Xpert.press

Die Reihe **Xpert.press** vermittelt Professionals in den Bereichen Softwareentwicklung, Internettechnologie und IT-Management aktuell und kompetent relevantes Fachwissen über Technologien und Produkte zur Entwicklung und Anwendung moderner Informationstechnologien.

Gerhard Weiß · Ralf Jakob

Agentenorientierte Softwareentwicklung

Methoden und Tools

Mit 93 Abbildungen und 78 Tabellen

Springer

Gerhard Weiß
e-mail: *weissg@in.tum.de*

Ralf Jakob
e-mail: *jakob@in.tum.de*

Institut für Informatik
Technische Universität München
Boltzmannstr. 3
85748 Garching

Bibliografische Information der Deutschen Bibliothek
Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.ddb.de abrufbar.

ISSN 1439-5428
ISBN 3-540-00062-3 Springer Berlin Heidelberg New York

Springer ist ein Unternehmen von Springer Science+Business Media
springer.de

Umschlaggestaltung: KünkelLopka Werbeagentur, Heidelberg
Satz: durch die Autoren unter Benutzung eines Springer LaTeX-Makropakets
Herstellung: LE-TeX Jelonek, Schmidt & Vöckler GbR, Leipzig
Gedruckt auf säurefreiem Papier 33/3142/YL - 5 4 3 2 1 0

Vorwort

Softwareentwicklung wird zwar immer komplexer – aber auch immer faszinierender.
B. Oestereich

Gegenstand dieses Buches

Agentenorientierte Softwareentwicklung findet aufgrund ihres Potentials seit einigen Jahren wachsende Aufmerksamkeit in Forschung und Praxis. Charakteristisch für diese junge Art der Softwareentwicklung ist es, dass die Struktur und Funktionalität von Softwaresystemen unter dem Blickwinkel einer Menge von Agenten entworfen, implementiert und analysiert wird. Unter einem Agent wird dabei eine abgrenzbare Einheit verstanden, die in der Lage ist, die ihr zugewiesenen Aufgaben autonom, flexibel und bei Bedarf in kooperativer und kompetitiver Interaktion mit anderen Agenten und menschlichen Benutzern zu bewältigen. Agentenorientierung ist kompatibel mit anderen Ansätzen wie beispielsweise Objekt- und Komponentenorientierung, bietet jedoch auf einer qualitativ unterschiedlichen Abstraktionsebene – nämlich der Ebene von Organisation, Sozialität und Wissen – eine neue, intuitiv klare und innovative Systemsicht, die sich in besonderem Maß eignet für die Handhabung von Systemkomplexität und die Realisierung von verteilten, offenen und eingebetteten Anwendungen. Solchen Anwendungen kommt mit zunehmender Rechnervernetzung und Plattforminteroperabilität in den verschiedensten industriellen, kommerziellen und wissenschaftlichen Bereichen, und generell im Rahmen von zukunftsträchtigen Informationsverarbeitungsparadigmen wie Grid Computing, Pervasive Computing und Mobile Computing, eine zentrale Bedeutung bei. Ein Schwerpunkt im Bereich der agentenorientierten Softwareentwicklung liegt auf Entwicklungsmethoden und -tools, die eine softwaretechnische Umsetzung der agentenorientierten Systemsicht unterstützen. Eine sorgfältige Auswahl solcher Methoden und Tools bildet den Gegenstand dieses Buches.

Ziele des Buches

Das vorliegende Buch will eine methoden- und toolzentrierte Einführung in die agentenorientierte Softwareentwicklung bieten, die in anschaulicher Weise zweierlei vermittelt:

- den Aufbau, die Verwendung und die Leistungsmerkmale von derzeit verfügbaren agentenorientierter Methoden und Tools; und
- die Systemsichtweise und die grundlegendenen Konzepte, die für die agentenorientierte Softwareentwicklung charakteristisch sind.

Mit dieser inhaltlichen Konzeption zielt das dieses Buch vor allem darauf ab, seinen Leserinnen und Lesern bei ihren Einstieg in den Bereich der agentenorientierten Software, bei ihrer kritischen Einordnung von Agentenorientierung als Softwareentwicklungsansatz, und bei ihrer Entscheidung über den Einsatz dieses Ansatzes im Rahmen eigener Entwicklungsarbeiten – seien dies studienbegleitende Softwarepraktika oder kommerzielle Softwareprojekte – behilflich zu sein. Last but not least will dieses Buch Neugier wecken und zur weiteren Exploration dieses jungen und vielversprechenden Ansatzes ermuntern.

Angesprochener Leserkreis

Das Buch richtet sich grundsätzlich an alle, die sich für agentenorientierte Softwareentwicklung interessieren. Besonders eignet es sich für

- Studierende, als Ergänzungstext für Veranstaltungen zu Themenkomplexen wie „Softwareentwicklung“, „(Multi-)Agententechnologie“ und „Künstliche Intelligenz“.
- Dozenten, als Materialquelle für ihre Veranstaltungen zu diesen Themenkomplexen; und
- Softwareentwickler, als Entscheidungshilfe bei der Auswahl von agentenorientierten Methoden oder Tools.

Webseite zum Buch

Unter http://www7.in.tum.de/~weissg/AOSE-MT sind die in diesem Buch enthaltenen Abbildungen und Tabellen sowie Verweise auf weiterführendes Material rund um „agentenorientierte Softwareentwicklung“ verfügbar.

Danksagung

Wir danken allen Kollegen, die sich die mühevolle Arbeit des Korrektur- und Probelesens gemacht haben. Die Fertigstellung dieses Buches hat weitaus mehr Wochenenden und Freizeit beansprucht, als wir ursprünglich vorgesehen hatten; für das dafür entgegengebrachte Verständnis bedanken wir uns ganz herzlich bei unseren Freunden und Familien. Schließlich möchten wir uns verlagsseitig bei Herrn Dr. Hermann Engesser, Frau Pannewig-Vogt und Herrn Holzwarth bedanken für die hervorragende (und auch überaus geduldige!) Zusammenarbeit.

Garching, *Gerhard Weiß*
Juli 2004 *Ralf Jakob*

Inhaltsverzeichnis

Teil I

Einführung

1

Agentenorientierung in der Softwaretechnik

The important thing in science is not so much to obtain new facts as to discover new ways of thinking about them.

W. Bragg

Gegenstand dieses Buches sind Methoden und Tools für die agentenorientierte Softwareentwicklung, also für den Entwurf, die Analyse und die Implementierung von Softwaresystemen unter dem Blickwinkel einer Menge von Agenten. Zur besseren Einordnung dieser Methoden und Tools gibt dieses Kapitel einen einführenden Überblick über Agentenorientierung in der Softwaretechnik. Der Überblick fokussiert auf die drei Fragen

- Was ist unter „Agent“ zu verstehen?
- Wodurch zeichnet sich die Agentenorientierung aus?
- Wie sieht die aktuelle Forschungs- und Anwendungslandschaft aus?

und gibt zahlreiche Verweise auf weiterführende Literatur und Web-Ressourcen.

