

BDI-Architekturen in Theorie und Praxis

Seminar Artificial Life
Sommersemester 2000

Ernst Bachmann¹, Manuel Klimek²

¹email: bachmann@fmi.uni-passau.de

²email: klimek@fmi.uni-passau.de

Inhaltsverzeichnis

1	Was sind BDI-Agenten	3
1.1	Überblick	3
1.2	Zusammensetzung	3
1.2.1	Belief	3
1.2.2	Desire	3
1.2.3	Intention	4
1.3	Decision Trees	4
1.3.1	Was sind Decision Trees?	4
1.3.2	Notwendige Modifikationen	6
1.3.3	Belief-accessible worlds	6
1.3.4	Desire-accessible worlds	8
1.3.5	Intention-accessible worlds	8
1.4	Dynamic Constraints	9
1.5	Reaktionsbeschreibung	9
2	Erläuterung einer Implementierung des Modells	11
2.1	Modellierung	11
2.1.1	Umgebung -> Belief	11
2.1.2	Ziele -> Desire	11
2.1.3	Strategie -> Intention	11
3	Anwendungsbeispiele	12
3.1	OASIS (Airport management system)	12
3.2	Air-Combat Modeling	13
4	Schlusswort	13

1 Was sind BDI-Agenten

1.1 Überblick

Mit der massiven Einführung von E-Commerce im Internet gewinnen Software-Agenten zunehmend an ökonomischer Bedeutung. In ihrer einfachsten Form nehmen sie Bestellungen entgegen und informieren den Kunden per E-Mail über den Stand der Bearbeitung. Die Industrie fordert jedoch schon jetzt immer mächtigere Systeme, mit dem Ziel, auch komplexere Arbeiten von Maschinen verrichten zu lassen. Um aber den gestiegenen Anforderungen gerecht zu werden, muss ein flexibles Design gewählt werden, welches in der Lage ist, die Erkenntnisse der Forschung in klar strukturierter Form an den Anwendungsentwickler weiterzugeben. Ein von vielen Stellen propagiertes Agenten-Design ist **BDI**. Diese Arbeit bietet einen Überblick über die einzelnen Komponenten des Modells und seine Vor- bzw. Nachteile. Zunächst wird auf die aus dem Namen hervorgehende Gliederung in drei Komponenten (**B**elief, **D**esire, **I**ntention) genauer eingegangen und später die Realisierung anhand einer konkreten Implementierung besprochen.

1.2 Zusammensetzung

Die erste Frage, wenn man sich ein neues Modell betrachtet, lautet meist: "Warum so und nicht anders?". In diesem Fall wurde die Kritik zunächst von zwei Seiten laut. Die Verfechter der klassischen Entscheidungstheorie stellten in Frage, ob wirklich alle drei Komponenten notwendig seien. Auf der anderen Seite vertraten Soziologen die Meinung, dass nur drei Komponenten wohl kaum die gesamte Bandbreite der Fragestellung abdecken könnten. Sehen wir uns nun die einzelnen Komponenten in Bezug auf diese Kritik etwas genauer an.

1.2.1 Belief

Mit den Beliefs bezeichnen wir alles Wissen, das der Agent über seine Umwelt besitzt. Da die Daten der Umwelt sich meist sehr oft ändern und keineswegs immer digital vorliegen, sollen die Beliefs im Gegensatz zur klassischen Modellierung die Umwelt nicht exakt modellieren, sondern die im Augenblick im Agenten vorherrschende **Vorstellung** von der Umgebung beschreiben. Die Beliefs repräsentieren also nur mit einer gewissen Wahrscheinlichkeit die wirklichen Umgebungsdaten. Diese Unschärfe muss bei der Entscheidungsfindung berücksichtigt werden.

Da im Allgemeinen eine einmalige Messung nicht den Gesamtzustand des Systems erfassen kann, ein Entscheidungsalgorithmus aber auf einer möglichst umfassenden Beschreibung der Umgebung arbeiten sollte, müssen die Umgebungsdaten über einen längeren Zeitraum gespeichert werden. Dies zeigt, dass Beliefs zur effizienten Implementierung eines komplexeren Agenten auf jeden Fall notwendig sind.

