Entwurf einer generischen Agentenarchitektur

Jan Dalski

HAW-Hamburg / Mastersemester 1
24. Februar 2014

1 Einleitung

Seit dem Beginn der Forschungen an der *Künstlichen Intelligenz* (KI) vor einem halben Jahrhundert ist deren Siegeszug ungebrochen. Ob Routineaufgaben, Handel oder Informationsverarbeitung – *Software-Agenten*, eigenständig agierende Programme, sind inzwischen aus vielen Bereichen nicht mehr wegzudenken.

Auch in der Simulation besitzt die KI einen hohen Stellenwert. Im Vergleich zu konventionellen, analytischen Modellen (Makromodelle, z.B. durch Differentialgleichungen oder *System Dynamics*) verfolgt die *Multiagentensimulation* (MAS) einen anderen Ansatz. Sie basiert auf einer *bottom-up*-Vorgehensweise, bei der die erwünschten Einsichten durch die Verwendung einer Vielzahl von Agenten entstehen.

Während in analytischen Modellen die Individuen nicht explizit berücksichtigt werden, haben sie in Multiagentensimulationen eine Schlüsselfunktion inne: Durch autonomes Verhalten und Interaktion entstehen emergente Effekte; das Gesamtverhalten des Systems ergibt sich aus den Wechselwirkungen im Inneren. Anstelle einer hochkomplizierten und schwankungsanfälligen „Systemformel“ werden die Einzelbestandteile des Systems modelliert. Insbesondere in komplexen Anwendungsgebieten, wie der Prognosenerstellung oder der Simulation von Umwelt/Natur und Wirtschaftskreisläufen bietet die MAS neue Perspektiven.

1.1 Motivation und Zielsetzung

Obwohl es viele Toolkits zum Entwurf von Agenten und Multiagentensystemen gibt[1], so weisen sie diverse Limitationen auf. Da die eigenen Agenten als Erweiterungen für diese Toolkits entwickelt werden, findet eine Bindung an die jeweilige Plattform statt. Eingebaute Strukturen und Standards (wie z.B. der FIPA-konforme Nachrichtenaustausch) können sich als Einschränkung oder Hindernis erweisen, ebenfalls lassen sich Performanz- und Skalierungsprobleme so nicht umgehen.

Das übergeordnete Ziel soll es sein, eine generische Architektur zu entwerfen, die allen diesen Ansprüchen genügt. Die Aufgabe dieses Dokumentes liegt darin, die zu bewältigenden Probleme aufzuzeigen und typische Agentenarchitekturen sowie spezielle Ansätze vorzustellen. Außerdem soll es als grobe Planungsgrundlage für die anstehenden Arbeitsschritte dienen.

[1] Als Beispiele seien hier JADE, MaDKit & JATLite genannt (alle Java-basierend).

2 Anforderungen

2.1 Einsatzgebiet

Die Architektur soll Anwendung in dem *MARS*-Framework[2] finden. Da dieses Multiagentensimulationen beliebiger Art ermöglichen soll, wird eine generische und offene Architektur benötigt. Entscheidend ist, daß diese Architektur dem Anwender möglichst wenig Restriktionen auferlegt und als konfigurierbare Grundlage dient.

Dem Nutzer soll hierzu eine Konfigurationsmöglichkeit per DSL (*Domain Specific Language*) geboten werden. Außerdem ist eine modulare Organisation Grundvoraussetzung, damit domänenspezifische Funktionalität (z.B. in Form von Plugins) ergänzt werden kann.

Als primärer Anwendungsfall im Rahmen des Entwurfes dient die Modellierung des Abdoulaye-Waldes. Hierbei handelt es sich um ein vom Raubbau betroffenes Naturschutzgebiet in Zentral-Togo, welches agentenbasiert simuliert werden soll. Weitere Einsatzgebiete liegen in der Räuber-Beute-Simulation; außerdem soll die Fußgängersimulation *WALK* [4] auf diese neue Architektur portiert werden.

2.2 Probleme bei generischen Ansätzen

Eines der Hauptprobleme, welches allen generischen Ansätzen gemein ist, liegt in der Auswahl eines angemessenen Abstraktionsniveau. Je allgemeiner die Architektur formuliert ist, desto weniger konkret (und damit nutzbar) ist sie im Endeffekt. Es gilt also, einen Kompromiß zwischen einer abstrakten und einer verwendbaren Lösung zu finden. Damit verbunden ist die Aufgabe zu entscheiden, welche Aspekte in die gemeinsame Struktur miteinfließen und was eher als optionales Modul realisiert werden sollte bzw. dem Anwender überlassen bleibt.

