Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Seminararbeit

Sven Tennstedt

Agentenbeschreibung: Expressiv AI (Façade) im Vergleich zu traditionellen Ansätzen

Fakultät Technik und Informatik
Department Informatik

Inhaltsverzeichnis

1 Einleitung

1.1 Problemstellung

Agentensysteme stellen eine interessante Softwarearchitektur dar. Sie sind modular und zwischen den einzelnen Komponenten bestehen nur wenig Abhängigkeiten. Daher können Agentensysteme besonders für Umgebungen attraktiv sein, in denen ein System sehr flexibel auf Veränderungen reagieren soll, also eine gewisse Selbstorganisation ermöglicht, und es damit auf die Robustheit eines Gesamtsystems ankommt.

Das IFlat-Projekt der Hochschule für Angewandte Wissenschaften Hamburg z. B. besitzt eine Anzahl von verschiedenen Geräten, die miteinander interagieren sollen. Der Fernseher kann zum einen zum Anschauen von Filmen benutzt werden, aber genauso kann er das Bild der Kamera vor der Haustür anzeigen, sowie Mails darstellen oder als Bildtelefon dienen. Die Stereoanlage kann den Ton zum Film ausgeben, Musik abspielen oder die Stimme eines Gesprächsteilnehmers beim Telefonieren übertragen.

Alle Geräte im IFlat-Projekt sollen miteinander und den Bewohnern interagieren. Neu gekaufte Geräte sollen sich leicht integrieren lassen und mobile Geräte sollen sich an der Wohnung selbstständig an- und abmelden. Das IFlat-Projekt kann daher als Multiagenten-System betrachtet werden, in dem die einzelnen Geräte die Agenten darstellen.

Zum einen ist es für den Nutzer wünschenswert, wenn die Geräte ihre Dienste selbstständig untereinander bekannt geben würden, zum anderen je nach Kontext die sinnvollste Konstellation von Geräten für eine Aufgabe zusammenstellen.

Jeder Benutzer hat seine eigenen Vorstellung davon, was er als sinnvoll erachtet. So wäre es von Vorteil, wenn er das Verhalten seiner Wohnung bzw. seiner Geräte bestimmen könnte.

1.2 Agentenverhalten beschreiben

Das Verhalten eines Agenten zu modellieren kann allerdings schnell zu einer komplexen Aufgabe werden. Ich möchte in dieser Arbeit daher verschiedene Methoden vergleichen wie Agentenverhalten beschrieben werden können. Als Gegenstand der Untersuchung dient mir das interaktive Drama Façade. Ein Ziel bei Façade war es ein möglichst glaubwürdiges Verhalten der synthetischen Protagonisten zu modellieren. Um dies zu Erreichen wurde versucht eine Beschreibungssprache für Agentenverhalten zu verwenden, mit der auch Nicht-Programmierer[1] umgehen können. Die Methode von Façade möchte ich den beiden klassischen Ansätzen gegenüberstellen. Einerseits in Form der reaktiven Agenten, für die sich die Subsumption-Architektur durchgesetzt hat. Andererseits die deliberativen Agenten, für die ich Jadex von der Universität Hamburg[2] heranziehe. Die Verhalten von Agenten zu

[1] siehe z. B. Rosseburg (2008)

[2] Jadex (2008)

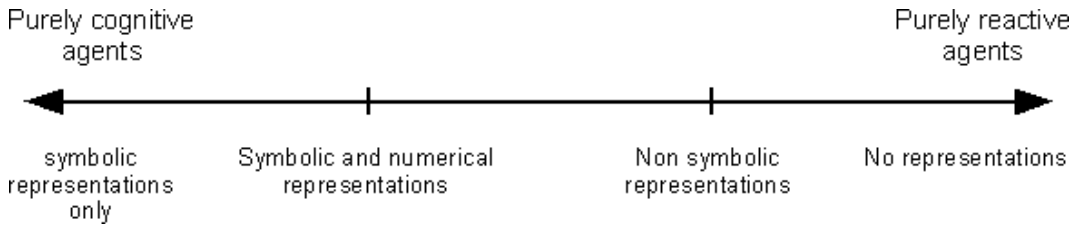

Abbildung 1: Die Unterscheidung zwischen kognitiv und reaktiv führt zu dieser zweckmäßigen Achse, um die kognitive Kapazität eines Agenten zu ermessen, damit er komplexe Aufgaben bewältigen und seine Aktionen planen kann. (Ferber (2001) S. 20)

beschreiben ist in beiden Ansätzen grundverschieden. Beide Ansätze stelle ich hier vor, bevor ich näher auf Façade eingehe. Am Schluss vergleiche ich die verschiedenen Ansätze miteinander und ziehe ein Fazit.

