

Seminar zum Thema „Künstliche Intelligenz“ – Seminararbeit

Software-Agenten

Autor: Kay Hasselbach · mi5482

Inhalt:

1. Einführung	3
1.1. Definition Agent	3
1.2. Wozu braucht man Agenten?	3
1.3. Beispiele zur Motivation	3
2. Grundlagen	5
2.1. Umwelt des Agenten	6
2.2. Unterschiedliche Betrachtungsweisen	6
2.3. Rationalität	6
2.4. Parallelität und Nebenläufigkeit	7
3. Agenten-Strukturen	8
3.1. Komponenten eines (Software-)Agenten	8
3.1.1. Umweltkopplung	8
3.1.2. Umweltmodell	8
3.1.3. Steuerung und Management	8
3.1.4. Fähigkeiten	8
3.1.5. Entscheidungskomponente	8
3.2. Einfacher Reflex-Agent	9
3.3. Modelbasierter Reflex-Agent	10
3.4. Zielbasierter Agent	11
3.5. Nutzenbasierter Agent	12
3.6. Stimulus Response oder andauernde Tätigkeit	12
3.7. Lernende Agenten	12
4. Architekturen	13
4.1. BDI-Architektur (Belief – Desire – Intention)	13
4.2. Schichtenmodell	14
4.2.1. Horizontale Architektur	15
4.2.2. Vertikale 1-Paß-Architektur	15
4.2.3. Vertikale 2-Paß-Architektur	16
4.3. Hybride Architekturen	16
5. Beispiel: „RoboCup“	17
5.1. Was ist RoboCup?	17
5.2. Die Simulationsliga	18
5.3. Das AT Humboldt	19
6. Quellen	20

1. Einführung

1.1. Definition Agent

Aufgrund der sehr unterschiedlichen Anforderungen, die an Agenten gestellt werden können, sowie der äußerst großen Anzahl an Szenarien, in denen der Einsatz von Agenten denkbar ist, ist es schwierig eine allumfassende, alle denkbaren Aspekte berücksichtigende, Definition zu geben. Daher werde ich mich hier auf eine allgemeine Arbeitsdefinition¹ beschränken, die den meisten Situationen gerecht wird.

Definition: *Ein Software-Agent ist ein längerfristig arbeitendes Programm, dessen Arbeit als eigenständiges Erledigen von Aufträgen oder Verfolgen von Zielen in Interaktion mit einer Umwelt beschrieben werden kann.*

Um ein Programm als Agent bezeichnen zu können, muss es also autonom arbeiten und mit einer Umwelt interagieren, sowie Aufträge entgegennehmen oder Ziele verfolgen können. Ein weiteres Kriterium ist die andauernde Verfügbarkeit (längerfristige Arbeit).

1.2. Wozu braucht man Agenten?

Die Frage könnte auch lauten: „Zur Lösung welcher Probleme braucht man Agenten?“. Es handelt sich hierbei also um eine Konkretisierung, bzw. Anwendung, der in 1.1. vorgenommenen, allgemeinen Definition. Aspekte, wie das eigenständige Verfolgen von Zielen (bzw. erledigen von Aufträgen) und die Interaktion mit einer Umwelt zeigen, worin sich Agenten von klassischer Software, die den Anwender nur beim Erfüllen seiner Aufgaben hilft, unterscheiden. Ein Agent hilft nicht beim Erledigen von Aufgaben, er erledigt die Aufgaben. Dies wird im Folgenden an einigen Beispielen verdeutlicht.

1.3. Beispiele zur Motivation

Es folgen drei Beispiele zur Motivation. Sie verdeutlichen außerdem, wie unterschiedlich mögliche Einsatzgebiete für Agenten sein können.

Klassifikation von Teilen:

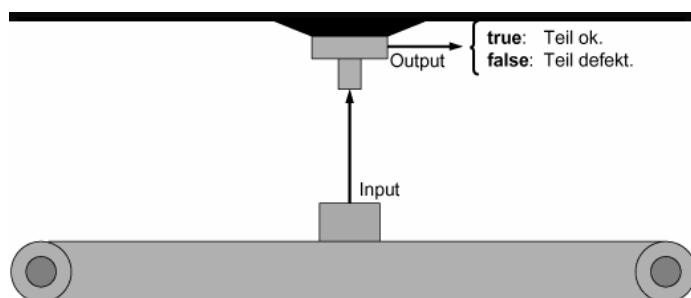


Abbildung 1.3.1.

¹ Vgl. Handbuch der künstlichen Intelligenz, Kap. 24.2.1

Hierbei handelt es sich um ein Beispiel für den einfachsten denkbaren Agenten. Seine Aufgabe ist es, Teile (z.B. in der Produktionswirtschaft), die auf einem Laufband durch sein Sichtfeld laufen, in jeweils eine von zwei möglichen Kategorien einzuordnen: Entweder ist das Teil intakt, oder es ist defekt. Als Input bekommt er über seine Sensoren ein Bild von dem zu klassifizierenden Teil. Anhand dieser Momentaufnahme trifft der Agent seine Entscheidung und gibt als Output über seine Aktoren entweder `true` (Teil ist intakt) oder `false` (Teil ist defekt) zurück.

Verkaufsagent²:

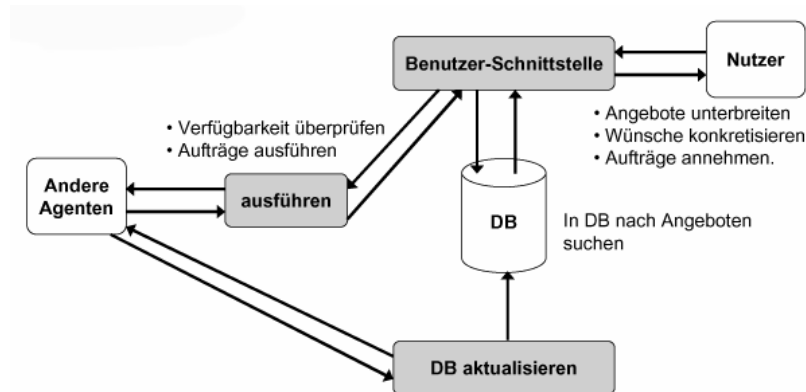


Abbildung 1.3.2.

Schon deutlich komplexer ist das Beispiel des elektronischen Verkaufsagenten. Seine Aufgabe besteht zum einen darin, Anfragen entgegen zu nehmen und nach passenden Angeboten (z.B. für Reisen) zu suchen. Denkbar ist auch, dass Kunden noch gar keine konkreten Vorstellungen haben und der Agent durch Beratung dabei hilft, Wünsche zu konkretisieren. Des Weiteren muss der Verkaufsagent auch mit anderen Agenten kommunizieren, um seine Datenbank mit Angeboten zu aktualisieren oder Aufträge (z.B. das Buchen einer Reise) für den Benutzer auszuführen.