1.1 Das Agentenkonzept

Geschichtliche Anmerkungen

Das Agentenkonzept, welches der agentenorientierten Softwareentwicklung zugrunde liegt, stammt aus der (Verteilten) Künstlichen Intelligenz. Damit reichen die Wurzeln dieses Konzepts zurück bis in die fünfziger Jahre des vergangenen Jahrhunderts. In der vergangenen Dekade hat sich das Agentenkonzept überaus erfolgreich in verschiedenen Bereichen der Informatik etabliert und die Agenten- und Multiagententechnologie gilt heute als außerordentlich vielversprechende Zukunftstechnologie im IT-Bereich. Erste Überlegungen zu agentenorientierter Softwareentwicklung finden sich Anfang der neunziger Jahre [126, 134]. Das wachsende Interesse an diesem Softwareentwicklungsansatz führte schließlich im Jahre 2000 zur Gründung der internationalen Workshop-Reihe

„Agent-oriented software engineering“ (AOSE) [67, 128, 129, 302]. (Weiterführende geschichtliche Betrachtungen zur Entwicklung des Agentenkonzepts in der verteilten und nicht-verteilten Künstlichen Intelligenz finden sich in [305, Appendix A].)

„Schwache“ Charakterisierung

Der Agentenbegriff war und ist Gegenstand intensiver Diskussionen und Präzisierungsbemühungen (z.B. [122, 142, 222]). Bislang gibt es keine standardisierte Definition des Agentenkonzeptes, aber in den vergangenen Jahren zeichnete sich ein breite Akzeptanz der folgenden Charakterisierung – in der Literatur häufig als „schwache“ Charakterisierung oder „schwache“ Bedeutung (weak notion) des Agentenbegriffs bezeichnet – ab [307]:

> *Ein Agent ist eine abgrenzbare (Software/Hardware-)Einheit, die in der Lage ist, die ihr vorgegebenen Aufgaben flexibel, interaktiv und autonom zu verfolgen.*

Dabei liegt den Schlüsselmerkmalen eines Agenten, die in dieser Charakterisierung genannt werden, folgende Vorstellung zugrunde:

- *Flexibilität.* Ein Agent ist in der Lage, sowohl reaktiv als auch proaktiv zu handeln. Reaktiv bedeutet, dass er in angemessener Zeit und auf geeignete Weise auf Änderungen in der Umgebung, in der er agiert, und auf Änderungen in den Anforderungen, die an ihn gestellt werden, reagiert. Proaktiv bedeutet, dass er vorausschauend, planend und ausgerichtet auf ein oder mehrere Ziele handelt. Flexibilität, zusammengesetzt aus Reaktivität und Proaktivität, ist also die Fähigkeit, möglicherweise unerwartete Ereignisse handhaben und zugleich plan- und zielorientiert handeln zu können.
- *Interaktivität.* Ein Agent ist in der Lage, mit seiner Umwelt – insbesondere also mit menschlichen Akteuren und mit anderen Agenten – zu interagieren. Die Interaktionen selbst können dabei auf sehr hohem Niveau stattfinden, d.h. sie können ausgesprochen kommunikations- und wissensintensiv sein, und sie dienen der Koordination mit Dritten, also der Aktivitätsabstimmung und Handhabung wechselseitiger Abhängigkeiten [199]. Koordination wird dabei sowohl im Sinne von Kooperation (gemeinsame Verfolgung von möglicherweise gemeinsamen Plänen und Zielen) als auch im Sinn von Kompetition (Verfolgung von sich teilweise oder sogar ganz ausschließenden Zielen) verstanden. Beispiele für Interaktionsformen, die für Agenten als typisch erachtet werden, sind Verhandlung und Konfliktlösung im Rahmen kooperativer Planungsaktivitäten und kompetitiver Verkaufsprozesse. Interaktivität erfordert eine präzise Schnittstelle und diese verschattet üblicherweise

sämtliche Agenten-Interna. Generell bezeichnet also Interaktivität alle (höheren) sozialen – kommunikativen, kooperativen und kompetitiven – Fähigkeiten, die ein Agent besitzt.

- *Autonomie.* Ein Agent ist in der Lage, im Rahmen seiner Aufgabenbearbeitung weitgehend selbständig und ohne Rücksprache und Abstimmung mit Dritten (menschlichen Benutzern oder anderen Agenten) zu entscheiden, welche Aktivitäten er ausführt. Häufig wird dabei gefordert oder implizit angenommen, dass die vom Agenten zu treffenden Entscheidungen nichttrivial sind, also beispielsweise umfangreiche Wissensverarbeitung erfordern oder in ihren Auswirkungen signifikant sind. Ein Agent besitzt damit in gewissem Umfang Entscheidungsbefugnis und Handlungsfreiheit und unterliegt insofern nur in eingeschränktem Maß der Kontrolle durch Dritte. In letzter Konsequenz impliziert Autonomie die Fähigkeit eines Agenten, seine Komplexität und die Komplexität seiner Anwendung selbstständig handzuhaben und damit vor allem auch seine Benutzer unter Wahrung ihrer Interessen zu entlasten.

Zu beachten ist, dass jedes dieser drei Schlüsselmerkmale in unterschiedlicher Ausprägung und Intensität vorliegen kann und damit der Übergang von „Agent“ zu „Nicht-Agent“ fließend ist.

„Starke“ Charakterisierung

Einen alternativen und weit verbreiteten Zugang zum Agentenkonzept bietet die so genannte starke Charakterisierung oder starke Bedeutung (strong notion), gemäß der ein Agent eine (Hardware/Software-)Einheit ist, die – in Analogie zum Menschen – mentale Haltungen beziehungsweise Zustände (mental attitudes) besitzt. Drei Arten von mentalen Zuständen spielen im Bereich der agentenorientierten Softwareentwicklung eine herausragende Rolle:

- *informationsbezogene Zustände* wie beispielsweise Wissen, Vermutungen und Annahmen;
- *konative Zustände* wie beispielsweise Intentionen, Pläne und Verpflichtungen (anderen oder sich selbst gegenüber); und
- *affektive Zustände* wie beispielsweise Ziele, Präferenzen und Wünsche.

Eine weitere Art von mentalen Zuständen, die unterschieden werden kann, sind emotionale Zustände. Agenten, die künstliche Emotion – Freude, Erstaunen, Angst, usw. – zeigen können (z.B. in Mimik und Gestik), werden seit einigen Jahren verstärkt insbesondere im Kontext von multimedialen Mensch-Maschine-Schnittstellen thematisiert (z.B. [52, 66]).

Verhältnis der beiden Charakterisierungen

Während die schwache Charakterisierung primär generische funktionale Merkmale eines Agenten erfasst, betrifft die starke Charakterisierung primär die Architektur und interne (Kontroll-) Struktur und damit generische strukturelle Merkmale eines Agenten. Beispielsweise kann aus der Feststellung, ein Agent besitzt Wissen, unmittelbar abgeleitet werden, dass er eine Strukturkomponente besitzt (eine „Wissensbasis"), in der all sein Wissen abgelegt ist, und aus der Feststellung, ein Agent verfolgt Pläne, lässt sich ableiten, dass er eine „Planbasis" sowie eine plankonforme Ablaufsteuerung benötigt. Die schwache und die starke Charakterisierung überlappen sich zumindest teilweise und sind als sich ergänzende Perspektiven des Agentenkonzepts zu verstehen – tatsächlich liegen den allermeisten Arbeiten in Forschung und Anwendung beide Charakterisierungsansätze zugrunde, d.h. die beiden Ansätze werden in aller Regel in Kombination statt in Reinform angewendet.