2 Agentensysteme

Ein Agentensystem, genauer ein Multiagentensystem[3], ist ein Architekurkonzept, in dem die einzelnen Softwarekomponenten aus Agenten bestehen, die mit ihrer Umwelt und miteinander interagieren, sowie untereinander kommunizieren.

2.1 Was ist ein Agent?

Der Begriff Agent impliziert, dass eine Person oder Entität im Auftrag einer anderen Person oder Entität ein Ziel verfolgt. Im Softwarebereich bezeichnet dieser Begriff eine Komponente, die nicht unbedingt im Auftrag einer anderen Person oder Entität agiert, sondern allgemein, dass diese Komponente getrieben von inneren Zielen proaktiv handelt um diese zu erfüllen.

Michael Wooldridge[4] klassifiziert intelligente Agenten über drei Eigenschaften:

Reaktivität Intelligente Agenten können ihre Umwelt wahrnehmen und reagieren unmittelbar auf Veränderungen in dieser Umwelt (im Rahmen ihrer Design-Ziele)

Proaktivität Intelligente Agenten sind befähgit zielgerichtetes Verhalten zu zeigen indem sie die Initiative für ihr Handeln übernehmen (im Rahmen ihrer Design-Ziele)

Soziale Fähigkeiten Intelligente Agenten sind fähig mit anderen Agenten (und möglicherweise Menschen) zu interagieren (im Rahmen ihrer Design-Ziele)

[3] siehe dazu z. B. Ferber (2001), Wooldridge (2002) und Tennstedt (2007)

[4] Wooldridge (2002)

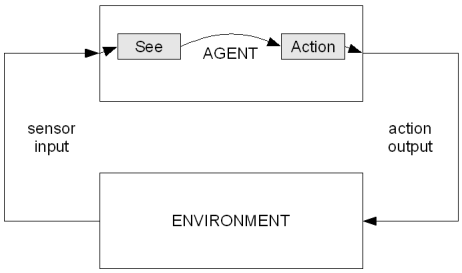

Abbildung 2: Reaktiver Agent. Der Zustand seiner Wahrnehmung („See“) ist direkt mit einer Handlung („Action“) gekoppelt. (Wooldridge (2002) S. 34)

Günther[5] und Ferber[6] differenzieren diese Eigenschaften weiter aus. Doch die drei Eigenschaften beschreiben das Konzept eines intelligenten Agenten für diese Ausarbeitung ausreichend.

Zu unterscheiden sind zwei vom Konzept her grundverschiedene Agententypen[7]: Die reaktiven Agenten (siehe Abbildung 2), die von ihrer Umwelt keine interne Repräsentation erzeugen und ausschließlich reizgesteuert sind und die kognitiven oder auch deliberativen Agenten(siehe Abbildung 5 S. 7), die intern ein Abbild ihrer Umwelt erzeugen und daraus ihre Handlungen planen.

In der Praxis gibt es keine klare Trennung zwischen diesen beiden Typen. Die Abbildung 1 zeigt eine Achse, dessen Enden die beiden Typen darstellen. Zwischen den Extremen ist jede Art der Mischung der Konzepte möglich.

Wie wird nun das Verhalten von Agenten beschrieben?

2.2 Reaktive Agenten

Bei den reaktiven Agenten hat sich die Subsumption-Architektur von Brooks[8] durchgesetzt. Die Reaktionen des Agenten sind hier prioriätengesteuert. Unterschiedliche Verhaltensmuster werden mit bestimmten Sensoren bzw. Sensorwerten verknüpft und jeweils einer Priorität zugeordnet. Ein Verhalten mit einer höheren Priorität verdrängt alle Verhalten mit niedrigeren Prioriäten.

