Entwurf einer generischen Agentenarchitektur für MAS

HAW-Hamburg / Anwendungen II
29. Juli 2014
Jan Dalski

1 Einleitung

Mein Ziel besteht darin, eine generische Architektur für Multiagentensimulationen zu entwerfen. Diese Architektur soll offen für alle Anwendungsdomänen sein und in dem MARS-Framework eingesetzt werden können. Sie soll den besten Kompromiß zwischen einer abstrakten, nicht einschränkenden Lösung und einer Plattform mit vorgefertigten Strukturen und Services bilden, um die Entwicklung von Simulationsagenten zu erleichtern und zu beschleunigen.

1.1 Bisherige Recherche

In AW-1 wurden zwei generische Ansätze vorgestellt: Die *Generic Agent Architecture* (GAA) [5] und die *Generic Architecture for Multi-Agent Simulations* (GEAMAS) [3]. Bei näherer Betrachtung erwiesen sich diese Architekturen aber als sehr vage, was dem generischen Aspekt geschuldet ist. Da Simulationen gewöhnlich für eine spezielle Domäne konzipiert werden, klammern generische Ansätze viele Teilbereiche bewußt aus. Aufgrund dieser hohen Abstraktion sagen sie nur wenig über die Konzeption eines (einzelnen) Agenten aus und eignen sich damit nur bedingt als Grundlage für meine Architektur.

1.2 Ziele dieses Papiers

Neben diversen anderen Hürden, wie beispielsweise die Ausführung durch die Simulationsumgebung oder die Darstellung der Umwelt soll es in diesem Papier um drei agentenrelevante Probleme gehen. Diese werden nachfolgend kurz erläutert, bevor dann in den nächsten Abschnitten Lösungsansätze präsentiert und auf Eignung untersucht werden.

- **Dynamische Planung:** Da für eine Vielzahl von Agenten eine zielorientierte Planungslogik (GOAP) sinnvoll wäre, soll eine solche Struktur in den Agentenkern integriert werden. Um ein realistisches, robustes Verhalten des Agenten zu fördern, ist es angemessen, ein erweitertes Verfahren zu implementieren, das je nach Lage Ziele pausieren, repriorisieren oder abbrechen kann.
- **Interaktionsdesign:** Es muß ein Basiskonzept für den Entwurf, die Repräsentation und die Ausführung von (Inter-)Aktionen gefunden werden. Wie werden optionale und obligatorische Interaktionen zwischen mehreren Agenten realisiert?
- **Emotionale Agenten:** Für die Mehrzahl der Anwendungsgebiete dürfte die Simulation von Menschen oder Tieren im Vordergrund stehen. Dass diese in ihrem Handeln von Emotionen beeinflußt werden, sollte für aussagekräftige Resultate berücksichtigt werden. Es wäre daher wünschenswert, eine Basisstruktur hierfür zu integrieren, die der Domänenentwickler später nutzen kann.

2 Lösungsansätze

Hier werden nun drei Papiere vorgestellt, welche zur Lösung der eben genannten Probleme hilfreich erscheinen. Zunächst wird der Anspruch an das Papier und der erhoffte Vorteil formuliert, dann wird das Konzept präsentiert. In der nächsten Sektion werden sie in Relation zu meinem Vorhaben gesetzt und auf Nutzen untersucht.

2.1 *Suspending and resuming Tasks in BDI Agents*

Beginnend soll das Problem der Zielplanung für GOAP-Agenten betrachtet werden. Wie eingangs erwähnt, ist die Verwendung einer erweiterten Planungslogik erstrebenswert, welche neben der Zielerstellung und -terminierung auch Möglichkeiten zur Neuplanung, Repriorisierung und Unterbrechung beherrscht. Dadurch soll die Robustheit und Autonomie der Agenten erhöht werden, da diese so flexibler auf sich ändernde Umweltbedingungen reagieren können und dem Entwickler zudem mehr Entwurfsmöglichkeiten zur Verfügung stehen.

Typische Gründe für das Aus- oder Absetzen von Zielen sind Ressourcenkonflikte und geänderte Kontextbedingungen. Es kann aber auch sein, daß kein Plan zur Erreichung des Zieles mehr verfügbar ist, oder eine Pausierung aufgrund von Synergieeffekten mit anderen Zielen hilfreich erscheint (wenn beispielsweise zwei Ziele den gleichen Zwischenzustand erreichen wollen). Ein Ziel kann aber auch zugunsten eines wichtigeren Zieles (Repriorisierung) oder auf Befehl von anderen Agenten (Autorisierung vorausgesetzt) abgesetzt werden.

