AOSE

Agentenorientiertes Software Engineering

Autor:
Dr. Gerhard Weiß
SCCH GmbH
http://www.scch.at

Agentenorientiertes Software Engineering (AOSE) bezeichnet ein Paradigma in der Softwareentwicklung, in dessen Mittelpunkt das Konzept eines Agenten – einer wissensverarbeitenden, flexiblen, interagierenden und autonomen Einheit – steht. Gegenstand von AOSE sind Vorgehensweisen, Methoden, Techniken und Tools für die Erstellung und Handhabung von agentenorientierter Software, also von Software, deren Struktur und Funktionalität unter dem Blickwinkel einer Menge von Agenten betrachtet wird. Im Folgenden werden ausgewählte zentrale Aspekte des agentenorientierten Software Engineering erläutert.

1. Das Agentenkonzept

Das Agentenkonzept, welches dem agentenorientierten Software Engineering zugrunde liegt, stammt ursprünglich aus der (Verteilten) Künstlichen Intelligenz und hier speziell aus dem Bereich der Agenten- und Multiagenten-Technologie. Damit reichen die Wurzeln dieses Konzepts zurück bis in die fünfziger Jahre des vergangenen Jahrhunderts. In der vergangenen Dekade hat sich das Agentenkonzept erfolgreich in verschiedenen Bereichen und Gebieten der Informatik, insbesondere auch auf dem Gebiet der Software und des Software Engineering, etabliert. Der Agentenbegriff war lange Zeit Gegenstand intensiver Diskussionen und Präzisierungsbemühungen und ist dies zum Teil auch heute noch. Im Folgenden werden zwei Charakterisierungen des Agentenkonzepts vorgestellt, die sich in der jüngeren Literatur etabliert haben.

„Schwache Charakterisierung". In den vergangenen Jahren zeichnete sich ein breite Akzeptanz der folgenden Charakterisierung des Agentenkonzepts – in der Literatur häufig als schwache Charakterisierung oder schwache Bedeutung (weak notion) bezeichnet – ab:

Ein Agent ist eine abgrenzbare Software/Hardware-Einheit, die in der Lage ist, wissensgestützt die ihr vorgegebenen Aufgaben flexibel, interaktiv und autonom zu bearbeiten.

Dabei liegen den Schlüsselmerkmalen eines Agenten, die in dieser Charakterisierung genannt werden, folgende Vorstellungen 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. 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 zu handhaben 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 in AOSE 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 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.

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 – in der Tat liegen den allermeisten Arbeiten in Forschung und Anwendung beide Charakterisierungsansätze zugrunde, d.h. die beiden Ansätze werden in aller Regel in Kombination 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 agentenorientierten Software Engineering ebenso wie auf dem Gebiet der Agententechnologie wird das Wort „Agent“ häufig in attributierter Form verwendet.

Beispiele für solche Attributierungen, mit denen üblicherweise das wichtigste Merkmal des jeweiligen Agenten hervorgehoben werden soll, sind „autonomer Agent“, „kooperierender Agent“, „reaktiver Agent“, „adaptiver Agent“ und „rationaler Agent“. Üblich ist auch die Bezeichnung „intelligenter Agent“, mit der betont wird, dass einem Agent aufgrund seiner Flexibilität, Interaktivität und Autonomie eine gewisse Intelligenz zugesprochen werden kann.

Neben der Betonung einzelner Merkmale ist es auch üblich, den Begriff „(Software-)Agent“ dadurch zu präzisieren, dass er mit seinem Einsatzbereich oder -zweck attributiert wird. Bekannte Beispiele hierfür sind: Informationsagent, Interface-Agent, Wrapper-Agent, Transaktionsagent, Verkaufsagent, Assistenzagent, virtueller Agent und mobiler Agent.

2. Merkmale und Potential des agentenorientierten Software Engineering

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, Objektorientierung, Komponentenorientierung, Aspektorientierung, Modellorientierung, Architekturorientierung, Patternorientierung, Aufgabenorientierung und – meist im Kontext von betrieblichen Informationssystemen – Prozessorientierung. In diese Liste ist Agentenorientierung als eine neue, im Entstehen begriffene Systembetrachtungsweise einzuordnen.