Weitere agentenspezifische Merkmale

Sehr häufig werden die obigen Charakterisierungen – insbesondere die schwache Charakterisierung – erweitert und konkretisiert, indem zusätzliche Merkmale mit „Agent-Sein" assoziiert werden. Zu den prominentesten dieser Merkmale gehören:

- *Situiertheit/Eingebettetheit.* Ein Agent ist sensorisch und/oder aktorisch eng mit seiner Umwelt gekoppelt, er agiert und interagiert also unmittelbar in einem konkreten sozio-technischen Umfeld und nicht etwa nur in einem abstrakten Modell dieses Umfelds.
- *Lernfähigkeit/Adaptivität.* Ein Agent optimiert selbständig seine Funktionalität in Hinblick auf die an ihn gestellten und sich im Laufe der Zeit möglicherweise ändernden Anforderungen.

Sonstige Merkmale, die in der Literatur häufig als elementar für „Agent-Sein" bezeichnet werden und die aus softwaretechnischer Sicht bemerkenswert sind, sind beispielsweise *Persistenz* (ein Agent realisiert nicht nur eine einmalige Berechnung sondern agiert über einen längeren Zeitraum); *Rationalität* (ein Agent agiert im Rahmen seiner Fähigkeiten und Kenntnisse und in Hinblick auf die Erfüllung seiner Aufgaben und Ziele bestmöglich, er maximiert also „so gut er kann seine Erfolgsaussichten"); *Gutartigkeit* (ein Agent handelt nicht absichtlich entgegen der Interessen seiner menschlichen Benutzer); und *Abgeschlossenheit* (ein Agent ist eine funktional abgeschlossene und ausführbare Entität).

Anmerkung zum Sprachgebrauch

Im Bereich der agentenorientierten Softwareentwicklung und Agententechnologie wird das Wort „Agent" häufig in attributierter Form verwendet. Da die Vielfalt dieser attributierten Formen

auf den ersten Blick durchaus sehr verwirrend wirken kann, werden im Folgenden die gebräuchlichsten Attributierungen vorgestellt.

Die Bedeutung, die den einzelnen agentenspezifischen Merkmalen zukommt, ist abhängig vom konkreten Forschungs- beziehungsweise Anwendungsfokus. Aus diesem Grund werden Merkmale, denen besondere Bedeutung zukommt, häufig explizit attributiv benannt; Beispiele für solche Attributierungen sind „autonomer Agent“, „kooperierender Agent“, „reaktiver Agent“, „adaptiver Agent“ und „rationaler Agent“. Üblich ist auch die Bezeichnung „*intelligenter Agent*“, mit der dem Umstand Rechnung getragen wird, dass einem Agent aufgrund seiner Flexibilität, Interaktivität, Autonomie, Lernfähigkeit, Rationalität, usw. durchaus eine gewisse Intelligenz im umgangssprachlichen Sinn zugesprochen werden kann. (Ein eindeutiger und allgemein anerkannter Intelligenzbegriff existiert nicht. In der Intelligenz-Forschung herrscht weitgehend Übereinstimmung darüber, dass sich menschliche Intelligenz aus verschiedenen „Teil-Intelligenzen“ wie beispielsweise logisch-mathematische Intelligenz, emotionale Intelligenz, soziale Intelligenz und kollektive Intelligenz zusammensetzt.)

Neben der Betonung einzelner Merkmale ist es auch üblich, den Begriff „Agent“ dadurch zu präzisieren, dass er mit dem Einsatzbereich oder -zweck oder einer sonstigen herausragenden (nicht-agentenspezifischen) Eigenschaft des jeweiligen konkreten Agenten attributiert wird. Bekannte Beispiele hierfür sind: Informationsagent, Interface-Agent, Wrapper-Agent, Transaktionsagent, Verkaufsagent, Assistenzagent, virtueller Agent und mobiler Agent.

Mit der Bezeichnung „agentenorientierte Software“ wird üblicherweise Software bezeichnet, die agentenorientiert – also unter dem Blickwinkel einer Menge von Agenten – entwickelt wurde. (Ein einzelner Softwareagent ist demnach ein Spezialfall von agentenorientierter Software.) Dieser Sprachgebrauch wird auch für dieses Buch übernommen.

1.2 Merkmale und Potential

Zu den großen Fortschritten, die in der Softwaretechnik erzielt wurden, zählt die Herausbildung und softwaretechnische Umsetzung von grundlegenden Systembetrachtungsweisen, die eine erfolgreiche, systematische und effiziente Entwicklung von Softwaresystemen unterstützen. Beispiele für diese Betrachtungsweisen sind Strukturorientierung [87, 238], Objektorientierung (z.B. [51,

61], Komponentenorientierung (z.B. [258, 280]), Aspektorientierung (z.B. [166, 191]), Modellorientierung (z.B. [205, 273]), Architekturorientierung (z.B. [263]), Patternorientierung (z.B. [64, 124]), Aufgabenorientierung [237] und – meist im Kontext von betrieblichen Informationssystemen – Prozessorientierung (z.B. [28]). In diese Liste ist Agentenorientierung als eine neue, im Entstehen begriffene Systembetrachtungsweise einzuordnen. Welchen Nutzen eine Betrachtungsweise bringt, ist eine Frage, die abschließend nur in der Praxis und basierend auf langjähriger Erfahrung zu beantworten ist. Agentenorientierung ist, wie beispielsweise auch Komponenten- und Architekturorientierung, zu jung, um eine solche empirisch gestützte Antwort geben zu können. Im Folgenden werden grundlegende qualitative Merkmale von Agentenorientierung beschrieben, die es nahelegen, dieser Betrachtungsweise ein sehr hohes Nutzen- und Akzeptanzpotential in der Softwaretechnik zu bescheinigen. Im Wesentlichen sind diese Merkmale auch der Grund für das rasch wachsende Interesse, das die agentenorientierte Softwareentwicklung seit einigen Jahren erfährt.

Systemsicht und Abstraktionsebene

Agentenorientierung legt die Metapher eines Softwaresystems als eine menschliche Organisation nahe und eröffnet damit eine innovative, qualitativ anspruchsvolle und zugleich intuitiv verständliche Sicht auf Software. Innovativ und qualitativ anspruchsvoll ist diese Sicht, weil sie es erlaubt, Softwaredesign als Organisationsdesign zu verstehen. Damit erschließt sich dem Softwareentwickler ein breiter Fundus an organisationstheoretischen Konzepten und Techniken (z.B. [81, 123]), die softwaretechnisch eingesetzt werden können. Intuitiv verständlich ist diese Sicht, weil der Umgang mit organisationalen Begrifflichkeiten zu unserem Alltag gehört. Es bereitet deshalb keine Schwierigkeit, sich ein Softwaresystem als Organisation (oder auch als Zusammenschluss von mehreren Organisationen) vorzustellen, in der Softwareeinheiten (Agenten) unter Berücksichtigung von vorgegebenen Berechnungs- und Verhaltensvorschriften (Regeln, Normen, Gesetzen, usw.) Aufgaben erledigen und zu diesem Zweck beispielsweise autonom verhandeln, (Ressourcen-)Konflikte lösen, dynamisch übergeordnete organisationale Einheiten (z.B. Teams) bilden und auflösen, innerhalb dieser übergeordneten Einheiten bestimmte Rollen (z.B. „Ressourcenverwalter“ und „Serviceanbieter“) spielen und als Rolleninhaber bestimmte Verpflichtungen wahrnehmen.