1.2.2 Desire

Die Desires eines Agenten werden durch eine Menge von Zuständen der Umgebung kodiert, die der Agent erreichen soll. Um einen dieser Zustände zu erreichen interagiert der Agent über definierte Schnittstellen mit der Umgebung. Da einige der erwünschten Zustände konkurrierende Ziele

darstellen, d.h. eine ausgewählte Interaktion eines der Ziele erreichen und gleichzeitig das Erreichen eines anderen Zieles verhindern kann, muss der Agent die Ziele an Hand von Prioritäten gewichten.

Natürlich muss jeder zielgerichtet arbeitende Agent über Informationen verfügen, welche Situationen für ihn günstig erscheinen (eine Payoff-Funktion). Implizit stellt diese Payoff Funktion aber genau die Desires des Agenten dar.

1.2.3 Intention

Eine Folge von Interaktionen mit der Umwelt, die einen erwünschten Zustand herbeiführen soll, nennt man einen Plan. Die Menge der aktiven Pläne nennt man die **Intentions** des Agenten. Aktive Pläne dürfen nicht gegenläufig sein und werden vom Agenten gleichzeitig verfolgt, bis eine sogenannte termination-rule eintritt (siehe 1.4).

Intentions werden benötigt, sobald die Entscheidungsfunktion nicht-trivial ist und damit ein nicht zu vernachlässigendes Zeitintervall zur Ausführung beansprucht. Selbst wenn es problemlos möglich ist, den Anforderungen der Umgebung zu genügen, wenn man iteriert die Entscheidungsfunktion berechnet und interagiert, kann es vorkommen, dass Ereignisse während der Ausführung der Entscheidungsfunktion eintreten. Es ist nun suboptimal, den Event einfach zu ignorieren, und zeitlich nicht tragbar, die Entscheidungsfunktion neu zu starten, da dann ja wieder ein Event eintreffen könnte. Dieses Problem kann man mit Intentions lösen, die sich durch termination rules (siehe 1.4) optimal an die Erfordernisse der aktuellen Situation anpassen lassen.

1.3 Decision Trees

1.3.1 Was sind Decision Trees?

Decision Trees sind ein Ansatz, um mögliche Entwicklungen und Zustände einer "Welt" abzubilden und algorithmisch betrachten zu können. Decision Trees kommen ursprünglich aus der klassischen Entscheidungstheorie, lassen sich aber auch zum Spezifizieren und Implementieren von BDI-Agenten verwenden.

Ein Decision Tree besteht aus drei verschiedenen Knotenarten, wobei die Wurzel des Baumes den Anfangszustand der Umgebung darstellt (siehe Abbildung 1).

- **"Decision nodes"**

Diese Knoten modellieren eine Entscheidung des Systems. Für jede mögliche Aktion, die zu diesem Zeitpunkt durchgeführt werden kann, existiert ein Zweig zu einem weiteren Teilbaum, in dem davon ausgegangen wird, dass die entsprechende Aktion ausgeführt wurde.

- **"Chance nodes"**

Mit den Chance nodes werden mögliche Ereignisse in der Umgebung dargestellt. Für jedes mögliche Umweltereignis existiert ein Unterbaum, in dem angenommen wird, dass dieses Ereignis eingetreten ist. Zu jedem Chance node gehört eine Zufallsvariable, die angibt, wie wahrscheinlich jeder Unterbaum ist. Da es in komplexen analogen Umgebungen unmöglich ist, alle Varianten der Entwicklung zu betrachten, muss hier eine starke Diskretisierung der Umgebung vorgenommen werden.