Während grundlegende Wahrnehmungs- und Handlungsoptionen bei allen Agenten anzunehmen sind, variiert der Entscheidungsprozeß je nach Agententyp[3] deutlich: Soll nur reaktiv gehandelt werden oder werden Ziele verfolgt? Entsprechend müssen Entscheidungsbäume erzeugt oder Ziele und Pläne definiert werden. Ebenfalls damit verbunden ist die unterschiedliche Nachfrage nach einem „Gedächtnis“ – einer Wissensbasis, die das Wissen, die Vermutungen und möglicherweise auch Gefühle oder Moralvorstellungen eines Agenten beinhaltet. Auch sollte ein eventuelles Lernvermögen berücksichtigt werden und damit in dieser Architektur umsetzbar sein.

Eine weitere Fragestellung resultiert aus der Erweiterung durch den Benutzer. Soll ein SDK zur Entwicklung von Modulen bereitgestellt werden, oder wird es von dem Anwender verlangt, selbst Programmcode zu schreiben – was zwar weniger benutzerfreundlich, dafür aber auch nicht einschränkend wäre? Im optimalen Sinne sollten beide Optionen möglich sein.

[2] *Multi Agent Research & Simulation*, siehe http://www.mars-group.org.
[3] Man betrachte bspw. einen Baum, einen Fußgänger und ein Beutetier, etwa eine Gazelle.

3 Agenten

3.1 Definition

Auch wenn es keine einheitliche Definition für den Begriff des Agenten gibt, so herrscht Konsens über dessen wichtigste Merkmale. Das folgende Zitat aus [2] drückt diese Eigenschaften aus:

> „An autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future."
>
> *Stan Franklin & Art Graesser, 1997*

1. **Autonomie:** Der Agent trifft selbständig Entscheidungen, er wird nicht gesteuert (im Gegensatz zu einem Aktor).
2. **Unabhängigkeit:** Ein Agent verfolgt seine eigenen Ziele[4].
3. **Evolution:** Zur Maximierung des eigenen Nutzens paßt ein Agent sein Verhalten an, er verfügt über Lernfähigkeit.
4. **Umweltbezug:** Ein Agent ist Teil seiner Umwelt und kann nicht losgelöst von ihr betrachtet werden, da die Umwelt seinen Wahrnehmungs- und Handlungshorizont festgelegt.

3.2 Grundlegende Funktionsweise

Ein Agent besteht konzeptionell aus drei Einheiten, welche sequentiell in periodischen Zeitintervallen ausgeführt werden (Taktung). Über Sensoren erfolgt im ersten Schritt eine Wahrnehmung der Umwelt. Diese Eingabe wird anschließend im Agentenprogramm ausgewertet und es wird eine (Re-)Aktion bestimmt. Die Ausführung dieser Handlung, also die Wirkung auf die Umwelt als Konsequenz auf deren Beobachtung erfolgt im dritten Schritt mittels Aktuatoren.

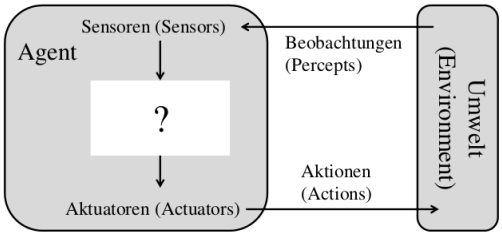

Abbildung 1: Funktionsprinzip eines Agenten, entnommen aus [5]

[4] Rein reaktive Agenten tun dies nicht, sie kennen keine Ziele.

4 Klassische Architekturen

4.1 Reaktive Agenten

Reaktive bzw. *reflexive* Agenten fällen keine Entscheidungen, sondern agieren über Entscheidungsbäume. Sie sind im Allgemeinen zustandslos, verfügen also über kein eigenes Wissen. Eine Ausnahme bildet die Erweiterung der *beobachtenden* Agenten, welche eine Repräsentation der Umwelt erzeugen und dadurch Änderungen und Auswirkungen analysieren können.

