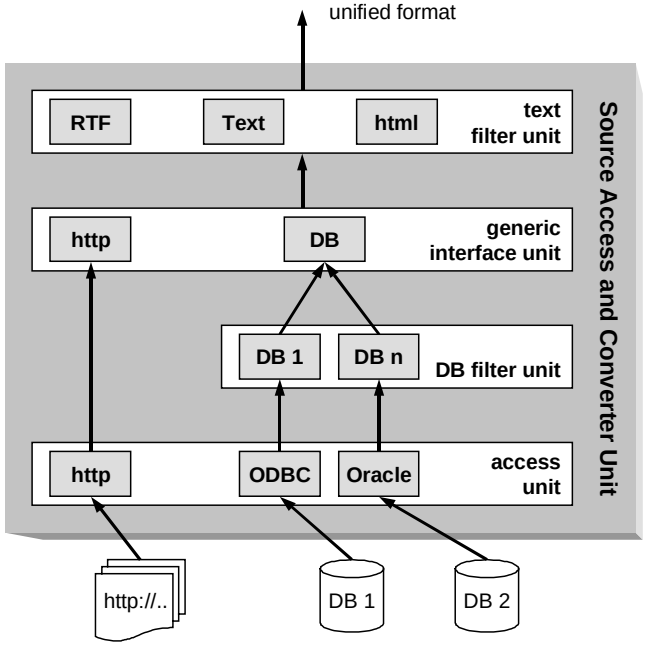

Abbildung 13: Framework Datenquellenzugriff- und Konvertereinheit

Generic Interface Unit (Generische Schnittstellen)
Die Komponenten dieser Ebene dienen im wesentlichen der Entkopplung allgemein benötigter Funktionalität der einzelnen Datenbereiche (DB, http, Textdateien) von den speziellen Treibern der Zugriffseinheit. Zu dieser Funktionalität gehört z.B. die Anreicherung der Daten mit Referenzen auf die Quellen, die allgemeine Zugriffssteuerung, das Updateverhalten und die Wahl entsprechender Filter.
In den generischen Schnittstellen ist es jedoch nur möglich, als Updatevariante das *Polling* (zeitgesteuerte Abfrage) zu implementieren. Andere Varianten, wie *Snooping* (Abfangen neuer Daten während der Eingabe) oder *Listener* (Warten auf eine Nachricht der Datenquelle – z.B. Trigger in Datenbanken), können wegen ihrer Abhängigkeit von den jeweiligen Datenquellen nur innerhalb der Zugriffseinheit realisiert werden.

Text Filter Unit (Textfiltereinheit)
In dieser Ebene werden verschiedene Filter zur Verfügung gestellt, um die unterschiedlichen Eingabeformate in verarbeitbare Textketten zu transformieren. Diesen Komponenten kommt dabei die Aufgabe zu, den Text aus den umgebenden Layoutangaben zu extrahieren, mit Informa-

tionen zur Relevanz im Dokument zu versehen (z.B. Kennzeichnung von Überschriften und Hervorhebungen) und bei Bedarf Zeichenkonvertierungen vorzunehmen. Am Ende besitzen die Daten, ganz gleich aus welcher Datenquelle sie kommen, ein einfaches einheitliches Format, das als Eingabe für den *Index Manager* verwendet wird.

Index Manager (Indexverwaltung)

Der *Index Manager* bereitet den Datenstrom von der Textfiltereinheit auf und legt ihn als Index mit Hilfe eines *DB Interface* in einer externen Datenbank ab. Dabei müssen neben dem Einfügen neuer auch Funktionen für das Ändern und Löschen bestehender Einträge vorhanden sein. Durch die Verwendung eines höher integrierten DBMS (Database Management System) kann auch der größte Teil aus dem *Index Manager* in das DBMS verlegt werden. Weiterhin kommt dem *Index Manager* die Aufgabe zu, den angeschlossenen Agentenmodulen über die Tatsache der Änderung von Indexeinträgen in Kenntnis zu setzen.

DB Interface (Datenbankschnittstelle)

Das *DB Interface* stellt alle notwendigen Funktionen zum Zugriff auf den Index bereit und abstrahiert damit von dem realen Index der entweder als besondere Struktur vorliegen kann oder mit Hilfe eines DBMS realisiert ist. Im letzten Fall kann durch die Verwendung der *Access Unit* innerhalb der *Source access Unit* die Unabhängigkeit von einem bestimmten Datenbanksystem gewahrt werden. Die Komponenten, die Zugriff auf den Index erhalten, sind neben dem *Index Manager* auch die Agenten über eine Schnittstelle der *Agent Runtime Unit*. Es muß deshalb unterschieden werden, durch wen der Zugriff erfolgt, damit die Agenten nicht den Index selbst verändern können.

Admin Interface (Administratorschnittstelle)

Über das *Admin Interface* (dargestellt in Abbildung 15) erhält der Administrator Zugriff auf den *Data source Manager* und damit indirekt auch auf den *Index Manager*. Zu den durch diese Schnittstelle bereitgestellten Funktionen gehört das An- und Abkoppeln von Datenquellen, die Wartung des Index, das Einbinden weiterer Elemente in die Einheiten für den Datenzugriff und der Abruf von Statistiken über interne Verarbeitungsprozesse.

4.5.3.2 Modul 2: News Watcher Agent Environment (Agentenumgebung)

In diesem Modul erfolgt die eigentliche benutzerspezifische Überwachung der Datenquellen und die Rückmeldung an den Benutzer, d.h. es enthält die Agentenfunktionalität. Der Aufbau ist in Abbildung 14 dargestellt.

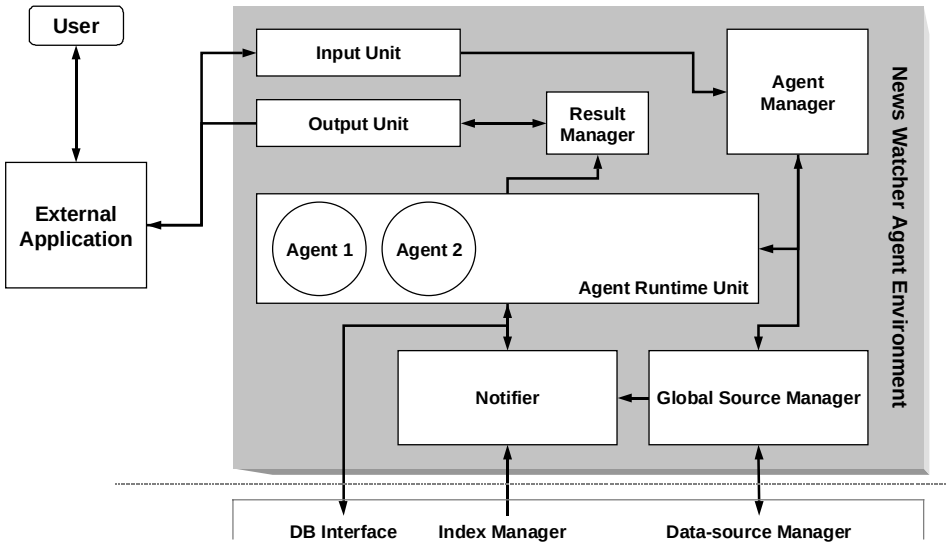

Abbildung 14: Framework Modul Agentenumgebung

Jeder Überwachungsauftrag wird mit Hilfe eines Agenten realisiert. Über die Schnittstellen der *Input Unit* können die Parameter an den *Agent Manager* übermittelt werden. Dieser erzeugt einen neuen Agenten, weist ihm ein Profil entsprechend der Suchparameter zu und übergibt ihn an die *Agent Runtime Unit*, die die Ausführung des Agenten kontrolliert und Schnittstellen zu den benötigten Diensten bereitstellt. Mittels des *Global Source Manager* kann sich ein Agent über die verfügbaren Datenquellen informieren und sich für die für ihn relevanten beim *Notifier* registrieren. Von diesem wird der Agent über erfolgte Datenänderungen informiert und kann durch Zugriff auf das *DB Interface* die Änderungen, entsprechend seines Profils gefiltert, abfragen. Wenn neue relevante Informationen vorliegen, stellt der Agent diese zusammen, übergibt sie der *Output Unit*, die je nach gewünschter Benachrichtigungsform diese sofort weiterleitet oder für spätere Anfragen vom *Result Manager* zwischenlagern läßt.

Input Unit (Eingabeeinheit)

Die *Input Unit* ist dem *Agent Manager* vorgelagert und dient der Entgegennahme von Kommandos zu bestimmten Agenten. Sie kann aber auch Statusinformationen der bearbeiteten Aufträge sowie zu einzelnen Agenten zurückliefern.
Durch die Auslagerung dieser Funktion in eine eigene Komponente kann, wie in Abbildung 15 genauer gezeigt, die Verbindung zur externen Anwendung über unterschiedliche Schnittstellen realisiert werden (z.B. Socketverbindung, direkter Zugriff über ein API). Die Eingabeeinheit stellt diese Schnittstellen zur Verfügung und leitet die Aufträge einheitlich an die entsprechenden Funktionen der Agentenverwaltung weiter. Dadurch ist es möglich, weitere Verbindungsarten durch Hinzufügen von Schnittstellenelementen zu unterstützen. Je nach Implementation kann dies auch dynamisch zur Laufzeit erfolgen.

Agent Manager (Agentenverwaltung)

Der *Agent Manager* ist die zentrale Komponente der Agentenumgebung. Über ihn erfolgt sowohl die eigentliche Kommunikation mit der Anwendung, d.h. entgegennehmen von Kommandos für einzelne Agenten und Rückmeldung von Statusinformationen als auch die Verwaltung der im System aktiven Agenten.
Über eine von der *Input Unit* angebotene Schnittstelle kann die Anwendung eine Benutzeranfrage in Form eines Profils der Agentenverwaltung übergeben, welche daraus einen Agenten kreiert und dessen Aufsicht übernimmt. Neben dem Erzeugen der Agenten werden auch Funktionen zum Bearbeiten, Löschen, Deaktivieren und Aktivieren angeboten.

Global Source Manager (Globale Ressourcenverwaltung)

Der *Global Source Manager* hat die Aufgabe, die in der Rechnerumgebung / Intranet / Internet verfügbaren Ressourcen, d.h. Module vereinheitlichter Datenquellen, zu lokalisieren, Verbindungen zu diesen aufzubauen und bei Bedarf neue Datenquellen anzufordern. Er liefert dem *Agent Manager* eine Liste der verbundenen Quellen und nimmt von diesem Aufträge für neu zu observierende Quellen an. Agenten haben ebenfalls die Möglichkeit, bei der Ressourcenverwaltung Informationen zu den aktuell verfügbaren Datenquellen einzuholen und je nach ihrem Profil den Suchraum auszuwählen.

Agent Runtime Unit (Agenten Laufzeitumgebung)

Die vom *Agentenmanager* kreierten Agenten agieren innerhalb einer Laufzeitumgebung, durch die sie Zugriff auf die indizierten Daten erhalten, sich beim *Notifier* anmelden und vom *Agentenmanager* kontrolliert werden können.
Wenn ein *Agent* neu angelegt wurde, fragt er einmal die gesamten Daten nach relevanten Informationen ab. Anschließend tritt der Agent nur noch in Aktion, wenn die Datenquelle signalisiert, daß neue Daten eingetroffen sind (siehe *Notifier*). Durch das Mitführen von Zeitstempeln der zuletzt erhaltenen Daten können die weiteren Abfragen auf die neu hinzugekommene Information eingegrenzt werden. Die aus der Abfrage resultierenden relevanten Informationen werden von den Agenten aufbereitet, mit der im Profil festgelegten Zustellungsart versehen und an den *Result Manager* übergeben.

Notifier (Benachrichtiger)

Der *Notifier* stellt die Verbindungen zu den *Indexmanagern* der angeschlossenen Datenmodule bereit. Die notwendigen Informationen dafür werden vom *Global Source Manager* bereitgestellt. Vom *Indexmanager* erhält der Notifier Informationen über Zeitpunkt (Timestamp) und Quellen-ID von neu eingetroffenen Daten.
Jeder aktive Agent muß sich beim Notifier registrieren, wobei die Liste der zu observierenden Datenquellen und der Updaterhythmus übergeben werden. Auf Grundlage dieser Daten ist es dem Notifier möglich zu entscheiden, welche Agenten über das Eintreffen der neuen Informationen benachrichtigt werden müssen.

Mit Blick auf die Autonomie von Agenten wäre es auch denkbar, daß der *Notifier* die Informationen der *Indexmanager* nur an die Agenten durchreicht und damit als einfacher Verteiler auftritt. Die Agenten müßten dann selbst prüfen, ob die Informationen relevant sind und Schlußfolgerungen für ihr Updateverhalten treffen. Es erscheint aber im Sinne eines ökonomischen Informationsflusses günstiger, an zentraler Stelle zu filtern und nur noch interessierte Agenten zu benachrichtigen. Andererseits könnte ein zentraler Filter bei einer sehr großen Anzahl von Agenten einen Flaschenhals bilden und die Verlagerung in Agenten dieses verhindern. Der zentrale Ansatz bildet jedoch für die meisten Anwendungsfälle die bessere Variante.

Result Manager (Ergebnisverwaltung)

Wie bei der *Agent Runtime Unit* beschrieben, erhält der *Result Manager* die Ergebnislisten der Agenten zusammen mit dem gewünschten Ausgabeformat und Aktualisierungsrhythmus. Je nach diesen Parametern werden die Listen sofort an die *Output Unit* zum Versand übergeben oder zwischengespeichert und nach Erreichen eines Zeitintervalls oder Datenmenge bzw. nach Aufforderung durch die externe Anwendung weitergereicht.
Auch diese Funktion sollte bei einer strengen Auslegung der Selbständigkeit von Agenten durch diese selbst übernommen werden, wie es auch in Abbildung 15 dargestellt ist. Mit dem Zwischenschalten der Ergebnisverwaltung erreicht man ein besser kontrollierbares I/O-Verhalten sowie Ressourcenminimierung bei den einzelnen Agenten.

Output Unit (Ausgabeeinheit)

Die endgültige Ausgabe der Überwachungs- und Suchergebnisse der Agenten erfolgt durch die *Output Unit*. Sie erhält diese vom *Result Manager* bzw. direkt von den Agenten (siehe dafür auch die Diskussion beim *Result Manager*). Vor dem Versand müssen die Resultate im gewünschten Format aufbereitet und als Parameter übergeben werden. Je nach Ausgabeart werden sie dem Benutzer direkt zugestellt (z.B. Email) oder der externen Anwendung übergeben. Mögliche Arten der Ausgabe sind:

- HTML-Seiten, die mit der Homepage des Benutzers in der Anwendung verlinkt sind,
- Benachrichtigung durch Email; entweder parallel zu den HTML-Seiten Information, daß sich diese geändert haben oder die Email enthält die neuen Ergebnisse selbst,
- API-Schnittstelle, die von einer externen Anwendung abgefragt werden kann.

Admin Interface (Administratorschnittstelle)

Über diese Schnittstelle erhält der Administrator Zugriff auf die zentralen Komponenten des Moduls. Zu den Funktionen, die von dieser Schnittstelle zur Verfügung gestellt werden, gehören:

- Überwachung der einzelnen Agenten (volle Funktionalität vom *Agentenmanager*)
- An- und Abhängen von Datenquellen
- Bewilligung der von Agenten angeforderten neuen Ressourcen (z.B. URL)

4.5.3.3 Interne Schnittstellen

Das Agenten- und das Datenmodul kommunizieren über drei Schnittstellen, die bei der Implementierung jedoch auch als eine Verbindung nach außen realisiert werden können.

1. Datasource-Manager $\leftrightarrow$ Global Source Manager
   Diese bidirektionale Verbindung dient zum einen dazu, der Agentenumgebung mitzuteilen, welche Ressourcen derzeit indiziert werden und welche Komponenten zur Datenanbindung und –konversion bereitstehen. In der entgegengesetzten Richtung kann die Agentenumgebung angeben, welche Ressourcen sie weiterhin indiziert haben möchte.
2. Index Manager $\rightarrow$Notifier
   Hierbei handelt es sich um eine einseitige Verbindung, durch die der Index Manger mitteilt, daß neue Daten eingetroffen sind. Dabei wird neben der Datenquellen-ID ein Zeitstempel übertragen, durch den die Agenten die Aktualität der Daten beurteilen können.
3. DB-Interface $\leftrightarrow$ Agenten
   Über diese Verbindung können die Agenten den Index abfragen und bekommen von der Datenbank die relevanten Informationen geliefert.

Wenn das Datenmodul komplett durch ein existierendes Softwareprodukt ersetzt werden soll, müssen diese Schnittstellen durch einen Wrapper dem Agentenmodul zur Verfügung gestellt werden. Die Verbindungen 1 und 2 sollten dabei recht einfach realisierbar sein, nur für den Zugriff auf den Index müssen die von den Agenten übergebenen und erwarteten Informationen in und von der Datenstruktur sowie den Funktionen der verwendeten Software übersetzt werden.

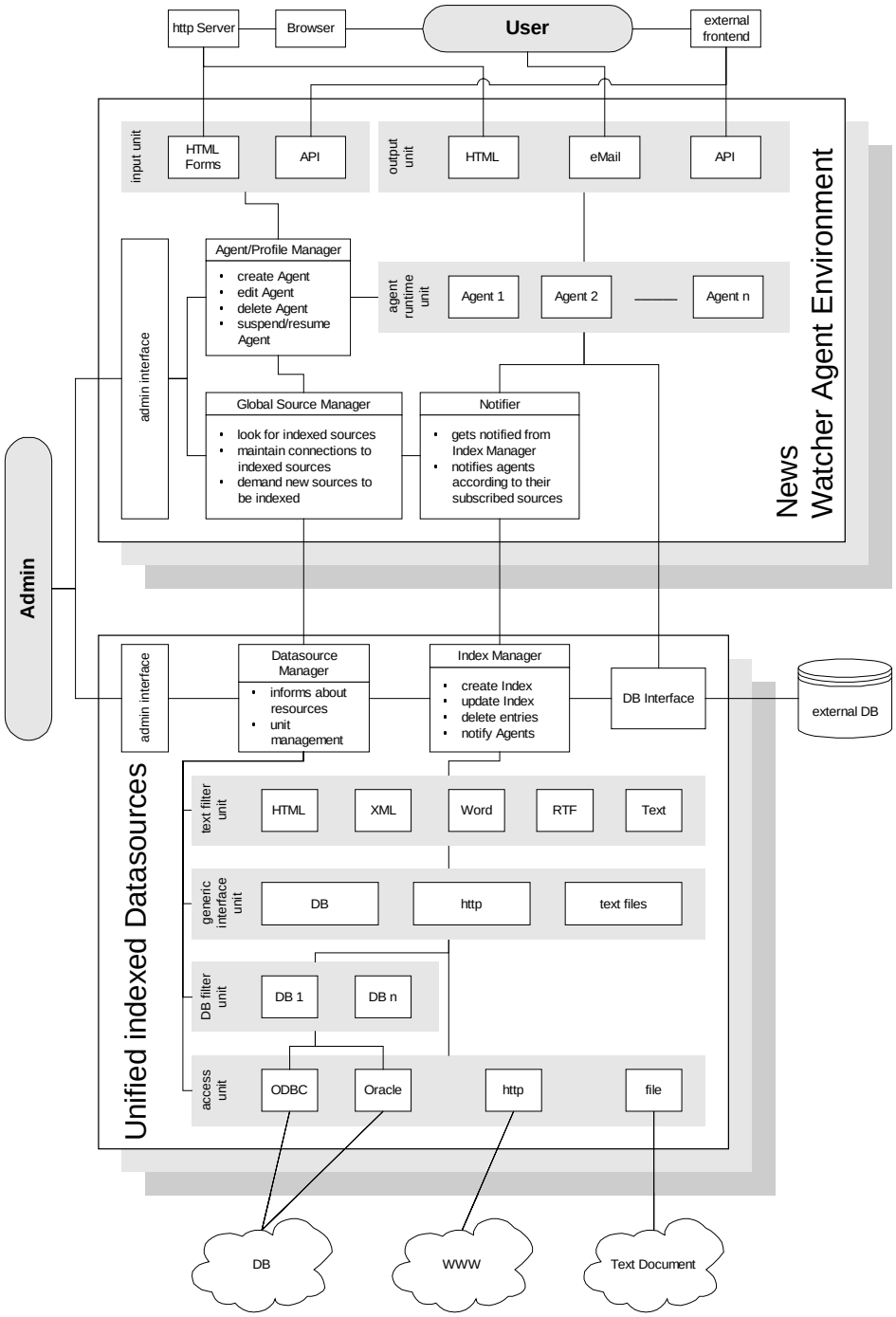

Abbildung 15: Framework Komplettansicht

4.6 Zusammenfassung

In diesem Kapitel wurde der Bedarf an einer integrierbaren eigenständigen *News Watcher*-Komponente anhand von aktuellen Projekten motiviert. Mit Berücksichtigung der Architektur und Bedürfnisse solcher Projekte wurde eine Anforderungsspezifikation erstellt, mit dem Ziel, möglichst vielen davon zu entsprechen. Anschießend wurde herausgearbei-

tet, daß sich für die Realisierung ein Agentensystem anbietet, wobei die Komplexität der einzelnen Agenten gering gehalten werden kann.