Elektronischer Taxifahrer³:

Dieses Beispiel wird wohl noch lange Zeit nicht besonders realistisch sein. Dennoch sei es erwähnt, da es ein besonders anschauliches Beispiel für einen autonom agierenden Agenten mit komplexen Zielen ist, der mit einer nichtdeterministischen und äußerst dynamischen Umwelt interagiert. Zum einen müsste ein solcher Agent selbständig im Straßenverkehr seine Fahrziele erreichen können und dabei auch auf unvorhergesehene Ereignisse adäquat reagieren können. Zum anderen muss er gleichzeitig noch weitere Vorgaben, wie Minimierung der Kosten und der Risiken sowie die Zufriedenstellung des Kunden soweit wie möglich einhalten können.

Man sieht also, dass als Einsatzgebiet für (Software-)Agenten viele unterschiedliche Szenarien denkbar sind. Gemeinsam haben diese Beispiele nur, dass ein Agent auf irgendeine Weise selbstständig Aufträge erfüllt.

² Vgl. Handbuch der künstlichen Intelligenz, Kap. 24.1.5

³ S. Russel, P. Norvig: Artificial Intelligence - A Modern Approach, Kap. 2.3.1

2. Grundlagen

2.1. Umwelt des Agenten

Die Umwelt ist die für den Agenten relevante Welt. Ihre Beschaffenheit hat maßgeblichen Einfluss auf die zugrunde liegende Architektur. Da, wie an verschiedenen Beispielen bereits gezeigt wurde, als Einsatzgebiet für Agenten sehr unterschiedliche Umwelten möglich sind, ist es schwer allgemeine Empfehlungen oder Implementationsanweisungen zu geben. Sehr viele Entscheidungen bezüglich Vorgehensweise und Architektur müssen eben anhand der konkreten Umwelt getroffen werden. Bei einer virtuellen Umwelt kann es außerdem möglich sein, dass der Entwickler diese mitgestalten kann, um so möglichst günstige Bedingungen für den oder die Agenten zu schaffen.

Prinzipiell lassen sich mehrere, für den Entwurf relevante, „Dimensionen“⁴ unterscheiden, wobei Überschneidungen möglich sind:

- **Vollständig beobachtbar vs. Teilweise beobachtbar:**
Eine Umgebung ist vollständig beobachtbar, wenn der Agent alle für ihn relevanten Aspekte der Umwelt beobachten kann. Gilt dies nicht, ist die Umwelt nur teilweise beobachtbar.
- **Deterministisch vs. Nichtdeterministisch:**
Wenn der nächste Zustand der Umgebung vollständig durch den aktuellen Zustand und die vom Agenten durchgeführten Aktionen vorausgesagt werden kann, ist die Umwelt deterministisch, andernfalls ist sie nichtdeterministisch (auch stochastisch genannt). Eine vollständig beobachtbare Umwelt wird dem Agenten als deterministisch erscheinen, wohingegen eine nur teilweise beobachtbare Umwelt ihm als stochastisch erscheinen kann.
- **Episodisch vs. Sequenziell:**
In einer episodischen Umgebung kann der Agent jeder Wahrnehmung eine Aktion zuordnen, ohne frühere Wahrnehmungen oder Aktionen berücksichtigen zu müssen. Viele Klassifizierungsaufgaben sind episodisch. In einer sequentiellen Umgebung können aktuelle Entscheidungen alle weiteren beeinflussen. Es existiert also eine Zustandskomponente. Es versteht sich, dass episodische Umgebungen wesentlich einfacher zu handhaben sind als sequenzielle.
- **Statisch vs. Dynamisch:**
Von einer dynamischen Umwelt spricht man, wenn sich ihr Zustand ändern kann, während der Agent eine Entscheidung trifft. Ansonsten spricht man von einer statischen Umwelt. Außerdem kann man eine dynamische Umwelt als nichtdeterministisch betrachten, wohingegen eine statische Umwelt deterministisch sein kann (aber nicht muss).
- **Diskret vs. Stetig:**
Diese Dimension kann sich auf den Zustand der Umwelt, die Behandlung der Zeit oder auf Wahrnehmung und Aktionen des Agenten beziehen. Klar ist, dass es in einem Programm keine echte Stetigkeit gibt. Eine physikalische Umwelt z.B. wäre aber echt stetig, was sich auf die eben erwähnten Aspekte auswirken würde.
- **Einzelagent vs. Multiagent:**
Man kann unterscheiden, ob sich ein Agent alleine in einer Umgebung befindet, oder

⁴ S. Russel, P. Norvig: Artificial Intelligence - A Modern Approach, Kap. 2.3.2

ob er mit anderen Agenten interagieren kann oder sogar muss. Dahinter verbirgt sich die Entscheidung, die ein Agent zu treffen hat, ob es sich bei einem anderen Objekt auch um einen Agenten handelt oder eben nicht. In einem *kooperativen* Umfeld mag das noch einfach erscheinen. In einer *konkurrierenden* Umgebung ist dieser Entschluss schon schwieriger, da Agenten ja versuchen könnten, andere Agenten zu täuschen, um so Vorteile für sich zu erzielen.

Diese 6 Dimensionen beeinflussen maßgeblich die Komplexität der Umwelt des Agenten. Dabei wäre eine nur teilweise beobachtbare, nichtdeterministische, sequenzielle, stetige und dynamische Multiagentenumgebung der komplexeste denkbare Fall, während eine vollständig beobachtbare, deterministische, episodische, diskrete und statische Einzelagentenumgebung der einfachste Fall wäre. Beliebige Zwischenformen sind denkbar. Wie bereits erwähnt, handelt es sich um keine echten Dimensionen, da sie sich teilweise gegenseitig beeinflussen. Es ist zum Erfassen und Einordnen der Problematik aber dennoch hilfreich sie zunächst als solche zu betrachten.

2.2. Unterschiedliche Betrachtungsweisen

Ein wesentlicher Unterschied besteht zwischen

- der Sicht des Entwicklers, also der Sicht zur Entwurfszeit und
- der Sicht des Agenten, also der Sicht zur Laufzeit.

Diese Unterscheidung ist wichtig, da die Sicht des Agenten häufig wesentlich eingeschränkter ist, als die Sicht des Entwicklers. Dies kann, wie bereits erwähnt, aufgrund der Beschaffenheit der Umwelt der Fall sein. Es kann aber auch, wie im Folgenden weiter ausgeführt wird, eine aus Effizienzgründen vom Entwickler getroffene Entscheidung sein. Der Entwickler muss also zur Entwurfszeit erkennen, welche Maßnahmen notwendig sind, damit der Agent zur Laufzeit die erwartete Leistung erbringt.

2.3. Rationalität

Es ist klar, dass ein Agent stets das Richtige tun sollte. Für einen allwissenden Agenten wäre das auch nicht schwer. Anhand seiner Vorgaben könnte er immer die optimale, und somit beste, Lösung finden. Meistens wird ein Agent aber nur über eine eingeschränkte Sichtweise verfügen. Dabei kann es sich um eine, aus Effizienzgründen, bewusst getroffene Entscheidung handeln.