Aufgrund der Klassifikationsstrategie spricht man auch von einer *Subsumptionsarchitektur*, der Begriff geht auf R. A. Brooks und seine 1991 veröffentlichten Thesen [1] zurück:

- Intelligentes Verhalten kann ohne explizite Repräsentation erzeugt werden.
- Intelligentes Verhalten kann ohne explizite abstrakte Schlußfolgerung erzeugt werden.
- Intelligenz ist eine emergente Eigenschaft bestimmter komplexer Systeme.

Die Vorteile dieses Ansatzes liegen in der Schlichtheit und Eleganz baumbasierter Modelle. Jedoch eignen sie sich nur für begrenzt komplexe Situationen und gestatten nur Reaktionen – ein Handeln aus sich heraus ist nicht möglich.

4.2 Proaktive (kognitive) Agenten

Als *proaktiv* werden Agenten bezeichnet, die selbst die Initiative ergreifen können. Sie besitzen eine Wissensbasis, verfolgen Ziele und enthalten eine Planungskomponente, welche Pläne zur Zielerreichung bildet. Da diese Agenten über ein Bewußtsein verfügen und „Denken“ können, werden sie auch als *kognitive* Agenten bezeichnet.

Ein typischer Vertreter der proaktiven Agenten ist der *zielbasierte* Agent. Er verfolgt ein oder mehrere Ziele und weiß, wie gewinnbringend seine Aktionen sind. Dies ermöglicht ihm, durch Kombination der (atomaren) Aktionen seines Fähigkeitenrepertoires einen Plan zur Zielerreichung zusammenzustellen.

Als Weiterentwicklung existiert der *nutzenbasierte* Agent. Dieser besitzt die Fähigkeit, den Nutzen eines Ziels und die Wahrscheinlichkeit der Zielerreichung zu ermitteln. Damit ist der Agent in der Lage, (gemäß seiner Handlungsmöglichkeiten) immer optimal zu agieren.

Oft spricht man in diesem Zusammenhang auch von *BDI*-Agenten – diese Agenten besitzen Wissen und Annahmen über die Umwelt *(Beliefs)*, verfolgen Ziele *(Desires)* und verfügen über Absichten *(Intentions)*, um diese Ziele zu erreichen.

4.3 Hybride Agenten / Schichtenmodelle

Die beiden oben vorgestellten Architekturen können zu einem *hybriden* Agenten kombiniert werden. Ein solches Schichtenmodell ist beispielsweise dann notwendig, wenn sowohl eine Agenda verfolgt, aber auch auf gewisse Ereignisse reaktiv reagiert werden soll.

Zur Konstruktion einer Schichtenarchitektur gibt es zwei Konzepte: Bei einem *horizontalen* Modell existieren ein reaktives und ein proaktives Agentenprogramm parallel. Dies resultiert in einer einfachen Umsetzung, aber auch in einem höheren Aufwand, da entweder beide Programme nebeneinander ausgeführt werden oder im Vorfeld per Mediator die Eingabe überprüft und einer Schicht zur Ausführung zugeteilt werden muß.

Bei einem *vertikalen* Modell hingegen bauen die Schichten aufeinander auf. Eine solche Architektur ist weniger komplex, dafür aber auch fehleranfälliger, da der Fehler einer Schicht Auswirkungen auf alle Folgeschichten besitzt. Es wird ferner zwischen einer *1-pass* Architektur (Informationsfluß hoch, oben Aktuator) und einer *2-pass* Architektur (Informationen hoch, Entscheidungen runter) unterschieden. Eine Übersicht bietet die untenstehende Grafik.