Ein solches Unterbrechungskonzept wird in [4] vorgestellt. Es ist ein generischer Mechanismus, der sich zum Einbau in gewöhnlichen BDI-Architekturen eignet und die Aussetzung und Wiederaufnahme von *Tasks* ermöglichen soll. Ein *Task* ist dabei ein Oberbegriff für Ziele, damit verbundene Pläne und Aktionen. Dieser Mechanismus soll transparent (falls nicht benötigt) sein, andernfalls parametrisierbar und leicht zu erweitern. Realisiert wird dies durch Meta-Aktionen, die der Planungsphase vorgeschaltet sind. Hier kann der Entwickler eigene Methoden zum Absetzen und Wiederaufnehmen eines Planes implementieren, welche analog zu Event-Handlern funktionieren. Dieses Konzept wurde für explizite Ziele (*achievement goals*) entworfen, ist aber möglicherweise auch für implizite Ziele (also zu haltende Zustände) nutzbar.

2.1.1 Metadaten eines Tasks

Die Verfolgung eines Zieles eines Planes wird durch drei Bedingungen gesteuert: Die *Vorbedingung* muß erfüllt sein, damit das Ziel in die Intentions-Datenstruktur aufgenommen werden kann. Während der Zielverfolgung muß die *Inbedingung* wahr bleiben, anderenfalls wird das Ziel abgesetzt. Die *Nachbedingung* gibt schlußendlich an, wann das Ziel (der Zielzustand) erreicht ist.

Daraus ergibt sich die Notwendigkeit, einem Ziel einen Ausführungszustand zuzuweisen. Das zugrundeliegende Prinzip ist das eines endlichen Automaten. Die Zustände sind *initialized*, *intended*, *suspended*, *succeeded* und *failed*, die Transitionen des Automaten entsprechen den Meta-Aktionen des Agenten. Die nachfolgende Abbildung veranschaulicht die Übergänge:

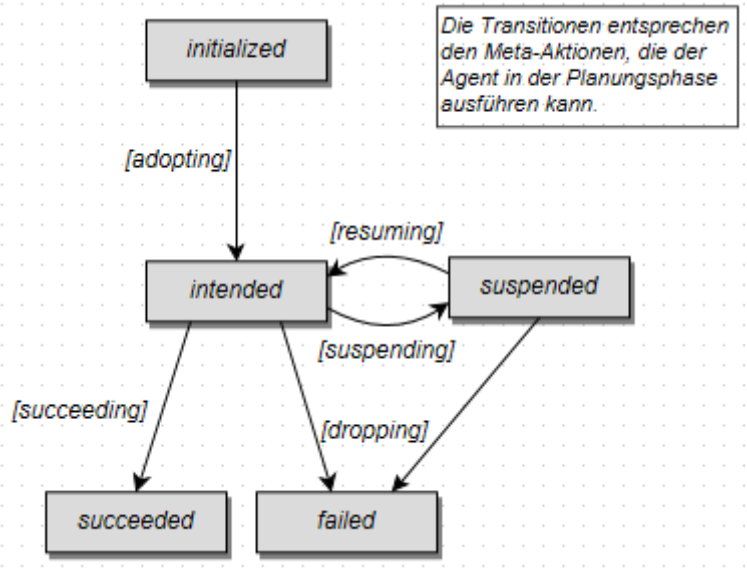

Abbildung 1: Zustände und Transitionen der Zielverwaltung

Auch wenn in obigem Kontext von Zielen gesprochen wurde, so gelten die Ausführungsbedingungen und -zustände für Pläne analog. Für Aktionen ist dies nicht der Fall; sie werden in diesem Kontext als atomar angesehen. Damit sind sie also nicht unterbrechbar und die Zustände und Bedingungen erübrigen sich.

Neben der Ausführbarkeit existieren noch weitere Größen, die Einfluß auf ein Ziel besitzen. Wann immer mehrere Tasks potentiell ausführbar sind, muß der Planungsalgorithmus eine Priorisierung vornehmen können, um anhand dessen den geeignetsten Kandidaten zur Ausführung zu finden.

Die Ermittlung der Priorität erfolgt dabei dynamisch aus beliebigen, domänenrelevanten Faktoren. Beispielsweise bietet es sich an, einen Task mit einem Nutzwert, einer Deadline und Ausführungskosten zu versehen. Aber auch interne (den Agenten betreffende) oder externe (die Umwelt betreffende) Abhängigkeiten oder eine Erfolgschance könnten von Bedeutung sein.

2.1.2 Unterbrechung

Zu Beginn jeder Planungsphase führt der Agent einen Abgleich seiner Ziele und Pläne mit der aktuellen Situation durch. Wenn für einen Task die Inbedingung nicht mehr gültig ist, muß er abgesetzt werden. Hierzu wird zunächst eine Wiederaufnahmebedingung erstellt und (zusammen mit eventuellen Metadaten) an den Task angehängt. Dann wird die *suspend()*-Methode aufgerufen. Dabei handelt es sich um einen leeren Funktionsrumpf, der von dem Entwickler durch eine eigene Funktion überschrieben werden kann – etwa um belegte Ressourcen freizugeben, den Task abzubrechen oder ein Fortfahren zu erzwingen.