Welchen Nutzen eine bestimmte Betrachtungsweise beziehungsweise ein bestimmtes Paradigma 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 solide, 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 AOSE 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, 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. 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. 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, 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 ausblendend 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. Erstens, sie ermöglicht eine gerichtete Zerlegung eines Softwaresystems in atomare Teile (Agenten) und aus ihnen zusammengesetzte Konstrukte (Agentengruppen). „errichtet“ 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) und „Sozialität“ (social level) 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;
- Parallelität/Nebenläufigkeit, d.h. die Verarbeitung von Daten erfolgt parallel/nebenläufig;
- 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. Generell sind diese drei Merkmale kennzeichnend für eine Vielzahl von Anwendungen, die auf neuen Modellen und Ansätzen zur computergestützten Informationsverarbeitung wie beispielsweise Grid Computing, Peer-to-Peer Computing, Web Computing, Pervasive und Ubiquitous Computing, Autonomic Computing und Mobile Computing 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. 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. Elementare Softwareeinheiten, die sich im Laufe dieser Evolution herausgebildet haben – monolithische Programme, Module, Prozeduren, Objekte, Komponenten und Services –, 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 der Aktivierungskontrolle} einer Softwareeinheit vorsieht („Selbst- statt Fremdaktivierung“, „Selbst- statt Fremdbestimmung“ und „Selbst- statt 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, 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 zu verstehen und im Kontext von Web Services wird Autonomie üblicherweise als wichtige Eigenschaft betrachtet (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, aber auch Sun's N1 Initiative, HP's Adaptive Enterprise Initiative und Microsoft's Dynamic Systems Initiative (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;
- haben Agenten und Komponenten das Merkmal der Abgeschlossenheit und den Fokus auf ihre Schnittstelle gemeinsam und das Agentenkonzept kann durchaus als Spezialisierung 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 Akzeptanzfaktoren. In den vergangenen Jahren entwickelten sich

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

zu Schwerpunktthemen im Bereich des agentenorientierten Software Engineering. 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.

Herausforderungen. Softwareentwicklung ist viel zu facettenreich und komplex als dass es ein „Allheilmittel“ – eine silver bullet – 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 und es wäre unrealistisch, anzunehmen, Agentenorientierung oder irgendein anderer Entwicklungsansatz sei ein solches. Der agentenorientierte Ansatz bietet eine Reihe von neuen, innovativen Konzepten und potentiellen Vorzügen. Im Folgenden wird auf drei grundsätzliche Herausforderungen hingewiesen, die es zur Realisierung dieser Vorzüge zu bewältigen gilt.

Eine leicht zu unterschätzende Herausforderung ist der sachliche und fundierte Umgang mit dem Agentenkonzept. Aufgrund seiner intuitiven Verständlichkeit kann dieses Konzept sehr 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. In der Tat finden sich viele Beispiele dafür, dass der Agentenbegriff zu leichtfertig verwendet wird und 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 wohl 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.

3. Aktuelle Schwerpunkte in Forschung und Praxis

Forschung. Aufgrund seines großen Potentials erfährt das agentenorientierte Software Engineering wachsende Aufmerksamkeit in

Forschung und Praxis. Ausdruck dessen ist auch die große Anzahl von Arbeiten, in denen unterschiedlichste Aspekte dieses Ansatzes thematisiert werden. Die zentralen Schwerpunkte in der aktuellen Forschungslandschaft der agentenorientierten Software sind

- Entwicklungsmethoden für agentenorientierte Software (z.B. Gaia, Prometheus und Tropos),
- Entwicklungstools und –plattformen (z.B. FIPA-OS, und JADE),
- Standardisierungen und Standardisierungsbemühungen (insbesondere durch IEEE FIPA)
- Sprachen für Programmierung und Spezifikation (insbesondere für die Spezifikation der Agent-Agent Koordination und Kommunikation wie z.B. FIPA-ACL, Jack und COOL),
- formale Ansätze zur Spezifikation und Verifikation des Programmverhaltens (z.B. LORA, DESIRE und RNS) und
- Architekturen und Designpatterns (z.B. INTERRAP und IMPACT).