In Abbildung 3 ist dieses Konzept verdeutlicht. Ein Roboter, ausgestattet mir meinem Frontbumper zur Kollisionerkennung und einem Kompassensor, soll solange seinem Kurs folgen, bis er auf ein Hindernis stößt. Geschieht dies, soll er diesem ausweichen und dann wieder auf Kurs gehen. Dieses Verhalten wird hier mit drei Tasks realisiert. Die höchste

[5] Günther (2003)
[6] Ferber (2001)
[7] Tennstedt (2007) S. 19
[8] Wooldridge (2002) S. 90 oder Pfeiffer und Scheier (2001) S. 199ff

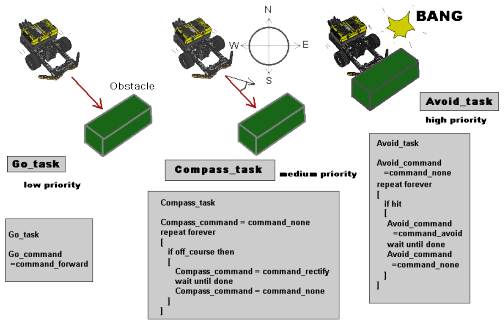

Abbildung 3: Konzeptionelle Sicht auf die Subsumption-Architektur. Dieses Beispiel: 3 verschiedene Tasks die je nach Sensorinput aktiviert werden. (von Boulette's Robotics page (2008))

Priorität bekommt der Avoid Task, die niedrigste der Task zum Vorwärtsfahren. Je nach Sensorinput und Priorität werden die Tasks aktiviert.

In der Regel wird die Subsumption-Architektur für mobile Agenten verwendet. Mit ihr lassen sich Agenten entwickeln, die über eine hohe Robustheit in ihrem Verhalten verfügen, d. h. zu jedem Zeitpunkt handlungsfähig bzw. entscheidungsfähig bleiben.

Für die Subsumption-Architektur kommen in erster Linie klassische textuelle Sprachen wie z. B. C, C++ oder Java zum Einsatz.

Betrachtet man z. B. den ct-bot[9], wie er an der Hochschule für Angewandte Wissenschaften Hamburg im Roboterlabor Verwendung findet, ist schnell zu sehen, dass trotz eines guten Frameworks selbst einfache Verhalten einen hohen Entwicklungsaufwand erfordern.

2.3 Deliberative Agenten

Stellvertretend für die deliberativen Agenten stelle ich in diesem Abschnitt das Multiagentensystem Jadex der Universität Hamburg[10] vor, das sich an der "Believe Desire Intention" (BDI) Metapher orientiert und die Middleware Jade[11] der Telecom Italia zur Agentenkommunikation einsetzt. Agentensysteme die den BDI-Ansatz verfolgen sind sogenannte Procedural Reasoning Systems (PRS).

Abbildung 4 zeigt abstrakt wie ein Multiagenten-System in Jadex aufgebaut ist. Eine Menge von Agenten ist in einer Umwelt situiert. Sie können mit dieser Umwelt über Sensoren und Aktoren interagieren. Zudem haben sie die Möglichkeit mit den anderen Agenten im System zu kommunizieren. Die Kommunikation basiert durch Jade auf dem FIPA[12] Standard.

[9] Entwickelt von Heise Zeitschriften Verlag GmbH & Co. KG (2006).
[10] Jadex (2008)
[11] Jade (2008)
[12] FIPA (2007)

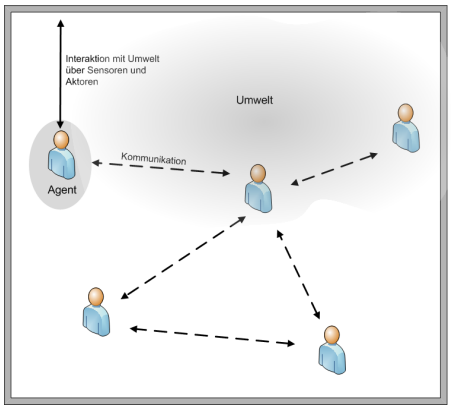

Abbildung 4: Funktionelle Sicht auf Jadex. Agenten situiert in einer Umwelt, interagieren mit Umwelt und untereinander über Kommunikation.

Agentenverhalten werden in Jadex in XML und Java modelliert. Zu den Bausteinen eines Verhaltens gehören *Plans*, *Beliefs* und *Goals*. Abbildung 5 verdeutlicht wie diese drei Bestandteile zu einem Ablauf verbunden werden, *Beliefs* repräsentieren das Weltmodell. Die funktionelle Basis von den Plänen (*Plans*), der Wissensbasis (*Beliefs*) und den Zielen (*Goals*) werden in Java programmiert. Die XML-Struktur, bei Jadex *Agent Description File* (ADF) genannt, fügt die einzelnen Bausteine zusammen und konfiguriert abschließend das Verhalten des Agenten.