Charakteristisch für die agentenorientierte Systemsicht ist vor allem auch, dass sie eine neue, von anderen Betrachtungsweisen verschiedene Abstraktionsebene bietet (weitere Ausführungen hierzu finden sich z.B. in [153, 318]). Der Schritt hin zu dieser Abstraktionsebene ist konform mit einer Entwicklung, die sich in

höheren Programmiersprachen widerspiegelt und die für die Programmierung im Großen eine notwendige Voraussetzung ist: die Zunahme des Abstrahierungsgrades, weg von der Maschinenebene hin zur Problemebene.

Komplexitätshandhabung

Software gilt als inhärent komplex und ihre Komplexität wird, wie schon in der Vergangenheit, auch weiterhin dramatisch zunehmen (siehe z.B. [62]). Ein entscheidendes Kriterium für die Beurteilung eines Sofwareentwicklungsansatzes ist damit seine Eignung zur Handhabung von Komplexität. Vier elementare Techniken zur Komplexitätshandhabung, denen in der Softwaretechnik sehr große Bedeutung zukommt [51, 203], sind:

- *Dekomposition*, also die Zerlegung in kleinere und damit übersichtlichere Teile, die in hohem Maß unabhängig voneinander entwickelt werden können.
- *Abstrahierung*, also die Erstellung eines Modells, welches die unwichtige Aspekte ausblended und die wesentliche Aspekte erfasst.
- *Strukturierung*, also die Bestimmung der (geordneten) Beziehungen und (gewünschten) Wechselwirkungen zwischen den Teilen des Gesamtsystems.
- *Wiederverwendung*, also die systematische Verwendung von Ergebnissen (Dokumenten und Prozessen), die in einem Softwareprojekt erzielt wurden, in zukünftigen Projekten.

Agentenorientierung unterstützt jede diese vier Techniken auf sehr natürliche Weise (weitere Betrachtungen hierzu finden sich z.B. in [153]). Erstens, sie ermöglicht eine gerichtete Zerlegung eines Softwaresystems in atomare Teile (Agenten) und aus ihnen zusammengesetzte Konstrukte (Agentengruppen). „Gerichtet“ bedeutet dabei, dass aufgrund der Semantik des Agentenkonzepts eine willkürliche und in Hinblick auf die Anwendung möglicherweise völlig ungeeignete Systemzerlegung unwahrscheinlich ist. Anders gesagt, das Agentenkonzept ist semantisch reichhaltig genug, um konkrete Hilfestellung and Anleitung bei der Systemzerlegung zu geben (vgl. diesbezüglich die Konzepte „Objekt“ und „Komponente“). Zweitens, sie erlaubt die Modellierung von Systemen und Anwendungen auf der Ebene von „Wissen“ (knowledge level [217]) und „Sozialität“ (social level [154]) und bietet damit vielfältige Möglichkeiten zur systematischen Abstraktion von Implementierungs- und Anforderungsdetails. Drittens, Beziehungen und Abhängigkeiten zwischen den einzelnen Teilen (Agenten) lassen sich unmittelbar aus den Interaktionen ableiten, die zwischen den Teilen im Rahmen ihrer Aufgabenbearbeitung erforderlich oder erlaubt sind. Das Spektrum möglicher Beziehungen reicht

dabei von der klassischen Client/Server-Strukturen über marktbasierten Strukturen bis hin zu Peer-to-Peer-Strukturen. Schließlich viertens, es gibt eine Reihe von agentenspezifischen Artifakten, die bei agentenorientierter Softwareentwicklung üblicherweise generiert werden und die sich für eine Wiederverwendung hervorragend eignen. Beispiele für solche Artefakte sind: ein einzelner Softwareagent (d.h. Programmcode, der einen Agenten realisiert); eine Menge (Team) von Softwareagenten, die gemeinsam eine bestimmte Aufgabe bearbeiten; agenteninterne Bestandteile (z.B. die Wissensbasis oder Planungskomponente eines Agenten); Architekturen von einzelnen Agenten und von Agententeams; Interaktionsstrukturen und -protokolle; und vollständige Agentenplattformen.

Anwendungsbreite

Agentenorientierung eignet sich in besonderem Maß zur Realisierung von Anwendungen, die durch folgende Merkmale ausgezeichnet sind:

- *Verteiltheit*, d.h. Daten, Information und Wissen liegen räumlich und/oder logisch verteilt vor und werden verteilt verarbeitet;
- *Offenheit*, d.h. die Anzahl und der Typ der Hardware- und Software-Komponenten, die in die Anwendung involviert sind, ist variabel und möglicherweise a priori (zur Designzeit) nicht genau bekannt; und
- *Einbettung* in komplexe – dynamische, schwer vorhersagbare, nur beschränkt einsehbare, heterogene, usw. – sozio-technische Umgebungen („situierte Anwendung“).

Mit zunehmenden technologischen Fortschritt, etwa in Rechnervernetzung und Plattforminteroperabilität, kommt solchen Anwendungen in den verschiedensten kommerziellen, industriellen und wissenschaftlichen Bereichen – von E-Commerce und E-Business über Fertigungslogistik und Telekommunikation bis hin zu Wissensmanagement und Simulation von sozialen und biologischen Prozessen – eine zentrale Bedeutung bei (siehe auch [319]). Generell sind diese drei Merkmale kennzeichnend für eine Vielzahl von Anwendungen, die auf neuen Modellen und Ansätzen zur Informationsverarbeitung wie beispielsweise Grid Computing (z.B. [120]), Peer-to-Peer Computing (z.B. [234, 260, 286]), Web Computing (z.B. [121]), Pervasive und Ubiquitous Computing [108, 202], Autonomic Computing [146] und Mobile Computing (z.B. [252]) basieren. Die Eignung für solche Anwendungen resultiert daraus, dass ihre Merkmale mit den drei Schlüsselmerkmalen eines Agenten – Flexibilität, Interaktivität und Autonomie – korrespondieren. Zum einen implizieren Verteiltheit und Offenheit eine

verteilte und offene Kontrollstruktur (die eine parallele und nebenläufige Verarbeitung ermöglicht) und damit insbesondere die Notwendigkeit, zur Realisierung der Anwendung Softwareeinheiten zu verwenden, die *autonom* (ohne zentrale Kontrolle) agieren können. Zum anderen implizieren die Merkmale Offenheit und Einbettung die Notwendigkeit, möglichst *flexible* Softwareeinheiten einzusetzen, also Softwareeinheiten, die beispielsweise in der Lage sind, trotz unerwarteter Änderungen in der technologischen Infrastruktur oder in den Benutzeranforderungen geeignet zu agieren. Und zum Dritten implizieren Verteiltheit, Offenheit und Einbettung gleichermaßen die Notwendigkeit, Softwareeinheiten einzusetzen, die in der Lage sind, zu *interagieren* (und zwar auf möglichst flexible und autonome Weise) , sei es beispielsweise zum Zweck des einfachen Datenaustausches oder zum Zweck der wissensbasierten Verhandlung über die Kosten für die Nutzung einer bestimmten Resource.