Vor der Entscheidung für eine Eigenentwicklung wurden zwei für diesen Bereich spezialisierte Softwareprodukte evaluiert, wobei sich zeigte, daß sie einen großen Teil der Anforderungen mit guten bis sehr guten Leistungen erfüllen konnten, jedoch bei einigen entscheidenden Kriterien zu inflexibel waren.

Unter Berücksichtigung dieser Mängel – Universalität und Integrationsfähigkeit – erfolgte die Ausarbeitung eines Framework für ein universelles *News Watcher*-Agentensystem, das einen modularen Aufbau besitzt, um Skalierbarkeit und die Verwendung existierender Software zu erleichtern. Eine wesentliche Entscheidung bei der Entwicklung der Architektur bestand darin, wie weit die Agenten den Aufgabenbereich alleine bearbeiten und welche Teile als Dienste der Agentenumgebung realisiert werden. Für den agentenzentrierten Ansatz spricht die bessere Skalierbarkeit auf Multiprozessormaschinen, während die Auslagerung in zentrale, von allen Agenten genutzte, Dienste eine effizientere Bearbeitung auf gering parallelisierter Hardware verspricht.

5 Entwicklung eines universellen *News Watcher*-Agenten

Nachdem im letzten Kapitel die Architektur eines universellen *News Watcher*-Agentensystems entwickelt wurde, beschreibt dieses Kapitel die Umsetzung dieses Framework in einen einsetzbaren Prototypen. Dafür müssen zunächst die vorhandenen Ressourcen und Bedingungen in der Konzeption berücksichtigt und daraus ein entsprechend angepaßtes Design entwickelt werden. Danach werden die programmtechnische Realisierung und anschließende Testläufe beschrieben. Mit Anmerkungen zu dem Einsatz in aktuellen Projekten sowie der Diskussion möglicher Weiterentwicklungen und Verbesserungen schließt das Kapitel.

5.1 Anpassung an projektbezogene Rahmenbedingungen

Für die Entwicklung des Prototyps, der in verschiedenen Projekten eingesetzt werden sollte, mußte der Schwerpunkt mit Rücksicht auf die zu Verfügung stehende Zeit und die Einsatzbedingungen verlagert werden. Mit einer realen Entwicklungszeit von maximal zwei Monaten galt es, den Komplexitätsgrad so weit wie möglich zu senken und trotzdem die Universalität nicht aufzugeben.

Die Rahmenbedingungen für den Einsatz des Agentensystems lassen sich wie folgt zusammenfassen:

- Es soll in WindowsNT Anwendungen integriert werden, die als Active Server Page[65] Anwendungen für den Internet Information Server[66] implementiert sind.
- Datenquellen sind vorrangig Oracle Datenbanken unterschiedlicher Version, deren Tabellen auch über Fremdschlüssel verknüpft sein können. Weiterhin soll es möglich sein, WWW-Seiten zu durchsuchen
- Die Anzahl der angemeldeten sowie der gleichzeitig aktiven Benutzer des Systems ist ungewiß. Bei erfolgreicher Einführung der Projekte sind jedoch mehr als 1000 angemeldete Benutzer möglich.

Daraus ergeben sich die Richtlinien für die Anpassung des Design:

1. Unterstützung der zu erwartenden Datenquellen, insbesondere von Datenbanken
2. Schnittstellenunterstützung für die bekannten Projekte
3. einfache Erweiterbarkeit auf neue Datenquellen und Schnittstellen

[65] Microsofts Variante mittels einer Skriptsprache dynamisch HTML-Seiten auf dem HTTP-Server zu generieren
[66] Microsofts Serveranwendungen für Internetdienste, enthält unter anderem den hier eingesetzten HTTP (WWW)-Server

4. effiziente Architektur, optimiert auf nicht oder nur schwach parallelisierte Plattformen

5.2 Designanpassung

Die im Framework vorgenommene Einteilung in zwei Module ist auch für den Prototyp sinnvoll, da es sich zum einen um unterschiedliche Funktionalitäten handelt, die als einzige Gemeinsamkeit den Index haben, zum anderen ist es so möglich, die Module auf getrennten Rechnern im Intranet laufen zu lassen und damit ohne großen Aufwand auf erhöhte Last zu reagieren.

Eine wesentliche Vereinfachung des Design kann erreicht werden, wenn die Dynamik des An- und Abmeldens von Datenquellen als gering eingestuft wird (normalerweise wird bei Datenbanken immer die selbe Datenquelle verwendet und ist ständig verfügbar). Ebenfalls kann in einem ersten Schritt von einer 1:1 Beziehung des Agentenmoduls und Datenmoduls ausgegangen werden.

Mit diesen Einschränkungen kann vollständig auf den *Global Source Manager* (siehe Abbildung 15 auf Seite 78) im Agentenmodul verzichtet werden, da davon ausgegangen werden kann, daß die Anwendung die indexierten Datenquellen kennt. Weiterhin kann auch das Gegenstück im Datenmodul stark reduziert werden und ausschließlich für die interne Verwaltung zur Verfügung stehen.

Weitere Vereinfachungen sind beim Agentensystem durch die Beschränkung auf eine zeitlich getriggerte gegenüber einer von Datenänderungen abhängigen Aktivitätssteuerung möglich. Dadurch entfällt der *Notifier* im Agentenmodul und der *Index Manager* muß nicht mehr bei jeder Änderung eine Nachricht versenden.

Insgesamt reduziert sich die Verbindung zwischen den Modulen auf den Zugriff auf den Index. Dieser wird – schon wegen der Zeitbeschränkung – als Tabelle in einer Datenbank realisiert (siehe Abbildung 16). Da das DBMS über Mechanismen für den gleichzeitigen Zugriff auf einen gemeinsamen Datenbestand verfügt, können beide Module als vollkommen voneinander entkoppelt betrachtet werden. Es ist demnach auch problemlos möglich, das Datenmodul komplett durch eine andere Software zu ersetzen, ohne am Agentenmodul etwas verändern zu müssen. Lediglich die Indexstruktur muß von dem Programm unterstützt werden.

Die strukturellen Veränderungen in den beiden Modulen gegenüber dem Framework , insbesondere ihre Autonomie, soll durch eine neue Bezeichnung zum Ausdruck gebracht werden. Sie werden entsprechend ihrer Hauptfunktion als Agentenserver und Indexserver bezeichnet.

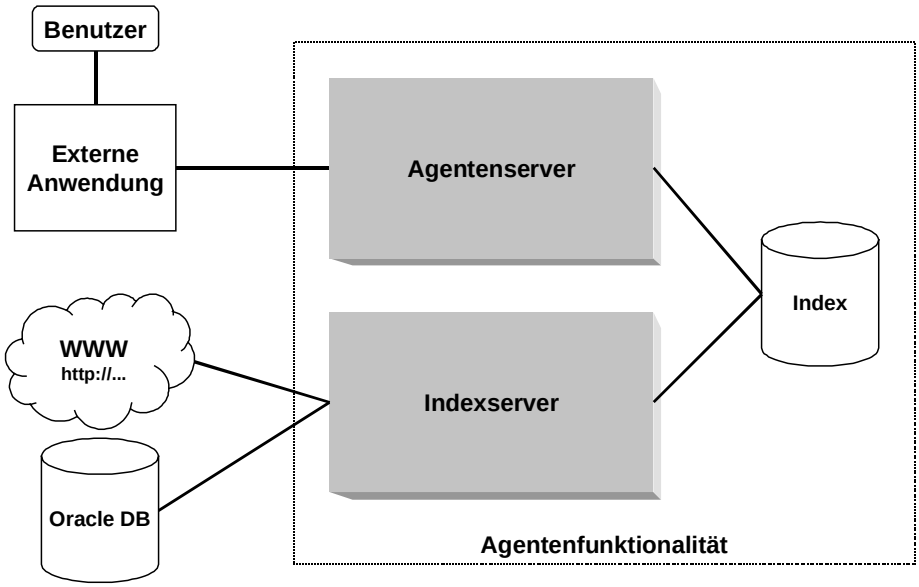

Abbildung 16: Prototypdesign Überblick

Die Änderungen im Indexserver gibt Abbildung 17 wieder. Im Zuge der Vereinfachung wurde versucht, die Grundstruktur der beiden Server weitestgehend identisch zu gestalten, so daß sie auf einer einheitlichen Codebasis aufbauen können und damit Zeit bei der Implementation eingespart werden kann. Dieses wird unter anderem in der Existenz der Komponente *Aktionsbearbeitung* ersichtlich, die mit den vorgelagerten Schnittstellen in beiden Servern vorkommt (vgl. Abbildung 18).

Die *Aktionsbearbeitung* regelt die Kommunikation mit der externen Anwendung, speziell administrative Aufgaben werden durch sie an die entsprechenden Komponenten im Server verteilt. Aus diesem Grund muß sie über einen Sicherheitsmechanismus verfügen, der nur autorisierten Anwendungen bzw. Benutzern den Zugriff erlaubt.

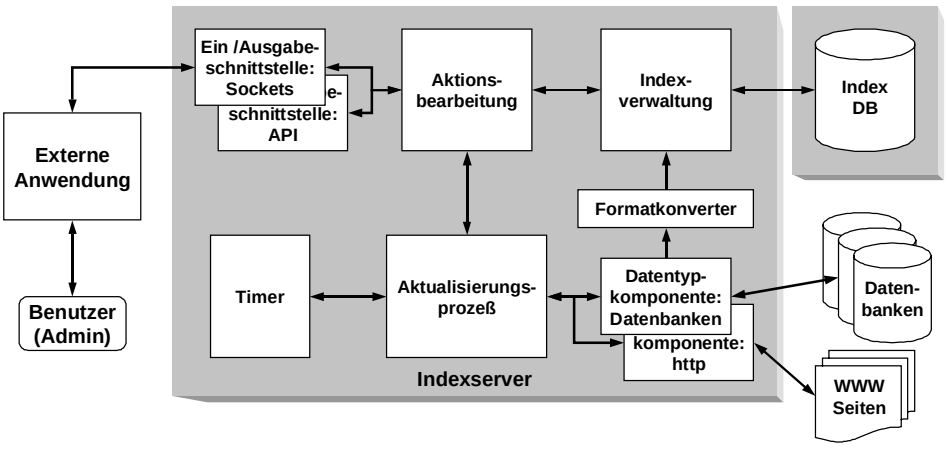

Abbildung 17: Prototypdesign Indexserver

Für die Anbindung der Datenquellen wurde das Framework ebenfalls überarbeitet. Die unteren zwei, bei Datenbanken die unteren drei,

Schichten (*access unit* bis *generic interface unit* in Abbildung 13) wurden für die verschiedenen Datentypen (z.B. Datenbank, WWW) in jeweils einer *Datentypkomponente* zusammengefaßt. Der Grund dafür ist, daß sie kaum identische Programmlogik aufweisen, für wenige Datentypen (1-2) dieser Ansatz geringeren Aufwand verspricht und die Erweiterung auf einen neuen Datentyp durch Hinzufügen einer einzigen Komponente problemlos möglich ist. Eine separierte Ebene für das Datenformat, hier als *Formatkonverter* bezeichnet, bleibt bestehen, da ein Textformat in unterschiedlichen Datentypen vorkommen kann und sich diese Schicht gut als allgemeiner Filter realisieren läßt.

Um den Umfang der *Datentypkomponenten* einzuschränken, wurde der Abgleich des Index mit den Datenquellen in einen eigenen globalen *Aktualisierungsprozeß* verlegt. Da dieser nicht auf Besonderheiten der einzelnen Datenquellen eingehen kann, kommt für die automatische Aktualisierung nur eine Zeitsteuerung in Frage. Mittels eines programmierbaren Zeitgebers (*Timer*) werden die Aktionen zur Aktualisierung gesteuert.

Wie schon weiter oben begründet, ist die Architektur des *Agentenservers* (Abbildung 18) ähnlich der des *Indexservers* (Abbildung 17). Die Komponenten der *Schnittstellen*, *Aktionsbearbeitung*, *Timer* und *Aktualisierungsprozeß* arbeiten wie weiter oben beschrieben. Die Aktionsbearbeitung muß außer den administrativen Funktionen vor allem Methoden für die nutzerspezifische Verwaltung der Agenten bereitstellen.

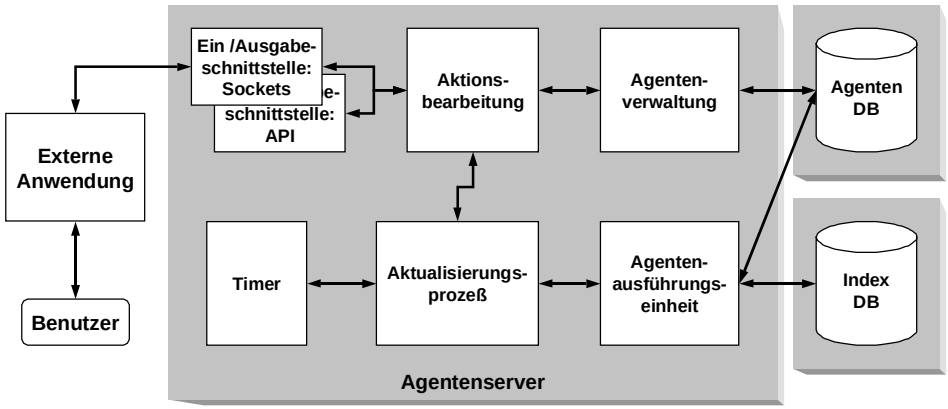

Abbildung 18: Prototypdesign Agentenserver

Um den begrenzten Hardwareressourcen im Hinblick auf eine große Anwenderzahl zu begegnen, sind die Agenten nicht als separate, kontinuierlich laufende Prozesse ausgelegt. Der jeweilige Zustand eines Agenten ist in einer *Agentendatenbank* festgehalten und wird, wenn der Agent in Aktion treten soll (z.B. für den Abgleich neuer Indexeinträge mit seinem Profil), von der *Agentenausführungseinheit* ausgelesen. Diese bildet daraufhin eine Instanz des entsprechenden Agent. Der Vorteil dieser Vorgehensweise liegt unter anderem darin, daß zu einem Zeitpunkt nicht von allen Agenten Hauptspeicher- und CPU-Ressourcen beansprucht werden, vielmehr geschieht dies nur von den aktuell aktiven Agenten.

Die *Agentenausführungseinheit* bildet sozusagen den eigentlichen parametrisierbaren Agenten. Sie ist für die Suche nach relevanten Informationen im Index als auch für deren temporäre Speicherung, gruppiert nach den jeweiligen Agenten, zuständig. Damit ist es relativ einfach möglich, die Aktivitäten der einzelnen Agenten zeitlich zu koordinieren. Sowohl der sequentielle Zugriff der Agenten auf den Index als auch die kontrollierte parallele Suche durch mehrfache Instanzen der *Agentenausführungseinheit* lassen sich realisieren.

Die Steuerung der Aktivierung von Agenten erfolgt wieder durch den *Aktualisierungsprozeß*. Dieser kann sowohl individuelle zeitliche Vorgaben der Agenten berücksichtigen als auch nach globalen Regeln vorgehen (z.B. Ausführung bei niedriger Systemlast oder zu Zeitpunkten mit gewöhnlich niedriger Auslastung).

Auf die Spezifizierung einer speziellen Ausgabekomponente wurde verzichtet, da in den anvisierten Projekten davon ausgegangen werden konnte, daß die entsprechenden Anwendungen ebenfalls Zugriff auf die Datenbank mit den Ergebnissen der einzelnen Agenten besitzen. Dies ist auch vorteilhaft, da die universelle Aufbereitung der Resultate für beliebige Anwendungen nur sehr aufwendig realisiert werden kann. Das liegt an den für eine individuelle Präsentation der Daten im Front-end der Anwendung zu berücksichtigenden Parametern. Dazu gehört die Bestimmung des zurückzuliefernden Elements (nur Referenz, Überschrift zur Fundstelle, Textbereich), dessen Umfang (wenige Wörter, Absatz, Zusammenfassung) sowie seine Formatierung.

5.3 Implementierung

5.3.1 Kriterien zur Auswahl der Programmiersprache

Die Wahl der Programmiersprache zur Umsetzung eines Projektes kann entscheidend für die Entwicklungszeit und den Erfolg sein. Die Auswahl sollte deshalb sorgfältig durch Vergleich der Leistungsfähigkeit der Sprache mit den Anforderungen des Projektes erfolgen. Für die Realisierung des universellen *News Watcher*-Agentensystems fiel nach eingehender Prüfung die Entscheidung auf Java von SUN Microsystems. Folgende Gründe waren ausschlaggebend:

- Java ist für nahezu jede Rechnerplattform verfügbar und binärkompatibel, d.h. ohne Neuübersetzung lauffähig. Dadurch wird der universellen Charakters des *News Watcher*-Systems unterstützt.
- Java verfügt über eine leistungsfähige Datenbankschnittstelle (JDBC) für die von praktisch allen Datenbankherstellern Treiber existieren und die den Austausch des DBMS durch Wechsel des Treibers ermöglicht (sogar zur Laufzeit). Damit bleibt die Unabhängigkeit von bestimmten Herstellern gewahrt und vorhandene Software kann genutzt werden.

- Die objektorientierte Programmiersprache unterstützt den Objektcharakter von Agenten sowie die Aufteilung der Programmlogik (Zerlegbarkeit) und die Wiederverwendbarkeit von Codeteilen (Zeitvorteil). Weiterhin ermöglichen Vererbung und abstrakte Klassen die Auswahl spezifischer Komponenten (z.B. Datentyp, Formatfilter) zur Laufzeit.
- Durch das RMI-Konzept (*Remote Method Invocation*) und einfacher Socketprogrammierung wird die Implementierung verteilter Programme unterstützt. Dadurch wird eine bessere Skalierbarkeit erreicht sowie die von der externen Anwendung getrennte Ausführung ermöglicht.
- Mittels dynamisch nachladbarer Klassen können zur Laufzeit Schnittstellen, Filter und unterstützte Datentypen erweitert werden.
- Java besitzt ein klares objektorientiertes Basismodell, automatische Speicherplatzverwaltung (*garbage collection*) und verzichtet auf Zeiger. Dadurch wird ein ‚sauberes' Design erzwungen, welches weniger Fehlerquellen und damit kürzere Entwicklungszeiten ermöglicht (speziell Fehlerbeseitigung beschränkt sich im wesentlichen auf Fehler beim Umsetzen des Algorithmus).
- Der größte Nachteil von Java gegenüber anderen Programmiersprachen ist die durch den Interpreter bedingte langsamere Ausführung. Dieses ist jedoch für diese Anwendung vernachlässigbar, da der größte Teil der Ausführungszeit von der Datenbank beansprucht wird.

Die meisten anderen Programmiersprachen sind entweder nicht auf allen Plattformen verfügbar, nicht binärkompatibel, nicht objekt-orientiert oder unterstützen keine bzw. nur wenige DBMS.

5.3.2 Umsetzung der angepaßten Konzeption

Wie in dem Design für den Prototyp vorgesehen wurde das System als zwei voneinander unabhängige Server (Java-Anwendungen) entwickelt. Eine logische Sicht auf die Klassen, aus denen die Server bestehen ist im Anhang 7.1 Abbildung 22 für den Agentenserver und in Abbildung 23 für den Indexserver dargestellt. Die dafür verwendeten Symbole der UML-Notation sind in Abbildung 21 mit Erläuterungen aufgeführt.