Das bedeutet aber, dass das beste Verhalten für den Agenten nicht voraussagbar ist. Er muss also auf Grundlage seines Wissens über die Umwelt die Entscheidung treffen, die ihm als die Beste erscheint. Als Grundlage hierfür dient eine vom Entwickler vorgegebene Leistungsbewertung, die der Agent versucht zu maximieren. Allerdings hat er (gerade in dynamischen Umgebungen) nicht ewig Zeit seine Entscheidung zu treffen, da ein zu langes Zögern den Agenten daran hindern würde, seine Leistungsbewertung zu maximieren.

Ein Rationaler Agent wählt also auf effiziente Weise⁵ auf Grundlage seines Wissens über die Umwelt immer die Aktionen aus, von denen er erwarten kann, dass sie seine Leistungsbewertung⁶ maximieren.

⁵ Vgl. Handbuch der künstlichen Intelligenz, Kap. 24.3.10

2.4. Parallelität und Nebenläufigkeit

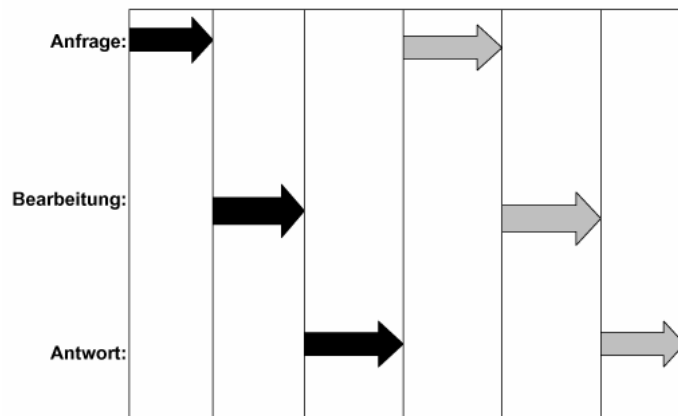


Abbildung 2.4.1: Sequentielle Abarbeitung von Aufträgen

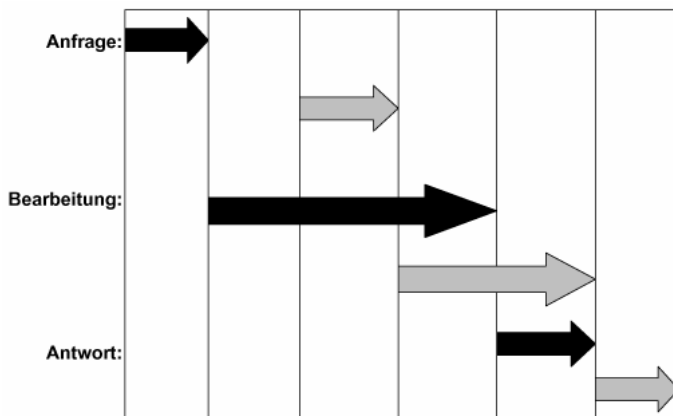


Abbildung 2.4.2: Parallele Abarbeitung von Aufträgen

Häufig ist es nötig, mehrere Ziele nebenläufig zu verfolgen. So würde es beispielsweise wenig Sinn machen, wenn der Verkaufsagent nur eine Aufgabe zurzeit wahrnehmen würde (Abbildung 2.4.1). Stattdessen wird er sinnvoller Weise mehrere Kunden-Anfragen gleichzeitig bearbeiten sowie mit anderen Agenten kommunizieren um seine Datenbestände zu aktualisieren oder Angebote auszuhandeln (Abbildung 2.4.2).

Es ist klar, dass dabei die jeweiligen Lese-/Schreib-Zugriffe auf die Datenbestände koordiniert werden müssen (Synchronisation). Es können sich hierbei aber, beispielsweise bei der Nachbildung menschlicher Verhaltensweisen, noch wesentlich komplexere Problemstellungen bezüglich der Koordination der unterschiedlichen Prozesse ergeben (Siehe Abschnitt 4.1).

⁶ S. Russel, P. Norvig: Artificial Intelligence - A Modern Approach, Kap. 2.2.2

3. Agenten-Strukturen

3.1. Komponenten eines (Software-)Agenten

Es lassen sich einige Komponenten, bzw. Bestandteile eines Agenten unterscheiden, die im Folgenden kurz erläutert werden.

3.1.1. Umweltkopplung

Hiermit ist die Anbindung des Agenten an seine Umwelt über Sensoren und Aktuatoren gemeint. Die Sensoren dienen dazu, Reize der Umgebung in eine angemessene interne Datenstruktur zu überführen. Die Aktuatoren führen die vom Agenten ausgewählten Aktionen in seiner Umwelt aus.

3.1.2. Umweltmodel

Bei dem Umweltmodel handelt es sich um eine interne Repräsentation des aktuellen Zustands der Umwelt. Kann der Agent alle Aspekte seiner Umwelt erfassen und verwalten, handelt es sich dabei um Wissen, andernfalls um Annahmen, die natürlich auch falsch sein können und korrigiert werden müssen, sobald der Agent dies bemerkt.

3.1.3. Steuerung und Management

Anhand dieser Komponente unterscheiden sich die verschiedenen Agenten-Strukturen und Architekturen. Hierbei handelt es sich um die „Programmlogik“ des Agenten, die die verschiedenen Komponenten miteinander verbindet.

3.1.4. Fähigkeiten

Fähigkeiten sind Pläne (im einfachsten Fall Aktionen), die vom Agenten ausgewählt, und kombiniert werden können, um ein bestimmtes Ziel zu erreichen; häufig auch als Planbibliothek bezeichnet. Es ist natürlich durchaus denkbar, dass solche Pläne auch parametrisiert werden können. Beispielsweise könnte der elektronische Taxifahrer in seiner Planbibliothek einen Plan „Fahre in die Richtung“ haben. Hier wäre die Richtung der Parameter.

3.1.5. Entscheidungskomponente

Diese Komponente ist für Auswahl von Handlungen, also von Fähigkeiten, zuständig. Später wird sich zeigen, dass solch ein Entscheidungsfindungsprozess sehr komplex sein kann und somit ein zentraler Bestandteil der „Programmlogik“ ist.