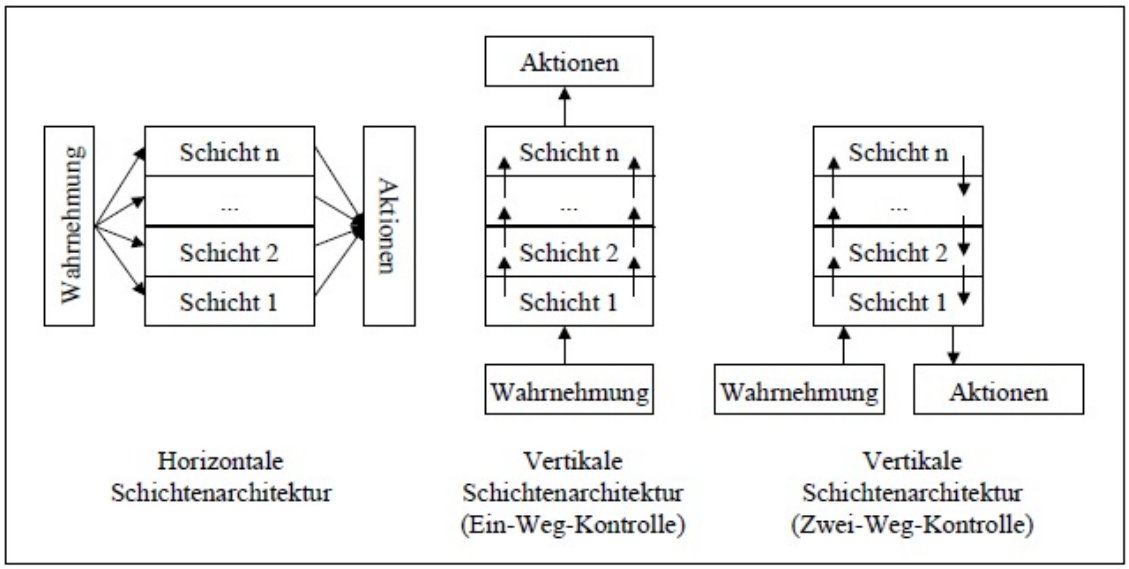

Abbildung 2: Schichtenarchitekturen [6]

Im folgenden Abschnitt sollen nun zwei spezielle Ansätze präsentiert werden. Während die erste Architektur wahlweise reaktiv oder proaktiv verwendbar ist, implementiert die zweite ein dreistufiges Schichtenmodell. Es folgt zunächst eine Vorstellung dieser Architekturen, im Anschluß wird dann deren Nutzen für mein Vorhaben einer generischen Architektur untersucht.

5 Spezielle Ansätze

5.1 GAA

Die *Generic Agent Architecture for Multiagent Systems* wurde von José Vidal und Paul Buhler entwickelt und ist im gleichnahmigen Paper [7] anhand einer Implementation für *Robocup* vorgestellt. Es handelt sich bei dieser Architektur um ein sehr geradliniges und flexibles Konzept, welches den Anspruch besitzt, sowohl reaktiv als auch proaktiv verwendbar zu sein.

Sie basiert auf einer hierarchischen Gliederung der Eingabe und der ausführbaren Aktionen. Bei der Eingabe wird zwischen drei verschiedenen Typen unterschieden: Mit *SensorInput* gibt es eine Oberklasse für Sensoreingaben, alle Arten der Wahrnehmung externer Informationen werden in ein Objekt diesen Types überführt. *Message* stellt einen dedizierten Kommunikationskanal für den Nachrichtenaustausch mit anderen Agenten bereit. Als dritte Gruppe gibt es die *Events*, mit denen ein Agent interne Ereignisse auslösen kann – beispielsweise zur Realisierung von Timeouts. Diese drei Klassen stellen nur ein Grundgerüst dar; es ist Aufgabe des Benutzers, diese Hierarchie domänenspezifisch zu erweitern.

Analog erfolgt eine Klassifikation der ausführbaren Aktionen: Es wird unterschieden zwischen dem (Langzeit-)Verhalten eines Agenten (*Behaviour*), welches aus einer Menge atomarer Aktionen zwecks Zielerreichung besteht und Kommunikationstypen (*Conversation*)[5]. Diesen beiden Typen liegt die Oberklasse *Activity* zugrunde. Sie deklariert drei Methoden, in denen sämtliche Anwendungs- und Steuerlogik untergebracht wird:

- **canHandle:** (Steuerlogik) Diese Funktion überprüft, ob diese Aktivität anhand der Eingabe ausführbar ist. Dafür können sowohl die Inhalte der Eingabe als auch zusätzlich die internen Zustände (Wissensbasis, BDI-Agenten) berücksichtigt werden. Da diese Prüfung aber pro Eingabe auf jeder Aktivität aufgerufen werden muß, sollte sie *schnell* sein.
- **handle:** (Anwendungslogik) Führt die Aktivität aus. Erzeugt dazu eine oder mehrere atomare Aktionen. Mehrstufige Pläne ermöglichen komplexes Verhalten, dabei kann ein Agent interne Variablen setzen, um Zustände zwischen mehreren Ausführungen zu speichern. Nach Abarbeitung wird die Aktivität gelöscht. Für gewöhnlich enthält ein Agent nie endende Aktivitäten (Hauptroutinen).
- **inhibits:** (Steuerlogik) Überprüfung, ob diese Aktivität eine andere (und ggf. deren Teilbaum) verhindert. Diese Funktion ermöglicht eine Einschätzung, wie sinnvoll diese Aktivität im Vergleich zu anderen ist. Die Rückgabe kann von einem Wahrheitswert (Subsumptionsverhalten, reaktiv) bis zu einer Nutzwertanalyse anhand interner Zustände (BDI) reichen.