Für den Prototyp wurde als Kommunikationsschnittstelle mit den Servern eine Socketverbindung implementiert, da diese die größtmögliche Flexibilität hinsichtlich des Verbindungsaufbaus mit anderen Anwendungen verspricht sowie eine netzwerkweite Kommunikation erlaubt. Als Protokoll wird eine ressourcenschonende Eigenentwicklung eingesetzt, die die Daten in durch Klammern hierarchisch verschachtelten Listen überträgt. Der Grund dafür ist, neben der minimalen Netzwerkbelastung, die einfache Implementierung der En- und Dekodiermodule, wodurch der Integrationsaufwand in externe Anwendungen verringert wird.

Zuständig für die Kommunikation auf Protokollebene ist die jeweilige *AdminProtokoll*-Klasse. Sie entspricht weitestgehend der im Design beschriebenen *Auftragsbearbeitung*. Ihr Aufgabenbereich umfaßt die Entgegennahme der Anforderungen des Clients, Auslösen der entsprechenden Aktionen und Zustellen der Ergebnisse bzw. Status der Bearbeitung (erfolgreich oder fehlerhaft). Weiterhin ist sie für die Autorisierungsprozedur verantwortlich, nach deren erfolgreichem Abschluß erst der Zugriff auf die anderen Funktionen freigegeben wird. Damit ein Angreifer, der die Verbindung zwischen Anwendung und Server abhört nicht einfach Kenntnis über die korrekte Autorisierung erhält, wird eine Kombination von zufällig generierten öffentlichen Schlüssel und geheimen festen Schlüssel, der sowohl Anwendung als auch Server bekannt ist, verwendet. Diese werden von der Anwendung miteinander verknüpft und an den Server zurückgeschickt, der das Ergebnis mit seinem eigenen privaten Schlüssel überprüft und bei Gleichheit die Verbindung für Anfragen öffnet.

Um die Integration der Server in ein bestimmtes Projekt ohne großen Programmieraufwand seitens der Anwendung durchzuführen, sind alle relevanten Parameter als Attribut-Wert Paare in einer lesbaren Form in einer Systemdatei zusammengefaßt und können mit einem beliebigen Editor angepaßt werden. Zu einem späteren Zeitpunkt kann für die Manipulation dieser Datei ein entsprechendes Front-end zum Einsatz kommen. Parameter, die sich zur Laufzeit ändern (z.B. verbundene Datenquellen) werden in einer Tabelle der Agentensystemdatenbank abgelegt.

Zur Unterstützung einer einfachen Installation wurden alle verwendeten Klassen in ein Archiv zusammengepackt. Damit braucht nur dieses Archiv zusammen mit den Initialisierungsdateien auf dem Zielrechner kopiert zu werden. Da die Größe des Archivs aufgrund des Interpreteransatzes von Java gering ist (siehe auch die statistischen Daten auf Seite 88), muß für den Index- und den Agentenserver kein separates Archiv angelegt werden. Je nach gewünschter Funktionalität werden die entsprechenden Klassen aus dem Archiv aufgerufen.

Indexserver

Der Zugriff auf den Index erfolgt durch den *IndexManager*, welcher der Indexverwaltung aus der Designphase entspricht. Die Aufgabe des ebenfalls dort angegebenen *Aktualisierungsprozesses* wird durch den *indexUpdateThread* wahrgenommen, der über die Klasse *scheduleList* sowohl intervall- als auch tageszeitbasiert das Update des Index veranlassen kann. Für das Update selbst ist der *Indexmanager* verantwortlich, der die bei ihm registrierten Komponenten für die unterschiedlichen Datentypen anweist, neue Daten zu übermitteln. Diese werden von einer eigenen Klasse *createIdx* verarbeitet, die mit Hilfe einer Filterklasse *wordSeparator* die Daten in Wörter zerlegt, diese mit einer Stoppwortliste vergleicht und in eine Liste neu aufzunehmender Token einträgt. Diese Liste wird mit weiteren Quellenangaben vom *IndexManager* in den Index eingefügt.

Für die Anbindung eines bestimmten Datentyps muß eine eigene Klasse von der abstrakten Klasse *dataType* abgeleitet werden, die Methoden für die Überwachung einer Datenquelle (Anmelden, Auslesen von Änderungen, Abmelden) besitzt. Die verfügbaren Datentypklassen werden beim Start des Indexserver aus den Parametern ausgelesen und dynamisch geladen. Das Hinzufügen weiterer Typen durch ein Kommando ist damit einfach zu realisieren, wird aber vom aktuellen Prototyp nicht unterstützt. Die Anbindung einer Datenquelle, für die eine entsprechende Datentypklasse existiert, ist jedoch zur Laufzeit über Kommandos möglich.

Derzeit ist nur eine Datentypklasse realisiert. Sie dient der Überwachung von relationalen Datenbanken (*DBsrcManager*). Ihr Konzept sieht vor, bei den überwachten Datenbanken einen Trigger automatisch zu erzeugen, der die Änderungen an einzelnen Tabellen überwacht und in einer Logtabelle speichert. Diese wird bei einer Indexaktualisierung ausgelesen und die darin referenzierten geänderten Daten an den *IndexManager* übergeben. Wichtig ist die Unterstützung von durch Fremdschlüssel verknüpften Tabellen. Da eine direkte Abbildung im Index nicht möglich ist, wird sichergestellt, daß bei einem Update eines Eintrages alle mit ihm verbundenen Einträge mit aktualisiert werden (zumindest wird der Zeitstempel dieser Einträge aktualisiert). In der Ergebnisliste der Agenten, die über verschiedenen Tabellen suchen, erscheinen dann die Einträge aller Tabellen und können von der Anwendung ausgewertet werden.

Die Anbindung einer neuen Datenbank erfolgt durch die Angabe von Verbindungsdaten (JDBC-Treiber, Login, Passwort) sowie durch die Übergabe einer Datei mit Strukturinformationen zur Datenbank. Diese Datei enthält den Namen der Datenbank, die zu überwachenden Tabellen mit den interessierenden Attributen, Angaben zum Primärschlüssel (um Datensätzen eindeutig zuordnen zu können) und eventuelle Fremdschlüsselverweise. Da sich die Fremdschlüsselverweise nur auf schon vorher definierte Tabellen beziehen dürfen wird vermieden, daß eine ringartige Verkettung auftritt.

Agentenserver

Wie der Indexserver besitzt auch der Agentenserver eine Entsprechung für den *Aktualisierungsprozeß* in der Klasse *agentResultUpdateThread*. Die Updatezyklen lassen sich hier ebenfalls entweder als Intervalle oder Tageszeiten spezifizieren. Wie im Design vorgeschlagen, wird das Update von einem *ResultManager* durchgeführt, der sich die notwendigen Parameter zu dem Agenten vom *agentManager* (Agentenverwaltung) besorgt und damit als Instanz des Agenten auftritt. Obwohl der *agentResultUpdateThread* so ausgelegt wurde, daß die Agenten nacheinander aktiviert werden, kann es zu parallel tätigen Agenten kommen, da der Agentenserver mehrere Verbindungen zur Anwendung zuläßt und diese über verschiedene Kommandos Einfluß auf die Aktivität einzelner Agenten nehmen können.

Durch die Parallelität (Multithreading[67]) war hier besonders darauf zu achten, daß die Klassen threadsicher ausgelegt sind, d.h. daß sie keine globalen Variablen verwenden, die von einem Thread ohne Wissen eines anderen geändert werden können und daß kritische Methoden, die auf bestimmte Ressourcen zugreifen, synchronisiert werden.

Statistische Daten

- Anzahl der Quelltextzeilen: 10725
- Anzahl der Klassen: 55
- Größe des JAR[68]: ca. 160 kB

5.3.3 Funktionsweise des Prototyps

Anhand von Anwendungsfällen wird in diesem Abschnitt die Funktionsweise des Prototyps erklärt. Gleichzeitig werden dabei die verwendeten Datenstrukturen beschrieben. Als wesentlich für das Verständnis des Prototyps werden folgende Anwendungsfälle erläutert:

- Hinzufügen von Datenquellen,
- Aktualisierung des Index,
- Anlegen eines Agenten,
- Aktualisierung der Ergebnisliste eines Agenten.

Die ersten beiden Fälle beziehen sich auf den Indexserver, während die letzten zwei den Agentenserver betreffen.

Hinzufügen von Datenquellen

Nach dem Start des Indexservers liest dieser die zuletzt beobachteten Datenquellen aus einer Systemdatenbank aus und fügt sie den für den jeweiligen Datentyp zuständigen Komponenten hinzu. Wie dieses abläuft, wird durch den weitgehend analogen Fall des Hinzufügens einer relationalen Datenbank als neue Datenquelle erklärt.

Der Administrator veranlaßt durch das Kommando ‚addSource(...)' (siehe Anhang 7.4) den Indexserver, eine neue Datenquelle zur Überwachung aufzunehmen. Als Parameter sind der Datentyp, ein Bezeichner für die Datenquelle sowie weitere datentypspezifische Angaben zu übergeben. Bei Datenbanken bestehen diese spezifischen Parameter aus dem Namen einer Datei mit JDBC-Verbindungsparametern und einer Datei mit Strukturdaten der Tabellen.

[67] Beim *Multithreading* laufen mehrere Ausführungsstränge (*Threads*) eines Programms gleichzeitig bzw. quasi-gleichzeitig (falls weniger Prozessoren als *Threads* verfügbar sind, teilen sich mehrere *Threads* einen Prozessor über ein Zeitscheibenverfahren)

[68] Archivdatei, in der alle neu entwickelten Klassen enthalten sind (JAR steht für Java Archive und bedeutet wörtlich ‚Schüssel')

Als nächstes wird von der *Aktionsverarbeitung* überprüft, ob eine *Datentypkomponente* für die Datenquelle zur Verfügung steht. Gibt es eine solche, werden die Parameter an diese mittels der Methode ‚addSource()' weitergeleitet. Die Komponente ist dann für die weitere Überwachung der neuen Datenquelle zuständig.

Im besprochenen Beispiel ist der ‚DBsrcManager' (siehe auch Anhang 7.1) für die Datenquelle zuständig. Mittels der Angaben für die JDBC-Schnittstelle wird eine Verbindung zu der zu beobachtenden Datenbank hergestellt. Um die Änderungen in der Datenbank verfolgen zu können, werden eine Log-Tabelle sowie Trigger für alle zu überwachenden Tabellen (die Information wird der Strukturdatei entnommen) erzeugt. War dies erfolgreich, trägt die Komponente die Datenquelle in eine Liste von zu überwachenden Datenbanken ein. Falls im Kommando angegeben war, daß auch die bisherigen Informationen der neuen Datenquelle suchbar sein sollen, wird der Inhalt aller zu beobachtenden, d.h. zu indexierenden, Attribute der Tabellen ausgelesen und in den Index aufgenommen. Dieser Vorgang der Aufnahme von Daten in den Index, wird im folgenden Anwendungsfall beschrieben.

Aktualisierung des Index

Da die Agenten ausschließlich im vom Indexmanager verwalteten Index suchen können, muß dieser kontinuierlich mit den Datenquellen abgeglichen werden. Dies erfolgt beim Prototyp über eine Zeitsteuerung. In der derzeitigen Version werden alle beobachteten Datenquellen gemeinsam aktualisiert. Die Zeitpunkte für Aktualisierungen kann der Administrator durch die Spezifizierung von Intervallen (Kommando ‚setIndexUpdateIntervall') oder Tageszeiten (Kommando ‚addFixedIndexUpdateTime') vorgeben. Mit diesen Angaben wird der *Timer* (siehe Abbildung 17) programmiert.

Der im eigenen Thread laufende *Aktualisierungsprozeß* wird vom *Timer* über eine anstehende Aktualisierung benachrichtigt. Dieser fordert daraufhin alle Datentypkomponenten der Reihe nach auf, die Änderungen der von ihnen beobachteten Datenquellen an den Index weiterzuleiten. Die Datenbankkomponente vollzieht danach für alle angebundenen Datenbanken folgende Schritte:

- für jede einzelne Tabelle wird die Log-Tabelle mit den Verweisen auf die geänderten Datensätze ausgelesen,
- für jeden geänderten Datensatz werden die Inhalte der zu indizierenden Attribute ausgelesen und einzeln, versehen mit Angaben zur Herkunft, Format (derzeit nur Unterscheidung zwischen ANSI-Text, Datum und ID) und Zeitstempel der Aktualisierung an den Indexmanager übergeben (vorher wurde der Indexmanager angewiesen, existierende Einträge für diesen Datensatz zu löschen),
- sind mit der aktuellen Tabelle weitere Tabellen über Fremdschlüssel verbunden, so werden alle Datensätze, die mit dem aktuell bearbeiteten verknüpft sind, ebenfalls als geändert betrachtet und an den Indexmanager übergeben,

- nachdem alle Tabellen abgearbeitet sind, werden die Einträge aus der Log-Tabelle gelöscht, die zum Zeitpunkt des Auslesens dieser Tabelle vorhanden waren (dadurch können Änderungen, die während des Abgleichs in den Originaldaten vorgenommen wurden, bei der nächsten Aktualisierung berücksichtigt werden)

Im Idealfall finden alle Indexänderungen innerhalb einer Transaktion statt. Durch die begrenzte Größe des *Rollback*-Puffers[69] können jedoch nur eine bestimmte Anzahl von Änderungen während einer Transaktion realisiert werden (die Größe wird durch einen Wert in der Initialisierungsdatei festgelegt). Dadurch kann es im schlechtesten Fall dazu kommen, daß ein Agent während der Aktualisierung des Index diesen durchsucht und fremdschlüsselverknüpfte Inhalte noch nicht komplett vorliegen.

Der Indexmanager übergibt den zu indizierenden Text, den Formattyp sowie eine Stoppwortliste an eine *Tokenizer*-Klasse (‚createIdx'). Diese liefert eine Liste aller im Text vorkommenden Wörter (ausgenommen der Stoppwörter) zusammen mit ihrer Wortfrequenz[70]. Jeder Eintrag dieser Liste wird mit Quellenangabe und Zeitstempel in den Index aufgenommen.

Anlegen eines Agenten

Für jede Anfrage an den Index muß ein neuer Agent angelegt werden. Ein Agent ist deshalb einem Benutzer eindeutig zugeordnet und liefert, solange er aktiviert ist, kontinuierlich den Vorgaben entsprechende Neuzugänge im Index. Die für die Erstellung des Agenten notwendigen Profildaten[71] gliedern sich deshalb in:

- Benutzerdaten (Benutzer-ID, Email-Adresse, Agentenname),
- Aktionsdaten (Intervall der Benachrichtigung, Zeitstempel der letzten Ergebnisaktualisierung, Zeitstempel der letzten Benachrichtigung, Aktivierungsflag) und
- Suchausdruck.

Der Suchausdruck besteht aus Sucheinträgen, d.h. einer Liste von (Teil)-Wörtern, die alle mit Parametern versehen sind. Neben der Angabe des Suchraums[72] sind das ein Operator (‚<', ‚>', ‚<>', ‚=', ‚like'), der die Art des Vergleichs des Wortes mit dem Suchraum angibt und ein boolescher Wert (‚AND', ‚OR'). Für das Verständnis der Suchraumbeschreibung sowie der Wirkungsweise des booleschen Wertes ist die Kenntnis der Indexarchitektur aus Anhang 7.2 notwendig.

[69] Puffer, der alle Änderungen während einer Transaktion aufnimmt; erst nach erfolgter Bestätigung der Gültigkeit aller Aktionen (*commit*) wird sein Inhalt freigegeben

[70] Anzahl des Vorkommens eines Wortes im Text

[71] Siehe auch Kommandobeschreibung für ‚createAgent' im Anhang 7.5

[72] Definiert die Teilmenge des Index, auf die sich ein Sucheintrag bezieht

Die Beschränkung des Suchraums kann über alle 3 Stufen des Index erfolgen (vertikale Unterteilung in der 3. Stufe). Wird z.B. nur die erste Stufe angegeben, bezieht sich der Sucheintrag auf die gesamte Datenquelle. Der boolesche Wert dient der Ergebniszusammenführung aus den einzelnen Sucheinträgen. Er wird im nächsten Anwendungsfall genauer beschrieben.

Die gesamten Profildaten werden der *Agentenverwaltung* (siehe Abbildung 18) übergeben, die diese überprüft und in zwei Tabellen der Agentendatenbank ablegt (eine Tabelle für Benutzer- und Aktionsdaten, eine für die Sucheinträge). Damit besitzt der Agentenserver alle notwendigen Daten, um einen entsprechend parametrisierten Agenten zu erzeugen. Um Speicherplatz und Rechenleistung zu sparen, geschieht dies nur, wenn der Agent seine Ergebnisliste aktualisieren soll. Die andere Zeit befindet er sich sozusagen in einem ‚schlafenden' Zustand.

Aktualisierung der Ergebnisliste eines Agenten

Neben der Möglichkeit, die Ergebnisliste eines einzelnen Agenten aktualisieren zu lassen (Kommando ‚updateAgentResults'), gibt es wie beim Index die zeitbasierte automatische Aktualisierung aller Agenten. Auch hier können Intervalle und Tageszeiten vorgegeben und vom *Timer* verwaltet werden. Der *Aktualisierungsprozeß* ruft beim Erreichen eines vom *Timer* registrierten Zeitpunktes die *Agentenausführungseinheit* auf, mit der Aufgabe, die Ergebnislisten aller Agenten zu aktualisieren.

Die *Agentenausführungseinheit* liest der Reihe nach die Profildaten aller aktivierten Agenten aus und startet einen Suchprozeß mit den jeweiligen Suchausdrücken. Dieser kann damit als Instanz des Agenten betrachtet werden. Die vom Suchprozeß gelieferten Resultate (Referenzen auf die Originaldaten) werden in einer Ergebnistabelle der Agentendatenbank unter der ID des Agenten abgelegt und können von der externen Anwendung ausgelesen werden. Wie der Suchprozeß den Suchausdruck auf den Index anwendet wird nachfolgend erläutert.

Die verschiedenen Sucheinträge, aus denen der Suchausdruck besteht, müssen zuerst geeignet zusammengefaßt werden. Dies erfolgt durch die Bildung von Gruppen mit gleichen Suchräumen. Innerhalb einer Gruppe wird nach Einträgen mit dem booleschen Wert ‚AND' und ‚OR' getrennt. Diese Werte beziehen sich jeweils auf eine *Dateneinheit* (siehe Anhang 7.2). Für jede Gruppe wird daraufhin eine Ergebnismenge in Form von Dateneinheiten ermittelt. Dazu werden für die ‚AND'-markierten Sucheinträge die Mengen der Dateneinheiten aus dem Suchraum, die dem jeweiligen Vergleichsausdruck (Operator und Suchwort) entsprechen, geschnitten, für die ‚OR'-markierten werden diese vereinigt. Die beiden resultierenden Mengen werden abschließend miteinander geschnitten.

Auf diese Weise erhält man für jeden im Profil spezifizierten Suchraum eine Ergebnismenge von Dateneinheiten. Für deren Zusammenfassung gelten folgende Regeln:

- die Ergebnismenge aus einem Suchraum S, der durch die Stufe x definiert wurde, wird mit der Ergebnismenge des Suchraums S_1, der sich aus S durch Einschränkung mittels Stufe x+1 ergibt, geschnitten (z.B. Datenquelle = Datenbank: Ergebnismenge aus Datenbankebene wird mit Ergebnismenge aus Tabellenebene geschnitten); d.h. innerhalb eines Pfades der Suchraumhierarchie werden die Ergebnismengen der einzelnen Ebenen miteinander geschnitten
- Ergebnismengen der einzelnen Pfade der Suchraumhierarchie werden miteinander vereinigt;
  Ausnahme: Pfade, die sich nur in der 3. Stufe unterscheiden werden wie ein Suchraum behandelt, d.h. die Ergebnismenge für alle „AND" Einträge wird mittels Schnittmenge gebildet, die für „OR" Einträge mittels Vereinigung und danach werden diese Mengen zusammen mit der aus den oberen Hierarchiestufen geschnitten

Am Ende liefert diese Vorgehensweise eine einzige Ergebnismenge. Mit diesem Ansatz kann ein Agent für die Suche über mehreren Datenquellen unterschiedlichen Typs definiert werden (ein Beispiel für die Verarbeitung eines Suchausdrucks gibt Anhang 7.3).