3.2. Einfacher Reflex-Agent

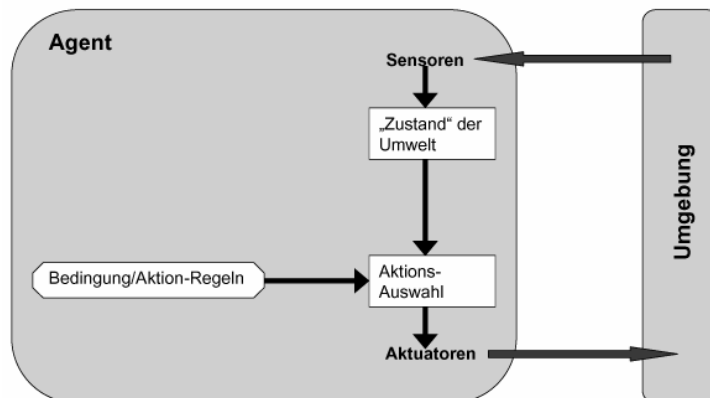


Abbildung 3.2.1. Einfacher Reflex-Agent⁷

Bei dem einfachen Reflex-Agenten handelt es sich um den einfachsten denkbaren Agenten. Über seine Sensoren empfängt der Agent eine Wahrnehmung, die als Momentaufnahme der Umwelt anzusehen ist. Nur aufgrund dieser Wahrnehmung wählt der Agent aus einer Menge von Bedingung/Aktion-Regeln eine passende Regel aus und ordnet mit ihr dem aktuellen Zustand der Umwelt eine Aktion zu.

Pseudocode:

```

var
  Rules rules; /* Eine Menge von Bedingung/Aktion-Regeln (Fähigkeiten). */

repeat
  percept := sense(sensoryInputs());
  /* Umweltkopplung: Wartet auf Input und bringt ihn in eine geeignete
  interne Darstellung */
  rule := match(percept, rules); /* Regel auswählen */
  action := act(rule); /* Umweltkopplung zu den Aktuatoren */
forever
  
```

Der zu Beginn vorgestellte Agent zur Klassifikation von Teilen entspricht genau dieser Struktur. Er befindet sich in einer episodischen Umwelt und kann seine Entscheidungen ausschließlich aufgrund des aktuellen Zustandes treffen. Dies ist nur in vollständig beobachtbaren Umgebungen möglich. Erfüllt eine Umwelt diese Bedingung nicht, ist sie also nur teilweise beobachtbar, sind mit dieser Struktur Endlosschleifen häufig unvermeidbar, da der Agent so keine Chance hat, festzustellen, wenn eine vorherige Aktion keine Wirkung gehabt hat. Er würde dieselbe Aktion einfach immer wieder auswählen. Für teilweise beobachtbare Umgebungen sind also komplexere Strukturen nötig.

⁷ S. Russel, P. Norvig: Artificial Intelligence - A Modern Approach, Kap. 2.4.2

3.3. Modelbasierter Reflex-Agent

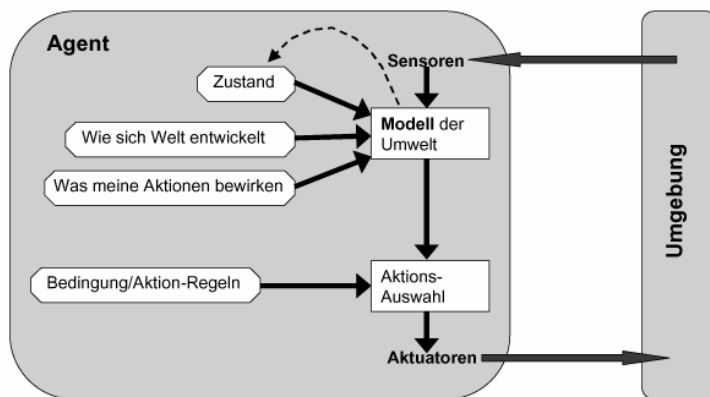


Abbildung 3.3.1. Modelbasierter Reflex-Agent⁸

Der Modelbasierte Reflex-Agent wählt Aktionen, im Gegensatz zum einfachen Reflex-Agenten, auf Grundlage eines Umweltmodells aus. Dieses Umweltmodell beinhaltet nicht nur die aktuelle Wahrnehmung, sondern auch den letzten Zustand der Umwelt, die zuletzt ausgeführte Aktion, sowie Wissen um die Veränderungen in der Umwelt und darum, was die eigenen Aktionen bewirken. Aus dem Delta der letzten beiden Aspekte kann der Agent dann bezüglich der Wirkung / des Erfolges seiner letzten Aktion Rückschlüsse ziehen (Siehe Abbildung 3.3.1). Es ist klar, dass die Auswahl von Bedingung-/Aktion-Regeln hier komplizierter ist, als beim einfachen Reflex-Agenten. Es ist z.B. denkbar, dass es sich hier um parametrisierbare Aktionen handelt.

Pseudocode:

```

var
  Beliefs bel; /* Repräsentation der aktuellen Annahme über die Welt. */
  Rules rules; /* Eine Menge von Bedingung/Aktion-Regeln (Fähigkeiten). */
  Action action; /* Die vorherige Aktion, anfänglich keine. */

repeat
  percept := sence(sensoryInputs());
  /* Umweltkopplung: Wartet auf Input und bringt ihn in eine geeignete
     interne Darstellung */
  bel := update(percept, bel, action); /* Umweltmodell aktualisieren */
  rule := match(bel, rules); /* Regel auswählen */
  action := act(rule); /* Umweltkopplung zu den Aktuatoren */
forever

```

Dieses Umweltmodell bezeichnet man auch als Annahmen (Beliefs) über die Umwelt. Diese Annahmen können, wie bereits erwähnt, auch falsch sein aufgrund nicht beobachtbarer Aspekte der Umwelt. Im Gegensatz zum einfachen Reflex-Agenten ist der Modelbasierte Reflex-Agent aber in der Lage diese nicht beobachtbaren Aspekte (den Nichtdeterminismus) zu verwalten.

Dennoch ist das Wissen um den Aktuellen Zustand der Welt nicht immer ausreichend, um die richtigen Entscheidungen treffen zu können. Das Verfolgen von längerfristigen Zielen, sowie das Aufgreifen neuer Ziele sind so nicht möglich. Hierfür benötigt man Zielbasierte Agenten.

⁸ S. Russel, P. Norvig: Artificial Intelligence - A Modern Approach, Kap. 2.4.2

3.4. Zielbasierter Agent

Der Zielbasierte Agent ordnet also nicht nur einem Zustand der Welt (bzw. einer Annahme über den Zustand der Welt) eine Aktion zu, sondern er bestimmt zunächst ein erstrebenswertes Ziel, das er zu erreichen versucht. Dabei gibt es kurz, mittel und langfristige Ziele. Einige lassen sich unmittelbar mit einer einzigen Aktion erreichen, für andere sind zur Erreichung Pläne notwendig. Prinzipiell könnte ein Agent auch mehrere Ziele gleichzeitig verfolgen. Zunächst wird, zur Vereinfachung, aber nur das Verfolgen eines Ziels gleichzeitig betrachtet. Mehrere Ziele gibt es hier nur, wenn sie aufeinander aufbauen (Teilziele) oder sequentiell verfolgt werden können. Auf das nebenläufige Verfolgen mehrere Ziele wird in Abschnitt 4.1. näher eingegangen.