Autonomie als Systemeigenschaft

Aus softwaretechnischer Sicht stellt Autonomie das markanteste und in seinen Auswirkungen weitreichenste Merkmal des Agentenkonzeptes und damit der Agentenorientierung dar [218]. Dieses Merkmal, wenngleich es auf den ersten Blick „radikal“ und „revolutionär“ erscheinen mag, kann als ein natürlicher nächster Schritt in der Evolution generischer Softwareprinzipien verstanden werden [153]. Elementare Softwareeinheiten, die sich im Laufe dieser Evolution herausgebildet haben – monolithische Programme, Module, Prozeduren, Objekte und Komponenten –, weisen einen wachsenden Grad an Lokalität und Kapselung von Daten und von Zustandskontrolle auf. All diesen Softwareeinheiten ist gemeinsam, dass ihre Aktivierung über externe Ereignisse (z.B. dem Startbefehl durch einen Benutzer oder dem Empfang einer Nachricht von einer anderen Softwareeinheit) erzwungen werden kann – die Einheiten entscheiden also nicht selbständig, ob sie z.B. auf eine Nachricht hin tatsächlich aktiv werden (eine Berechnung ausführen, Daten zur Verfügung stellen, usw.). Agentenorientierung überwindet diese Einschränkung, indem sie mit der Autonomieeigenschaft zusätzlich die Kapselung der Kontrolle über die Aktivierung einer Softwareeinheit vorsieht („Selbst- statt Fremdaktivierung“, Selbst- statt Fremdbestimmung“ und „Selbststatt Fremdverantwortung“). In der Literatur wird diese erweiterte Kapselung häufig durch eine vergleichende Gegenüberstellung des Agentenkonzepts mit dem Objektkonzept im Sinne des derzeit marktdominanten objektorientierten Paradigmas erläutert (z.B. [153, 225, 305], Analoges gilt für das Verhältnis von Agenten- und Komponentenkonzept): während Objekte neben ihrer Identität („Wer?“) und ihren Zustand („Was?“) ihr *passives* Verhalten

(„Was, falls aktiviert“?) kapseln, kapseln Agenten zusätzlich Freiheitsgrade in ihrer (Inter-)Aktivität und damit *aktives* Verhalten („Wie, wann und mit wem, falls überhaupt?“). Zwei bekannte Slogans, in denen dieser Unterschied zum Ausdruck kommt, sind: „objects do it for free, agents do it because they want to“ und „objects do it for free, agents do it for money“.

Der Schritt hin zu Softwareautonomie lässt sich nicht nur historisch motivieren, sondern spiegelt auch praktischen Bedarf wider. Zum einen implizieren, wie oben dargestellt, eine Reihe von Anwendungen indirekt die Notwendigkeit, Software mit Autonomie auszustatten. Zum anderen wird Autonomie als Systemeigenschaft immer häufiger auch direkt, gleichsam „per Definition“, gefordert. So ist es beispielsweise üblich, ein Peer-to-Peer System als ein sich selbstorganisierendes System von gleichberechtigten autonomen Einheiten [234] zu verstehen und in [179] wird Autonomie als wichtige Eigenschaft von Web Services genannt (zusätzlich zu den in der W3C Definition von Web Services genannten Eigenschaften). Autonomie als gewollte beziehungsweise notwendige Eigenschaft von IT Systemen steht – in unterschiedlichen Nuancen und Ausprägungen („self-governing“, „self-structuring“, „self-healing“, „self-repairing“, u.ä.) – auch im Mittelpunkt von verschiedenen Initiativen, die in den vergangenen Jahren von führenden Vertretern der IT Branche lanciert wurden. Zu nennen ist hier insbesondere IBM's Autonomic Computing Initiative [146], aber auch Sun's N1 Initiative [276], HP's Adaptive Enterprise Initiative [138] und Microsoft's Dynamic Systems Initiative [204] (wobei die Schwerpunkte der drei letztgenannten Initiativen fast ausschließlich im Server- und Infrastrukturbereich liegen). Gemeinsam ist diesen Initiativen die Vision von autonomen IT Systemen, die ihre Komplexität und die Komplexität ihrer Umgebung von ihren menschlichen Benutzern abschirmen.

Verträglichkeit

Mitentscheidend für das Potential eines neuen Ansatzes – einer Betrachtungsweise, einer Technik, einer Methode, usw. – ist auch seine Verträglichkeit mit bereits bestehenden und in der Praxis etablierten Ansätzen. Agentenorientierung ist in hohem Maß verträglich mit anderen Ansätzen. Insbesondere erhebt die agentenorientierte Betrachtungsweise *nicht* den Anspruch, andere Betrachtungsweisen zu ersetzen oder auszuschließen. Beispielsweise

- ergänzen sich die Abstraktionsniveaus von Agenten- und Objektorientierung auf sinnvolle Weise [224, 266];
- haben Agenten und Komponenten das Merkmal der Abgeschlossenheit und den Fokus auf ihre Schnittstelle gemeinsam und das Agentenkonzept kann durchaus als Spezialisie-

rung oder Generalisierung (je nach Sicht) des Komponentenkonzepts verstanden werden;

- hat Agentenorientierung durch ihren Fokus auf organisationale Strukturen (auf der Ebene der individuellen Agenten) einen engen Bezug zur Architekturorientierung;
- hat die Agentenorientierung mit ihren Fokus auf Interaktivität und damit auf Folgen von aufeinander abgestimmten Aktionen einen grundlegende Gemeinsamkeit mit der Prozessorientierung; und
- betont die Agentenorientierung ähnlich wie die Aufgabenorientierung die Bedeutung der Erfassung von übergreifenden Aufgaben (also von Aufgaben auf Akteurs- statt z.B. Objektebene) und ihrer Abhängigkeiten.

Damit integriert Agentenorientierung verschiedene Kernaspekte anderer Ansätze und bietet grundsätzlich auch die Möglichkeit, in Kombination mit anderen Ansätzen angewandt zu werden. Weitere Betrachtungen zum Verhältnis von agentenorientierter Softwaretechnologie und Mainstream-Softwaretechnologie finden sich in [215].

Weitere Akzeptanzfaktoren

In den vergangenen Jahren entwickelten sich

- die Methoden- und Toolunterstützung und
- die *Standardisierung* von agentenspezifischen Konzepten, Konstrukten und Techniken

zu Schwerpunktthemen im Bereich der agentenorientierten Softwareentwicklung. Dies ist deshalb besonders betonenswert, da ein Ansatz, der nicht von leistungsfähigen Standards, Methoden und Tools getragen wird, in der Softwarepraxis nur sehr schwer Fuß fassen kann.