5.4 Test

Bevor der Prototyp des universellen News-Watcher Agentensystems in den im Kapitel 5.5 beschriebenen Projekten zum Einsatz kam, wurden während der Entwicklungszeit und vor der Freigabe der Version 1.0 umfangreiche Testläufe durchgeführt, um Programmfehler und Schwachstellen bei der Performanz zu erkennen und zu beseitigen.

Als Testsystem kam folgende Hard- und Software zum Einsatz:

- Hardware: Compaq PC mit Pentium II Prozessor, 233 MHz Prozessortakt, 96 MB Hauptspeicher, 4 GB Plattenkapazität
- Betriebssystem: Windows NT® 4.0 Service Pack 3
- Laufzeitumgebung des Agentensystems: Java™ Virtual Machine von Sun Version 1.1.7
- DBMS für Anwendungsdaten, Agentendaten und Index: Oracle 8.0.5
- JDBC-Treiber zum Zugriff auf die Datenbank: Oracles 8.0.5 OCI JDBC-Treiber

Zum Test des Indexserver erhielt das Agentensystem Zugriff auf die dynamischen Daten aus dem in 4.1.1 vorgestellten Projekt. Diese wiesen meist tägliche Neuzugänge sowie ein kontinuierliches Entfernen veralteter Daten auf und wurden in einem stündlichen Zyklus vom Indexserver überprüft. Nach Überarbeitung der einzelnen Inhaltsformate (Text, ID, Datum usw.) funktionierte das automatische Indexieren problemlos und ausreichend schnell (ca. 25 kB Daten pro Minute).

Bei den Testläufen zeigte sich auch, daß der Index durch die Angabe von Metainformationen zu jedem Eintrag (hierarchische Quellenangabe, Zeitstempel, Rankingwert) und durch die auf dieser Tabelle angelegten Indizes ein mehrfaches der eigentlichen Daten an Speicherplatz verbrauchte. Ein Aufteilen des in einer Tabelle gehaltenen Index auf mehrere miteinander verknüpfte Tabellen (Normalisierung) könnte diesem entgegenwirken, jedoch nur, wenn durchschnittlich mehr als drei Token pro Dateneinheit zu indexieren sind. Bei den aktuellen Projekten dienten jedoch vor allem Datenbanken als Datenquelle, die teilweise nur ein zu indexierendes Element pro Einheit aufwiesen.
Weiterhin muß geklärt werden, inwiefern eine Verteilung auf mehrere Tabellen die Zugriffszeit sowie den benötigten Speicher bei der Suche beeinflußt, da erst mit JOIN-Operationen die Tabellen miteinander zu verknüpfen sind. Weil jedoch für diesen Prototyp der Schwerpunkt auf der Umsetzung eines *universellen* News-Watcher Agentensystems lag, konnte für die Optimierung des Index keine Zeit aufgewendet werden.

Für den Agentenserver wurde ein Belastungstest entwickelt, der die gleichzeitige Aktivität mehrerer Benutzer (bzw. mehrerer Verbindungen zur Anwendung) simuliert, da Probleme vor allem bei einem parallelen Arbeiten und kontinuierlichem Indexzugriff zu erwarten waren. Während des Tests, der 4 Tage dauerte und in dem 5 Clients gleichzeitig Aktionen auf dem Server initiierten und dann für 0-10 Sekunden pausierten, lief der Agentenserver stabil und es wurden jeweils die erwarteten Antworten an die Clients zurückgeschickt. Die angeforderten Aktionen reichten von der Verwaltung der Agenten (erstellen, löschen) bis zur augenblicklichen Aktualisierung der Ergebnisliste. Weiterhin mußte der Agentenserver die zyklischen Aktualisierungen der registrierten Agenten sowie sporadische Aktionen von Benutzern verarbeiten. Die Logdatei, die während dieses Zeitraums auf über 40 MB anwuchs, hatte ebenfalls keine Fehler verzeichnet. Insgesamt mußte der Agentenserver in dieser Zeit rund 290.000 Anfragen bearbeiten.

Vor diesem endgültigen Testlauf erfolgten einzelne Tests unter Verwendung von Oracle Version 8.0.3. Dabei traten sporadische, nicht direkt reproduzierbare Ausnahmefehler (Exceptions) während der Kommunikation mit der Datenbank auf. Es stellte sich heraus, daß hierfür der JDBC-Treiber von Oracle verantwortlich war. Eine vorübergehende Lösung stellte die Verwendung von Suns JDBC-ODBC Bridge dar, die den Zugriff auf einen unter Windows installierten ODBC-Treiber für die Datenbank ermöglicht. Durch die Konzeption des Agentensystems konnte dies einfach durch die Änderung von wenigen Parametern in der Konfigurationsdatei erledigt werden. Änderungen am Programm selbst waren nicht notwendig.

Insgesamt kann nach den Testläufen gesagt werden, daß der Prototyp, wie er hier implementiert wurde, für Datenquellen im zweistelligen Megabytebereich sowie für einige 100 Agenten auf einem zeitgemäßen Rechnersystem ausreicht. Besonders durch die Flexibilität der JDBC-Schnittstelle zusammen mit der Abstraktionsschicht für Datenbanken im Agentensystem konnte ein Aspekt des universellen Anspruchs realisiert

werden. Zur Unabhängigkeit gegenüber den Projekten, in denen das System eingesetzt werden soll, gibt der nächste Abschnitt Auskunft.

5.5 Betrieb

Nachdem sich das Agentensystem als stabil und ausreichend performant gezeigt hatte, mußte sich noch zeigen, ob sich die Integration in ein Projekt einfach durchführen läßt und ob es den jeweiligen Ansprüchen genügt. Während meines Aufenthaltes am Fraunhofer IAO boten sich dafür die zwei Projekte an, die schon unter 4.1 vorgestellt wurden.

Das endgültige Agentensystem ist in einer eigenen Dokumentation [Boe98] mit Schwerpunkt auf die Integration in eine Anwendung beschrieben. Dort wurde auch der Name dafür festgelegt: UNWAS (für *Universal News-Watcher Agent System*).

5.5.1 Handwerks-Centrum

Nachdem in 4.1.1 die Zielrichtung und Konzeption dieses Projektes vorgestellt wurden, folgt hier die Darstellung des technischen Umfelds, in welches das Agentensystem integriert werden soll. Danach werden die Arbeitsschritte zum Anpassen beschrieben und zum Schluß Screenshots vom laufenden System aus Anwendersicht gezeigt.

„Handwerks-Centrum“ ist ein verteiltes dynamisches WWW-System. Es basiert auf einem Microsoft Internet Information Server (WWW-Server, Windows NT® 4.0) gekoppelt mit einer ORACLE-Datenbank, welche über die Middleware des Microsoft Internet Information Server, Active Server Pages (ASP), angesprochen wird. Die WWW-Seiten werden „on-the-fly“ (dynamisch) generiert. Dabei wird die Anwendungslogik durch ASP-Module realisiert. Die Programmierung erfolgt mit Visual Basic Script. Komplexere Funktionen werden als Komponenten ausgelagert und über COM angesprochen.

In dieses Systems wurde UNWAS integriert, ohne daß an der bestehenden logischen Struktur Änderungen vorgenommen werden mußten. Um von den ASP-Seiten die Socketverbindung zu nutzen, wurde eine COM-Komponente in Java realisiert (mittels J++ unter Verwendung der Java-Virtual-Machine von Microsoft), welche die Socketkommunikation als Methodenaufrufe kapselt und dabei auch die Paßwortberechnung erledigt. Für das En- und Dekodieren der Parameter in und aus der Klammernotation wurde eine zweite COM-Komponente erstellt, welche die entsprechende Funktionalität der Serverschnittstelle der Anwendung zur Verfügung stellt.

Die Notwendigkeit für die Bereitstellung der Komponenten liegt darin begründet, daß Visual Basic Script nicht für die Socketprogrammierung geeignet ist und auf die Unterstützung von in anderen Sprachen geschriebenen Komponenten angewiesen ist. Durch den Einsatz von Java

waren die Komponenten jedoch einfach zu realisieren. Da COM ein Komponentenmodell vornehmlich für Windows ist, widerspräche es dem Ansatz des UNWAS, dieses direkt als COM-Komponente auszulegen. Durch die Erstellung der externen Komponente bleibt die Unabhängigkeit des UNWAS von bestimmten Betriebssystemen gewahrt und beliebige Windowsprogramme profitieren von der zweiten Möglichkeit, auf das Agentensystem zuzugreifen.

Die Anpassung der Systemdateien des UNWAS war in diesem Fall unkritisch, da das „Handwerks-Centrum“ bereits während der Entwicklung des Agentensystems für den Test des Indexservers als Datenquelle verwendet wurde. Vor allem das Front-End für die Agentenverwaltung durch den Benutzer sowie die Auswertelogik für die von den Agenten gelieferten Ergebnisse mußten noch im Projekt implementiert werden.

Abbildung 19 zeigt die Seite zum Einrichten eines Agenten – hier Assistent genannt – durch den Benutzer. Neben dem Namen des Agenten muß ein Suchprofil ausgefüllt werden, daß für die Bestimmung relevanter Neuzugänge in den Daten notwendig ist. Dabei wird von dem Formular eine Einschränkung der möglichen Suchparameter vorgenommen. Für die korrekte Kombination dieser Parameter ist die Anwendung verantwortlich.

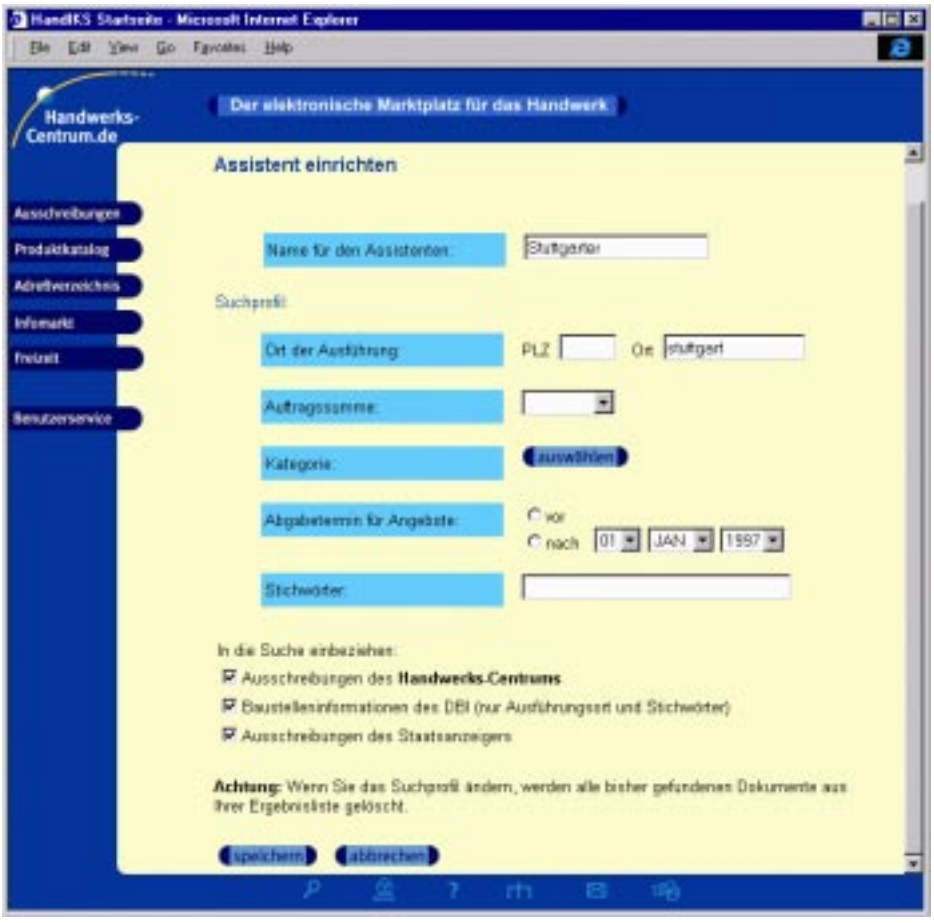

Abbildung 19: Screenshot Handwerks-Centrum - Agenten einrichten

Weitere Screenshots, welche die agentenspezifischen Teile des Frontends zeigen, sind im Anhang 7.2 zu finden. Dort wird in Abbildung 26 die Übersichtsseite zur Verwaltung der einzelnen Agenten eines Benutzers gezeigt, Abbildung 27 zeigt die individuelle Startseite eines Benutzers, auf der ihm der aktuelle Stand der gefundenen Ergebnisse seiner Agenten mitgeteilt wird und Abbildung 28 zeigt die Ergebnisliste eines bestimmten Agenten.

5.5.2 ELPRO

Nach einer Evaluierungsphase verschiedener Anbieter agentenorientierter Lösungen (unter anderem die in 4.4 vorgestellten), fiel die Entscheidung für den Einsatz des UNWAS als unkomplizierte, flexible Middlewarelösung.

Der ELPRO-Server, der auf einem Intel-PC unter Windows NT 4.0 läuft, basiert auf dem Internet Information Server von Microsoft als WWW-Server, wobei Cold Fusion zur dynamischen Generierung der HTML-Seiten eingesetzt wird. Die Ausschreibungsdaten liegen in einer Datenbank, die unter Oracle Version 7.3 läuft. Ein Prototyp von ELPRO ist unter http://www.elpro.net erreichbar.

Die Verbindung zwischen der Anwendungslogik von ELPRO mit dem Agentensystem wird durch die im vorherigen Projekt entwickelten COM-Komponenten realisiert, da auch hier die Skriptsprache für die Socketprogrammierung schlecht geeignet ist.

Außer der Programmierung der Benutzerschnittstelle der Agenten in ELPRO mußte als wesentliche Anpassungsarbeit eine Beschreibungsdatei für die Ausschreibungsdatenbank erstellt werden, die deren Struktur und die zu indizierenden Attribute beinhaltet. Längere Zeit beanspruchte die erstmalige Indizierung der gesamten Datenbank durch den Indexserver, um den Index auf den gleichen Stand, wie die Datenbank zu bringen. Insgesamt beanspruchte die manuelle Anpassungen der Benutzerschnittstelle die meiste Zeit. Wie hoch diese ausfällt, hängt nicht unwesentlich von der Architektur der Anwendung und der verwendeten Programmierumgebung ab.

Der vorangegangene Screenshot sowie die in Anhang 7.7 gezeigten Bilder demonstrieren die Integration des Agentensystems in ELPRO aus Benutzersicht. Abbildung 20 zeigt die Maske zur Erstellung eines Agenten. Die Eingabefelder sind entsprechend der Datenstruktur der Ausschreibungsdaten angelegt und damit ausschließlich auf diese Datenquelle ausgerichtet. In Abbildung 29 ist die Verwaltungsseite aller Agenten eines Benutzers zu sehen und in Abbildung 30 die Zusammenfassung der gefundenen Informationen eines Agenten.

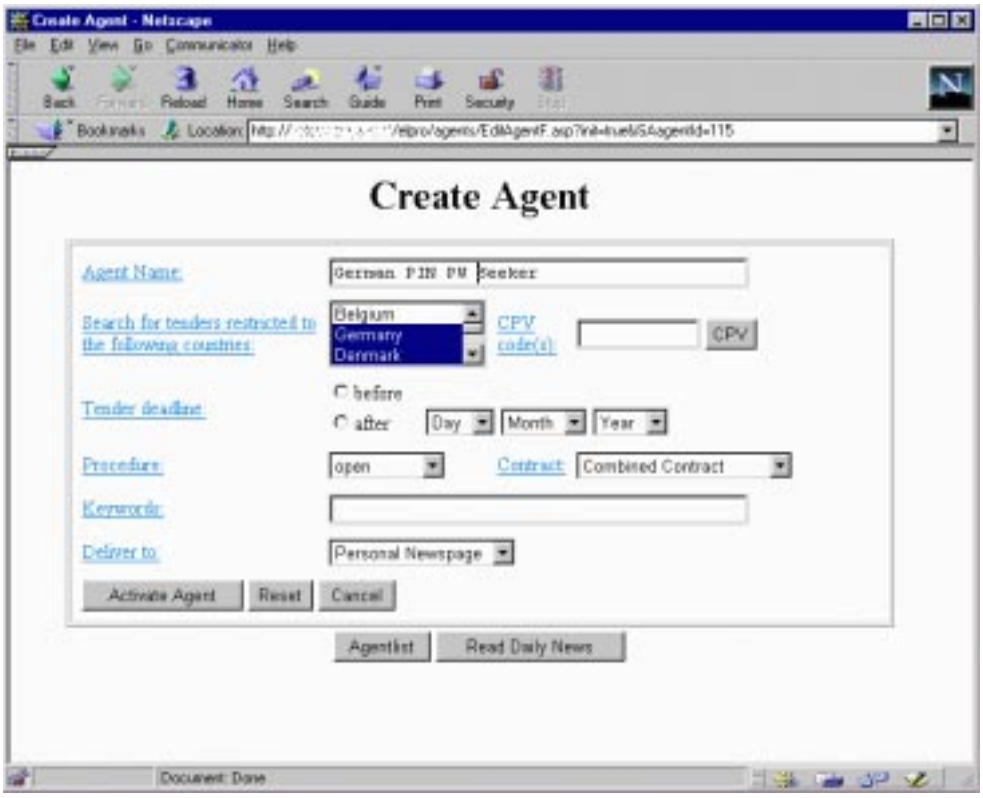

Abbildung 20: Screenshot ELPRO - Anlegen eines Agenten

5.6 Entwicklungsstand und Ausblick

In diesem Abschnitt werden die Merkmale des aktuellen Prototyps übersichtlich im Vergleich zur Anforderungsspezifikation des Framework aus Kapitel 4 zusammengestellt. Daraus werden sinnvolle kurzfristige Erweiterungen der aktuellen Implementierung als auch längerfristige Verbesserungen in der Architektur abgeleitet.

5.6.1 Merkmale der aktuellen Version

Anforderungen an universelles News-Watcher Agentensystem	Umsetzung im aktuellen Prototyp
Integration in verschiedene Anwendungen / Skalierbarkeit	
Unterstützung verschiedener Betriebsysteme und Rechnerarchitekturen	100 % reine Javaanwendung, d.h. er läuft auf allen Systemen mit virtueller Javamaschine ab Version 1.1
Ausführungsort unabhängig von integrierender Anwendung	Agentensystem ist für sich lauffähig und kann durch Socketkommunikation netzwerkweit angesprochen werden

Anforderungen an universelles News-Watcher Agentensystem	**Umsetzung im aktuellen Prototyp**
Aufteilung der Funktionseinheiten des Agentensystems im Netzwerk	Trennung von Agenten- und Indexserver; jeder kann für sich auf einem eigenen Rechner laufen; Verbindung nur über Indexdatenbank, auf die über das Netzwerk zugegriffen werden kann
paralleler Einsatz mehrerer Agentensysteme auf verschiedenen Rechnern	Mehrere Indexserver können einen gemeinsamen Index verwalten (wenn alle mit verschiedenen Datenquellen arbeiten) und mehrere Agentenserver können auf einen gemeinsamen Index zugreifen, die Daten der Agenten müssen jedoch in verschiedenen Tabellen – besser Datenbanken – liegen. Das Verwalten mehrerer Indextabellen durch einen Indexserver oder die Abfrage verschiedener Indextabellen durch einen Agentenserver ist nicht möglich.
generisches Konzept mit leichter Integrationsmöglichkeit in bestehende Umgebungen	
abgeschlossene, für sich lauffähige Komponente	✓
einfach zu implementierende Schnittstellen, für Funktionsaufruf und Parameterübergabe	Socketkommunikation mit Protokoll basierend auf durch Klammernotation hierarchisch gegliederten Listen (leicht in Java oder C zu implementieren); unter Windows stehen COM-Komponenten zur Verfügung, welche die Kommunikation auf einen Funktionsaufruf beschränken;
verschiedene Arten von Schnittstellen	Derzeit ist nur Socketkommunikation realisiert; unter Windows ist zusätzlich Kommunikation über COM-Komponente möglich (lokale oder netzwerkweite Verbindung); Keine API-Schnittstelle oder Java-Bean-Komponente[73] (lassen sich jedoch als Erweiterung einfügen oder auf die Socketschnittstelle aufsetzen)
Verwendung von standardisierten Protokollen, die von verschiedenen Systemen und Programmbibliotheken unterstützt werden	In dieser Version wird kein standardisiertes Protokoll verwendet

[73] Ein *Java Bean* ist eine wiederverwendbare Software-Komponente; das *Bean*-Konzept realisiert den ComponentWare-Ansatz für Java

Anforderungen an universelles News-Watcher Agentensystem	**Umsetzung im aktuellen Prototyp**
Verwendung vorhandener Datenverwaltungssysteme	Daten der Agenten sowie der Index können von einem beliebigen relationalen DBMS verwaltet werden, mit den Bedingungen, daß es Trigger und Transaktionen unterstützt sowie einen JDBC- oder ODBC-Treiber mitbringt
Einfache Möglichkeit zur Verwaltung des Agentensystems	Im wesentlichen können die Server durch Änderungen einzelner Parameter in den Systemdateien an ihren Einsatzort angepaßt werden. Weitere administrative Funktionen sind über dieselbe Schnittstelle wie die Verwaltung der Agenten erreichbar. Mitgelieferte kurze ASP-Seiten zeigen beispielhaft, wie ein einfaches Verwaltungswerkzeug als WWW-Formular aussehen kann (der Einsatz bedingt jedoch den Internet Information Server von Microsoft)
Bereitstellung allgemeiner Suchfunktionalität	
Unterstützung verschiedener Datenquellen	✓, die Datentypkomponenten können mehrere Datenquellen verwalten; der Index enthält die Einträge aller Datenquellen, so daß die Agenten parallel über alle Quellen suchen können
Unterstützung verschiedener Datentypen	✓, pro Datentyp wird eine Datentypkomponente benötigt; derzeit existiert nur eine für relationale Datenbanken
Unterstützung verschiedener Datenformate	✓, die Klasse zur Zerlegung der Daten in Token (Wörter, Zahlen usw.) kann verschiedene Formate unterscheiden; derzeit sind nur einfacher ANSI-Text, ID und Datum implementiert
Einbindung neuer Datenquellen zur Laufzeit	✓, durch ein Administratorkommando an den Indexserver; Datentypkomponente für Datenquelle muß geladen sein
Erweiterung auf neue Datentypen und –formate möglichst ohne Änderungen der Komponente	Neue Datentypen lassen sich durch erstellen einer entsprechenden Datentypkomponente hinzufügen, jedoch muß der Indexserver neu gestartet werden. Neue Formate lassen sich nur durch Änderung des Quelltextes hinzufügen.