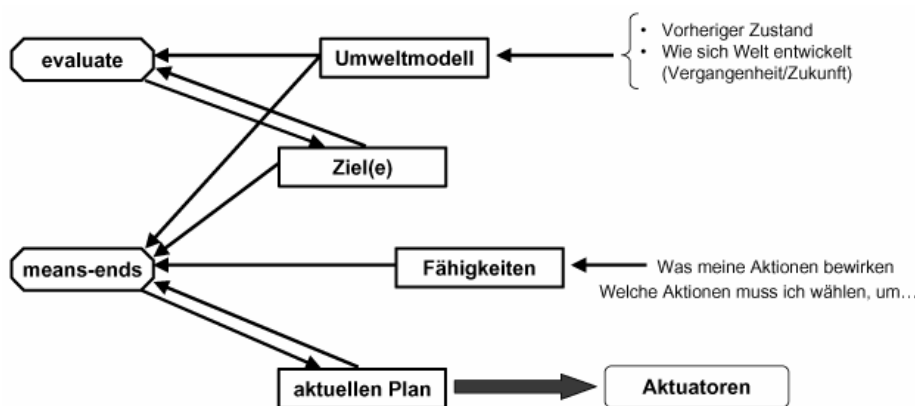


Abbildung 2.4.1. Zielbasierter Agent: Deliberation und Means-ends-reasoning⁹

Wie in Abbildung 2.4.1 zu sehen ist, besteht der Entscheidungsprozess jetzt aus zwei Stufen:

1. Deliberation: Der Agent aktualisiert mit Hilfe seines Umweltmodells seine Ziele. Dabei können neue Ziele hinzugefügt und alte entfernt werden, wenn sie nicht mehr erreicht werden können oder bereits erreicht sind.
2. Means-ends-reasoning: Mit Hilfe seines Umweltmodells, seiner Ziele und Fähigkeiten (Planbibliothek) aktualisiert der Agent seinen Plan, den er aktuell verfolgt. Dabei ist es möglich, dass der aktuelle Plan verworfen und gegen einen neuen Plan ausgetauscht wird, der als geeigneter erscheint. Dabei ist ein geeigneter Mittelweg zwischen Flexibilität und Stabilität im Verhalten zu finden (Doch dazu mehr in Abschnitt 4.1).

Die Aktuatoren schließlich führen die nächste anstehende Aktion aus.

Pseudocode:

```

var
  Beliefs bel; /* Repräsentation der aktuellen Annahme über die Welt. */
  Option goal; /* Das aktuelle Ziel / die aktuell gewählte Option */
  Plan pla; /* Aktueller Plan */

repeat
  percept := sence(sensoryInputs());
  /* Umweltkopplung: Bringt den eintreffenden Input in eine geeignete
  interne Darstellung */
  bel := update(percept, bel); /* Umweltmodell aktualisieren */
  goal := evaluate(bel, goal); /* Ziel bestimmen */
  pla := means_ends(bel, goal, pla); /* Plan aktualisieren */
  action := act(pla); /* Umweltkopplung zu den Aktuatoren */
forever
  
```

⁹ Vgl. Handbuch der künstlichen Intelligenz, Kap. 24.5.6

Trotz der Trennung dieser beiden Entscheidungsphasen zur besseren Strukturierung darf der Agent in der ersten Phase keine Ziele auswählen, für deren Erreichung er keine Fähigkeiten besitzt. Er muss also die *Realismus-Forderung* erfüllen. Dazu muss der Agent in der ersten Phase abschätzen können, welche Ziele er mit seinen Fähigkeiten erreichen kann. Außerdem ist eine Zielrevision vorzusehen, für den Fall, dass sich der Agent „verschätzt“.

3.5. Nutzenbasierter Agent

Der Nutzenbasierte Agent ist nur eine „Erweiterung“ des Zielbasierten Agenten. Er besitzt zusätzlich eine Nutzenfunktion, die Zustände oder Zustandsfolgen auf eine reelle Zahl abbildet, die den Grad des Nutzens beschreibt. Ziele / Teilziele mit hohem Nutzen werden gegenüber solchen mit niedrigem Nutzen bevorzugt. Es ist also gewährleistet, dass der Agent nicht einfach „irgendwie“ versucht seine Ziele zu erreichen, sondern dabei versucht den Nutzen zu maximieren. In Anlehnung an die getroffene Definition eines Rationalen Agenten lässt sich also sagen, dass die Begriffe „Rationaler Zielbasierter Agent“ und „Nutzenbasierter Agent“ äquivalent sind. Meistens meint man, wenn man von einem Zielbasierten Agenten spricht damit auch einen Nutzenbasierten Agenten.

3.6. Stimulus Response oder andauernde Tätigkeit

Der einfache und der Modelbasierte Reflex-Agent arbeiten, wie der Name schon sagt, nach dem Stimulus-Response-Prinzip. Sie reagieren auf Reize, bzw. Anfragen. Das Verhalten des Ziel- und des Nutzenbasierten Agenten ist da schon wesentlich komplexer. Hier spricht man von andauernder Tätigkeit. Sie kümmern sich selbst um die regelmäßige Aktualisierung ihres Umweltmodells und um das Anstoßen neuer Entscheidungsprozesse.

Natürlich sind auch Mischformen denkbar. So kann etwa der Verkaufsagent durchaus ein andauernd tätiger, Ziel-, bzw. Nutzenbasierter Agent sein. Er wird beispielsweise selbstständig mit anderen Agenten kooperieren um seine Datenbank zu aktualisieren, Angebote auszuhandeln oder Aufträge zu vergeben. Dem Benutzer hingegen wird er als Stimulus-Response-Agent erscheinen, der auf seine Anfragen reagiert.

3.7. Lernende Agenten

Es ist oft nicht möglich und nicht sinnvoll, das komplette Wissen über die Umwelt statisch im Agenten abzulegen, da hierfür zum einen eine zu große Datenbank notwendig wäre, um alle Eventualitäten zu erfassen. Zum anderen ist es schlicht und ergreifend zu unflexibel. Nur geringfügige Änderungen an der Umwelt des Agenten würden dazu führen, dass der Agent von seinen Entwicklern angepasst werden muss. In sehr einfachen Umgebungen mag das noch akzeptabel sein. Doch in komplexeren Umgebungen wird es notwendig, dass der Agent sein Wissen über die Umwelt und die Wirkungsweise seiner Aktionen darin selbst aktualisiert, dass er also *selbstständig lernt*.

Hierzu kann der Agent nach der Ausführung von Aktionen deren Erfolg bewerten (ein Feedback generieren), um so seine Auswahlkriterien für künftige Aktionen zu verbessern. Außerdem kann es sinnvoll sein, dass der Agent zusätzliche Aktionen ausführt, um neues

Wissen zu generieren. Hierzu berechnet ein Problemgenerator Aktionen, von denen er sich einen Lernerfolg verspricht.