Silver Bullet?

Softwareentwicklung ist viel zu facettenreich und komplex als dass es ein „Allheilmittel“ – eine silver bullet [59] – geben könnte, welches immer (oder wenigstens meistens) unter Einhaltung des verfügbaren Zeit- und Kostenrahmens zu einem optimalen Softwaresystem führt. So ist bekanntermaßen die Objektorientierung kein solches Allheilmittel (z.B. [27, 60, 117]) und es wäre natürlich auch unrealistisch, anzunehmen, Agentenorientierung oder ein anderer Entwicklungsansatz sei ein solches. Wo die kritischen Schwachstellen des agentenorientierten Ansatzes im Detail liegen, wird sich in der Praxis zeigen. Im Folgenden wird auf drei grundsätzliche Herausforderungen, die Agentenorientierung mit sich bringt, hingewiesen.

Herausforderungen

Eine leicht zu unterschätzende Herausforderung ist der sachliche Umgang mit dem Agentenkonzept. Aufgrund seiner intuitiven Verständlichkeit kann dieses Konzept schnell dazu verleiten (vor allem im Falle fehlender Vorkenntnisse und Erfahrungen mit agentenorientierter Entwicklung und Agententechnologie), den Bezug zur softwaretechnischen Relevanz und Machbarkeit und nicht zuletzt zu den eigentlichen Anforderungen an das zu bauende System zu verlieren. Mindestens zwei der in [308] genannten Fallstricke der agentenorientierten Softwareentwicklung stehen in unmittelbaren Zusammenhang mit dieser Herausforderung:

- „You confuse buzzwords with concepts“ und
- „You see agents everywhere“.

Der zweitgenannte Fallstrick führt häufig dazu, dass auch solche Systemeinheiten als „Agenten“ bezeichnet werden, die dieser Bezeichnung schlichtweg nicht gerecht werden – ein solcher oberflächlicher Umgang mit dem Agentenkonzept kann eine geeignete und sinnvolle agentenorientierte Systemrealisierung ganz erheblich erschweren oder sogar unmöglich machen.

Eine zweite zentrale Herausforderung ist die korrekte und präzise Erfassung und Spezifikation von Autonomie als Softwareeigenschaft. Diese Herausforderung, die mit wachsenden Bedarf an autonomen Informationssystemen über den agentenorientierten Ansatz hinaus von zentraler Bedeutung ist, wirft auch wichtige Fragen der Systemsicherheit und des Datenschutzes auf. Dies folgt daraus, dass ein Softwareagent als autonome Einheit im Rahmen der ihm übertragenen Verantwortlichkeiten möglicherweise Entscheidungen treffen kann, die beispielsweise signifikante Konsequenzen finanzieller oder rechtlicher Art für seine menschlichen Benutzer haben. Als Entwickler steht man also vor der Aufgabe, Autonomie weder zu restriktiv noch zu großzügig zu fassen, da sonst erwünschte Effekte (z.B. Benutzerentlastung) nicht erzielt beziehungsweise unerwünschte Effekte (z.B. emergente Instabilität des Gesamtsystems) nicht ausgeschlossen werden können.

Eine dritte und die herausragendste Herausforderung ist die Anbindung des agentenorientierten Softwareansatzes an praxisrelevante Qualitäts- und Entwicklungsstandards. Zwar wurden, wie oben bereits angemerkt, diesbezüglich in den vergangenen Jahren enorme Fortschritte erzielt, jedoch reichen diese noch nicht aus, um einen breiten und umfassenden industriellen und kommerziellen Einsatz von agentenorientierter Softwaresystemen zu ermöglichen. Konkrete Vorschläge und Anregungen zur Realisierung dieser Anbindung werden beispielsweise in [194, 215, 222] und [196, Abschnitte 4 und 7] gemacht.

1.3 Schwerpunkte in Forschung und Anwendung

Aufgrund seines großen Potentials erfährt der agentenorientierte Ansatz zur Softwareentwicklung seit einigen Jahren ein rasch wachsendes Interesse in Forschung und Praxis. Ausdruck findet dieses Interesse in einer inzwischen kaum mehr überschaubaren Anzahl von Arbeiten, in denen die unterschiedlichsten Aspekte dieses Ansatzes thematisiert werden. Im Folgenden wird, nach Schwerpunkten geordnet, ein Überblick über die aktuelle Forschungs- und Anwendungslandschaft der agentenorientierten Software(entwicklung) gegeben.

Entwicklungsmethoden

Mit dem Interesse an agentenorientierter Software wuchs vor allem auch das Interesse an agentenorientierten Entwicklungsmethoden. Eine Vielzahl solcher Methoden sind inzwischen verfügbar, die sich nach ihrer disziplinären Verwurzelung in vier Gruppen unterteilen lassen:

- *Agententechnologie* als Ausgangspunkt. Der Fokus dieser Methoden liegt auf agentenspezifischen Abstraktionen (Gruppe, Organisation, Rolle, usw.) und Verfahren (z.B. zur wissensgestützten Koordination). Typische Vertreter dieser Gruppe sind *Gaia* [310], *Aalaadin* [105, 135], *SODA* [229], *EXPAND* [54] und *ADELFE* [49].
- *Objektorientierung* als Ausgangspunkt. Diese Methoden resultieren aus agentenspezifischen Erweiterungen von bestehenden objektorientierten methodischen Verfahren und Prinzipien. Typische Vertreter dieser Gruppe sind MASSIVE [189], *KGR* [169], *MaSE* [86], *Prometheus* [236], *ODAC* [127], *PASSI* [242], *AOAD* [63] und *MASB* [210].
- *Requirements Engineering* als Ausgangspunkt. Eine Methode, die eng an Techniken und Formalismen angelehnt ist, die aus dem Bereich des Requirements Engineering stammen, ist *Tropos* [57]. Agentenorientiertes Requirements Engineering [314] weist interessante Parallelen zum zielorientierten Requirements Engineering [216, 317] auf. Diese Parallelen sind zum Teil darin begründet, dass der Modellierung von (menschlichen) Agenten im Requirements Engineering eine große Bedeutung beigemessen wird [288]; Betrachtungen zum Agentenkonzept im Requirements Engineering finden sich z.B. in [315, 316].
- *Knowledge Engineering* als Ausgangspunkt. Der Fokus dieser Methoden liegt auf der Identifikation und Modellierung des Wissens, welches die einzelnen Softwareagenten besitzen. Die

zwei bekanntesten Vertreter sind *CoMoMAS* [130] und *MAS-CommonKADS* [147].

Die Einteilung in diese Gruppen spiegelt die Vielfalt der methodischen Ansätze wider. Eine Kombination von methodischen Elementen über diese Gruppen hinweg ist durchaus möglich, wie beispielsweise die Methode *MESSAGE* [103] zeigt. Einen guten Überblick über agentenorientierte Methoden kann man sich mit [48, 195, 295] verschaffen; einen weiteren hilfreichen, wenn auch nicht mehr ganz aktuellen Überblick bietet [32].