Anforderungen an universelles News-Watcher Agentensystem	**Umsetzung im aktuellen Prototyp**
Kombinationsmöglichkeit der Suchparameter	Einzelne Suchparameter lassen sich mit ‚AND' oder ‚OR' kennzeichnen, je nachdem ob sie notwendigerweise vorhanden sein müssen (‚AND') oder nur wenigstens einer aus der Liste (‚OR'). Durch Angabe von ‚=', ‚<', ‚>', ‚<>' und ‚like' kann wie bei Datenbanken der Suchausdruck mit den Daten auf gleich, kleiner/größer, ungleich oder teilweise Übereinstimmung (Wildcards) verglichen werden.
allgemeines Modell für die Verknüpfung von Suchparametern bei verschiedenen Datentypen	Ausgehend von den Datenbanken als mit am stärksten strukturierten Datenquellen können die Daten über drei Hierarchieebenen lokalisiert werden (bei Datenbanken: Datenbank ➔ Tabelle ➔ Datensatz), zusätzlich kann für die Suche die letzte Hierarchieebene, die auch als Ergebnis zurückgegeben wird, nochmals unterteilt werden (bei Datenbanken: Attribute); Die Agenten können über alle Hierarchieebenen suchen, d.h. für die Suchparameter kann der Suchraum auf bestimmte Ebenen eingegrenzt werden und es lassen sich Suchparameter mit unterschiedlichem Skopus verbinden (z.B. ein Parameter sucht über die komplette Tabelle einer Datenquelle, der andere schränkt die Treffermenge durch eine Bedingung an ein Attribut ein)
automatisierte Ausführung von Anfragen	
Individuelle (anwenderbezogene) Speicherung der Anfragedefinition	Für jede Anfrage wird ein Agent erzeugt, der die Parameter als Profil enthält; Ergebnisse werden zusammen mit der ID des Agenten gespeichert
Bestimmung der Ausführungshäufigkeit gespeicherter Anfragen	Nur globale (für alle Agenten geltende) Vorgabe für Aktivitätszyklus (Intervall oder Tageszeit(en))
verschiedene Möglichkeiten der Ergebniszusammenstellung und –übermittlung	Da es projektabhängig ist, wieviel und in welcher Form von den Originaldaten als Ergebnis zusammenzustellen ist, werden nur die Ergebnisse als Verweise auf die Daten in einer globalen Ergebnistabelle abgelegt, auf welche die Anwendung Zugriff hat. Die Anwendung ist selbst für die Zustellung verantwortlich

Anforderungen an universelles News-Watcher Agentensystem	Umsetzung im aktuellen Prototyp
Sicherheit der Daten	
Zugriff auf Komponente nur autorisierten Programmen erlauben	Der Zugriff auf die Funktionen der Server wird erst nach erfolgter Autorisierung über eine Kombination aus festem geheimem und veränderlichem öffentlichem Schlüssel freigegeben (siehe auch Seite 86)
Direktzugriffe eines Benutzers nur nach Autorisierung und nur auf eigene Daten gestatten	Da ein Benutzer nur über die Schnittstelle der Anwendung auf den Agentenserver zugreift, steht nach einmaliger Autorisierung der Anwendung das gesamte Funktionsangebot offen. Die Anwendung muß selbst dafür sorgen, daß ein Benutzer nur über definierte Masken indirekt auf das Agentensystem Einfluß nehmen kann. Eine Sicherheitsstufe besteht darin, das beim Aufruf der Verwaltungsfunktionen für die Agenten immer die Benutzer-ID übergeben werden muß und diese wird von der Anwendung, entsprechend dem angemeldeten Benutzer, eingesetzt

Die Gegenüberstellung zeigt, daß viele der in der allgemeinen Konzeption aufgestellten Bedingungen von dem Prototyp erfüllt oder wenigstens zu einem bestimmten Grade umgesetzt worden. Die verbliebenen, nicht realisierten, Punkte sowie erste Erfahrungen beim Einsatz lassen genügend Raum für zukünftige Verbesserungen. Diese sollen als nächstes diskutiert werden.

5.6.2 Weiterentwicklung

Wie in den vorangegangenen Abschnitten gezeigt wurde, befindet sich der Prototyp in einem Stadium, in dem er sich bereits gut für den Einsatz in verschiedenen Anwendungen eignet. Trotzdem sind einige Verbesserungen im aktuellen Design empfehlenswert und auf längere Sicht sollten auch die bestehenden Unterschiede zwischen dem Ansatz im Framework und der aktuellen Architektur überwunden werden.

5.6.2.1 Unmittelbare Verbesserungen

Die derzeitigen Möglichkeiten, Updatezyklen für die Agenten festzulegen, sind unzureichend, da sie sich immer auf alle aktiven Agenten beziehen. Unterschiedliche Aktivitätssequenzen, angepaßt an die jeweiligen Vorgaben der Benutzer, würden zu Ergebnislisten führen, deren Aktualität den Vorstellungen der Benutzer entspricht und das Agentensystem könnte entlastet werden, wenn nicht jedesmal alle Agenten ihre

Ergebnisse aktualisierten. Um die Koordination der Agenten zu vereinfachen wäre es auch denkbar, daß verschiedene Aktivitätsgruppen mit eigenen Aktualisierungssequenzen angeboten werden, denen die Agenten zugeordnet werden können.

Die Größe des Index beträgt durch die Verwendung einer einzigen Tabelle und der dadurch vielfach wiederholten Quellenangabe ein mehrfaches der eigentlichen indizierten Daten. Dieses kann bei größeren Datenquellen schnell zu Speicherplatzproblemen führen, insbesondere wenn für Transaktionen temporäre Tabellen in einem Rollbackpuffer angelegt werden müssen. Die Auslagerung der Quellenangabe sowie des Zeitstempels der letzten Aktualisierung in eigene Tabellen, die über einen Schlüssel mit dem Index verbunden sind, kann zu einer wesentlichen Einsparung führen.

Die Filter für die Zerlegung von Daten in einem bestimmten Format in einzelne Token zur Aufnahme in den Index sind in einer einzigen Klasse realisiert, d.h. für neue Filter muß der Quellcode dieser Klasse geändert und das Agentensystem neu übersetzt werden. Wesentlich effektiver und anwendungsfreundlicher ist die Definition einer allgemeinen abstrakten Klasse für Filter von der spezialisierte Filterklassen abgeleitet werden. Diese Methode, die auch schon bei den Datentypkomponenten zum Einsatz kommt, erlaubt die Erweiterung auf neue Filter zur Laufzeit des Systems. Für die Datentypkomponenten muß dafür nur noch eine entsprechende Funktion vorgesehen werden.

5.6.2.2 Langfristige strukturelle Änderungen

Das aktuelle Agentensystem verfügt über keine Ergebnisverwaltung. Derzeit werden nur neue Resultate in eine Tabelle eingefügt. Die Anwendung muß sich um das Abholen, Auswerten und Löschen der Daten kümmern. Es ist notwendig, ein Modell zu erstellen, daß den universellen Umgang mit den Resultaten erlaubt, so daß die Agenten selbst diese Aufgaben wahrnehmen können. Dabei müssen der Umfang der gelieferten Daten (nur Verweis auf Originaldaten, Zusammenfassung, kompletter Text), deren Formatierung, Art der Zustellung und Regeln für das Löschen durch Parameter individuell einstellbar sein. Dieses entspricht dem *Result Manager* in Verbindung mit der *Output Unit* im Design des Framework.

Eine wesentliche Entlastung des Index kann erreicht werden, wenn das Konzept des *Notifier* ebenfalls in der Implementierung berücksichtigt wird. Dadurch entfallen Anfragen der einzelnen Agenten an den Index, solange sich dieser für die zu überwachenden Datenquellen nicht geändert hat. Vorerst negativ zu werten ist der damit verbundene Wegfall der völligen Autonomie von Agenten- und Indexserver. Sowohl der *Global Source Manager* auf Seite des Agentenservers als auch der *Data Source Manager* im Indexserver müssen, zumindest teilweise, realisiert und Schnittstellen für netzwerkweite Verbindungen bereitgestellt werden.

Wenn schon Verbindungen zwischen den beiden Servern bestehen, sollte auch der Index ausschließlich vom Indexserver direkt angesprochen werden und die Agenten greifen über ihn auf den Index zu. Dadurch entfällt die Notwendigkeit, den Index in einer Datenbank abzulegen und eine spezialisierte Software zum Indizieren einer Datenquelle kann, mit einem Wrapper versehen, die Funktion des Indexservers bei Bedarf übernehmen.

Durch die Einführung des *Global Source Manager* wird es ebenfalls möglich, daß ein Agent über mehrere Indizes von verschiedenen Indexservern sucht. Bisher konnten zwar mehrere Indexserver einen Index erstellen und mehrere Agentenserver darauf zugreifen, der zentrale Index dürfte sich dabei jedoch als Flaschenhals erweisen.

Damit die Agenten auch einen qualitativen Vorteil, d.h. eine Verbesserung der Aktualität ihrer Ergebnisse aus der Existenz des *Notifiers* ziehen können, müssen sie individuell auf dessen Benachrichtigung reagieren (in Aktion treten) können. Dazu ist es notwendig, daß die Agenten als selbständige Objekte (mit eigenem Thread) im Agenteserver vorliegen und nicht wie bisher von einer Komponente pro Aktion instanziiert werden. Dieses ist ebenfalls für eine agentengesteuerte Ergebnisverwaltung erforderlich.

6 Zusammenfassung

Informationsverwaltung, Verarbeitung komplexer Aufgabenstellungen und Automatisierung sind die Gebiete, in denen die Agententechnologie Lösungen anbieten und dem Menschen Arbeit abnehmen will. Dieser weite Spielraum verhindert eine allgemeingültige Definition zur Bestimmung eines Agenten. Klassifikation, die auf wesentlichen Eigenschaften von Agenten beruhen, können besser als Gradmesser zur Unterscheidung von Agenten und anderer Software verwendet werden.

Die bislang realisierten einfachen Anwendungen von Agenten, hauptsächlich zur Informationsfilterung, zeigen, daß sich ‚intelligentes' Verhalten nur bedingt in einem einzelnen Agenten umsetzen läßt, vielmehr sind Fähigkeiten zur Kooperation mit anderen Agenten und die Möglichkeit, den Ausführungsort zu wechseln, geeignet, als Herausstellungsmerkmal von Agenten angesehen zu werden und die Software qualitativ zu verbessern. Zur Umsetzung dieser Ziele sind jedoch Standards unerläßlich, welche die Zusammenarbeit und die Ausführung mobilen Codes regeln. Die Akzeptanz und der Erfolg dieser Standards hängen entscheidend davon ab, daß sie zum einen genaue Vorgaben zu Schnittstellen und Architekturen enthalten, zum anderen den Entwicklern viel Freiheit beim Entwurf und der Implementierung von Agenten einräumen. Vielversprechend sind dabei die Arbeiten von OMG und FIPA.

Der Einsatz von Agenten als Repräsentanten ihrer Benutzer mit Entscheidungsvollmacht ist derzeit noch nicht möglich, da wichtige Fragen der Vertrauenswürdigkeit in Entscheidungen und ‚sozialem' Auftreten sowie rechtliche Haftbarkeit von Anwendern und Software geklärt werden müssen. Bis dahin werden Agenten entweder auf Intranets oder die Interaktion mit kostenfreien Diensten beschränkt bleiben.

Das im Rahmen dieser Diplomarbeit entstandene universelle *News Watcher*-Agentensystem (der spätere Prototyp erhielt den Namen UNWAS – *universal news watcher agent system*) wurde in einem mehrstufigen Prozeß entwickelt. Ausgehend von einer Analyse der Anforderungen an eine allgemeine Überwachungskomponente für Informationsquellen sowie der notwendigen Eigenschaften von Agenten in einem solchen System erfolgte die Ausarbeitung eines Framework mit dem Schwerpunkt der Universalität. Es zeigte sich, daß stationäre reaktive Agenten für diese Aufgabe ausreichen, da sie zum Zugriff auf die zu überwachenden entfernten Datenquellen lokale Dienste in Anspruch nehmen können.

Der zweite Schritt bestand in der Überarbeitung des Framework, um es in der zur Verfügung stehenden Zeit als Prototyp realisieren zu können. Dafür mußten Vereinfachungen vorgenommen werden, die in Einschränkungen der Universalität und Skalierbarkeit sowie in stärkerer Anpassung an die als Einsatzziel vorgesehenen Projekte bestanden.

In der nächsten Phase wurde die angepaßte Variante realisiert. Dabei zeigte es sich, daß für die Entwicklung eines derartigen Agentensystems, in dem keine Kooperation zwischen den Agenten vorgesehen ist, herkömmliche Techniken der Softwareentwicklung ausreichend sind. Die Umsetzung des Prototypen erfolgte in Java, daß sich vor allem durch seine Plattformunabhängigkeit und durchgängig verfügbare Datenbankschnittstelle für diese Aufgabe qualifizierte.

Während der anschließenden Testphase wurden Implementierungsfehler beseitigt und das Laufzeitverhalten des Systems überprüft. Dabei konnten der Stabilität und Performanz gute Noten gegeben und damit das Design bestätigt werden.

Die Bewährungsprobe des universellen Anspruchs (auch wenn dieser in der angepaßten Variante etwas eingeschränkt wurde) war die Integration des UNWAS in zwei Projekte („Handwerks-Centrum“ und ELPRO). Dieses erwies sich als unproblematisch, wobei die Hauptanpassungszeit für die Entwicklung des Front-end benötigt wurde.

Abschließend wurden Erweiterungen und Verbesserungen des Prototypen diskutiert. Diese bestehen im wesentlichen in der vollständigen Umsetzung der Konzeption des Framework, d.h. die vorgenommenen Vereinfachungen sollten in einem späteren System entfallen.