Nun kann der Task auf *suspended* gesetzt werden. Dieser Vorgang wird rekursiv für alle Untertasks wiederholt. Für ein Ziel wird also auch der Plan, und damit alle anstehenden Aktionen[1] als inaktiv markiert. Wenn der Plan noch nicht begonnen wurde, kann er auch

[1] Es können durchaus mehrere Handlungen anstehen, wenn die Parallelbearbeitung der Tasks möglich ist.

verworfen und bei Bedarf neu erstellt werden (er darf nur nicht fehlschlagen, da der Agent ansonsten bei Wiederaufnahme nach Alternativplänen suchen würde). Für ein Ziel, welches noch nicht über einen Plan verfügt, brauchen keine weiteren Schritte unternommen werden.

2.1.3 Fortführung

Neben der Validierung der aktiven Tasks muß in jedem Planungszyklus auch eine Überprüfung der pausierten Ziele und Pläne auf Fortführung vorgenommen werden. Die Voraussetzung dazu ist die Erfüllung der Wiederaufnahme-Bedingung. Wenn sie wahr ist, wird eine (standardmäßig leere) *resume()*-Methode aufgerufen, die der Entwickler mit eigener Anwendungslogik füllen kann (beispielsweise um zielrelevante Daten wiederherzustellen). Danach wird das Ziel als wieder verfügbar markiert und der dazugehörige Plan überprüft bzw. ein neuer Plan erstellt.

Bei der Validierung eines Planes muß die Inbedingung überprüft werden, um die Gültigkeit im aktuellen Kontext festzustellen. Wenn dies der Fall ist, werden die nächsten Schritte des Planes aktiviert und das Ziel kann in der nächsten Iteration wieder verfolgt werden[2]. Anderenfalls wird der Plan entweder abgebrochen und durch einen Alternativplan ersetzt, oder (falls keine Alternative besteht) pausiert. In letzterem Falle wäre das Ziel zwar gültig, aber nicht erreichbar – weshalb es wieder deaktiviert werden kann.

2.2 *IODA - Interaction-Oriented Design of Agents*

2.2.1 Besonderheiten der Interaktionen in MABS

In gewöhnlichen Multiagentensystemen beschränkt sich die Interaktion zwischen Agenten oftmals auf Organisations- oder Synchronisationsaspekte. Die Umsetzung erfolgt häufig über den Entwurf von Protokollen; Modellierungskonzepte wie endliche Automaten oder Petri-Netze kommen hier zum Einsatz. Die Informationsweiterleitung wird durch simplen Nachrichtenaustausch vorgenommen.

Bei MABS hingegen sind die Interaktionen der Schlüssel zu dem gewünschten, emergenten Verhalten. Sie stellen „physische“ Aktionen zwischen den Agenten dar, bspw. das Fällen eines Baumes durch einen Holzfäller. Dabei sind die Umgebung und/oder mehrere Agenten simultan involviert, wobei Synchronisation und Berechtigungen berücksichtigt werden müssen: Eine Anfrage zur Kooperation darf von dem Zielagenten ablehnt werden, eine Pflichtaktion – die Gazelle wird vom Geparden getötet – hingegen nicht.

Die Interaktionen in einer Simulation mittels reinem Nachrichtenaustausch abzubilden, ist restriktiv und im Falle von integrierten Interoperabilitätsstandards wie *FIPA-ACL* oder *KQML* zusätzlicher Ballast – solche Protokolle sind dafür bestimmt, Kommunikation zwischen heterogenen, verteilten Systemen zu ermöglichen. Stattdessen sollten sie in einem Simulationssystem eine zentrale Rolle spielen – sowohl im Entwurf der Simulation als auch bei der Agentenspezifikation.

[2] Es kann natürlich trotzdem geschehen, daß ein anderes Ziel eine höhere Priorität besitzt.

Das *Interaction-Oriented Design of Agents (IODA)* [2] umfaßt sowohl eine Entwurfsmethodik, als auch ein formales Modell und Umsetzungskonzept für Interaktionen in MABS. Das Ziel ist es, eine klare Trennung von Modell und Implementation zu wahren und dadurch die Domänenspezialisten möglichst lange am Entwicklungsprozeß beteiligen. Hierzu erfolgt eine Interaktionsspezifikation unabhängig von den Agenten, die diese später einsetzen sollen. Die Agenten selbst werden in *Familien* (Begriff in Anlehnung an die Biologie gewählt) eingeteilt und können Vererbungshierarchien bilden. So entsteht ein generisches Framework, welches sich durch eine hohe Wiederverwendbarkeit auszeichnet.

2.2.2 Repräsentation der Interaktionen

IODA verwendet eine homogene Struktur für Interaktionen und „normale“ Aktionen. Letztere werden im IODA-Kontext auch als *degenerate interactions* bezeichnet und können als reflexive Interaktionen angesehen werden. Aufgrund dieser Zusammenfassung lassen sich alle Interaktionen als folgenden semantischen Block ausdrücken:

$I = (id, cardinality, agents, primitives, preconditions, trigger, actions)$

Eine Interaktion ist demnach ein Tupel, bestehend aus einem Bezeichner, einer Kardinalität (wird nachfolgend erklärt), den IDs der beteiligten Agenten, eine Zuordnung von Funktionsprimitiven an alle beteiligten Agenten, Vorbedingungen, einem Auslöser und der eigentlichen Aktionslogik.