[5] Hierzu werden Protokolle implementiert, üblicherweise als endliche Automaten.

Zur Planung und Ausführung von Aktivitäten wird der *ActivityManager* genutzt. Diese Klasse verwaltet alle Aktivitäten und wählt pro Ausführungsschritt anhand des Eingabeobjektes eine Aktion zur Ausführung aus[6]. Hierzu werden zunächst alle ausführbaren Aktivitäten ausgewählt („canHandle“) und die Untermenge aller nicht-blockierten Aktivitäten gebildet (Durchlauf mit „inhibits“ pro Aktivität und über die ganze Menge). Aus dieser Menge wird dann eine zufällige Aktivität ausgewählt und mit „handle“ ausgeführt.

5.2 GEAMAS

Die Abkürzung GEAMAS steht für *Generic Architecture for Multi-Agent Simulations* [3] und dient als zweite „Inspirationsquelle“. Sie stammt von Pierre Marcenac und Sylvain Giroux und wurde zur Entwicklung von Vorhersagemodellen für komplexe Natursysteme (wie z.B. Vulkanausbrüche oder Lawinen) entwickelt. Das Verhalten dieser dynamischen und komplexen Systeme soll durch eine Aufsplittung in viele Einzelkomponenten nachgebildet werden. Diese werden agentenorientiert implementiert und bilden einen hierarchischen Verbund. Das Resultat ist ein *Agent aus Sub-Agenten*, wobei allen Agenten die gleiche Struktur zugrunde liegt (Rekursion) und sowohl kognitive als auch reaktive Architekturen als Bestandteile verwendet werden können. In den so entstehenden emergenten Effekten innerhalb eines Agenten liegt der Schlüssel zur Simulation von Systemen mit *selbstorganisierter Kritikalität*.

Zur Umsetzung eines Agenten aus Subagenten schlägt die GEAMAS drei Abstraktionsstufen vor, dargestellt in Abbildung 3. Auf der obersten Stufe (dem *Society Model*) findet die Simulationssteuerung statt. Hier existieren Makro-Agenten, die als Simulationsschnittstelle dienen, für die Einhaltung globaler Spezifikationen (z.B. Simulationsparameter) sorgen und die untergeordneten Agenten gemäß ihrer Rollen bzw. Verhalten gruppieren.

Bei diesen „untergeordneten“ Agenten der zweiten Schicht (dem *Agent Model*) handelt es sich um autonome, kognitive Agenten, die als Bestandteile der Gesellschaft fungieren (*High-Level Agents*). Sie bestehen aus einer Gruppe von Aktivitäten der untersten Stufe und abstrahieren von deren feingranularer Komplexität. Durch die Interpretation des Verhaltens dieser *Zellen* und dementsprechendem Handeln entsteht Verhaltensemergenz (internes Verhalten des Stufe-II-Agenten). Das externe Verhalten bezeichnet die Reaktion auf äußere Ereignisse (Störeinflüsse). Ein solcher Agent besitzt darüber hinaus Evolutionsfähigkeiten: Er ist in der Lage, sich selbst zu restrukturieren, indem er seine Verknüpfungen zu anderen Agenten ändert. Durch diese Selbstorganisation der Agenten als Resultat auf externe Ereignisse kann ein System ohne initiale Beschränkungen simuliert werden.

Die unterste Ebene beherbergt Mikro-Agenten, auch *Third-Level-Agents* oder *Cells* genannt. Sie sind ausschließlich reaktiv und besitzen kein Wissen über die Umwelt oder globale Zustände (können aber durchaus interne Zustände besitzen und sich so weiterentwickeln). Diese Zellen reagieren auf Ereignisse, die zu feingranular

[6] Events werden priorisiert behandelt.

wären, als daß sie von kognitiven Agenten verstanden würden. Dazu kommunizieren sie mit anderen Agenten über semantikfreie Signale (müssen vom Empfänger interpretiert werden), wobei das Signal mit der Entfernung abnimmt. Zellen können also nur mit ihrer unmittelbaren Nachbarschaft kommunizieren. Diese Kommunikationseigenschaften resultieren in *nicht-deterministischem* Verhalten (analog zu Naturphänomenen oder biologischen Vorgängen).