Nachfolgend beziehe ich mich auf das Beispiel aus dem Jadex Tutorial[13], in dem ein Translator-Agent erstellt wird, der einzelne Wörter vom Englischen ins Deutsche übersetzt. Um diese Aufgabe zu bewältigen benötigt der Agent als erstes eine Wissensbasis (*Beliefs*).

Im ADF kann ein *Beliefset* erstellt werden, das eine Wissensdatenbank darstellt und zu dem Wissen des einzelnen Agenten hinzugefügt werden kann:

```
<beliefset name="egwords" class="Tuple" exported="true">
  <fact>new Tuple("milk", "Milch")</fact>
  <fact>new Tuple("cow", "Kuh")</fact>
  <fact>new Tuple("cat", "Katze")</fact>
</beliefset>

<beliefs>
  <beliefsetref name="egwords">
    <concrete ref="transcap.egwords" />
  </beliefsetref>
</beliefs>
```

[13] Braubach und Pokahr (2007)

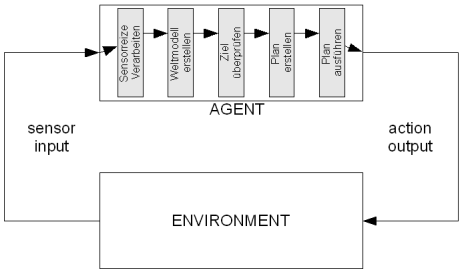

Abbildung 5: Deliberative Agenten erstellen aus den Sensordaten ein Weltbild, überprüfen daran ob sie ihre Ziele erreicht haben, wenn nicht, erstellen oder verändern sie Handlungspläne und führen diese aus.

Als nächstes benötigt der Agent einen Plan, wie er englische Wörter in deutsche umwandeln kann:

```
<plan name="egtrans">
  <body>new EnglishGermanTranslationPlanC1()</body>
  <trigger>
    <messageevent ref="request_translation"/>
  </trigger>
</plan>
```

Für einen ganz einfachen Agenten wie diesen, der nur in eine Richtung übersetzt, ist das Agentenverhalten in Jadex hiermit ausreichend beschrieben.

Im Tutorial wird der Agent immer weiter ausgebaut, so dass später noch Ziele definiert werden. Zum Beispiel um in beide Richtungen übersetzen zu können oder eine weitere Zielsprache zu haben wie z. B. Französisch.

```
<goals>
  <achievegoal name="translate">
    <parameter name="direction" class="String"/>
    <parameter name="word" class="String"/>
    <parameter name="result"
      class="String" direction="out"/>
  </achievegoal>
</goals>
```

Wie an diesen Beispielen zu sehen ist, spiegeln sich die Java Notationen im ADF wider. Eine Person, die die Agentenbeschreibung erstellen möchte, muss folglich nicht nur die Struktur der Pläne und Ziele in der Javaumgebung kennen, sondern zudem die Java-Syntax beherrschen.

Abbildung 6: Spielszene aus Façade

3 Expressiv AI

3.1 Façade

Façade[14] ist ein interaktives Drama, in dem jemand ein befreundetes Ehepaar besucht, das er oder sie schon lange nicht mehr gesehen hat. Die drei wollen ihre alte Freundschaft wieder aufleben lassen. Doch der Freund bzw. Freundin gerät in einen aufkeimenden Ehestreit, dessen Verlauf von seiner bzw. ihrer Reaktion abhängt.

Der Spieler schlüpft in die Rolle dieses Freundes und ist bei Trip und Grace zum Dinner eingeladen. Noch bevor der Spieler die Wohnung des Ehepaars betritt hört er durch die Tür, wie die beiden eine Meinungsverschiedenheit ausfechten.

Abbildung 6 zeigt eine typische Situation aus dem Spiel. Der Spieler sieht die Spielwelt wie bei Egoshootern aus der Ich-Perspektive. Nur anders als bei Egoshootern unterhält sich der Spieler bei Façade mit den simulierten Personen. Dem Spieler steht es frei sich in der Wohnung der beiden umzuschauen, Dinge aus einem Regal zu nehmen und zu betrachten. Er kann ans Telefon gehen, wenn es klingelt. Alles hat Auswirkung auf den Ablauf des Spiels. Am meisten beeinflusst aber das, was der Spieler “sagt”. Er unterhält sich wie bei einem Chat mittels der Tastatur mit dem Ehepaar.