Durch diese generische, unabhängige Definition mit generischer Semantik, die auf abstrakten Primitiven operiert, ist ein Polymorphismus möglich. Agenten, die zur Ausführung oder Teilnahme an einer Interaktion befähigt sein sollen, müssen Implementationen für diese Primitive liefern (indem sie eine Schnittstelle implementieren). Als Beispiel sei eine *EatInteraction* betrachtet: Es ist für Beobachter zweifelsfrei erkennbar, wenn z.B. ein Wolf und ein Mensch etwas essen, da das zugrundeliegende Prinzip identisch ist – auch wenn die konkrete Ausführung (für gewöhnlich) anders erfolgt.

2.2.3 Rollen

IODA unterteilt die an einer Interaktion involvierten Agenten in zwei Gruppen. Die Urheberagenten ($agents_{source}$) initiieren eine Interaktion, sie führen diese aktiv aus. Die konträre Rolle bilden die Zielagenten ($agents_{target}$), sie sind passiv beteiligt. Je nach Rolle unterscheiden sich die Primitive, die ein Agententyp zu implementieren hat. Die *Kardinalität* gibt das Rollenverhältnis an:

$cardinality(I) = (|agents_s|, |agents_t|)$

Interaktionen mit einer Kardinalität von (1,*) (also nur ein Urheber) entsprechen dabei der Normalform. Um die Koordination zu gewährleisten, sind alle Interaktionen in diese Form zu überführen, indem weitere Quellagenten gegebenenfalls als Zielagenten einzutragen sind. Als neue Bedingung für die Ausführung der Interaktion kommt die Zustimmung dieser Zielagententeilmenge hinzu.

In den Beispielen von IODA werden nur Interaktionen der Kardinalitäten (1,0) und (1,1) verwendet.

2.2.4 Agentenstruktur & Ausführung

Die Struktur der in IODA verwendeten Agenten folgt der geläufigen Phaseneinteilung in Wahrnehmung, Planung und (Inter-)Aktion. Das Agentenprogramm kann beliebiger Art sein (reaktiv, kognitiv, hybrid), in der Beispielanwendung kommt eine reaktive Planungskomponente zum Einsatz. Zur Ausführung der Agenten stellen die Entwickler eine einfache Simulationsumgebung namens *JEDI* (*Java Environment for Design of Interactions*) vor. Diese ermöglicht eine sequentielle Ausführung der Agenten in zufälliger Reihenfolge in einer zeitdiskreten Umwelt.

Dem IODA-Konzept liegt die Idee der Reifikation (Verdinglichung) von Interaktionen zugrunde. Da eine Interaktion als ein Tupel mit diversen Attributen definiert ist, wird sie programmiertechnisch durch eine Klasse repräsentiert. Die Erzeugung einer solchen Instanz und die Zuweisung konkreter Werte (Agentenreferenzen) geschehen im Rahmen der Planungsphase des (Urheber-)Agenten. In der Aktionsphase führt dieser das Interaktionsobjekt dann aus. Die Teilnahme für die Zielagenten ist verpflichtend, sie besitzen keinen Einfluß auf das Stattfinden der Interaktion. Bei der Überführung von Kooperationsaktionen in die Normalform muß daher ein weiterer Mechanismus zum Tragen kommen, um die Zustimmung im Vorfeld einzuholen.

2.2.5 Entwurfsphase

Für die Erstellung eines gültigen, formalen Modells ist die Beteiligung der Domänenexperten unabdingbar. Um diesen Prozeß zu strukturieren, basiert die IODA-Methodik auf einer einfachen Abfolge von Spezifikationen, die aufeinander aufbauen und so schrittweise das Simulationsmodell entstehen lassen. Diese sieben Schritte werden nachfolgend kurz zusammengefaßt.

1. **Vorannahmen treffen:** Zunächst müssen grundlegende Entwurfsentscheidungen festgelegt werden. Wie wird die Simulation ausgeführt, zeitdiskret (feste Intervalle) oder kontinuierlich? Wie wird die Umwelt darstellt, was kann ein Agent wahrnehmen und wie erfolgt das? Diese Annahmen bilden die Basis für alle späteren Schritte.

2. **Interaktionen festlegen:** Die erste richtige Entwurfsstufe zeigt deutlich den Fokus von IODA: Welche Prozesse sollen in der Simulation berücksichtigt werden? An dieser Stelle werden die zu beobachtenden Interaktionen festgehalten. Dies erfolgt zunächst in grober Form – Spezifizierung kommt später!

3. **Agententypen bestimmen:** Nun werden die Typen (in IODA: *Familien*) der Agenten bestimmt, welche in den Interaktionen involviert sind. Ebenfalls werden deren Spezialisierungen (Vererbungsstrukturen) identifiziert und (z.B. in einem Graphen) hierarchisch dargestellt.