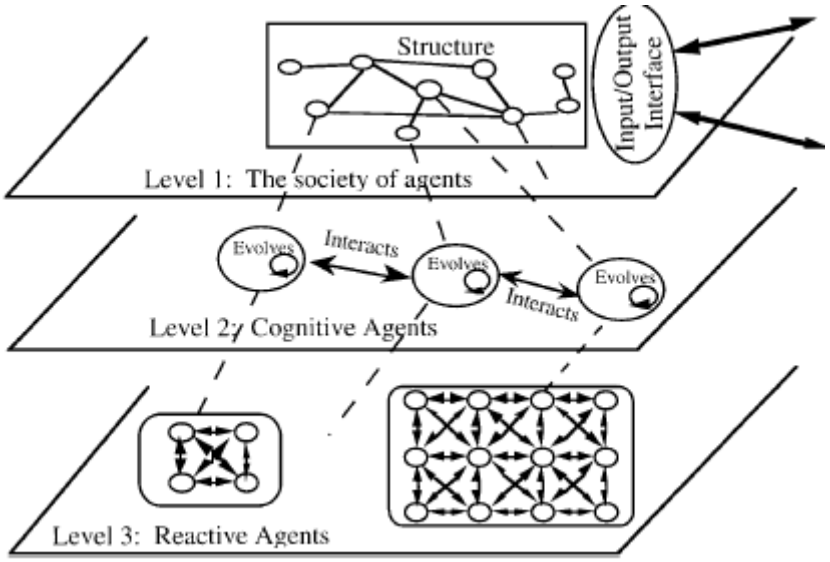

Abbildung 3: Die Abstraktionsstufen der GEAMAS (aus [3])

Der Schlüssel zur Emergenz liegt in der Interaktion von Agenten. Die Möglichkeit zur Kommunikation wird in der GEAMAS durch die Verknüpfungen zwischen den Agenten und deren Reorganisationsfähigkeit ausgedrückt. Es ist dabei ein Informationsaustausch innerhalb einer Schicht (wie bereits bei den Zellen erwähnt, die über Stimuli miteinander agieren) und zwischen den Schichten möglich.

Bei der Inter-Schichten-Kommunikation gibt es zwei Fälle: Die Kommunikation mit dem übergeordneten Agenten und den Informationsfluß „nach unten“. Wenn Information nach oben weitergeleitet wird, spricht man von *Rekomposition*. Dieser *bottom-up* Ansatz reagiert auf die Dynamik in unteren Schichten durch Weiterleitung primitiver Signale (siehe Beschreibung des Zellen-Layers) – beispielsweise ausgelöst durch die Überschreitung von Schwellwerten innerhalb des niederen Agenten.

Das Gegenstück stellt die *Dekomposition* dar. Sie bezeichnet die Informationsweitergabe von einem Agenten an dessen Subagenten. So kann eine Benachrichtigung über externe Events, wie geänderte Umwelteinflüsse oder einzuhaltener Regeln (Naturgesetze, Simulationsparameter) erfolgen.

6 Ausblick

6.1 Bestandsaufnahme

6.1.1 GAA

Die GAA verspricht ein einfaches, flexibles Design für vielfältige Anwendungen. Diesem Anspruch wird sie auch gerecht – jedoch auf Kosten der Konkretheit. Viele Aspekte werden bewußt ausgelassen, insbesondere was den proaktiven Gebrauch betrifft. Ziele, deren Erstellung, (Re-)Priorisierung und Verknüpfung mit Aktivitäten werden genauso wenig erwähnt wie die Repräsentation der Umwelt bzw. der Wissensbasis. Es liegt also im Ermessen des Nutzers, die BDI-Logik selbst hinzuzufügen.

Auch gibt es keinen Aufschluß darüber, wie die Transformation der Sensoreingabe erfolgen soll, oder in welchem Maße eine Aggregation von Eingaben sinnvoll ist. Diese Fragestellung resultiert aus dem potentiell hohen Aufwand der Aktivitätsauswahl pro Eingabeobjekt innerhalb des ActivityManagers – eine mögliche Komplexitätsbremse ist zu befürchten.