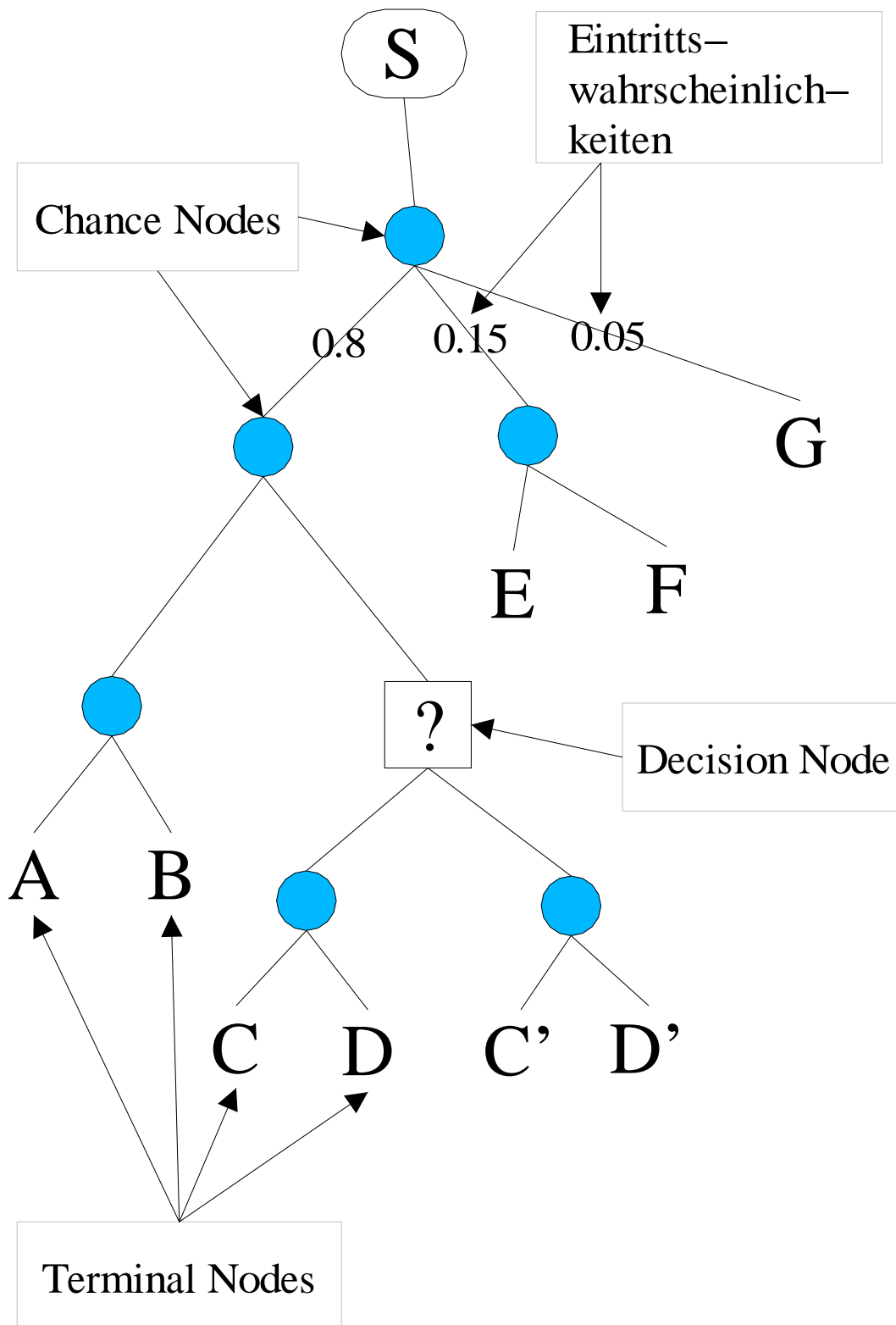


Abbildung 1: decision trees

- **"Terminal nodes"**

Die Terminal nodes sind die Blätter des Entscheidungsbaumes. Zu ihnen gehört jeweils eine Gewichtung, die angibt, wie erstrebenswert das erreichte Szenario ist (Payoff Funktion).

Eine Entscheidungsfunktion kann nun über den Baum laufen und die Folge von Aktionen auswählen, die z.B. den größten erwarteten Gewinn abwirft.

1.3.2 Notwendige Modifikationen

Dieses Verfahren lässt sich natürlich nur anwenden, wenn man vollständiges Wissen über die Umwelt hat und weiß, welche Ereignisse überhaupt auftreten können.

Mit einigen Änderungen lässt sich das Modell aber auch in Situationen einsetzen, in denen nur unzureichendes Wissen über Wahrscheinlichkeiten vorhanden ist und sich das Problem dynamisch ändern kann. Zu diesem Zweck wird der Baum so modifiziert, dass Belief, Desire und Intention unabhängig voneinander dargestellt werden können.

1.3.3 Belief-accessible worlds

Diese Modifikation liefert eine Menge von allen Bäumen, die sich durch die zufällige Entwicklung der Umwelt ergeben können. Diese Bäume enthalten nur decision nodes und terminal nodes.

Der Algorithmus benötigt eine Liste von Bäumen, in der sich zu Beginn nur der vollständigen Decision Tree befindet, der jeden möglichen Pfad und damit auch Endzustand enthält und einen Zeiger auf den aktuell zu bearbeitenden Baum in der Liste.