Wenn der Spieler nichts tut oder sagt, nimmt die Handlung unabhängig ihren Lauf. Die Handlung kennt keine Triggerpositionen, an denen zwingend auf Spieleraktionen gewartet wird. Der Verlauf der Handlung passt sich dynamisch an die Aktionen des Spielers an und die Geschichte spitzt sich zu einem dramatischen Höhepunkt zu[15].

[14] Mateas und Stern (2003) und Mateas (2001b)
[15] Mateas (2001b)

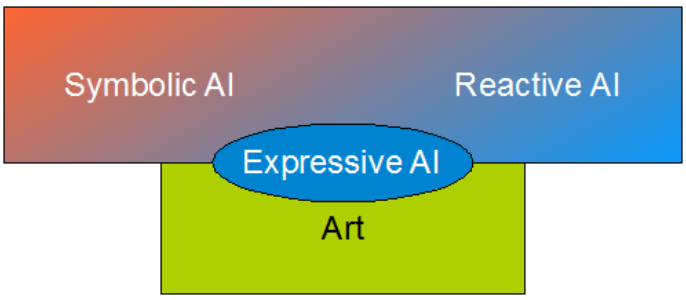

Abbildung 7: Einordnung von Façade als Expressiv AI in die traditionellen Ansätze der KI

3.2 Einordnen von Expressiv AI (Artificial Intelligence))

Expressiv AI wird von Mateas in Mateas (2001a) zwischen den beiden klassischen Ansätzen der symbolischen und reaktiven künstlichen Intelligenz (KI) und der Kunst eingeordnet (siehe Abbildung 7). Bei Expressiv AI steht im Vordergrund **"cultural artifacts"** zu modillieren, anstatt einen allgemeinen Ansatz für KI zu schaffen. **"cultural artifacts"** heißt, dass es sich um kulturelle Aspekte handelt, die modelliert werden sollen, so wie glaubwürdige Agenten ("beliefable agents"), simulierte Personen, die sich in ihrer begrenzten Domäne so verhalten, wie man es von echten Menschen erwarten würde.

3.3 A Behavior Language (ABL)

ABL ist aus der *believable agent language* Hap entstanden[16], eine Sprache, die geschaffen wurde um "believable agents" zu modellieren. Dem entsprechend verfügt sie über Ausdrucksmöglichkeiten um einem simulierten Agenten eine Persönlichkeit zu geben. ABL erweitert Hap um sogenannte "Joint Goals", um gemeinsame Ziele von mehreren Agenten zu definieren. Die Syntax von ABL ist zu Hap etwas abgewandelt, um sie etwas javaähnlicher werden zu lassen. Denn die Sprache ist in Java programmiert und erzeugt ihrerseits auch Java-Code.

Im Folgenden zeige ich ein Beispiel, wie Verhalten in ABL modelliert werden.

In Façade wartet Trip zu Beginn darauf, dass man als Gast anklopft. Wenn dieses Ereignis eintritt, gibt er zu verstehen, dass er das Klopfen gehört hat, geht dann zur Tür, öffnet sie und begrüsst den Gast. Dieses Verhalten wird in ABL folgendermaßen ausgedrückt:

```
sequential behavior AnswerTheDoor() {
    WME w;
    with (success test { w = (KnockWME) } ) wait;
    act sigh();
    subgoal OpenDoor();
```

[16] Mateas und Stern (2004)

```
    subgoal GreetGuest();
    mental_act { deleteWME(w); }
}
```

WME bedeutet “Working Memory”. In dem Working Memory sind Ereignisse gespeichert. Zu Beginn des Verhaltens wird geprüft, ob dort das Anklopfenereignis aufgetreten ist, wenn nicht, wird solange gewartet, bis es eintritt. Mit dem *act* steuert man die Aktoren, in diesem Fall den virtuellen Körper des Agenten. *Subgoal* sind Unterziele, die zu erreichen sind. In ABL sind dies andere *behaviors*, die somit rekursiv aufgerufen werden. Ein *mental act* ist ein interner Vorgang wie Berechnungen oder, wie hier im Beispiel, für das Löschen des Ereignisses aus dem Working Memory.