7 Anhang

7.1 Logische Sicht auf die Klassenstruktur des universellen News-Watcher-Agentensystem Prototyps in UML-Notation

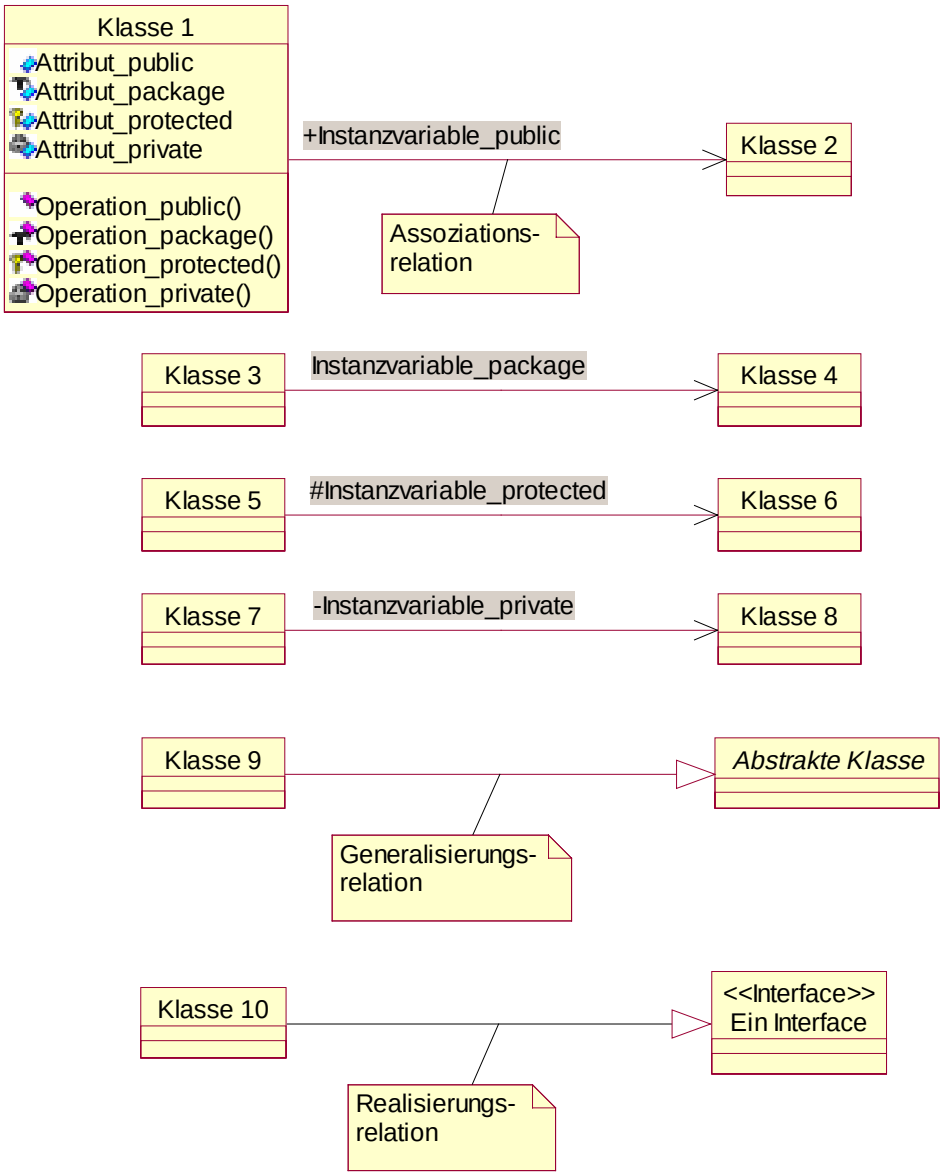

Abbildung 21: Legende der verwendeten UML-Symbole

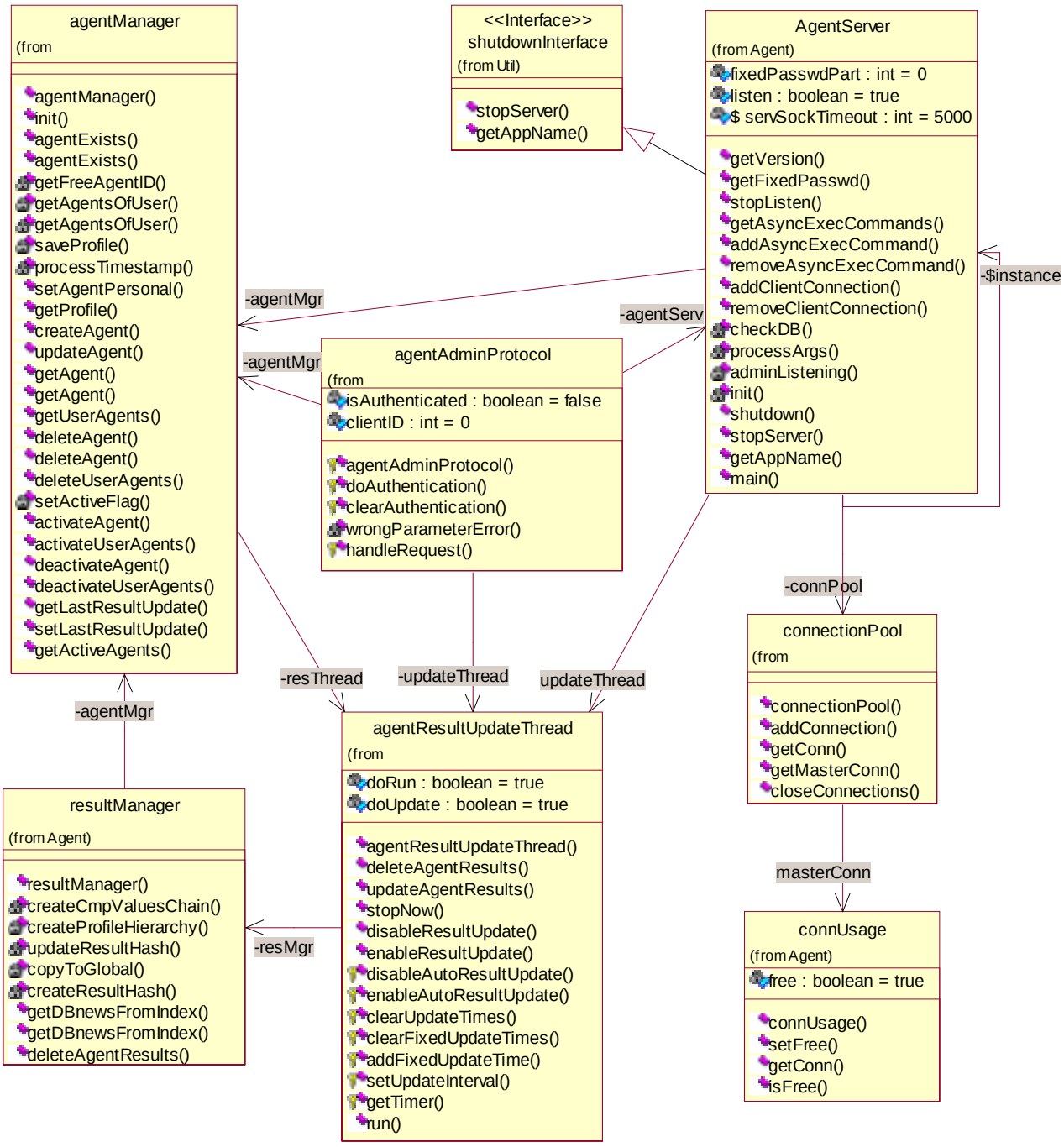

Abbildung 22: Logische Sicht auf Agentenserver (ausgewählte Klassen)

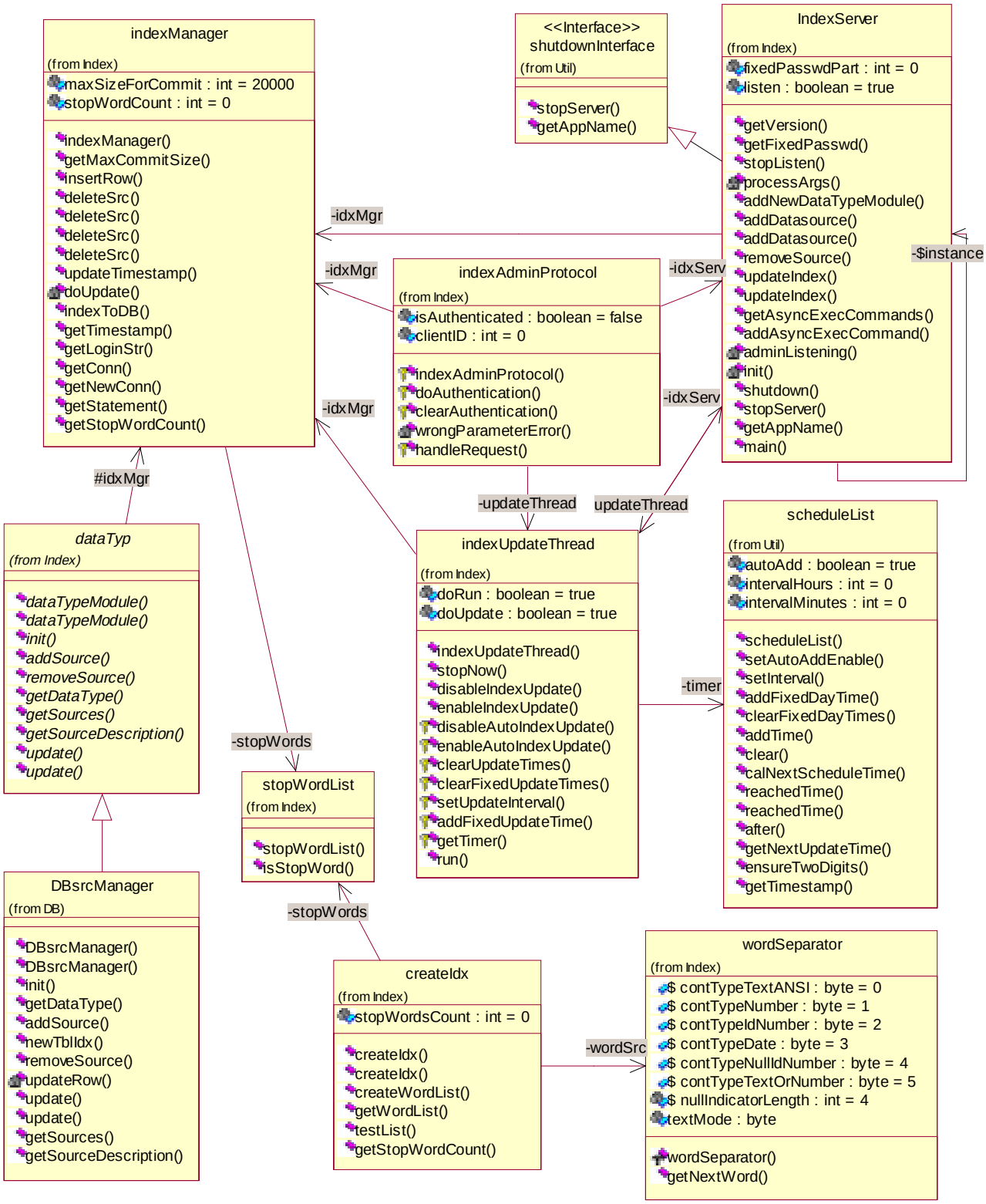

Abbildung 23: Logische Sicht auf Indexserver (ausgewählte Klassen)

7.2 Indexarchitektur

Der Aufbau des Index ist unter dem Gesichtspunkt einer einheitlichen Repräsentation der verschiedenen Datentypen vorgenommen worden. Er abstrahiert damit von den Besonderheiten der jeweiligen Datenquelle unter Beibehaltung der notwendigen spezifischen Informationen. Die Architektur des Index ist in Abbildung 24 dargestellt und wird im folgenden erläutert.

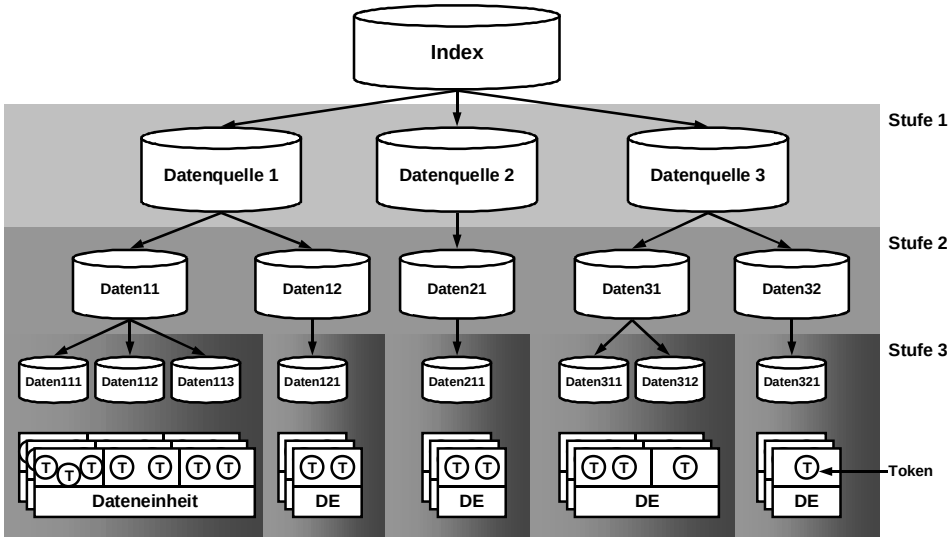

Abbildung 24: Indexarchitektur des Prototyps

Die im Index vorliegenden Daten werden durch 3 Hierarchiestufen unterteilt, um Suchraumbeschränkungen vornehmen und Fundstellen den entsprechenden Originaldaten zuordnen zu können. Die erste Stufe dient dem Separieren der einzelnen Datenquellen. In der nächsten Stufe erfolgt die Untergliederung der einzelnen Quellen. Die letzte Stufe gliedert die jeweiligen Teilmengen sowohl horizontal als auch vertikal. In der horizontalen Teilung werden die Daten in einzelne Dateneinheiten mit eindeutiger ID unterteilt, die als Ergebnis einer Suche zurückgeliefert werden und damit die Granularität der Ergebnisse bestimmen. Dies können z.B. Dokumente oder Datensätze sein. Die vertikale Teilung dient der feineren Beschränkung des Suchraumes auf Bereiche der Dateneinheiten, z.B. Überschriften und Abschnitte von Dokumenten oder Attribute von Datensätzen. Die folgende Tabelle verdeutlicht die Abbildung realer Datentypen auf die Indexstruktur.

Datentyp	Stufe 1	Stufe 2	Stufe 3 horiz.	Stufe 3 vert.
Datenbank	eindeutiger Name für die Datenbank	Tabellennamen	Wert des Primärschlüssels	Attributnamen
WWW	eindeutiger Name für Kollektion (z.B. Softwarehersteller)	HTTP-Adresse (Wurzelverzeichnis einer WWW-Site)	Pfadname einer HTML-Seite	Seitenbereiche (z.B. HEAD, TITLE, BODY)

7.3 Beispiel für die Verarbeitung eines Suchprofils

Das folgende Beispiel soll die Verarbeitung des Suchausdrucks der Profildaten eines Agenten bei dessen Ergebnisaktualisierung illustrieren. Als Datenquelle wird eine Datenbank über Gebrauchtwagen angenommen, die wenigstens zwei Tabellen für technische Daten und eine Preisliste besitzt. Die Tabelle für technische Daten enthält unter anderem die Attribute Farbe und Laufleistung, die Preisliste das Attribut Preis. Weiterhin besitzen beide Tabellen das Attribut Wagentyp.

In einem fiktiven Profil wird nach allen Einträgen gesucht, die als Schlüsselwort „Trabant“ oder „Jaguar“ besitzen. Weiterhin wird der Suchraum einmal eingegrenzt auf rote Autos mit einer Laufleistung von weniger als 150.000 km, im zweiten Fall auf einen Preis kleiner 10000. Ein entsprechender Suchausdruck hätte folgende Einträge:

Suchraum			Such-Token	Operator	boolescher Wert
Stufe 1	Stufe 2	Stufe 3			
Auto_DB			Trabant	=	OR
Auto_DB			Jaguar	=	OR
Auto_DB	Technische Daten	Farbe	rot	=	AND
Auto_DB	Technische Daten	Laufleistung	150000	<	AND
Auto_DB	Preisliste	Preis	10000	<	OR

In Abbildung 25 ist dargestellt, wie die aus den Sucheinträgen resultierenden Ergebnismengen miteinander verknüpft werden. Im ersten Schritt werden aus der Menge aller Dateneinheiten diejenigen ausgewählt, die als Token „Trabant“ bzw. „Jaguar“ enthalten. Anschließend wird im linken Strang diese Menge auf die Einheiten eingeschränkt, die in der Tabelle „Technische Daten“ vorkommen und im Farbattribut den Wert „rot“ sowie im Laufleistungsattribut ein Token kleiner als „150000“ aufweisen.

Obwohl es sich bei diesen beiden Einschränkungen um getrennte Suchräume handelt, werden sie wie einer behandelt (und damit die Ergebnisse wegen der Angabe „AND“ über die Schnittmenge verknüpft), da es sich um die dritte Selektionsstufe handelt (siehe Ausnahmeregelung auf Seite 92).

Im rechten Strang erfolgt nur eine Selektion aus der Trabant-Jaguar Menge auf die Datensätze in Tabelle „Preisliste“, die im Preisattribut einen Wert kleiner „10000“ besitzen. Bei einem einzelnen Selektionskriterium ist die Angabe der Verknüpfungsart ohne Bedeutung.

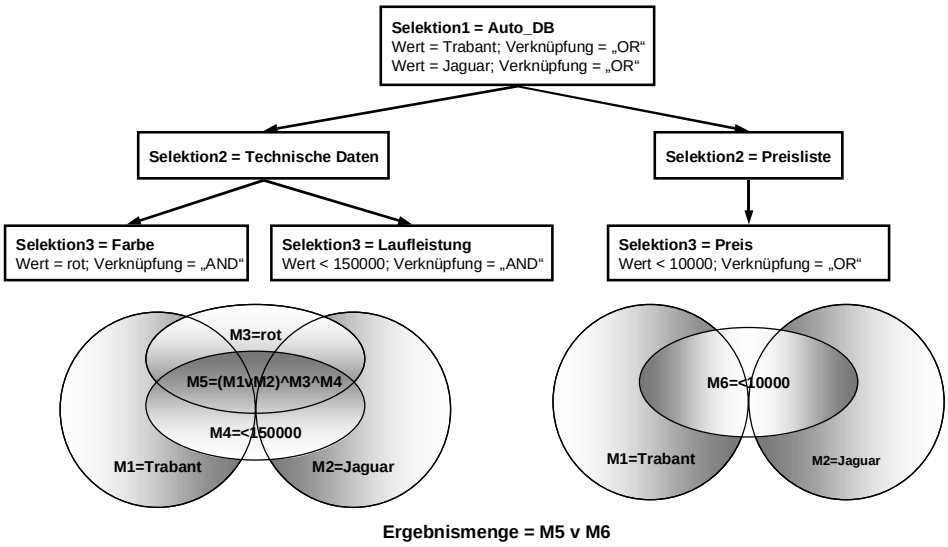

Abbildung 25: Profilverarbeitung am Beispiel einer Datenbank

Da sich die beiden Selektionsstränge auch oberhalb der dritten Stufe unterscheiden, werden die Ergebnismengen vereinigt. In diesem Beispiel erscheint es sinnvoll, die Mengen nach einem bestimmten Kriterium zu schneiden anstatt zu vereinigen. Dieses kann auf Ebene der Dateneneinheiten (siehe Anhang 7.2) nicht geleistet werden, weshalb in diesem Fall die externe Anwendung für die Zusammenführung dieser Resultate zuständig ist. Um sicherzugehen, daß der Agent auch Ergebnisse aus beiden Tabellen liefert, müssen diese dem Indexserver als über Fremdschlüssel verknüpfte Tabellen bekannt sein. Dadurch sind die Zeitstempel im Index für verbundene Einträge identisch.

7.4 Protokoll des Indexservers

- Jede Aktion besteht aus einem Anforderungs/Antwort Paar und wird immer vom Client initiiert; jedes Datenpaket schließt mit einem Newline-Zeichen ab
- Zu übertragende Daten werden in *Klammernotation* kodiert:

<data>	::=	<data_name>(<parameter>,<parameter>,...)
<data_name>	::=	<encoded_string>
<encoded_string>	::=	Zeichenkette, in dem die Zeichen ()\," mit vorangestelltem \ kodiert sind
<parameter>	::=	<data> || <encoded_string> alles, was zwischen Komma und/oder Klammern steht, gehört zum Parameter (auch Leerzeichen u.ä.)

- weitere in der Notation verwendete Datentypen

<timestamp>	::=	Zeitstempel, bestehend aus 14 Zeichen: YYYYMMDDHH24MISS mit YYYY – Jahr (4 stellig); MM – Monat (2 stellig); DD – Tag (2 stellig); HH24 – Stunden (24 Stunden-Format); MI – Minuten (2 stellig); SS – Sekunden (2 stellig)
<Y/N>	::=	entweder „Y“ für „ja“ oder „N“ für „nein“
<okMessage>	::=	Zeichenkette, die bei positivem Ausgang einer Aktion zurückgeliefert wird. Sie ist in der Systemeinstellungen-Datei definiert.
<errorMessage>	::=	Zeichenkette, die bei negativem Ausgang einer Aktion zurückgeliefert wird. Sie ist in der Systemeinstellungen-Datei definiert.

- Jede Aktion kann eine <errorMessage> Nachricht liefern, falls die Anzahl der Parameter fehlerhaft ist
- Es wird zwischen Aktionen die Authentifikation benötigen und Start-Aktionen, die dieses nicht voraussetzen, unterschieden.
- Die Authentifizierung (Login) wird über folgende Prozedur erreicht:
  - Der Client sendet die Anforderung ‚getID()‘ und erhält eine Zahl (Ziffernfolge) im Bereich eines 32-bit Integerwertes (varWert)
  - Aus einer anderen Quelle bezieht der Client einen 2. 32-bit Integerwert (festWert). Dieser muß dem Indexserver bekannt sein – er ließt ihn aus der Systemeigenschaften-Datenbank unter Eigenschaftsname ‚ClientLogin‘ und Attribut ‚indexserv‘ aus (Name und Attribut können in der globalen Systemeinstellungen-Datei geändert werden)
  - Mittels folgender Formel werden beide Werte zu einem neuen Wert – dem Paßwort – kombiniert: $Passwort = (((festWert * \mathrm{var}Wert) + festWert) \bmod 100000)$
  - Das Paßwort wird mit ‚login(<Paßwort>)‘ zurück zum Server geschickt. Wenn als Antwort <okMessage> übermittelt wird bzw. die Verbindung nach Erhalt der Antwort weiterbesteht, war die Authentifizierung erfolgreich.