Der aktuelle Baum wird durchlaufen, bis man auf eine chance node stößt. Dann wird für jedes Kind eine Kopie des Baumes angelegt, in der das Kind direkt mit dem Vater der aktuellen node verbunden wird, d.h. die chance node entfernt wird. Jeder so entstandene Baum wird an die globale Baumliste angehängt. Danach wird der aktuelle Durchlauf durch den Baum abgebrochen und der nächste Baum der globalen Liste wird zum aktuellen Baum. Dieser Algorithmus wird solange wiederholt, wie es noch einen nächsten Baum in der Liste gibt.

Zur Verdeutlichung werden nun die belief accessible worlds anhand eines einfachen Beispiels entwickelt. Zunächst enthält die Liste nur den zu modifizierenden kompletten decision tree. Nun wird dieser Baum rekursiv durchlaufen, um die erste chance node zu finden (Abbildung 2).

- **1. Schritt**

Die aktuelle chance node wird gelöscht und für jedes der zwei Kind-Knoten ein Baum erzeugt, bei dem der Kind-Knoten als Vater den Vater des aktuellen Knotens erhält. In diesem Beispiel wird jeder der Kind-Knoten zur Wurzel. Die zwei neuen Bäume werden ans Ende der Liste angefügt und der aktuelle Baum gelöscht.

Jetzt wird in der Liste der Reihe nach jeder Baum durchlaufen, bis man wieder auf eine chance node stößt. (Abbildung 3).

- **2. Schritt**

Wie auch im 1. Schritt wird die aktuelle chance node gelöscht und für die beiden Kind-Knoten ein Baum angelegt, bei dem der jeweilige Kind-Knoten direkt mit dem Vater der gelöschten chance node (hier die decision node) verbunden ist. Wieder werden die zwei neuen Bäume an die Liste angehängt und der aktuelle Baum entfernt.

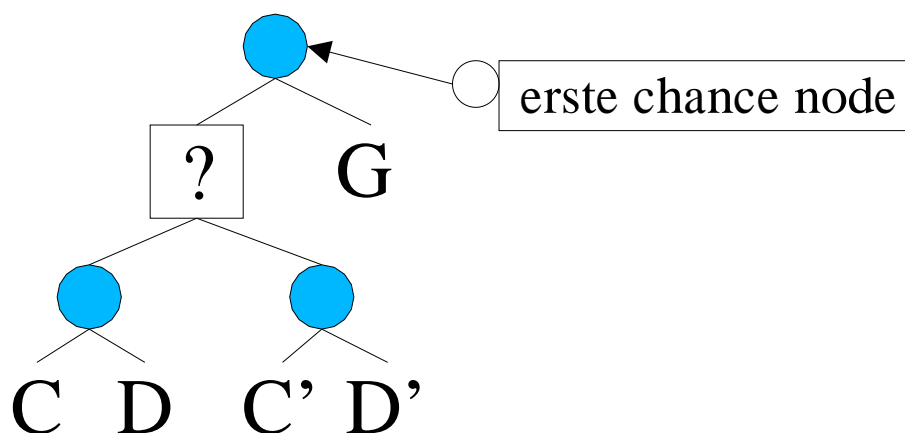


Abbildung 2: Anfangszustand

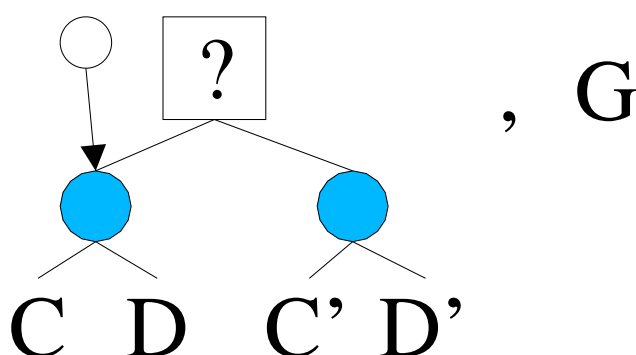


Abbildung 3: Nach dem 1. Schritt

Wie in Schritt 1 wird in der Liste der Reihe nach jeder Baum durchlaufen, bis man auf eine chance node stößt. Der erste Baum besteht nur aus der terminal node G , enthält also keine chance node. Im zweiten Baum werden wir wieder fündig (Abbildung 4).