Das *OpenDoor* Subgoal ist in Façade in zwei Varianten implementiert. Die ich ebenfalls vorstelle:

```
sequential behavior OpenDoor() {
    precondition {
        (KnockWME doorID :: door)
        (PosWME spriteID == door pos :: doorPos)
        (PosWME spriteID == me pos :: myPos)
        (Util.computeDistance(doorPos, myPos) > 100)
    }
    specificity 2;
    // Too far to walk, yell for knocker to come in
    subgoal YellAndWaitForGuestToEnter(doorID);
}

sequential behavior OpenDoor() {
    precondition { (KnockWME doorID :: door) }
    specificity 1;
    // Default behavior - walk to door and open
}
```

Die Varianten unterscheiden sich in ihrer *Precondition*. Ist Trip zu weit von der Tür entfernt (die obere Version), ruft er den Gast herein, anstatt zur Tür zu gehen. Ist er nahe genug an der Tür, geht er hin und öffnet sie selber.

Es fällt auf, dass in den *Behaviors* keine Pläne definiert werden. Das liegt daran, dass das Multiagenten-System einen Metaplan erstellt, in dem es zu bestimmten Zeiten des Spielverlaufs *Behaviors* zur Verfügung stellt oder sogar ausblendet, damit sich die Handlung stetig zuspitzt[17].

[17] Mateas (2006)

4 Vergleich

4.1 Konzeptionelle Ebene

4.1.1 Reaktive Agenten

Die Subsumption-Architektur hat in diesem Vergleich eine Sonderstellung, da sie ein Konzept darstellt und auf verschiedene Weise implementiert werden kann. Daher bleibe ich hier allgemein.

Reaktive Agenten sind in der Praxis vorwiegend mobile Systeme. Dies ist bedingt dadurch, dass unsere Umwelt stark nicht-deterministisch ist und sich die Subsumption-Architektur in diesem Zusammenhang als sehr robust erwiesen hat. Die Agenten agieren vorwiegend eigentständig und kommunizieren wenig mit anderen Agenten.

4.1.2 Jadex – Deliberative Agenten

Bei Jadex hingegen steht die Kommunikation der Agenten im Mittelpunkt. Die Agenten besitzen eine gewisse Repräsentation ihrer Umwelt, verfolgen Ziele und erstellen Pläne, wie sie diese Ziele erreichen können. Jadex dient in erster Linie dazu ein Multiagenten-System aus Informationsagenten aufzubauen.

4.1.3 Façade

Bei Façade sollen virtuelle Peronen möglichst glaubwürdig erscheinen. Die Agenten besitzen ähnlich wie die BDI-Agenten von Jadex eine Repräsentation ihrer Umwelt und verfolgen Ziele. Das Konzept eines Planes existiert hier allerdings nicht in den Agenten, sondern im Multiagenten-System selbst, das kontrolliert, dass sich die Situation im Spiel immer einem dramatischen Höhepunkt entgegen geht.

4.2 Beschreibungsebene

4.2.1 Reaktive Agenten

Das Agentenverhalten wird hier zumeist “fest verdrahtet” in Programmcode ausgedrückt. Eine Änderung des Verhalten kann nur durch Änderung der Codebasis erreicht werden. Der Verhaltensdesigner muss das komplette System verstehen und systemnah programmieren.

4.2.2 Jadex

Durch den allgemeinen Ansatz bei Jadex ist es erforderlich ebenfalls sehr systemnah zu programmieren und ein Verhaltensdesigner muss auf jeden Fall Java programmieren, da selbst in dem ADF z. T. Java-Ausdrücke verwendet werden.

4.2.3 Façade– ABL

ABL ist direkt auf die Bedürfnisse der Aufgabendomäne von Façade zugeschnitten und abstrahiert daher in einem hohen Grad von dem darunter liegenden System. Ein Verhaltensdesigner benötigt zwar gewisse Kenntnisse von dem Aufbau des Systems und eine Vorstellung von den Abläufen, braucht sich aber nicht auf die Ebene des Systems zu begeben.

5 Fazit

Ein allgemeiner Ansatz, wie ihn z. B. Jadex verfolgt, resultiert darin, dass in der Agentenbeschreibungssprache Ausdrücke verwendet werden, die aus dem darunter liegenden System stammen und den Benutzer damit zwingen das System verstehen zu müssen.