4. **Interaktionsmatrix bilden:** Basierend auf diesen Daten wird nun die Rohform der Interaktionsmatrix erstellt. Sie enthält als Spalten- und Zeilenkopf die Agententypen, die Zellen beschreiben die möglichen Interaktionen. Die Zeile zu einer Interaktion identifiziert den Agententyp des Initiators, während die Spalte den Zielagententyp abbildet. Daneben kann sie eine Metrik beliebiger Art (bspw. Euklidische Distanz) beinhalten, welche die maximale Interaktionsdistanz angibt. Eine zusätzliche „Nullspalte“ dient zur Auflistung der reflexiven Aktionen. Als Beispiel ist auf der nächsten Seite die Matrix einer einfachen Räuber-Beute-Simulation abgebildet.

Table 2 Raw Interaction Matrix of an ecosystem simulation, where wolves, sheep, goats and grass agents evolve

Source \ Target	∅	Grass	Sheep	Goat	Wolf
Grass	(Grow)				
Sheep	(Move)	(Eat, d = 0)	(Procreate, d = 1)		
Goat	(Move)	(Eat, d = 0)		(Procreate, d = 1)	
Wolf	(Move)		(Eat, d = 3)	(Eat, d = 3)	(Procreate, d = 1)

The element *(Eat, d = 0)* at the intersection of the row starting with *Sheep* and the column starting with *Grass* is read "Sheep agents are able to initiate the Eat interaction with a target Grass agent at a maximal distance of 0 from the source"

Abbildung 2: Beispielsmatrix einer Räuber-Beute-Simulation aus dem IODA-Papier

5. **Interaktionsspezifikation:** Nach der Interaktionserfassung erfolgt jetzt die Konkretisierung derselben. Es werden der Auslöser, die Vorbedingungen und die Aktionslogik ermittelt. Hieraus lassen sich später die Primitive ableiten, welche die teilnehmenden Agenten zu implementieren haben. An dieser Stelle ist eine Abwägung des Detailgrades zu treffen: Je konkreter die Spezifikation, desto leichter erfolgt die Implementation – aber sie ist auch weniger generisch und schlechter wiederzuverwenden. Möglicherweise existiert auch schon eine passende Interaktion aus einer vorherigen Simulation: In diesem Fall kann sie wiederverwendet und der Entwicklungsprozeß dadurch beschleunigt werden.

6. **Planungslogiken festlegen:** Jetzt wird der Fokus auf die Agententypen verlagert: Es gilt zunächst, die Funktionsweise des Agentenprogrammes zu bestimmen. Bei reaktiven Agenten reicht es aus, jeder initiierbaren Interaktion in der Matrix (die dem Agententyp entsprechende Zeile) eine Priorität hinzuzufügen. Die Planungslogik braucht dann lediglich aus der Menge möglicher Interaktionen (Auslöser + Vorbedingung erfüllt) diejenige mit der höchsten Priorität auswählen. Es können aber auch kognitive Verfahren verwendet werden, bspw. GOAP oder hybride Ansätze.

7. **Agenten-Feinspezifikation:** Da alle Rahmenbedingungen festgelegt sind, können die Agenten nun umgesetzt werden. Hierzu müssen alle abstrakten Primitive der Interaktionen implementiert werden, zu deren Ausführung bzw. Teilnahme eine Agentenfamilie befähigt sein soll[3]. Außerdem müssen die Attribute, die von diesen Algorithmen modifiziert werden, erfaßt und umgesetzt werden. Die Sensorik der Agenten und deren Wahrnehmungsbereiche (in IODA: *Halo*), ist zu erstellen, ebenso die Initialisierungsroutinen. Auch muß die Planungslogik implementiert werden, welche vom einfachen, reaktiven Controller (siehe oben) bis zum beliebig komplexen Teilsystem reichen kann.

Entgegen dem Anschein der absoluten Abfolge obiger Phasen können diese auch vermischt durchlaufen werden – indem beispielsweise zunächst die beteiligten Aktoren und dann erst deren Interaktionen identifiziert werden. Die einzige Anforderung liegt darin, das Interaktionsdesign vor dem Entwurf der Agenten durchzuführen. Anderenfalls wäre deren klare Trennung nicht mehr gewährleistet und das Vorgehen somit nicht mehr als interaktionsorientiert zu bezeichnen.

[3] Bezogen auf obiges Beipiel: Ein Wolf muß die Quellprimitive von *Move*, *Eat* und *Procreate* sowie die Zielprimitive von *Procreate* implementieren.

2.3 *EBDI - An Architecture for emotional Agents*

Die Mehrzahl der Agenten wird nutzenbasiert entworfen. Sie verwenden ein rationales Agentenprogramm, welches sich der Lage entsprechend stets optimal verhalten soll. Für viele Anwendungsgebiete ist dies wünschenswert, im Simulationsbereich ergeben sich jedoch einige Probleme. Die dort repräsentierten Entitäten – oftmals Menschen oder Tiere – können emotionsbehaftet sein, wodurch ein immer rationales Handeln der Glaubwürdigkeit entgegenwirkt.