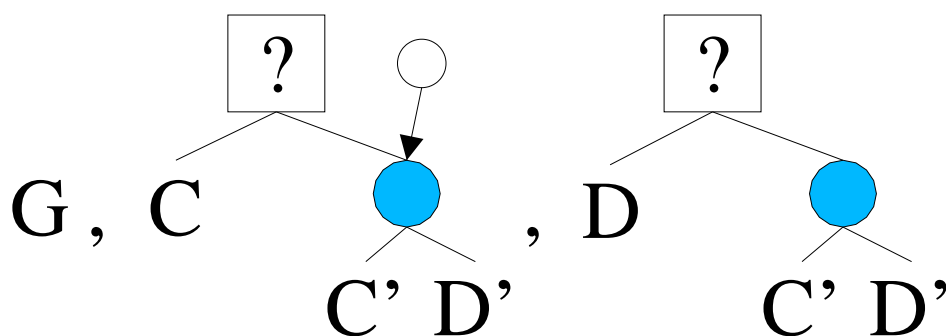


Abbildung 4: Nach dem 2. Schritt

- **3. Schritt**

Wir gehen wieder analog zu Schritt 1 und 2 vor. In Abbildung 5 sieht man, wie die Liste am

Ende dieses Durchlaufes beschaffen ist.

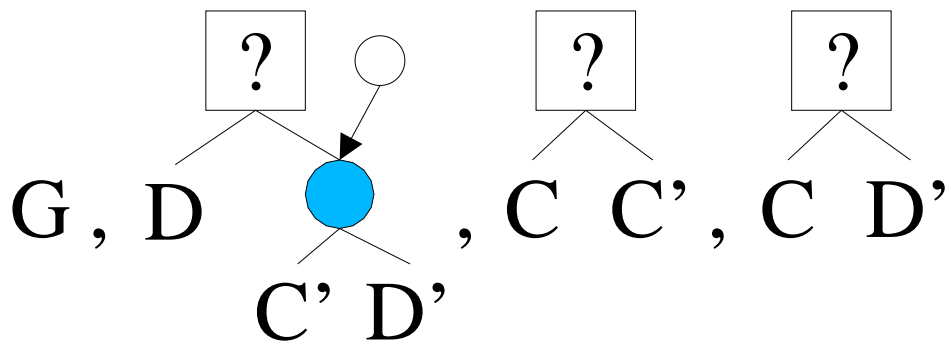


Abbildung 5: Nach dem 3. Schritt

- **Letzter Schritt**

Der Anfang erfolgt wie in den vorangegangenen Schritten. Diesmal gibt es jedoch am Ende des Laufes keine chance node mehr in der gesamten Liste. Somit sind die belief accessible worlds vollständig erzeugt. (Abbildung 6).

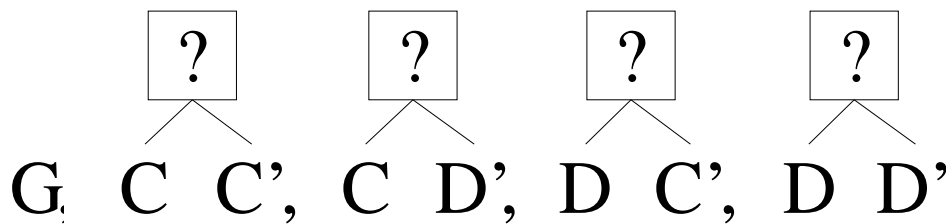


Abbildung 6: Endzustand

1.3.4 Desire-accessible worlds

Die Menge der "Desire accessible worlds" ist prinzipiell die selbe wie die Menge der "Belief accessible worlds", mit dem Unterschied, dass hier nicht die Wahrscheinlichkeit für das Auftreten einer bestimmten Situation von Interesse ist, sondern der jeweils erwartete Gewinn. Aus der Menge können alle Welten entfernt werden, die keinen Gewinn abwerfen, nun sind nur noch die Welten enthalten, die der Agent herbeiführen will, deswegen auch der Name "Desire accessible worlds".

1.3.5 Intention-accessible worlds

Aus dem Entscheidungsbaum kann der Agent mit der oben angesprochenen Entscheidungsfunktion für jede Situation die beste Abfolge von Aktionen auswählen. Diese beschreibt in jedem Baum der desire-accessible worlds genau einen Pfad, von welchem man per Payoff-Funktion den maximalen erwarteten Gewinn errechnet hat. Diesen Pfad speichert man als Baum, der nur noch einen Ast mit den gewählten Aktionen enthält. Im Endeffekt sollte für jede desire accessible world eine zugehörige intention accessible world existieren, die das optimale Verhalten in dieser Situation beschreibt.