Entwicklungstools und Plattformen

Inzwischen ist eine kaum überschaubare Anzahl von Tools verfügbar, die eine agentenorientierte Softwareentwicklung unterstützen. Die meisten dieser Tools sind frei erhältliche Prototypen, wobei seit einigen Jahren auch kommerzielle Tools angeboten werden. Einen guten Eindruck von der Vielfalt der derzeit verfügbaren Tools kann man sich auf den Webseiten [21, 200] verschaffen; Kurzbeschreibungen ausgewählter Tools finden sich auch in [195]. Zu den bekanntesten Entwicklungstools zählen u.a. das *FIPA-OS* Toolkit (z.B. [219, 220]), *JADE* [80]; *Zeus*-Toolkit [271], *MadKit* [197], *agentTool* [24, 25], *JAFMAS* [152], *JACK* [3] und *AGENTBUILDER* [16]. *JACK* und *AGENTBUILDER* werden kommerziell vertrieben. Neben diesen „general purpose“ Tools gibt es noch eine Reihe von Entwicklungstools, die auf spezielle Belange ausgerichtet sind, wie beispielsweise *SWARM* [278] auf die Entwicklung von agentenbasierten Simulationen, *Bee-gent* [43] auf die agentengestützte Integration existierender Anwendungen und *CRNS* [296] auf die agentenorientierte Spezifikation von autonomen Softwareverhalten.

Die meisten der oben genannten Entwicklungstools implizieren die Verwendung einer bestimmten Agentenplattform, die den Anwendungen zugrundegelegt wird. Diese Plattformen legen beispielsweise Details zur Identifikation von Agenten, zur Kommunikation zwischen Agenten und zu Services, die den Agenten zur Verfügung stehen, fest. Darüber hinaus gibt es eine Reihe von methoden- und toolunabhängigen Plattformen, die bei der Realisierung von agentenorientierten Anwendungen hilfreich sein können. Zu den prominentesten Beispielen hierfür gehören *RETSINA* [279] und *IMPACT* [149, 275], weitere Beispiele sind *OPAL* [247], *Comtec* [76] und *AAP* [1]. Verfügbar sind auch spezielle Plattformen für mobile agentenbasierte Anwendungen; hier sind insbesondere *LEAP* [209], *MicroFIPA-OS* [270] und *GRASSHOPPER* [148] zu nennen. Ein Entwickler, der auf bestehende Plattformen zurückgreifen will, hat also die Qual der Wahl, zumal zur Zeit keine vergleichende Evaluierung dieser Plattformen vorliegt.

Standards und Standardisierungsbemühungen

Die wichtigste Standardisierungsinstanz im Agentenbereich ist die Foundation for Intelligent Physical Agents, kurz FIPA. Im Anhang A werden die FIPA-Standardisierungsbemühungen zu Agentenplattformen vorgestellt (eine Liste von FIPA-konformen Plattformen ist auf der Webseite [115] verfügbar). Weitere FIPA-Standardisierungsbemühungen zielen zum einen auf eine als FIPA *AgentUML* (kurz *AUML*) [15, 42] bezeichnete agentenbasierte Erweiterung von Standard-UML (diese Erweiterungen betreffen beispielsweise die Modellierung von Agentenklassen, Interaktionsprotokollen und Sozialstrukturen) und zum anderen auf ein SPEM-basiertes Meta-Modell für den agentenorientierten Softwareentwicklungsprozess [116] ab. FIPA-Konformität gilt zunehmend als Voraussetzung für industrielle und kommerzielle agentenorientierte Software, weshalb dieser Standard auch von nahezu allen neueren Tools und Plattformen unterstützt wird.

Die Beschreibung und das Kommunizieren von Wissen spielt bei der Agentenorientierung eine wichtige Rolle. Als Quasi-Standard für die Beschreibung von Wissen und Metawissen gilt die logikbasierte Sprache KIF [167], für die auch ein ANSI-Standardisierungsvorschlag vorliegt [31]. Von KIF liegen verschiedene Varianten vor: neben Ontolingua [232], einer KIF-Erweiterung zur Beschreibung von Ontologien, sind dies insbesondere FIPA-KIF [114] und SUO-KIF [277], die beide eine geringere Mächtigkeit als KIF besitzen und als Kandidaten für zukünftige Standardisierungen gelten. Zwei Agenten-Kommunikationssprachen, die sich als Quasi-Standard etabliert haben, sind die „Knowledge Query and Manipulation Language“ KQML [109, 182] und die „FIPA Agent Communication Language“ FIPA-ACL [112]. Beide Sprachen sind angelehnt an die Sprechakttheorie, wobei KQML einen etwas größeren Sprachumfang (gemessen an der Zahl der verwendbaren Performative) besitzt. Problematisch mit beiden Sprachen ist, dass sie keine völlig eindeutige Semantik besitzen und damit unterschiedliche Dialekte zulassen [267].

Seitens der Object Management Group (OMG) gibt es zwei weitere wichtige Standardisierungsbemühungen: zum einen die Integration agentenspezifischer Konzepte in UML (Version 2.0) durch die „Agent Platform Special Interest Group“ (Agent PSIG) [227] und zum anderen die unter der Bezeichnung MASIF bekannte Standardisierung der Interoperabilität zwischen mobilen Agenten [228]. (Mobilität als Agenteneigenschaft ist auch Gegenstand der FIPA-Standardisierungsbemühungen.)

Weitere (Quasi-)Standards, Standardtechnologien und Standardisierungsbestrebungen, die generell für verteilte und offene Anwendungen und damit für den agentenorientierten Ansatz von Bedeutung sind (vgl. Seite 10), sind beispielsweise XML [313],

CORBA [77], Jini [159], UDDI [284], JXTA [161], SOAP [268], WSDL [312], WSCL [311], ebXML [99] und RosettaNet [255].

Sprachen

Für agentenorientierte Anwendungen sind folgende vier Sprachklassen von besonderer Bedeutung:

- Kommunikationssprachen;
- Koordinationssprachen;
- Programmiersprachen; und
- Ontologiesprachen.

Im Folgenden wird auf wichtige Vertreter dieser Klassen verwiesen.

Die gebräuchlichsten Kommunikationssprachen für Agentensysteme sind KQML und FIPA-ACL (siehe Seite 17). Eine gute Einführung in diese Sprachklasse gibt [178].

Koordinationssprachen dienen der Festlegung der Koordinationsabläufe und des Koordinationsmanagements in einem agentenorientierten Softwaresystem. Zu den bekanntesten agentenspezifischen Koordinationssprachen zählen *COOL* [36], *STL++* [261] und *SDML* [208]. *COOL* zeichnet sich durch die explizite Repräsentation von Koordinationswissen und ein integriertes regelbasiertes Konversationsmanagement aus; der Fokus von *STL++* und *SDML* liegt auf der expliziten Beschreibung von organisationalen Strukturen.