In *EBDI: An Architecture for Emotional Agents* [1] wird ein abstrakter Ansatz vorgestellt, wo eine generische BDI-Architektur um Emotionen angereichert wird. Hierzu findet ein Filterprinzip Anwendung: Die Emotionen überlagern die Wahrnehmung des Agenten und sind in der Lage, das Wahrgenommene verzerren, verstärken oder abschwächen zu können.

2.3.1 Ermittlung von Wissen & Emotionen

EBDI verwendet diverse Funktionen, mit denen anschließend die BDI-Routine erweitert wird. Mangels konkreter Domäne sind diese sehr abstrakt gehalten und kennzeichnen nur die logische Bedeutung und die verwendeten Daten.

- **3 *Belief Revision Functions* (BRF):** Diese Funktionen dienen zum Füllen der Wissensbasis des Agenten. $BRF_{see} : Env \mapsto B_p$ stellt die Wahrnehmung dar, welche aus der Umwelt eine Wissensteilmenge (B_p) extrahiert. Analog dazu bildet $BRF_{msg} : Cont \mapsto B_m$ das Wissen, welches aus Kommunikationen (*Content*) entsteht, ab. Die dritte Funktion, $BRF_{in} : ExIx(BuB_puB_m) \mapsto B$ repräsentiert das Nachdenken: Das neue Wissen B geht aus dem vorigen Wissen und den neuen Wahrnehmungen, in Verbindung gesetzt mit den aktuellen Intentionen und Emotionen, hervor.
- **2 *Emotion Update Functions* (EUF):** Sie ermitteln die primären und sekundären Emotionen und binden diese in den Entscheidungsprozeß ein. Bei primären Emotionen handelt es sich um unmittelbares, reflexhaftes Empfinden – sie sind eher reaktiver Natur (instinktiv) und können den Entscheidungsprozeß beschleunigen, wenn wenig Zeit zum Nachdenken besteht (in Extremsituationen). Sie berücksichtigen nur das gerade Wahrgenommene: $euf_1 : ExIx(B_puB_m) \mapsto E$. Sekundäre Emotionen hingegen sind Folgeemotionen, die entweder aus den primären Emotionen heraus entstehen (aus Angst folgt Wut) oder durch Nachdenken über die Situation gebildet werden: $euf_2 : ExIxB \mapsto E$.
- **Filterfunktion zur Beeinflussung der Absichten:** In dieser Funktion kommt der aktuelle Emotionszustand zum Tragen. Anhand des Wissens, der Ziele und der Absichten, in Verbindung gesetzt mit den Emotionen, werden die neuen Absichten gebildet: $filter : ExBxDxI \mapsto I$.

2.3.2 Basisstruktur des emotionalen Agenten

In jedem Durchlauf erfolgt zunächst die Wahrnehmungsphase (brf_{see}, brf_{msg}), aus der sich die primären Emotionen (euf_1) ableiten. Danach wird die Wissensbasis mit den neuen Beobachtungen und erhaltenen Nachrichten gefüllt (brf_{in}). Mithilfe von gewöhnlichen BDI-Funktionen werden dann die verfügbaren Optionen generiert ($options : BxI \mapsto D$) und ein Plan erstellt ($plan : IxAc \mapsto \pi$, mit $\pi =< a_1, ..., a_n >$ und $a_i \in Ac$).

Wenn jetzt noch Zeit verbleibt, werden die sekundären Emotionen (euf_2) betrachtet. Damit kann der Agent sein Wissen und seine Ziele und Absichten erneut evaluieren – andernfalls wird direkt der anliegende Plan ausgeführt. Die unterstehende Abbildung der EBDI-Routine in Pseudocode verdeutlicht den Ablauf:

EBDI-MAIN-LOOP

1 $E \leftarrow E_0$; ▷ E_0 are initial emotions
2 $B \leftarrow B_0$; ▷ B_0 are initial beliefs
3 $I \leftarrow I_0$; ▷ I_0 are initial intentions
4 **while** true
5 **do** $B_p \leftarrow$ *brf-see*(Env);
6 $B_m \leftarrow$ *brf-msg*$(Cont)$;
7 $E \leftarrow$ *euf1*$(E, I, B_p \cup B_m)$;
8 $B \leftarrow$ *brf-in*$(E, I, B \cup B_p \cup B_m)$;
9 $D \leftarrow$ *options*(B, I);
10 $I \leftarrow$ *filter*(E, B, D, I);
11 $E' \leftarrow E$
12 $E \leftarrow$ *euf2*(E, I, B);
13 **if** time permits **and** $E \neq E'$
14 **then** $B \leftarrow$ *brf-in*(E, I, B);
15 $D \leftarrow$ *options*(B, I);
16 $I \leftarrow$ *filter*(E, B, D, I);
17 $\pi \leftarrow$ *plan*(I, Ac);
18 *execute*(π)