1.4 Dynamic Constraints

Ein wichtiger Aspekt der BDI Architektur ist die Idee, eine einmal gefällte Entscheidung beizubehalten (= "commitment"). "Entscheidung" bedeutet hier das Festlegen einer intention zum Erreichen eines gewissen Zustandes. Der Agent kann nur seine intentions ändern, da er auf seine beliefs und desires keinen direkten Einfluss hat (die beliefs können nur indirekt über das Ausführen der Aktionen von intentionierten Pfaden beeinflusst werden.)

Das commitment stellt nun den Ausgleich zwischen zielgerichtetem Handeln und schnellen Reaktionen auf geänderte Umweltbedingungen dar. Ein commitment besteht üblicherweise aus zwei Teilen: die Situation, die der Agent beibehalten oder herbeiführen will und die Bedingungen, unter denen er bereit ist, seine Verpflichtung aufzugeben (= "termination condition"). In der klassischen Agententheorie wurde propagiert, die Situation ständig neu zu erfassen und in jedem Schritt die optimale auszuführende Aktion neu zu bestimmen. Mit der Einführung von Agentensystemen, die mit sehr komplexen Umgebungen interagieren und langwierige Optimierungsprobleme lösen müssen, um die beste Handlungsweise zu finden, hat sich dieser Ansatz jedoch als zu kostspielig erwiesen. Das andere Extrem wäre, den Agenten eine vom Programmierer vorgegebene Strategie verfolgen zu lassen. Dies ist jedoch nicht flexibel genug um sich an eine Vielfalt von Änderungen in der Umgebung anzupassen. BDI versucht nun den Mittelweg zu finden. Zu diesem Zweck wird vom Agenten anhand einer (komplexen) Optimierungsfunktion der optimale erreichbare Endzustand ermittelt. Der Pfad dorthin wird über die intention-accessible worlds gespeichert. Des weiteren erstellt der Agent eine (kleine) Menge von termination conditions, die festlegen, wann der soeben gewählte Plan nicht mehr durchführbar ist und ein neuer Optimierungsvorgang gestartet werden muß. Jetzt hat der Agent nur noch eine relativ kleine Menge von Bedingungen "ständig" zu überprüfen.

Damit ist der Agent beliebig an die Erfordernisse des Einsatzgebietes anpassbar. Denn mit verschiedenen Terminierungsbedingungen lassen sich sehr unterschiedliche Verhaltensweisen erzeugen: zum Beispiel könnte man sich einen *blindly-commited* Agent vorstellen, der jede Änderung an seinen Beliefs und Desires verbietet, wenn sie mit seinen Verpflichtungen kollidiert. Dieser Agent verhält sich, nachdem er die optimale Strategie einmal errechnet hat, komplett unflexibel. Ein *single-minded* Agent würde Änderungen an seinen Beliefs, ein *open-minded* Agent auch an seinen Desires zulassen, und bei Konflikten die entsprechende Verpflichtung fallen lassen. Man kann natürlich auch soweit gehen und einen Agenten bauen, der eine beliebige Änderung der Umwelt als termination condition betrachtet. Dieser Agent würde dann (eine sich stetige ändernde Umwelt vorausgesetzt) wieder versuchen ständig die optimale Handlungsweise zu errechnen. Im Normalfall wird der Agent seine termination conditions irgendwo zwischen den beiden Extremen ansiedeln.

1.5 Reaktionsbeschreibung

Die Reaktion des Agenten auf externe events kann nach [2, Seite 150] wie folgt als Pseudo-Code dargestellt werden:

```
initialize-state()  
repeat  
  options := option-generator(event-queue);
```

```
selected-options := deliberate(options);
update-intentions(selected-options);
execute();
get-new-external-events();
drop-successful-attitudes();
drop-impossible-attitudes();
end repeat
```

Zunächst wird eine Reihe von möglichen Plänen erzeugt (options). Danach ermittelt die Entscheidungsfunktion anhand aller möglichen options diejenigen, welche ausgeführt werden sollen. Diese werden mit den vorhandenen intentions verknüpft. Eine eventuell vorhandene auszuführende Aktion wird nun gestartet. Danach werden alle externen events erfasst und in eine interne Event-Liste eingefügt. Schon erreichte intentions und nicht mehr erreichbare desires werden dann aus den entsprechenden Strukturen entfernt.