Als Standardprogrammiersprache für agentenorientierte Softwaresysteme hat sich Java etabliert. Nicht nur wünschenswert sondern für die Programmierung im Großen unverzichtbar ist es allerdings, dass agentenspezifische Konzepte (wie z.B. „Agent“ oder „Team“) auf Programmiersprachenebene als vorgegebene, atomare Sprachkonstrukte zur Verfügung stehen, analog etwa dem Objekt- oder Klassenkonstrukt in objektorientierten Sprachen. Hier herrscht großer Forschungsbedarf, auch wenn erste Vorschläge für eine solche „agentenorientierte Programmiersprache“ bereits vorliegen. Zu nennen ist hier beispielsweise die auf Java aufsetzende *JACK Agent Language* (Seite 261), *PLACA* [283], *AgentSpeak(L)* [248] und *ConcurrentMetateM* [118]. Die erste kommerziell verfügbare Programmiersprache zur plattformunabhängigen Implementierung von mobilen Softwareagenten ist *Telescript* [298]. Verwiesen sei noch auf *ConGolog* [83] und *FLUX* [282], zwei agentenorientierte Sprachen aus dem Umfeld der Roboterprogrammierung.

Ontologiesprachen, also Sprachen zur Spezifikation von Ontologien, sind von wachsender Bedeutung für die agentenorientierte Softwareentwicklung, stellen in diesem Bereich aber keinen For-

schungsschwerpunkt dar. Geeignet für agentenorientierte Anwendungen sind traditionelle Ontologiesprachen wie *Ontolingua* und *LOOM* und mit zunehmender Web-Anbindung solcher Anwendungen vor allem die neueren Web-basierten Ontologiesprachen *DAML+OIL* und *OWL*. Einen sehr guten Überblick über diese und andere Ontologiesprachen gibt [131].

Formale Spezifikation und Verifikation

Mit wachsender Praxisakzeptanz von agentenorientierter Software gewinnen formale Methoden der Spezifikation (d.h. der Festlegung der Funktionalität bei gegebenen Kundenanforderungen) und der Verifikation (d.h. des Nachweises der Korrektheit der Software bei gegebener Spezifikation) enorm an Bedeutung, und umgekehrt ist die Verfügbarkeit solcher formalen Methoden von großer Bedeutung für die Praxisakzeptanz [94]. Auch wenn erste formale Ansätze vorliegen (einen Überblick gibt [306]), besteht hier ein deutlicher Forschungsbedarf. Die drei bekanntesten formalen Spezifikationsansätze sind der belief-desire-intention (BDI) [250] Ansatz, *LORA* [304] und *DESIRE* [55]. Pionierarbeit zur automatischen Kompilierung von Agentenspezifikationen ist in [254] beschrieben. Gute Beispiele für formale Verifikationsansätze sind [119] und [46]; der erstgenannte Ansatz ist axiomatisch, der zweite basiert auf Model Checking.

Neben den obigen Ansätzen, die logikbasiert sind, gibt es eine Reihe von Ansätzen, die auf anderen Standardformalismen aufsetzen. Beispielsweise basiert der in [95] beschriebene Spezifikationsansatz auf *Z* und der in [206] beschriebene Ansatz auf Petrinetzen. Umfangreiche Verwendung findet vor allem auch Standard-UML (einen guten Überblick gibt [41]) und agentenspezifische UML-Erweiterungen wie (siehe Seite 17).

Die explizite Spezifikation von Autonomie als Systemeigenschaft wird in [297] thematisiert; vorgeschlagen wird hier ein Formalismus namens *RNS*, der die explizite Angabe von Rechten und Pflichten eines Agenten vorsieht. Weitere formale Ansätze zur expliziten Autonomiespezifikation sind [37, 91, 192, 235].

Architekturen und Designpatterns

Aus agentenorientierter Sicht ist es üblich, zwei Architekturebenen zu unterscheiden: die Mirco-Ebene der individuellen Agenten, auf der das „Innenleben“ eines Agenten erfasst wird, und die Macro-Ebene von „Agentengemeinschaften“, auf der die Interaktionen und die Wechselwirkungen zwischen Agenten erfasst werden. Die Vielfalt der in der Literatur beschriebenen Architekturen ist nur schwer überschaubar. Einen ersten, einführenden Überblick über vorgeschlagene Micro-Architekturen kann man sich mit [212, 303] verschaffen; generelle Überlegungen zu diesen Architekturen finden sich in [186, 294]. Zu den bekannteren Micro-Architekturen zählen beispielsweise *INTERRAP* [213],

dMars [96], *ARCHON* [68], *OAA*^{TM} [233], *COUGAAR* [79] und *SOAR* [269]. Deutlicher Forschungsbedarf besteht zur Frage, nach welchen generellen Kriterien die Auswahl einer geeigneten Micro-Architektur erfolgen soll; Richtlinien für diese Auswahl gibt [214]. Beispiele für oft zitierte Macro-Architekturen sind *GPGP/TAEMS* [187], *RETSINA* [279], *IMPACT* [149, 275] und – für mobile agentenorientierte Systeme – *TuCSoN* [230]. Eine objektorientierte Makro-Architektur für Agentensysteme wird in [140, 141] vorgestellt. Da der Fokus der Makro-Architekturen meist auf Agent-Agent-Koordination liegt, werden sie auch als *Koordinations-Frameworks* bezeichnet. Eine abstrakte Klassifikation von Makro-Architekturen wird in [92] vorgestellt; unterschieden wird dabei in Anlehung an die Organisationstheorie zwischen „Markt“, „Hierarchie“ und „Netzwerk“. Beschreibungen von (Software-)Infrastrukturen für agentenorientierte Systeme und ihren Mikro/Makro-Architekturen finden sich in [290, 291].

Die Notwendigkeit zu Wiederverwendung hat im Bereich der agentenorientierten Software zur Identifkation verschiedenster Patterns für Agentenverhalten, -koordination und -organisation geführt. Beispiele hierfür sind [19, 78, 89, 93, 97, 98, 163, 176, 177, 190, 281, 265] und, speziell für mobile Agenten, [34, 88]. In Hinblick auf die Formalisierung der Wiederverwendbarkeit von Agenten sei noch auf [47] verwiesen.

Anwendungen

Agentenorientierte Software ist gerade dabei, die Forschungsstätten in Richtung kommerzieller und industrieller Anwendungspraxis zu verlassen. Dies zeigt sich beispielsweise auch daran, dass Firmen wie Siemens, IBM, Sun, Apple, Microsoft und SAP das Thema „Softwareagenten und Agententechnologie“ in eigenen Produkten und/oder Projekten aufgegriffen haben und dass es eine wachsende Zahlen von Firmen gibt, die auf agentenorientierte Software und ihre Entwicklung spezialisiert sind. Zu diesen Firmen zählen u.a. WHITESTEIN Technologies [299], agentscape [23], IKV++ Technologies AG [148], Agent Oriented Software Pty. Ltd. [3], Acklin [2], AGENTBUILDER [16], Agentis [20], emorphia [100], Lost Wax [193] und BNN Technologies [50].

Ein großer Anteil der Forschung zu agentenorientierter Software ist anwendungsnah ausgelegt und es gibt eine Vielzahl von prototypischen agentenbasierten Anwendungssystemen. Zu den Anwendungsdomänen und -typen zählen beispielsweise (die angegebenen Referenzen verweisen auf Überblicksartikel und -bücher, sofern verfügbar, andernfalls auf ein typisches Beispielsystem):

- E-Commerce [136];
- Geschäftsprozessmanagement [155];