Daraus läßt sich schließen, dass eine hinreichende Abstraktion von dem unterliegenden System nur durch eine domänenspezifische Agentensprache möglich ist, so wie es in Façade demonstriert wird. Für das IFlat-Projekt heißt das, dass eine Agentenbeschreibungssprache für intelligente Haushaltsgeräte und Unterhaltungselektronik geschaffen werden muss.

Literatur

[Jadex 2008] *Jadex BDI Agent System*. 2008. – URL http://vsis-www.informatik.uni-hamburg.de/projects/jadex. – letzter Zugriff 16.07.2008

[Jade 2008] *Java Agent DEvelopment Framework (JADE)*. 2008. – URL http://jade.tilab.com/. – letzter Zugriff 16.07.2008

[Boulette's Robotics page 2008] : *Boulette's Robotics page*. Juli 2008. – URL http://www.convict.lu/Jeunes/RoboticsIntro_old.htm

[Braubach und Pokahr 2007] BRAUBACH, Lars ; POKAHR, Alexander: *Jadex Tutorial*, Juni 2007. – URL http://vsis-www.informatik.uni-hamburg.de/projects/jadex/jadex-0.96x/tutorial/tutorial.pdf. – letzter Zugriff 16.07.2008

[Ferber 2001] FERBER, Jacques: *Multiagentensysteme*. Addison-Wesley, 2001. – ISBN 3-8273-1679-0

[FIPA 2007] *FIPA specifications*. 2007. – URL http://www.fipa.org/. – letzter Zugriff 25.07.2008

[Günther 2003] GÜNTHER, Görz (Hrsg.): *Handbuch der Künstlichen Intelligenz*. Oldenbourg, 2003. – ISBN 3-486-27212-8

[Heise Zeitschriften Verlag GmbH & Co. KG 2006] *c't-Bot und c't-Sim*. 2006. – URL http://www.heise.de/ct/projekte/ct-bot/. – Zugriff: Juni.2007

[Mateas 2001a] MATEAS, Michael: Expressiv AI / Carnegie Mellon University. URL http://www-2.cs.cmu.edu/~michaelm/publications/Leonardo2001.pdf, 2001. – Forschungsbericht. letzter Zugriff 24.07.2008

[Mateas 2001b] MATEAS, Michael: *A Preliminary Poetics for Interactive Drama and Games*. 2001. – URL http://www-2.cs.cmu.edu/~michaelm/publications/DigitalCreativity2001.pdf. – letzter Zugriff 24.07.2008

[Mateas 2006] MATEAS, Michael: *Expressive Intelligence: Artificial Intelligence, Games and New Media*. Video. October 2006. – URL http://deimos3.apple.com/WebObjects/Core.woa/Browse/itunes.stanford.edu.1357108180.01357108182.1356831578?i=1550820088. – letzter Zugriff 25.07.2008

[Mateas und Stern 2003] MATEAS, Michael ; STERN, Andrew: *Façade: An Experiment in Building a Fully-Realized Interactive Drama*. 2003. – URL http://www.

interactivestory.net/papers/MateasSternGDC03.pdf. – letzter Zugriff 24.07.2008

[Mateas und Stern 2004] MATEAS, Michael ; STERN, Andrew: *A Behavior Language: Joint Action and Behavioral Idioms*. In: PRENDINGER, H. (Hrsg.) ; ISHIZUKA, M. (Hrsg.): *Life-like Characters. Tools, Affective Functions and Applications*, Springer, 2004. – URL http://www.interactivestory.net/papers/MateasSternLifelikeBook04.pdf. – letzter Zugriff 24.07.2008

[Pfeiffer und Scheier 2001] PFEIFFER, Rolf ; SCHEIER, Christian: *Understanding Intelligence*. Bradford Books, 2001. – ISBN 0-262-16181-8

[Rosseburg 2008] ROSSEBURG, Kai: *Blended Learning*. 2008. – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2008/rosseburg/folien.pdf. – letzter Zugriff 16.07.2008

[Tennstedt 2007] TENNSTEDT, Sven: *Genese sozialen Verhaltens in Multiagentensystemen durch genetische Verfahren*, Hochschule für Angewandte Wissenschaften Hamburg, Bachelorarbeit, 2007. – URL http://users.informatik.haw-hamburg.de/~kvl/tennstedt/bachelor.pdf

[Wooldridge 2002] WOOLDRIDGE, Michael: *MultiAgent Systems*. Wiley, 2002. – ISBN 0-471-49691-X