Diese grundlegende Algorithmisierung lässt dem Programmierer noch jede Menge Freiheiten im Design. Wie diese Strukturen implementiert werden, hängt natürlich hauptsächlich vom Anwendungsgebiet des Agenten ab. Einige Vorschläge erhält man, wenn man sich bereits existierende Implementierungen von BDI-Agenten näher betrachtet.

2 Erläuterung einer Implementierung des Modells

Die oben eingeführten Begriffe wollen wir nun anhand eines kleinen Beispiels präzisieren: OASIS (siehe 3.1) enthält unter anderem einen Aircraft Agent. Dieser modelliert ein spezielles Flugzeug. Der Agent ist eingebettet in das OASIS System und muss mit den dort bereitgestellten Komponenten zusammenarbeiten.

Der Aircraft Agent kann mit seiner Umgebung interagieren, indem er die Flughöhe bzw. die Geschwindigkeit variiert.

2.1 Modellierung

2.1.1 Umgebung -> Belief

Die Umgebung des Flugzeugs besteht im Wesentlichen aus den Winddaten und der ETA, die von globalen OASIS-Komponenten bereitgestellt werden. Im Entscheidungsbaum repräsentiert jede Änderung der ETA bzw. der Windverhältnisse eine chance node. Jede mögliche Ankunftszeit wird durch eine terminal node repräsentiert, jede mögliche Änderung der Geschwindigkeit bzw. der Flughöhe durch decision nodes.

2.1.2 Ziele -> Desire

Das Flugzeug verfolgt zwei zum Teil gegensätzliche Ziele:

- Einhaltung des vorhergesehenen Ankunftsplans
- Minimierung des Treibstoffverbrauchs durch Regulierung von Geschwindigkeit und Flughöhe

Die desire-accessible worlds stellen hierbei die belief-accessible worlds ohne diejenigen Zweige dar, bei denen die Ankunftszeit nicht mit der vom System vorgegebenen ETA übereinstimmen.

2.1.3 Strategie -> Intention

Aus den desire-accessible worlds wird nun nur der Zweig mit dem minimalen Treibstoffverbrauch ausgewählt. Dazu wird jeder mögliche Zweig per Payoff Funktion (= Treibstoffverbrauch) bewertet.

3 Anwendungsbeispiele

3.1 OASIS (Airport management system)

Dies ist das Gesamtsystem, das unter anderem aus dem eben genauer erläuterten Aircraft Agent besteht. Zur Umgebung dieses Systems gehören u.a. die Wetter- und Windbedingungen, Zustände der Start und Landebahnen und die Anwesenheit von anderen Flugzeugen. Die meisten dieser Bedingungen unterliegen einer ständigen, nicht vorhersehbaren Wandlung, das System muss sich also schnell an veränderte Rahmenbedingungen anpassen können. Hinzu kommt noch, dass der Zustand der Umgebung nie vollständig bekannt ist, sondern nur stückchenweise in Erfahrung gebracht werden kann. Zum Beispiel erhält das System Wetterdaten per Funk von Flugzeugen, d.h. unregelmäßig und lokal begrenzt. Da sämtliche Änderungen asynchron passieren können, kann es sein, dass sich die Bedingungen auch während eines Berechnungslaufes ändern.

Für jedes Flugzeug wird ein *Aircraft Agent* angelegt, der eine vorgegebene Menge an Wegpunkten abfliegt. Dabei sind die Windverhältnisse auf dem Weg im Allgemeinen unbekannt, so dass für jede mögliche Windrichtung und Geschwindigkeit eine *belief-accessible world* erzeugt würde.

Der Agent hat dabei die Wahl, wie er die Wegpunkte erreicht: er kann die Geschwindigkeit und Flughöhe variieren, solange er den letzten Wegpunkt, den Flughafen, zur vorgegebenen Ankunftszeit erreicht.

Diese Wahlmöglichkeiten lassen sich als einzelne Äste in den oben angesprochenen Belief-accessible worlds darstellen.