- Aktionen, die für den Indexserver definiert sind (‚params' listet die zu übergebenden Parameter, ‚return' beschreibt den Rüchgabewert):
  - *Erlaubte Aktionen ohne Authentifizierung:*
    - **getVersion**()
      fragt nach der Version des Servers
      return: Zeichenkette mit Servernamen, Versionskennung und Änderungsdatum
    - **getID**()
      fragt nach der Sitzungsnummer (dient der Paßwortberechnung – s.o.)
      return: Ziffernfolge, die einen 32-bit Integerwert repräsentiert
    - **login**(<passwd>)
      sendet das Paßwort zur Authentifizierung am Server
      params: <passwd> das berechnete Paßwort (s.o.)
      return: <okMessage> falls Login erfolgreich war
      „bye" falls Paßwort falsch war (Verbindungsabbruch)
  - *Interne* Aktionen*:*
    - **stop**()
      initiiert das Herunterfahren des Agentenservers
      return: "server stop"
    - **getRunningCommands**()
      fragt nach aktuell laufenden (parallelen) Aktionen
      return: "commands(<request1>,<request2>,...)"
  - *Aktualisierungs-Aktionen:*
    - **clearFixedIndexUpdateTimes**()
      entfernt alle Einträge der Liste mit festen Aktualisierungszeiten (wird für Bestimmung des nächsten Aktualisierungszeitpunktes genutzt)
      return: <okMessage>
    - **clearIndexUpdateTimes**()
      entfernt alle Einträge aus der Liste mit den nächsten Aktualisierungszeiten
      return: <okMessage>
    - **enableIndexUpdate**()
      aktiviert die zeitgesteuerte Aktualisierung des Index
      return: <okMessage>
    - **disableINdexUpdate**()
      deaktiviert die zeitgesteuerte Aktualisierung des Index
      return: <okMessage>
    - **enableAutoIndexUpdate**()
      aktiviert das automatische Hinzufügen des nächsten Aktualisierungszeitpunktes
      return: <okMessage>
    - **disableAutoIndexUpdate**()
      deaktiviert das automatische Hinzufügen des nächsten Aktualisierungszeitpunktes
      return: <okMessage>
    - **setIndexUpdateInterval**(<hours>,<minutes>)
      überschreibt das bisher definierte Intervall zwischen zwei Aktualisierungen des Index

params: <hours> Intervall-Stunden
<minutes> Intervall-Minuten
return: <okMessage>
<errorMessage> falls die Parameter fehlerhaft waren

- **getNextIndexUpdateTime**()
  fragt nach dem nächsten Aktualisierungszeitpunkt
  return: <timestamp>
  <errorMessage> falls die Liste mit den Aktualisierungszeiten leer ist
- **addIndexUpdateTime**(<hour>,<minute>)
  fügt einen weiteren Eintrag in die Liste mit den Aktualisierungszeiten hinzu
  params: <hour> Stunde (0-23)
  <minute> Minute (0-59)
  return: <okMessage>
  <errorMessage> falls die Parameter fehlerhaft waren
- **addFixedIndexUpdateTime**(<hour>,<minute>)
  fügt einen weiteren Eintrag in die Liste mit festen Tageszeiten hinzu
  params: <hour> Stunde (0-23)
  <minute> Minute (0-59)
  return: <okMessage>
  <errorMessage> falls die Parameter fehlerhaft sind
- **updateSourceIndex**(<source_name>)
  aktualisiert den Index für die angegebene Datenquelle
  params: <source_name> eindeutiger Name der Datenquelle (max. 40 Zeichen)
  return: <okMessage>
  <errorMessage> falls Datenquelle nicht existiert oder ein Fehler während der Aktualisierung auftrat

- *Datenquellenspezifische* Aktionen*:*
  - **addSource**(properties(propentry(datatype,<datatype_name>),
    propentry(name,<unique_datasource_name>),
    <additional_datatype_specific_propentries>), <Y/N>)
    fügt eine Datenquelle zur Observierung dem entsprechenden Modul für diesen Datenquellentyp hinzu; die Parameter der Datenquelle werden in der Systemeigenschaften-Datenbank gespeichert
    params: <datatype_name>
    Bezeichnung des Datenquellentyps (z.B. "database")
    <unique_datasource_name>
    ein eindeutiger Bezeichner für die neue Datenquelle (max. 40 Zeichen)
    <additional_datatype_specific_propentries>
    weitere, für den Datenquellentyp spezifische Attribut-Wert Paare (z.B. für Datenbanken:
    "connPrpFile" – Name der Datei mit JDBC-Daten,
    "dbDefFile" – Name der Datei mit Strukturdaten der Datenbank)
    <Y/N> falls „Y“ wird ein neuer Index für die Datenquelle erzeugt (alte Indexeinträge für diese Datenquelle werden zuvor gelöscht)

return: <okMessage> falls der letzte Parameter „Y“ war
(die Aktion wird in einem eigenen Thread ausgeführt und Fehler erscheinen nur in der Logdatei)
<okMessage> falls der letzte Parameter „N“ war und keine Fehler bei der Ausführung auftraten
<errorMessage> falls der Datenquellenbezeichner schon existiert oder die Datenquelle nicht hinzugefügt werden konnte

- **removeSource**(<source_name>,<Y/N>)
entfernt die Datenquelle von der Aktualisierungsliste sowie von der Systemeigenschaften-Datenbank, d.h. auch nach einem Neustart des Indexservers wird die Datenquelle nicht mehr observiert
params: <source_name> eindeutiger Bezeichner der Datenquelle
<Y/N> falls „Y“ werden alle Indexeinträge dieser Datenquelle gelöscht
return: <okMessage> falls der letzte Parameter „Y“ war
(die Aktion wird in einem eigenen Thread ausgeführt und Fehler erscheinen nur in der Logdatei)
<okMessage> falls der letzte Parameter „N“ war und keine Fehler bei der Ausführung auftraten
<errorMessage> falls der Datenquellenbezeichner nicht existiert oder die Datenquelle nicht entfernt werden konnte
- **getSource**(<source_name>)
fragt nach den Parametern der Datenquelle
params: <source_name> eindeutiger Bezeichner der Datenquelle
return: properties(propentry(datatype,<datatype_name>),
propentry(name,<unique_datasource_name>),
<additional_datatype_specific_propentries>)
<errorMessage> falls der Datenquellenbezeichner nicht existiert
- **getSourceNames**()
fragt nach allen eindeutigen Namen der aktuell observierten Datenquellen
return: datasourcenames(<source1_name>,<source2_name>,...)
mit <source_name> ::= <encoded_string>
(max. 40 Zeichen)
- **removeSourceFromIndex**(<source_name>)
entfernt alle Einträge der Datenquelle vom Index (Bereinigung des Index von Einträgen zu Datenquellen, die nicht mehr observiert werden)
params: <source_name> eindeutiger Bezeichner der Datenquelle
return: <okMessage> (die Aktion wird in einem eigenen Thread ausgeführt und Fehler erscheinen nur in der Logdatei)

7.5 Protokoll des Agentenservers

- Jede Aktion besteht aus einem Anforderungs/Antwort Paar und wird immer vom Client initiiert; jedes Datenpaket schließt mit einem Newline-Zeichen ab
- Zu übertragende Daten werden in *Klammernotation* kodiert:

<data>	::=	<data_name>(<parameter>,<parameter>,...)
<data_name>	::=	<encoded_string>
<encoded_string>	::=	Zeichenkette, in dem die Zeichen ()\," mit vorangestelltem \ kodiert sind; eine leere Zeichenkette wird als " (doppeltes Anführungszeichen) kodiert
<parameter>	::=	<data> || <encoded_string> alles, was zwischen Komma und/oder Klammern steht, gehört zum Parameter (auch Leerzeichen u.ä.)

- Agentendefinition in Klammernotation:

<agent>	::=	agent(<agent_id>, <user_id>, <eMail>, <interval>, <lastResultUpdate>, <lastNotification>, <active>, <agent_name>, <profile_entry1>, <profile_entry2>, ..))
<agent_id>	::=	Ganzzahl (32 bit) >= 100
<user_id>	::=	Ganzzahl (32 bit)
<eMail>	::=	<encoded_string> (max. 60 Zeichen) eMailadresse des Benutzers
<interval>	::=	Zeichenkette (max. 4 Zeichen) Intervall zwischen Benachrichtigungen per Email über neue Einträge in der Ergebnisliste
<lastResultUpdate>	::=	<timestamp> || „now“ || leerer String als <encoded_string> Zeitstempel der letzten Aktualisierung der Ergebnisliste; „now“ und leerer String können nur bei *createAgent* und *updateAgent* übergeben werden: • „now“: wird vom Server durch aktuellen Zeitstempel ersetzt • leerer String: wird vom Server durch einen Zeitstempel aus der Vergangenheit (1.1.1980) ersetzt
<lastNotification>	::=	<timestamp> || „now“ || leerer String als <encoded_string> Zeitstempel der letzten Benachrichtigung per Email „now“ und leerer String können nur bei *createAgent* und *updateAgent* übergeben werden (siehe Beschreibung bei <lastResultUpdate>)
<active>	::=	<Y/N> Agentenzustand („Y“ = aktiv, d.h. Ergebnisliste wird aktualisiert)

<agent_name>	::=	<encoded_string> (max. 20 Zeichen) Name des Agenten
<profile_entryX>	::=	profile(<source_name>, <subID>, <subSubSubID>, <word>,<boolean>, <operator>) ein Eintrag im Agentenprofil;
<source_name>	::=	<encoded_string> (max. 40 Zeichen) eindeutiger Name der Datenquelle
<subID>	::=	<encoded_string> (max. 40 Zeichen) Unter-ID der Datenquelle (z.B. Tabellenname bei Datenbanken)
<subSubSubID>	::=	<encoded_string> (max. 40 Zeichen) Unter-unter-unter-ID für Datenquelle (z.B. Attributname bei Datenbanken)
<word>	::=	<encoded_string> (max. 40 Zeichen) Suchpattern
<boolean>	::=	"AND","OR" definiert boolesche Verknüpfung der Suchpattern: • „AND" : Suchpattern muß gefunden werden • „OR" : nur eins der mit „OR“ ausgezeichneten Suchpattern muß gefunden werden
<operator>	::=	{<,>,<>,=,like} definiert, wie das Suchpattern mit dem Index verglichen wird

- weitere in der Notation verwendete Datentypen

<timestamp>	::=	Zeitstempel, bestehend aus 14 Zeichen: YYYYMMDDHH24MISS mit YYYY – Jahr (4 stellig); MM – Monat (2 stellig); DD – Tag (2 stellig); HH24 – Stunden (24 Stunden-Format); MI – Minuten (2 stellig); SS – Sekunden (2 stellig)
<Y/N>	::=	entweder „Y“ für „ja“ oder „N“ für „nein“
<okMessage>	::=	Zeichenkette, die bei positivem Ausgang einer Aktion zurückgeliefert wird. Sie ist in der Systemeinstellungen-Datei definiert.
<errorMessage>	::=	Zeichenkette, die bei negativem Ausgang einer Aktion zurückgeliefert wird. Sie ist in der Systemeinstellungen-Datei definiert.

- Jede Aktion kann eine <errorMessage> Nachricht liefern, falls die Anzahl der Parameter fehlerhaft ist
- Es wird zwischen Aktionen die Authentifikation benötigen und Start-Aktionen, die dieses nicht voraussetzen, unterschieden.
- Die Authentifizierung (Login) wird über folgende Prozedur erreicht:
  - Der Client sendet die Anforderung ‚getID()‘ und erhält eine Zahl (Ziffernfolge) im Bereich eines 32-bit Integerwertes (varWert)

- Aus einer anderen Quelle bezieht der Client einen zweiten 32-bit Integerwert (fest-Wert). Dieser muß dem Agentenserver bekannt sein – er liest ihn aus der Systemeigenschaftentabelle unter Eigenschaftsname ‚ClientLogin' und Attribut ‚agentserv' aus (Name und Attribut können in der globalen Systemeinstellungendatei geändert werden)
- Mittels folgender Formel werden beide Werte zu einem neuen Wert – dem Paßwort – kombiniert: $Passwort = (((festWert * \mathrm{var}Wert) + festWert) \bmod 100000)$
- Das Paßwort wird mit ‚login(<Paßwort>)' zurück zum Server geschickt. Wenn als Antwort <okMessage> übermittelt wird bzw. die Verbindung nach Erhalt der Antwort weiterbesteht, war die Authentifizierung erfolgreich.

- Aktionen, die für den Agentenserver definiert sind (‚params' listet die zu übergebenden Parameter, ‚return' beschreibt den Rüchgabewert):
  - *Erlaubte Aktionen ohne Authentifizierung:*
    - **getVersion**()
      fragt nach der Version des Servers
      return: Zeichenkette mit Servernamen, Versionskennung und Änderungsdatum
    - **getID**()
      fragt nach der Sitzungsnummer (dient der Paßwortberechnung – s.o.)
      return: Ziffernfolge, die einen 32-bit Integerwert repräsentiert
    - **login**(<passwd>)
      sendet das Paßwort zur Authentifizierung am Server
      params: <passwd> das berechnete Paßwort (s.o.)
      return: <okMessage> falls Login erfolgreich war
      „bye" falls Paßwort falsch war (Verbindungsabbruch)
  - *Interne* Aktionen*:*
    - **stop**()
      initiiert das Herunterfahren des Agentenservers
      return: "server stop"
    - **getRunningCommands**()
      fragt nach aktuell laufenden (parallelen) Aktionen
      return: "commands(<request1>,<request2>,...)"
  - *Aktualisierungs*-Aktionen*:*
    - **clearFixedResultUpdateTimes**()
      entfernt alle Einträge der Liste mit festen Aktualisierungszeiten (wird für Bestimmung des nächsten Aktualisierungszeitpunktes genutzt)
      return: <okMessage>
    - **clearResultUpdateTimes**()
      entfernt alle Einträge aus der Liste mit den nächsten Aktualisierungszeiten
      return: <okMessage>
    - **enableResultUpdate**()
      aktiviert die zeitgesteuerte Aktualisierung der Ergebnisliste
      return: <okMessage>
    - **disableResultUpdate**()
      deaktiviert die zeitgesteuerte Aktualisierung der Ergebnisliste
      return: <okMessage>

- **enableAutoResultUpdate**()
  aktiviert das automatische Hinzufügen des nächsten Aktualisierungszeitpunktes
  return: <okMessage>
- **disableAutoResultUpdate**()
  deaktiviert das automatische Hinzufügen des nächsten Aktualisierungszeitpunktes
  return: <okMessage>
- **setResultUpdateInterval**(<hours>,<minutes>)
  überschreibt das bisher definierte Intervall zwischen zwei Aktualisierungen der Ergebnisliste
  params: <hours> Intervall-Stunden
  <minutes> Intervall-Minuten
  return: <okMessage>
  <errorMessage> falls die Parameter fehlerhaft waren
- **getNextResultUpdateTime**()
  fragt nach dem nächsten Aktualisierungszeitpunkt
  return: <timestamp>
  <errorMessage> falls die Liste mit den Aktualisierungszeiten leer ist
- **addResultUpdateTime**(<hour>,<minute>)
  fügt einen weiteren Eintrag in die Liste mit den Aktualisierungszeiten hinzu
  params: <hour> Stunde (0-23)
  <minute> Minute (0-59)
  return: <okMessage>
  <errorMessage> falls die Parameter fehlerhaft waren
- **addFixedResultUpdateTime**(<hour>,<minute>)
  fügt einen weiteren Eintrag in die Liste mit festen Tageszeiten hinzu
  params: <hour> Stunde (0-23)
  <minute> Minute (0-59)
  return: <okMessage>
  <errorMessage> falls die Parameter fehlerhaft sind
- **updateAgentResults**(<agent_id>,<user_id>)
  Ergebnisaktualisierung für einen bestimmten Agenten
  params: <agent_id> die Agenten-ID deren Ergebnisliste aktualisiert werden soll
  <user_id> Benutzer-ID der der Agent gehört (Ganzzahl)
  return: <okMessage>
  <errorMessage> falls der Agent nicht existiert oder andere Fehler auftraten

- *Agentenspezifische* Aktionen*:*
  - **createAgent**(<agent>)
    erstellt einen neuen Agenten für den in <agent> übergebenen Benutzer; <agent_id> wird vom Agentensystem ausgefüllt
    params: <agent> Agentendefinition in Klammernotation (siehe oben); für das Ausfüllen der Zeitstempel siehe <agent>-Definition
    return: neue Agenten-ID (Ganzzahl)
    <errorMessage> falls der Agent nicht erstellt werden konnte
  - **updateAgent**(<agent>)

ändert die Agentendefinition für die implizit angegebene Agenten-ID;
alle Felder außer <agent_id> und <user_id> können geändert werden
params: <agent> Agentendefinition in Klammernotation (siehe oben)
für das Ausfüllen der Zeitstempel siehe <agent>-Definition
return: <okMessage>
<errorMessage> falls der Agent nicht existiert, nicht dem Benutzer gehört oder nicht geändert werden konnte

- **setEMail**(<agent_id>,<user_id>,<newEMail>,<newInterval>)
ändert die Emaileinstellungen in der Agentendefinition für den angegebenen Agenten; alle anderen Einstellungen und Daten bleiben erhalten
params: <agent_id> Agenten-ID deren Emaileinstellungen geändert werden
<user_id> Benutzer-ID des Agenten
<newEMail> neue Email-Adresse
<newInterval> neues Intervall für die Email-Benachrichtigung
return: <okMessage>
<errorMessage> falls der Agent nicht dem Benutzer gehört oder nicht geändert werden konnte
- **getAgent**(<agent_id>,<user_id>)
fragt nach der Agentendefinition für einen bestimmten Agenten (grundlegende Einstellungen und Profil)
params: <agent_id> Agenten-ID deren Definition gewünscht wird
<user_id> Benutzer-ID des Agenten
return: <agent> Agentendefinition in Klammernotation (siehe oben)
<errorMessage> falls der Agent nicht existiert oder die Parameter fehlerhaft waren
- **getUserAgents**(<user_id>)
fragt nach den Agentendefinitionen aller Agenten eines Benutzers
params: <user_id> Benutzer-ID deren Agentendefinitionen gewünscht werden
return: agents(<agent>,<agent>,...)
Klammernotation der Benutzeragenten
<errorMessage> falls die Parameter fehlerhaft waren oder ein Datenbankfehler aufgetreten ist
- **deleteAgent**(<agent_id>,<user_id>)
löscht den angegebenen Agenten
params: <agent_id> Agenten-ID die gelöscht werden soll
<user_id> Benutzer-ID des Eigentümers des Agenten
return: <okMessage>
<errorMessage> falls die Parameter fehlerhaft waren oder ein Datenbankfehler aufgetreten ist
- **deleteUserAgents**(<user_id>)
löscht alle Agenten eines Benutzers
params: <user_id> Benutzer-ID deren Agenten gelöscht werden sollen
return: <okMessage>
<errorMessage> falls bei mindestens einem Agenten ein Fehler aufgetreten ist
- **activateAgent**(<agent_id>,<user_id>)
aktiviert den angegebenen Agenten

params: <agent_id> Agenten-ID die aktiviert werden soll
<user_id> Benutzer-ID der der Agent gehört
return: <okMessage>
<errorMessage> falls der Agent nicht dem Benutzer gehört oder ein Datenbankfehler aufgetreten ist

- **activateUserAgents**(<user_id>)
  aktiviert alle Agenten eines Benutzers
  params: <user_id> Benutzer-ID deren Agenten aktiviert werden sollen
  return: <okMessage>
  <errorMessage> falls wenigstens ein Agent nicht aktiviert werden konnte
- **deactivateAgent**(<agent_id>,<user_id>)
  deaktiviert den angegebenen Agenten
  params: <agent_id> Agenten-ID die deaktiviert werden soll
  <user_id> Benutzer-ID der der Agent gehört
  return: <okMessage>
  <errorMessage> falls der Agent nicht dem Benutzer gehört oder ein Datenbankfehler aufgetreten ist
- **deactivateUserAgents**(<user_id>)
  deaktiviert alle Agenten eines Benutzers
  params: <user_id> Benutzer-ID deren Agenten deaktiviert werden sollen
  return: <okMessage>
  <errorMessage> falls wenigstens ein Agent nicht deaktiviert werden konnte

7.6 Screenshots der Agenten im Handwerks-Centrum

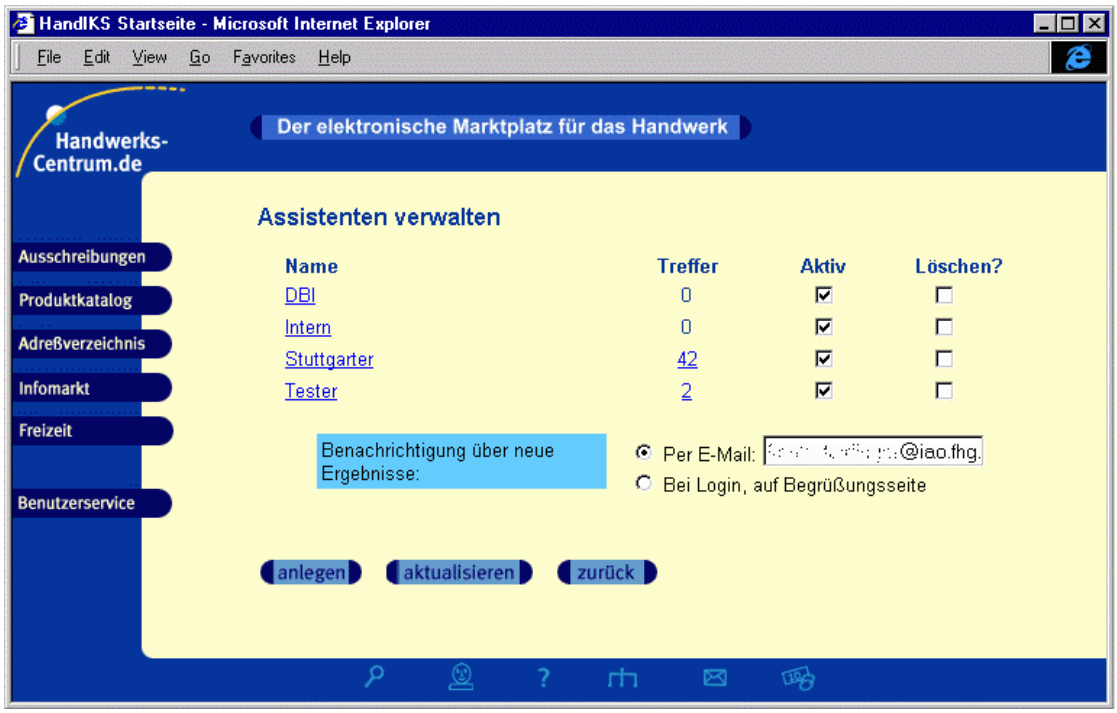

Abbildung 26: Screenshot Handwerks-Centrum – Agentenverwaltung

Abbildung 27: Screenshot Handwerks-Centrum – Benutzereinstiegsseite mit Informationen der Agenten