4. Architekturen

4.1. BDI-Architektur (Belief – Desire – Intention)

Ein Agent sollte, besonders in dynamischen Umgebungen, sinnvoller Weise mehrere Ziele gleichzeitig (also nebenläufig) verfolgen können. Gleichzeitig verfolgte Ziele dürfen aber untereinander nicht in Konflikt stehen. Hier gilt eine *starke Realismusforderung*. Stehen Ziele im Konflikt, muss eine Auswahl getroffen werden. Dabei stellt die Entscheidung, ob ein bisheriges Ziel zugunsten eines anderen aufgegeben wird eine besondere Schwierigkeit dar. Zum einen kann es sinnvoll sein, auf bessere Alternativen auszuweichen, zum anderen ist es aber eine Frage der Zuverlässigkeit, dass der Agent Begonnenes zu Ende führt. Es ist also abzuwägen zwischen Flexibilität auf der einen Seite (auf bessere Alternativen ausweichen) und Stabilität im Verhalten auf der anderen Seite (Begonnenes zu Ende verfolgen). Natürlich müssen auch Ziele, die nicht mehr erreicht werden können, aufgegeben werden.

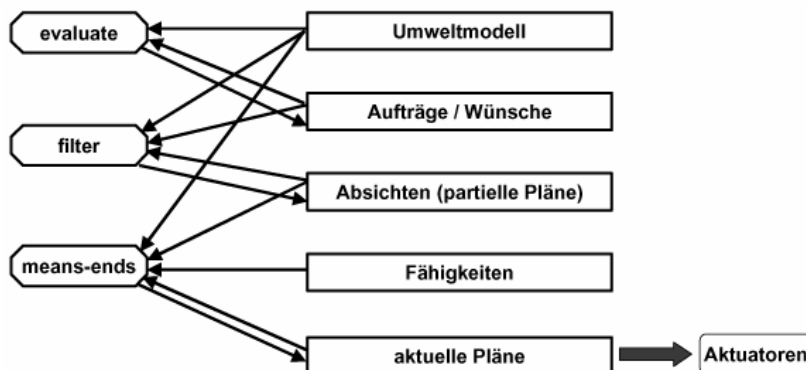


Abbildung 4.1.1. BDI-Architektur¹⁰

Die BDI-Architektur unterteilt die Menge der Ziele noch einmal in zwei Untermengen:

- In die Menge der Wünsche (desire): Hier gilt noch keine Realismus-Forderung.
- In die Menge der Absichten (intention): Hierbei handelt es sich um Verpflichtungen, denen der Agent nachzukommen hat. Hier gilt die starke Realismusforderung.

Diese Unterteilung hat den Vorteil, dass sie den Entscheidungsprozess besser strukturiert. Wünsche die zu einem bestimmten Zeitpunkt mit den Absichten in Konflikt stehen, können unter Umständen zu einem späteren Zeitpunkt zu Intentionen werden. Diese Vorgehensweise ist angelehnt an das menschliche Verhalten: Wünsche können „im Hinterkopf“ behalten werden.

Abbildung 4.1.1 verdeutlicht noch einmal den Ablauf. Zunächst werden also mit Hilfe des Umweltmodells die Wünsche aktualisiert. Dabei ist es durchaus denkbar, dass es eine Menge

¹⁰ Vgl. Handbuch der künstlichen Intelligenz, Kap. 24.5.7

von fest vorgegebenen, parametrisierbaren Wünschen gibt (Beim Taxifahrer beispielsweise: Fahrziel erreichen oder Fahrgast abholen. Die konkreten Koordinaten wären dann die Parameter). Auch wenn hier noch keine Realismus-Forderung gilt, kann es natürlich sinnvoll sein, zu überprüfen, ob Wünsche jemals erreicht werden können. Dies wird besonders deutlich, wenn man sich die Wünsche als Aufträge von einem Benutzer oder anderen Agenten vorstellt.

Im nächsten Schritt werden mit Hilfe des Umweltmodells die Wünsche und Absichten abgeglichen. Es wird also überprüft, welche Wünsche zu Absichten werden können. Dabei erhalten bereits bestehende Absichten Priorität. Sie stellen einen „Zulässigkeits-Filter“¹¹ dar. Wobei natürlich überprüft werden muss, ob bestehende Absichten noch erreicht werden können oder bereits erreicht sind. Diese Absichten lassen sich auch als partielle (also vergrößerte) Pläne bezeichnen, die erst im nächsten Schritt auf die konkreten Aktionen zurückgeführt werden.

Im letzten Schritt werden dann mit Hilfe des Umweltmodells, den Absichten und den vorhandenen Fähigkeiten die Pläne aktualisiert. Anschließend wird von der Akuatorik die nächste Aktion ausgeführt.

Pseudocode:

```
var
  Beliefs bel; /* Repräsentation der aktuellen Annahme über die Welt. */
  Options desires, intentions; /* Wünsche und Absichten/Verpflichtungen */
  Plans pla; /* Die aktuellen Pläne */

repeat
  percept := sence(sensoryInputs());
  /* Umweltkopplung: Bringt den eintreffenden Input in eine geeignete
     interne Darstellung */
  bel := update(percept, bel); /* Umweltmodell aktualisieren */
  desires := evaluate(bel, desires); /* Wünsche aktualisieren/ermitteln */
  intentions := filter(bel, desires, intentions);
  /* Verpflichtungen aktualisieren / neue hinzufügen */
  pla := means_ends(bel, intentions, pla); /* Pläne aktualisieren */
  action := act(pla); /*Umweltkopplung zu den Aktuatoren */
forever
```

4.2. Schichtenmodell

Eine Problematik, für die die BDI-Architektur keine Lösungen anbietet, ist die Tatsache, dass es eben kurz-, mittel- und langfristige Ziele gibt, sowie Einflüsse mit kurz, mittel oder langfristigen Auswirkungen. Das Aufgeben kurzfristiger Ziele hat eine andere Tragweite, als das Aufgeben langfristiger Ziele. Dies stellt eine sehr komplexe Problemstellung für die Entscheidungskomponente dar.

Eine Lösung dieses Problems besteht darin, die Fähigkeitskomponente „zu verselbstständigen“. Sie überprüft die Bedingungen für Fortsetzung, bzw. Modifizierung selbst. Es wird also eine Unterteilung von Teilzielen und Plänen gemäß ihrer Tragweite vorgenommen. Kurz-, mittel- und langfristige Entscheidungen werden auf mehrere Schichten verteilt. Jede Schicht trifft ihre Entscheidungen weitgehend selbst.

¹¹ Vgl. Handbuch der künstlichen Intelligenz, Kap. 24.5.7

Im Folgenden werden 3 unterschiedliche Arten von Schichten-Modellen vorgestellt, wobei die letzte der 3 Architekturen die ist, die dem beschriebenen Problem am besten gerecht wird.