Jeder dieser Schwerpunkte hat bereits eine Reihe von Prototypen und prototypischen Lösungsansätzen hervorgebracht. Darüber hinaus werden inzwischen auch kommerzielle Lösungen (insbesondere im Toolbereich) angeboten.

Praxis. Agentenorientierte Software kommt bereits in kommerzieller und industrieller Praxis zum Einsatz. Inzwischen haben Firmen wie beispielsweise Siemens, IBM, Sun, Apple und Microsoft das Thema “Softwareagenten und Agententechnologie” in eigenen Produkten und/oder Projekten aufgegriffen und es gibt bereits auch Firmen, die (u.a.) auf agentenorientierte Software und ihre Entwicklung spezialisiert sind. Zu diesen Firmen zählen u.a. WHITESTEIN Technologies, agentscape, Agent Oriented Software Pty. Ltd., Agentis Software, Lost Wax und SAVANNAH Simulations..

Ein großer Anteil der Forschung zu agentenorientierter Software ist anwendungsnah ausgelegt und es gibt eine Vielzahl von prototypischen agentenbasierten Anwendungssystemen. Zu den Domänen, in denen agentenorientierte Software inzwischen eingesetzt wurde, zählen beispielsweise

- E-Commerce,

- Geschäftsprozessmanagement,
- Fertigungssteuerung und -logistik,
- Gesundheitswesen,
- Telekommunikation,
- Verteilte Sensor-Netzwerke,
- Entertainment,
- Persönliche digitale Assistenten,
- Wissensmanagement,
- Simulation von komplexen Systemen,
- Mensch-Maschine-Interaktion und
- Informationsfilterung und -auswahl.

4. Literaturhinweise

Einen breiten methodologischen Überblick geben folgende beiden Bücher:

- F. Bergenti, M.-P. Gleizes & F. Zambonelli (Hrsg.): Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook. Kluwer Academic Publishers. 2004.
- B. Henderson-Sellers & P. Giorgini (Hrsg.): Agent-Oriented Methodologies: Idea Group Publishing. 2005.
- M. Luck, R. Ashri & M. D'Inverno (Hrsg.): Agent-Based Software Development. Artech House. 2004.
- G. Weiß & Ralf Jakob: Agentenorientierte Softwareentwicklung. Xpert.press, Springer-Verlag. 2005.

Grundlegende Betrachtungen zum agentenorientierten Softwareparadigma finden sich beispielsweise in folgenden Beiträgen:

- N.R. Jennings: On agent-based software engineering. Artificial Intelligence,117, pp. 277-296. 2000.
- N.R. Jennings & M. Wooldridge: Agent-Oriented Software Engineering. In J. Bradshaw (Hrsg.), Handbook of Agent

Technology. AAAI/MIT Press, 2002.

- G. Weiß. Agent Orientation in Software Engineering. Knowledge Engineering Review, 16(4), pp. 349–373, 2002.

Folgende Publikationen behandeln Anwendungen von agentenorientierter Software beziehungsweise Softwareagenten:

- S. Kirn (Hrsg.): Schwerpunktheft zur Agententechnologie. it – Information Technology. 2005.
- F. Klügel: Applications of Software Agents. Künstliche Intelligenz, Band 2/04 (Schwerpunktheft zu Anwendungen von Softwareagenten, herausgegeben von A. Heinzl und F. Rothlauf, pp. 5–10, 2004.
- V. Parunak: Agents in overalls: Experiences and Issues in the Development and Deployment of Industrial Agent-Based Systems. International Journal of Cooperative Information Systems, 9(3), pp. 209–227, 2000.
- V. Parunak: A Practitioners' Review of Industrial Agent Applications. Autonomous Agents and Multi-Agent Systems, 3(4), pp. 389–407, 2000.

Abschließende Hinweise:

- IEEE FIPA Standardisierungskomitee, siehe http://www.fipa.org/about/fipa_and_ieee.html
- International Journal of Agent-Oriented Software Engineering, http://www.inderscience.com/browse/index.php?journalID=174
- International Workshop Series on Agent-Oriented Software Engineering, z.B. http://www.cs.rmit.edu.au/agents/aose06/