Abbildung 3: Hauptroutine des EBDI-Agentenprogrammes

3 Fazit

3.1 Task-Unterbrechungskonzept

Das in [4] vorgestellte Konzept zur Task-Unterbrechung und Wiederaufnahme erscheint mir sehr geradlinig und intuitiv. Es basiert in erster Linie auf zusätzlichen Prüfbedingungen und Meta-Aktionen, welche rekursiv angewendet werden. Trotz dessen Einfachheit verspricht es aber einen großen Gewinn an Potential und Autonomie für die Agenten. Außerdem könnte ein Fehlen dieser Möglichkeiten andersherum betrachtet als Einschränkung aufgefaßt werden – und das soll ja vermieden werden!

Zur Implementation des Konzeptes sind im Papier die Semantiken der Verwaltungsfunktionen in *CAN*, einer generischen Sprache zur Beschreibung von Agentenprogrammen, gegeben. Dadurch wird die Umsetzung der Meta-Aktionen unterstützt. Auch dürfte die Integration in die bisherige Architektur leicht vonstattengehen, da keine tiefgreifenden Änderungen, sondern nur Ergänzungen nötig sind. Es scheint ausreichend, die Tasks mit Metadaten zu versehen und einen zusätzlichen Mechanismus zu entwerfen, welcher der eigentlichen Planungsphase vorgeschaltet wird und die De-/Reaktivierung übernimmt.

Eine mögliche Schwäche könnte in der Komplexität der Unterbrechung oder Verwerfung von übergeordneten Tasks bestehen. Ob dies immer und automatisiert möglich ist, bleibt ebenfalls fraglich. Außerdem wurden hier nur explizite Ziele (*achievement goals*) betrachtet. Inwiefern dieser Mechanismus für implizite Ziele (*maintenance goals*) nutzbar ist, muß noch betrachtet werden.

3.2 IODA

Mit IODA existiert ein eingängiges Entwurfs- und Interaktionskonzept, welches auf eine generische Entwicklung ausgelegt ist. Durch die Berücksichtigung verschiedener Rollen und mehrerer Teilnehmer ist es gut für Simulationen geeignet. Da andere Aspekte des Agenten nicht von Relevanz sind, läßt es sich einfach in eine bestehende Grundarchitektur integrieren.

Zudem paßt IODA gut zum bisherigen Entwurf, da die Aktionen bereits modular und hierarchisch geplant sind. Die Umsetzung kann leicht über eine Erweiterung der bisherigen Aktionen zu Interaktionen erfolgen, wobei noch die zusätzliche Abstraktion über Interaktionsprimitive notwendig wird. Diese Indirektionsschicht fehlte noch – hätte aber sowieso aus Gründen der Wiederverwendbarkeit ergänzt werden müssen. Zu klären bleibt die Frage, inwiefern die sequentielle Ausführung auf Parallelisierung erweitert werden kann und wie dann die Synchronisation der Interaktionen geregelt wird.

Eine kritische Betrachtung bei IODA ist bezüglich der Fairness notwendig: Ein Agent kann in einem Zeitintervall unter Umständen in allen Interaktionen beteiligt sein und auch selbst noch handeln. Hier müsste eine Einschränkung geschehen, beispielsweise durch Hinzufügen einer Ausführungszeit. Wenn ein Agent seine logische Zeit aufgebraucht hat, kann er keine weiteren Handlungen vornehmen. Hier könnten Prioritäten und Interaktionsarten von Relevanz werden.

3.3 EBDI

Die Autoren des EBDI-Papieres versprechen ein generisches, nicht einschränkendes Konzept, welches die Integration von Emotionen in eine gewöhnliche BDI-Architektur gestattet. So soll es alle gängigen Emotionskonzepte unterstützen und für beliebige Einsatzzwecke geeignet sein.

Aufgrund der sehr abstrakten Darstellung sehe ich es allerdings als zu wage an, um von Nutzen zu sein. Es werden weder Aussagen über die Repräsentation von Emotionen getroffen, noch sind deren Wirkungen spezifiziert. Lediglich eine allgemeine Funktionsabfolge und die darin beteiligten Daten sind in Form von Kreuzprodukten gegeben. Die Grundlage für eine nutzbringende Implementation fehlt somit.

Außerdem sieht dieses Konzept keine Erweiterung eines bestehenden Systemes vor, sondern muß in der kompletten Planungsphase verankert werden. Der ursprüngliche Wunsch eines dazuschaltbaren Emotionsmoduls ist so nicht realisierbar, stattdessen müßte es immer vorhanden sein.

Für die Verwendung in einer generischen Architektur ist eine eigene Darstellungsform zu finden, analog zu den abstrakten Interaktionen in IODA. Hierfür ist aber eine eingehende Untersuchung der beteiligten Faktoren und deren Einflüsse aufeinander nötig. Dies scheint mir ein sehr komplexes Teilgebiet, welches stark von der Art der simulierten Entitäten abhängt und psychologisches Wissen erfordert. Aus diesen Gründen werde ich die Integration von Emotionen bis auf weiteres zurückstellen.