4.2.1. Horizontale Architektur

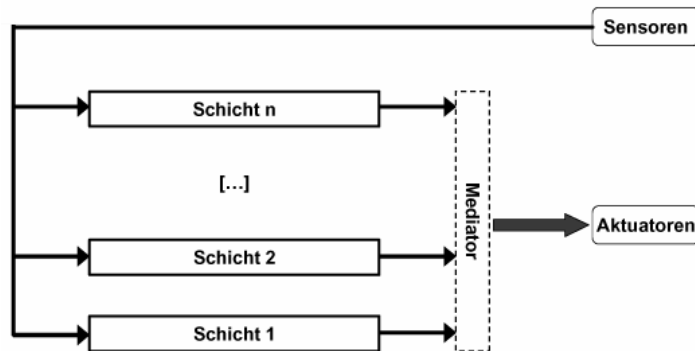


Abbildung 4.1.1. Horizontale Schichtenarchitektur¹²

Wie in Abbildung 4.1.1 zu sehen ist, sind hier alle Schichten über Sensoren und Aktuatoren mit der Umwelt verbunden. Um eine konsistente Ausgabe zu sichern ist gegebenenfalls noch ein Mediator notwendig, der den Output der Schichten entgegennimmt und überprüft. Hierbei handelt es sich prinzipiell um nichts anderes, als mehrere Prozesse/Threads, die gleichzeitig ablaufen und natürlich an bestimmten Stellen synchronisiert werden müssen.

4.2.2. Vertikale 1-Paß-Architektur

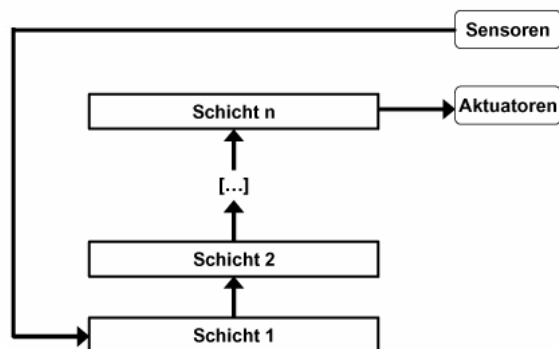


Abbildung 4.2.2.1. Vertikale 1-Paß-Architektur¹²

Bei der vertikalen 1-Paß-Architektur bekommt die unterste Schicht den sensorischen Input. Jede Schicht verarbeitet nun den Input, den sie von der darunter liegenden bekommt und gibt ihn an die darüber liegende weiter. Die oberste Schicht schließlich ist über die Aktorik mit der Umwelt verbunden. Man erkennt, dass es sich hier um nichts anderes handelt, als um Funktionen, die ihre Ergebnisse aneinander übergeben. Es handelt sich also um eine zeitliche Strukturierung eines Prozesses. Die BDI-Architektur kann als vertikale 1-Paß-Architektur betrachtet werden.

4.2.3. Vertikale 2-Paß-Architektur

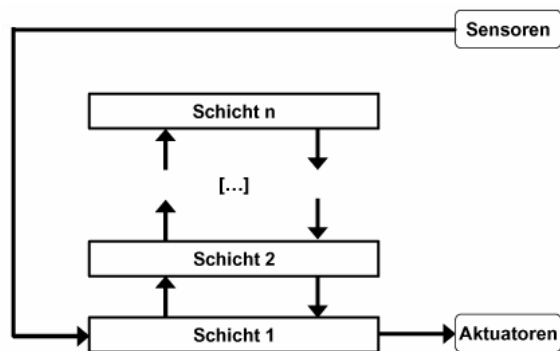


Abbildung 4.2.3.2. Vertikale 2-Paß-Architektur¹²

Die vertikale 2-Paß-Schichten-Architektur ist wohl das, was man sich als erstes Vorstellt, wenn man an ein Schichten-Modell denkt. Die unterste Schicht ist für die komplette Umweltkopplung zuständig. Sie empfängt sensorischen Input, verarbeitet ihn und gibt ihn an die darüber liegende Schicht weiter. Umgekehrt bekommt sie Anweisungen von der darüber liegenden Schicht und kümmert sich um eine geeignete Umsetzung über die Aktorik. Es findet zwischen den angrenzenden Schichten (vertikal) eine direkte Kommunikation statt. Bis auf die unterste Schicht kommunizieren alle anderen Schichten auf logischer Ebene mit der Umwelt.

Man kann erkennen, dass sich mit dieser Architektur auch die angesprochene Aufteilung von Entscheidungen unterschiedlicher tragweite realisieren lässt.

Gängige Schichten-Architekturen sehen 3 Schichten vor:

- Oberste Schicht: Kooperation und kooperative Planung.
- Mittlere Schicht: Individuelle Planung
- Unterste Schicht: Reaktives Verhalten.

Klar ist, dass auf der obersten Schicht, also bei der Kooperation mit Benutzern oder anderen Agenten, Stabilität im Verhalten wichtig ist. Es ist wichtig Zuverlässig zu erscheinen, so dass andere Teilnehmer wissen, dass sie sich auf einen verlassen können. Auf der untersten Schicht hingegen ist Flexibilität wichtig, da hier unter Umständen sehr kurzfristig auf Einflüsse reagiert werden muss. Klar ist auch, dass auf den verschiedenen Ebenen auch eine unterschiedliche zeitliche Sichtweise herrscht. Die oberste Schicht plant in viel längeren Zeiträumen als die unterste.

4.3. Hybride Architekturen

Man hat gesehen, dass BDI sich sehr gut für die Strukturierung des Entscheidungsprozesses eignet, sofern denn an menschliches Verhalten angelehnte Entscheidungskriterien maßgeblich sind. Schichten wiederum eignen sich sehr gut zur Strukturierung des Verfeinerungsprozesses von Absichten und Plänen.

Daher bietet es sich natürlich an, beide zu kombinieren: BDI zur Strukturierung des Entscheidungsprozesses und eine Schichten-Architektur zur Unterteilung der kur- mittel- und langfristigen Planung.

¹² Vgl. Handbuch der künstlichen Intelligenz, Kap. 24.5.9

Wie bereits erwähnt, wäre dabei die oberste Schicht für die kooperative Planung zuständig, die Stabilität im Verhalten fordert, während die unterste Schicht für das reaktive Verhalten zuständig ist, welches Flexibilität erfordert.

Das bekannteste Beispiel hierfür ist die InterRap-Architektur: Sie umfasst die erwähnten 3 Schichten Kooperation, Planung und Verhalten. Jede Schicht wiederum arbeitet nach dem BDI-Prinzip.

Es ist aber auch eine Verbindung von reaktivem Verhalten und komplexeren Entscheidungsstrukturen möglich. Prinzipiell kann man sagen, dass auch alle vorgestellten Strukturen in einem Agenten vorkommen können.

5. Beispiel: „RoboCup“

Schon immer war es (u.a.) der Zweck Wissenschaftlicher Experimente, neue Erkenntnisse zu gewinnen und vorhandene Ideen zu überprüfen (Beispiel Grundlagenforschung). Warum nicht auch in der KI?