Des Weiteren gibt es noch eine Reihe von globalen Agenten:

- Der **wind modeller** hat die Aufgabe eintreffende Informationen über die Wetterverhältnisse zu sammeln und zu einem grossen Gesamtbild zusammenzufügen.
- Der **trajectory checker** prüft die Flugbahnen der einzelnen aircraft agents auf Überschneidungen.
- **Coordinator** koordiniert den Informationsfluss zwischen den globalen Komponenten und den Aircraft Agents.
- Der **Sequencer** hat die Hauptaufgabe alle Flugzeuge sicher und in einer optimalen Reihenfolge zu landen. Als Eingabeparameter erhält er die Leistungsdaten eines Flugzeugs, den gewünschten Abstand zwischen zwei Fliegern, die Landebahnzuordnung, die Windverhältnisse und eine Kostenfunktion. Er wählt nun die beste Ankunftsreihenfolge aus und berechnet die Ankunftszeitpunkte der einzelnen Flieger. Sobald er sich für eine bestimmte Reihenfolge entschieden hat, verfolgt er diese *single-minded*, d.h. er verwirft sie nur, wenn entweder alle Flugzeuge gelandet sind oder er nicht mehr glaubt, dass das nächste Flugzeug seinen vorgesehenen Ankunftszeitpunkt noch einhalten kann.

In der klassischen Entscheidungstheorie hingegen würde man nach jeder Änderung der Umwelt die optimale Reihenfolge erneut berechnen, was hier allerdings zuviel Rechenzeit verbrauchen würde und für Bodenpersonal und Piloten eine enorme Belastung bedeuten würde, sollte sich der Plan zu häufig ändern.

Das OASIS System wurde 1995 erfolgreich am Flughafen Sydney getestet.

3.2 Air-Combat Modeling

SWARMM ist ein System zur Modellierung von Luftkämpfen. Es stellt Pilot-Agenten bereit, die dann z.B. in Simulationen als Sparring Partner für Menschliche Gegner dienen. Das System modelliert sensorisches Equipment und physikalische Daten des Flugzeuges, aber auch das taktische Wissen der Piloten.

Dazu ist das Gesamtsystem in drei Komponenten aufgeteilt:

- Ein 3D Grafik Subsystem, welches die Umgebung modelliert.
- Ein Subsystem zur physikalischen Modellierung der Flugzeuge, welches spezielle Modelle für die verschiedenen Arten von Flugzeugen, Waffen, Sensoren usw. besitzt.
- Ein Subsystem zur Modellierung des Piloten. Dieses System läuft auf dMARS und modelliert sowohl taktische Überlegungen als auch koordinierendes und synchronisierendes Verhalten des Piloten.

Das System ist auf gute Skalierbarkeit optimiert und wird von der Royal Australian Air Force zur Analyse von taktischen Möglichkeiten und Leistungsdaten eingesetzt, wobei bis zu 64 Piloten gleichzeitig modelliert werden.

4 Schlusswort

BDI ist mittlerweile eine anerkanntes Design für Software Agenten. Es ist offensichtlich, dass BDI nur für entsprechend komplexe Systeme mit hohen Anforderungen sinnvoll einsetzbar ist, da kleinere Aufgaben oft wesentlich einfacher zu implementieren sind.

Ein oft angebrachter Kritikpunkt ist, dass BDI keine Unterstützung für Lernverhalten und Multi-Agenten Kooperation auf Design-Ebene bietet. Georgeff meint dazu in [1, Seite 2ff], dass BDI auch auf diesem Level überhaupt keine Aussage machen will, sondern nur eine gemeinsame Basis für eine Vielzahl von Ansätzen bieten soll, damit die Grundlagen nicht ständig neu erarbeitet werden müssen. Zum Beispiel können spezielle Lern-Algorithmen problemlos in die Entscheidungsfunktion übernommen werden, während Multi-Agenten-Kommunikation an verschiedenen Stellen eingebaut werden kann, vom Prinzip her aber der BDI-Struktur in keiner Weise widerspricht. Wichtig ist wohl auch, dass dieses Design durch bereits bestehende Implementierungshilfen unterstützt wird und sich somit wahrscheinlich auch in Zukunft einer breiten Anhängerschaft erfreuen wird.

Literatur

- [1] M. Georgeff et al. *The Belief-Desire-Intention Model of Agency*. Springer, Berlin, 1998. in: Müller, J. P., Singh, M. P., Rao, A. S. (eds.): *Intelligent Agents V. Lecture Notes in Artificial Intelligence* 1555.
- [2] Rao A. Georgeff M. *Rational Software Agents: From Theory to Practice*. Springer, Berlin, 1998. in: Jennings, N. R., Wooldridge, M. J.: *Agent Technology - Foundations, Applications, and Markets*.