4 Weiteres Vorgehen

4.1 Aktueller Stand

Nach einem ersten, groben Architekturentwurf[4] ist inzwischen eine prototypische Implementation erfolgt, welche allerdings zum gegenwärtigen Zeitpunkt (Juli 2014) noch nicht in MARS integriert ist. Sie diente zu ersten Testzwecken (Umsetzung eines einfachen Räuber-Beute-Systems, Aufbau der Vererbungsstrukturen, Interaktion zwischen Umwelt und Agenten, ...) und soll in leicht abgewandelter Form weiterentwickelt werden.

Als Basis für alle Agenten dient eine abstrakte Klasse, welche das sequentielle Agentenprogramm in dessen Grundzügen beherbergt. Es wird eine Unterteilung in die Verantwortlichkeiten Wahrnehmung, Planung und Handlung vorgenommen; der abstrakte Agent operiert auf ebenfalls abstrakten Oberklassen dieser Komponenten, welche dann domänenspezifisch ausgeprägt werden können.

Die nebenstehende Abbildung veranschaulicht die Pipeline sowie die bisherige Idee der Umschaltung von aktiver und passiver Planung – basierend auf der Idee einer horizontalen Schichtenarchitektur des Agentenprogrammes. Dieser Ansatz kann künftig durch Integration des IODA-Prinzips vereinfacht werden.

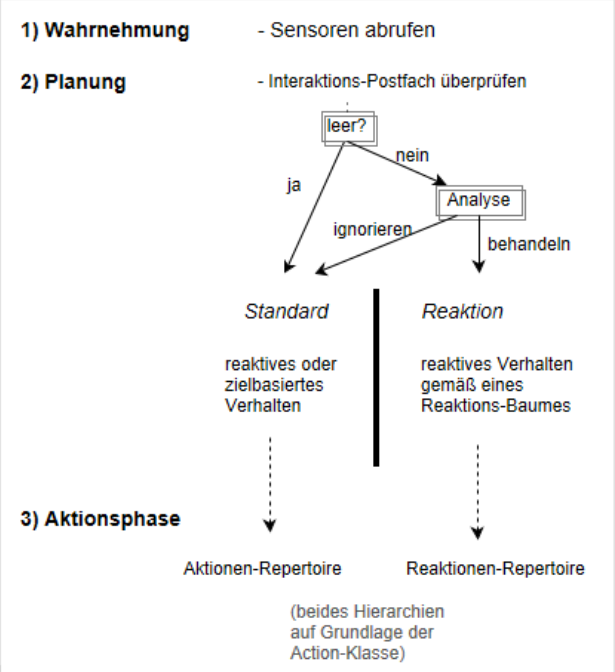

Abbildung 4: Das vorläufige Agentenprogramm

4.2 Planung & Ausblick

Das Primärziel ist zunächst, eine lauffähige Basisarchitektur mit rudimentärer Funktionalität bereitzustellen, die dann später verfeinert werden kann. Hierzu muß der oben erwähnte Prototyp an die Ausführungs- und Umweltschnittstellen von MARS angepaßt werden. Dann soll das IODA-Interaktionskonzept Einzug finden, wobei zunächst einfache, reaktive Agenten verwendet werden. Im Anschluß kann eine GOAP-Erweiterung erfolgen, wobei möglicherweise auf ein fertiges Modul zugegriffen werden kann.

Später läßt sich das in 2.1 präsentierte Konzept der dynamischen Zielauswahl ergänzen. Außerdem sollen die Agenten mit einer Konfigurationsschnittstelle zu Initialisierungs- und Änderungszwecken ausgestattet werden. Der Emotionsaspekt scheint hingegen zu komplex, um eine zufriedenstellende Abstraktion zu erstellen – stattdessen wäre es nutzbringender, freie Ressourcen auf die Komposition von Bibliotheken zu verwenden. So könnten Hierarchien von (Inter-)Aktionen, Agentenprogrammen, Sensoren usw. bereitgestellt werden, um Standardimplementationen zu bieten und den Entwicklungsprozeß zu beschleunigen.

[4] Ein Überblick über die Entwicklung ist einzusehen im *Confluence*, Agentenbereich.

Literatur

[1] Hong Jiang, José M. Vidal, and Michael N. Huhns. Ebdi: An architecture for emotional agents, 2007.

[2] Yoann Kubera, Philippe Mathieu, and Sébastien Picault. IODA: An interaction-oriented approach for Multi-Agent Based Simulations. *Journal of Autonomous Agents and Multi-Agent Systems*, 23(3):303–343, 2011.

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

[4] John Thangarajah, James Harland, David Morley, and Neil Yorke-Smith. Suspending and resuming tasks in bdi agents. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '08, pages 405–412, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems.

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