Diese kleine Analogie mag vielleicht an einigen Stellen etwas hinken. Dennoch erfasst sie den Kern des Problems: Wenn man die Ideen und Konzepte, die im Rahmen der KI entwickelt werden, nicht anhand konkreter Problemstellungen ausprobiert, wird man auch nichts über ihre Leistungsfähigkeit oder Relevanz aussagen können. Daher macht es auch keinen Sinn, über allgemeine Konzepte zur Entwicklung von Agenten zu sprechen, ohne anhand von konkreten Fragestellungen zu überprüfen, wie geeignet sie sind. Hier gibt RoboCup Antworten.

5.1. Was ist RoboCup?

Der RoboCup existiert seit 1997 und ist eine internationale Initiative mit dem Ziel, die Forschung und Lehre in den Gebieten KI und Robotik zu fördern. Dabei erlaubt Fußball als Problemstellung das Untersuchen einer Vielzahl von interessanten Fragestellungen, wie z.B. Design autonomer Agenten, Kooperationsverhalten, Entscheidungsfindung in Echtzeitumgebungen, Verwerten unscharfer Informationen, Bildverarbeitung, Steuerung, usw.

Es gibt verschiedene Liegen, wobei sich 2 wesentliche unterscheiden lassen:

- Liegen mit realen Robotern
- Die Simulationsliga

Letztere wird im Folgenden näher betrachtet.

5.2. Die Simulationsliga

Auf einem virtuellen Fußballfeld treten zwei Agenten-Mannschaften gegeneinander an. Die Simulation des Spiels erfolgt über den so genannten „Soccerserver“. Er nimmt die Kommandos von den Spielern entgegen und berechnet daraus die resultierende Bewegung von Spielern und Ball.

Die Spieler können alle 100ms (Mikrosekunden) eine Aktion ausführen (heißt: ein Kommando abschicken), alle 150ms können sie eine Sichtinformation empfangen. Die Zeit ist unterteilt in 10ms-Zeitpunkte, welche die kleinsten wahrnehmbaren Intervalle darstellen. Der Raum erscheint hingegen kontinuierlich (die Koordinaten sind also reelle Zahlen).

Den Spielern stehen folgende Kommandos zur Verfügung:

turn (drehen), **dash** (beschleunigen), **kick** (schießen), **turn-neck** (Kopf wenden), **say** (Rufen. Das gerufene hören alle Spieler in der Umgebung, auch die gegnerischen).

Komplexere Handlungen müssen als Folgen solcher Aktionen geplant und ausgeführt werden. Das bedeutet aufgrund der vorhandenen 100ms-Verzögerung auch, dass eine kürzere Folge von Kommandos die zum Ziel führt, optimaler ist, als eine lange. Es ist für ein erfolgreiches Spiel also auch wichtig, dass die Spieler sparsam mit dem Absetzen von Kommandos und dem Anfordern von Sicht-Informationen umgehen.

Die Informationen, die der Soccerserver den Spielern liefert, sind außerdem nicht ganz zuverlässig. Für weiter entfernte Objekte werden zufällige Verfälschungen vorgenommen. Aufgrund der Verzögerung bei der Beantwortung von View-Anfragen kann ein Spieler sich diese zufälligen Verfälschungen auch nicht durch eine Reihe von Anfragen „herausrechnen“. Die Ballbewegungen sind schwach nicht-deterministisch, es erfolgen zufällige Verfälschungen.

Das Spiel erfolgt unter Echtzeitbedingungen und die Umwelt ist (wie man sich denken kann) sehr dynamisch. Das bedeutet, dass auch nur begrenzte Zeit für das Fällern von Entscheidungen zur Verfügung steht.

Die Simulation des Spiels erfolgte ursprünglich in 2D. Seit 2004 gibt zusätzlich es die 3D-Simulationsliga.

5.3. Das AT Humboldt

Das Agenten-Team der Humboldt-Universität (AT Humboldt) war 1997 Weltmeister in Nagoya, 1998 Vize-Weltmeister in Paris und 2004 Vize-Weltmeister in der neu geschaffenen 3D-Simulationsliga in Lissabon.

Implementiert wurde der Agent des AT Humboldt in C++ in Anlehnung an eine Kombination aus BDI- und einer 2-Paß-Schichten-Architektur.

Die wichtigsten Eigenschaften sind dabei:

- Feingliedrige Fähigkeiten: Initialisierungsaktionen von Fähigkeiten werden weggelassen, wenn sie nicht erforderlich sind. Beispielsweise wird es beim Schießen des Balls in vielen Fällen erforderlich sein, ihn vorher abzustoppen, um ihn dann in die richtige Richtung schießen zu können. Kommt der Ball aber günstig, ist es besser, diese Initialisierungsaktionen wegzulassen und ihn direkt zu kicken. Hier ist die erwähnte Selbstständigkeit der Fähigkeitskomponente zu erkennen.
- Der Agent nimmt nur bei größeren Abweichungen Plankorrekturen vor, geringfügige werden ignoriert. Hiermit wird die Anzahl der Kommandos minimiert, die der Agent abschickt, was ihn schneller macht.
- Pläne können zeitweise fanatisch verfolgt werden. Dies bedient die Forderung nach Stabilität im Verhalten. Hat sich ein Spieler beispielsweise entschlossen, an eine bestimmte Stelle zu rennen, so sollte er diesen Plan erst einmal zu Ende verfolgen und ihn nicht permanent überprüfen und korrigieren.
- Kooperation ohne Kommunikation: Spieler können aufgrund vorgegebener Annahmen über das Verhalten der anderen Spieler miteinander kooperieren ohne dabei das „say“-Kommando benutzen zu müssen. Man kann leicht erkennen, dass für diese Eigenschaft die zuvor erwähnte Stabilität im Verhalten essentiell ist.

Aufgrund wechselnder Teams mit stark unterschiedlicher Programmier-Erfahrung war für das Projektmanagement eine klare Strukturierung nach Vorgaben der BDI-Architektur äußerst wichtig.

Hieran kann man auch sehr gut erkennen, welchen Vorteil Architekturen, wie BDI, aber auch ein Schichtenmodell bringen. Bei aller Kreativität und guten Ideen ist es beim Entwerfen und Implementieren intelligenter Agenten eben enorm wichtig, das Problem in der richtigen Weise strukturieren zu können, nicht nur für die Entwurfs-Phase, sondern auch für das gesamte Projektmanagement.

6. Quellen

Literatur:

- Handbuch der künstlichen Intelligenz, Kap. 24.1 – 24.5.15
- S. Russel, P. Norvig: Artificial Intelligence - A Modern Approach, Kap. 2

Links:

- www.robocup.org
- www.robocup.de/AT-Humboldt/
- www.findarticles.com/p/articles/mi_m2483/is_n3_v19/ai_21210027/print