Letztendlich soll die GAA aber auch nur eine Grundlage darstellen. Und in Bezug auf die divergenten Ansprüche bezüglich Wahrnehmung und Fähigkeiten, welche eine generische Architektur handhaben muß, liefert die GAA viele Ansätze zu Strukturierung und Aufteilung.

6.1.2 GEAMAS

Der GEAMAS liegt ein interessanter Ansatz zugrunde. Durch die Verwendung von „Agenten in Agenten“ entstehen intra-emergente Effekte, welche einen vielversprechenden Eindruck erwecken. Da allen Agenten dabei die gleiche Struktur zugrunde liegt, erleichtert diese Rekursion die Entwicklung. Außerdem können bei guter Kapselung Unteragenten als Module entworfen und in anderen Domänen wiederverwendet werden.

Fraglich ist nur, inwiefern immer eine sinnvolle Aufteilung eines Agententypes in diese Unteragenten gefunden werden kann. Darüber hinaus ist unklar, wie viel Mehraufwand dieser Zerlegung mit sich führt – sowohl in Bezug auf Implementation als auch auf Rechenaufwand – und wie groß der Mehrwert für die Simulation ausfällt.

6.2 Konferenzen

Es folgen einige Konferenzen und *Special Interest Groups*, die ihren Themenschwerpunkt auf der künstlichen Intelligenz oder Simulation bzw. Multiagentensystemen besitzen.

- **ACM SIGART:** Special Interest Group on Artificial Intelligence
- **ACM SIGSIM:** Special Interest Group on Simulation
- **IFAAMAS:** International Foundation for Autonomous Agents and Multiagent Systems
- **ECAI:** European Conference on Artificial Intelligence

6.3 Wichtige Personen

Die folgende Personenauflistung stellt keinesfalls eine vollständige Liste dar. Vielmehr handelt es sich um Personen, die an den gelesenen Papern mitgewirkt haben oder darin erwähnt wurden bzw. deren Tätigkeitsbereiche für mein weiteres Vorgehen von Interesse sein können.

- **Michael Wooldridge:** Professor of Computer Science, University of Oxford. Hat diverse Paper zu Agenten, Multiagentensystemen und agentenorientierter Programmierung veröffentlicht.
- **Rodney A. Brooks:** Er hat die Subsumptionsarchitektur entworfen und gehört zu den „reaktiven Vertretern“.
- **José M. Vidal:** Professor an der Universität von South Carolina. Weniger in der Forschung, dafür mehr in der Lehre und im Grundlagenwissen tätig. Hat viele Paper zu MAS und agentenbasierter Programmierung veröffentlicht.
- **Stan Franklin:** Arbeitet an der *University of Memphis* und forscht im kognitiven Bereich. Er hat eine Forschungsgruppe gegründet (*Cognitive Computing Research Group*) und ist der Autor von *Artificial Minds*.

6.4 Weiteres Vorgehen

Nachdem in diesem (dem ersten) Semester ein Themenbereich gewählt und untersucht wurde, steht im Folgesemester der Entwurf der Architektur und dessen prototypische Implementation an. Im Laufe des dritten Semesters soll dieser Prototyp im Rahmen von Projekt II fertiggestellt werden.

Damit liegt dann eine Experimentationsplattform vor, die als Grundlage für die Masterarbeit im vierten Semester dienen soll.

Literatur

[1] Rodney A. Brooks. Intelligence without reason. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'91, pages 569–595, San Francisco, CA, USA, 1991. Morgan Kaufmann Publishers Inc.

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

[3] Pierre Marcenac and Sylvain Giroux. Geamas: A generic architecture for agent-oriented simulations of complex processes. *Applied Intelligence*, 8(3):247–267, May 1998.

[4] Stefan Münchow, Ia Enukidze, Stefan Sarstedt, and Thomas Thiel-Clemen. Walk: A modular testbed for crowd evacuation simulation. In *6th International Conference on Pedestrian and Evacuation Dynamics*, PED 2012.

[5] Peter Norvig and Stuart J. Russell. Artificial intelligence: A modern approach (3rd edition). Prentice Hall, 2009.

[6] Institute of Information Systems at Frankfurt University. Softwareagenten. http://www.is-frankfurt.de/publikationenNeu/Softwareagenten.pdf.

[7] José M. Vidal and Paul Buhler. A generic agent architecture for multiagent systems. Technical report, 2001.