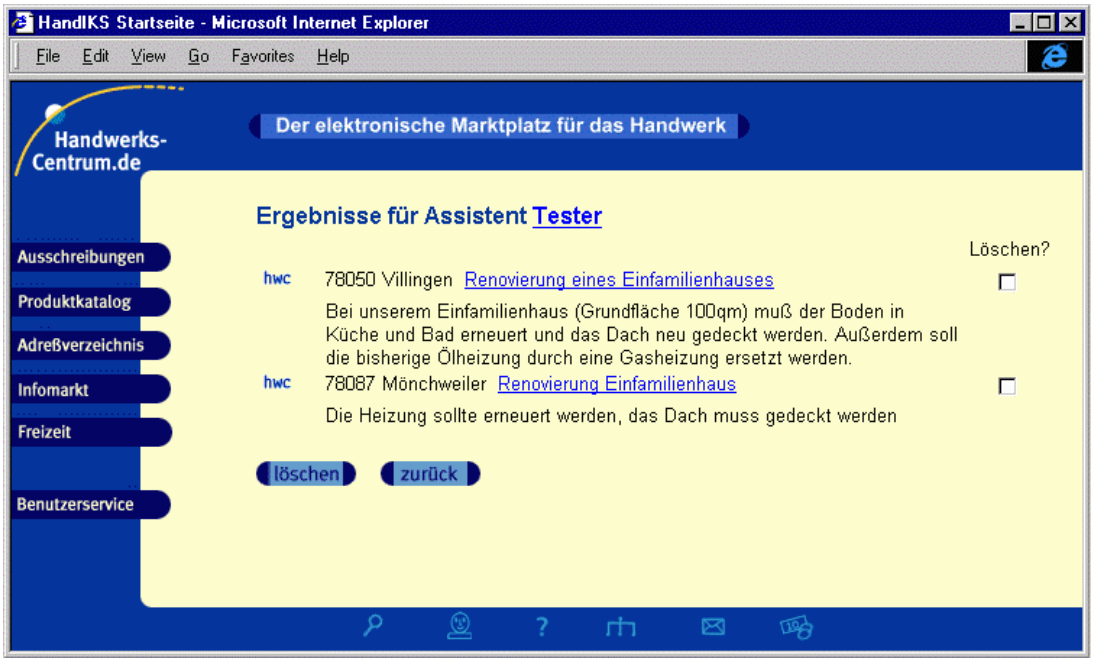

Abbildung 28: Screenshot Handwerks-Centrum – Ergebnisseite eines Agenten

7.7 Screenshots der Agenten in ELPRO

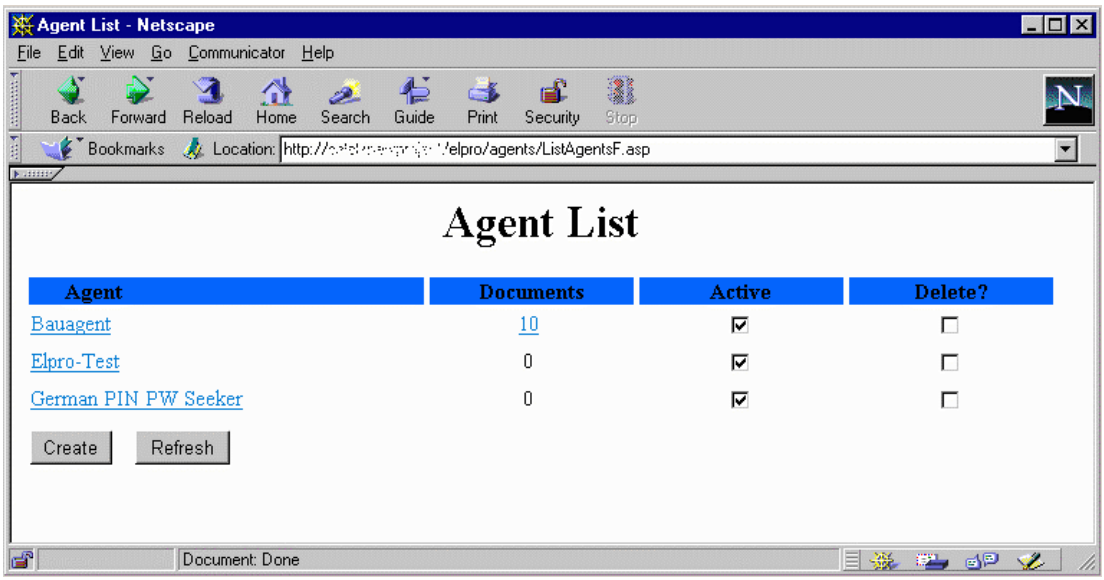

Abbildung 29: Screenshot ELPRO – Agentenverwaltung

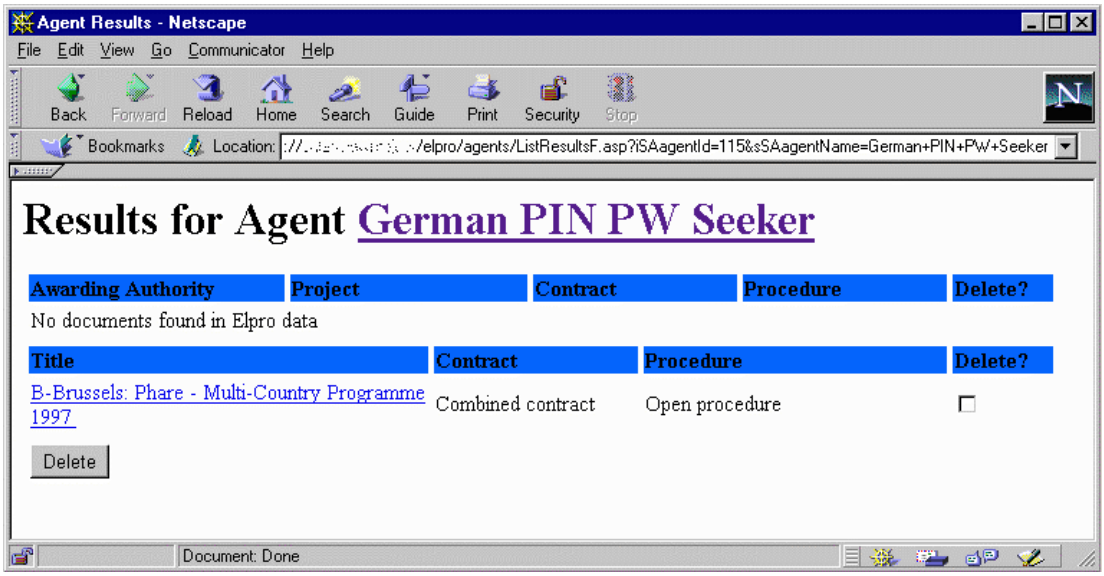

Abbildung 30: Screenshot ELPRO – Ergebnisliste eines Agenten

8 Literaturverzeichnis

[Bag96] Bager, J., Wortschätze heben – Volltext-Retrievalsysteme, c't Magazin für Computertechnik, (1996)8, S. 160 – 166, Verlag Heinz Heise

[Bal96] Balzert, H., Lehrbuch der Software-Technik – Software-Entwicklung, Spektrum Akademischer Verlag, Heidelberg et. al. 1996

[Boe98] Böhme, T., Dokumentation zu UNWAS (Universal News Watcher Agent System) Version 1.0, Fraunhofer IAO, Stuttgart, 1998

[Boo99] Booch, G., Rumbaugh, J., Jacobson, I., The Unified Modeling Language User Guide, Addison-Wesley, Reading et. al. 1999

[Bra97] Bradshaw, J., An Introduction to Software Agents, in: Software Agents, AAAI Press / The MIT Press, 1997

[Bre98] Brenner, W., Zarnekow, R., Wittig, H., Intelligente Softwareagenten: Grundlagen und Anwendungen, Springer-Verlag, Heidelberg et. al. 1998

[Dec97] Decker, K., Pannu, A., Sycara, K., Williamson, M., Designing Behaviors for Information Agents, in: Proceedings of the First International Conference on Autonomous Agents, S. 404 – 412, Marina del Rey 1997

[Etz95] Etzioni, O., Weld, D. S., Intelligent agents on the Internet: Fact, fiction, and forecast, in: IEEE Expert, 10(1995)4, S. 44 – 49

[Far96] Farquhar, A., Fikes, R., Rice, J., The Ontolingua Server: a Tool for Collaborative Ontology Construction, Report Nr. KSL–96–26, Knowledge Systems Laboratory, Stanford University, 1996

[Fin93] Finin, T., Weber, J. et. al., Specification of the KQML Agent-Communication Language, The ARPA Knowledge Sharing Initiative External Interfaces Working Group, in: Internet URL: http://www.cs.umbc.edu/kqml/papers/kqmlspec.pdf (Stand 01/1999)

[Fin94] Finin, T., Fritzson, R., McKay, D., McEntire, R., KQML as an Agent Communication Language, in: CIKM '94. Proceedings of the Third International Conference on Information and Knowledge Management, S. 456 – 463, ACM Press, 1994

[Fip97] FIPA, FIPA 97 Specification Part 5 Personal Assistant, S. 5, Genf, 1997

[Fip98] FIPA, FIPA '98, in: Internet URL: http://www.fipa.org/spec/FIPA98.html (Stand 12/1998)

[Fra97] Franklin, S., Graesser, A., Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents, in: Müller, J. P., Wooldridge, M.J., Jennings, N.R. (Hrsg.), Intelligent Agents III. Agent Theories, Architectures, and Languages, Springer-Verlag, Heidelberg et. al. 1997

[Fon94] Foner, L. N., What's an Agent, Anyway? A Sociological Case Study, in: Internet URL: http://foner.www.media.mit.edu/people/foner/Julia/Julia.html (Stand 1998)

[Fon97] Foner, L. N., Yenta: A Multi-Agent, Referral-Based Matchmaking System, in: Internet URL: http://foner.www.media.mit.edu/people/foner/Reports/Agents-97/Yenta-level-2.ps (Stand 01/1999)

[Fon98] Foner, L., A brief introduction to Yenta, in: Internet URL: http://foner.www.media.mit.edu/people/foner/yenta-brief.html (Stand 1998)

[Fow97] Fowler, M., Scott, K., UML Distilled – Applying The Standard Object Modeling Language, Addison-Wesley, Reading et. al. 1997

[Ful98] Fulcrum Knowledge Network™, in: Internet URL: http://www.fulcrum.com/english/products/fknhome.htm (Stand 1998)

[Gen92] Genesereth, M. R., Fikes, R. E. et. al., Knowledge Interchange Format, Version 3.0 Reference Manual, Technical Report Logic-92-1, Computer Science Department, Stanford University, 1992

[Gen94] Genesereth M.R., Ketchpel S.P., Software Agents, in: Communications of the ACM (CACM), 37(1994)7, S. 48 – 53

[Gen98] General Magic, Odyssey, in: Internet URL: http://www.genmagic.com/technology/odyssey.html (Stand 6/1998)

[Gil95] Gilbert, D. et al., The Role of Intelligent Agents in the Information Infrastructure, IBM, Research Triangle Park 1995

[Gil97] Gilbert, D., Intelligent Agents: The Right Information at the Right Time, in Internet URL: http://www.networking.ibm.com/iag/iagwp1.html (Stand 6/1998), 1997

[Gmd97] GMD FOCUS, IBM, Mobile Agent System Interoperability Facilities Specification, OMG TC Document, in Internet URL: http://www.omg.org/cgi-bin/doc?orbos/97-10-5

[Gru93] Gruber, T., A translation approach to portable ontology specification, in: Knowledge Acquisition, 5(1993)2, S. 199-220

[Gut98] Guttman, R., Moukas, A., Maes, P., Agent-mediated Electronic Commerce: A Survey. in: Knowledge Engineering Review, 13(1998)2

[Har95] Harrison, C. G., Chess, D. M., Kershenbaum, A., Mobile Agents: Are they a good idea?, in: Internet URL: http://www.research.ibm.com/massive/mobag.ps (Stand 5/1998), 1995

[Hay95] Hayes-Roth, B., An architecture for adaptive intelligent systems, in: Artificial Intelligence, 72(1995), S. 329 – 365

[Her96] Hermans, B., Intelligent Software Agents on the Internet: an inventory of currently offered functionality in the information society & a prediction of (near-)future developments, Tilburg University, Tilburg 1996

[Ing92] Ingwersen, P., Information Retrieval Interaction, Taylor Graham, London et. al. 1992

[Jen98] Jennings, N. R., Wooldridge, M. J. (Hrsg.), Applications of Intelligent Agents, in: Agent Technology - Foundations, Applications, and Markets, Springer-Verlag, Heidelberg et. al. 1998

[Kod96] Koda, T., Maes, P., Agents with Faces: The Effect of Personification, 5th IEEE International Workshop on Robot and Human Communication, Tsukuba, Japan, 1996

[Kru98/1] Krupansky, J., Producing a Comprehensive List of Web References for "Software Agent", in: Internet URL: http://www.basetechnology.com/aglpaper.htm (Stand 6/1998)

[Kru98/2] Krupansky, J., The Software Agent Puzzle, in: Internet URL: http://www.basetechnology.com/agpuzl.htm (Stand 6/1998)

[Lab97] Labrou, Y., Finin, T., A Proposal for a new KQML Specification, Technical Report CS-97-03, Computer Science and Electrical Engineering Department, University of Maryland Baltimore County, 1997

[Lan95] Lanier, J., Agents of Alienation, in: Journal of Consciousness Studies, (1995)2, S. 76 – 81

[Las94] Lashkari, Y., Metral, M., Maes, P., Collaborative Interface Agents, in: Procedings of AAAI'94 Conference, Seattle 1994

[Lar96] Larsson, J. E., Hayes-Roth, B., Gaba, D., Guardian: Final Evaluation, Report Nr. KSL–96–25, Knowledge Systems Laboratory, Stanford University, 1996

[Law98] Lawrence, S., Giles, C. L., Searching the World Wide Web, in: Science, 280(1998), S. 98 – 100

[Mae95] Maes, P., Artificial Life Meets Entertainment: Life like Autonomous Agents, Communications of the ACM (CASM), 38(1995)11, S. 108 – 114

[Mam98] Mamdani, A., Steiner, D., FIPA's Fifth Call for Proposals, in: Internet URL: http://www.fipa.org/cfp/cfp5.html (Stand 12/1998)

[Mey94] Meyers Lexikonredaktion, Meyers neues Lexikon: in 10 Bänden, Bd.1.A-Ben, Bibliographisches Institut & F. A. Brockhaus AG, Mannheim 1994

[Mit98] MIT Media Lab: Software Agents Group, in: Internet URL: http://agents.www.media.mit.edu/groups/agents (Stand 6/1998)

[Mul98] Muller, N. J., Systems Management: The Emerging Role of Intelligent Agents, in: Internet URL: http://www.ddx.com/agents.shtml (Stand 6/1998)

[Nwa96] Nwana, H. S., Software Agents: An Overview, in: Knowledge Engineering Review, 11(1996)3, S. 205 – 244

[Oas98] OASIS (Optimal Aircraft Sequencing using Intelligent Scheduling), in: Internet URL: http://www.aaii.oz.au/proj/oasis/oasis.html (Stand 1998)

[Obj97] ObjectSpace Voyager™, in: Internet URL: http://www.objectspace.com/voyager/index.html (Stand 6/1998), ObjectSpace, 1997

[Odu97] Odubiyi, J. B., Kocur, D. J., Weinstein, S. M., Wakim, N., Srivastava, S., Gokey, C., Graham, J., SAIRE - A Scalable Agent-based Information Retrieval Engine, in: Proceedings of the First International Conference on Autonomous Agents, Marina del Rey 1997

[Omg95] Object Management Group, CORBAfacilities: Common Facilities Architecture V4.0, in Internet URL: http://www.omg.org/corba/cf2.html

[Omg98] Object Management Group, CORBA / IIOP 2.2 Specification, in: Internet URL: http://www.omg.org/corba/corbaiiop.html, Stand 1/1999

[Pcd98] PC DOCS, DOCSFulcrum Product Overview, in: Internet URL: http://www.pcdocs.com/Products/Fproducts/KNProd3.htm, Stand 12/1998

[Pet96] Petrie, C. J., Agent-Based Engineering, the Web, and Intelligence, in: IEEE Expert: Intelligent Systems and Their Applications, 11(1996)6

[Rao91] Rao, A. S., Georgeff, M. P., Modeling rational agents within a BDI-Architecture, in: Technical Report 14, Australian AI Institute, Carlton 1991

[Rao95] Rao, A. S., Georgeff, M. P., BDI Agents: From Theory to Practice, in: Proceedings of the First International Conference on Multi-Agent-Systems (ICMAS), San Francisco 1995

[Ret99a] Reticular Systems, Agent Construction Tools, in: Internet URL: http://www.agentbuilder.com/AgentTools/index.html, Stand 2/1999

[Ret99b] Reticular Systems, AgentBuilder User's Guide Version 1.1 Rev. 4, 1999, verfügbar unter Internet URL: http://www.reticular.com/AgentBuilder/usermanual.html, Stand 2/1999

[Rus95] Russel, S. J., Norvig, P., Artificial Intelligence: A Modern Approach, Englewood Cliffs, 1995.

[Sal89] Salton, G., Automatic Text Processing. The Transformation, Analysis and Retrieval of Information by Computer, Addison-Wesley, Reading/MA, 1989

[Sta97] Stark, H., Agents on the Web: Catalyst for e-commerce - A white paper, in: Internet URL: http://www.ovum.com/innovate/ia2/ia2wp Stand 6/1998, Ovum Ltd., London 1997

[Tha96] Tham, C., Friedman, B., Common Agent Platform Architecture, in: Internet URL: http://www.agent.org/pub/satp/papers/satp.html

[Ver96] Verity, SEARCH´97™ Agent Server Toolkit User´s Guide V1.0

[Ver99] Verity, SEARCH´97™ Information Server User's Guide V3.6, in: Internet URL: http://www.verity.com/support/documentation/ISdoc/user36/cover.htm (Stand 02/1999)

[Vir98] Virdhagriswaran, S., Mobile Unstructured Business Object (MuBot)™ Technology, in: Internet URL:

http://www.crystaliz.com/logicware/mubot.htm
(Stand 6/1998), Crystaliz Inc.

[Vos97] Voss, A., Kreifelts, T., Soap: Social Agents Providing People with Useful Information, in: Internet URL: http://orgwis.gmd.de/projects/SAW/voss.html (Stand 6/1998), GMD Sankt Augustin

[Whi96] White, J., Prospectus for an Open Simple Agent Transfer Protocol, in: Internet URL: http://www.agent.org/pub/satp/prospectus3.html (Stand 01/1999)

[Won96] Wondergem, B.C.M., van Bommel, P., Huibers, T.W.C., van der Weide, Th., Towards an Agent-Based Retrieval Engine, in: Furner, J., Harper, D. J. (Hrsg.), Proceedings of the 19th BCS-IRSG Colloquium, S. 126 – 144, Robert Gordon University, Aberdeen 1997

[Woo95] Wooldridge, M., Jennings, N. R., Intelligent Agents: Theory and Practice, in: Knowledge Engineering Review, 10(1995)2, S. 115 – 152

[Woo98] Wooldridge, M., Agents and software engineering, in: AI*IA Notizie, XI(1998)3, S. 31 – 37

Erklärung

Ich versichere, daß ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Leipzig März 1999