

(12) NACH DEM VERTRAG ÜBER DIE INTERNATIONALE ZUSAMMENARBEIT AUF DEM GEBIET DES
PATENTWESENS (PCT) VERÖFFENTLICHTE INTERNATIONALE ANMELDUNG

(19) Weltorganisation für geistiges Eigentum
Internationales Büro



(43) Internationales Veröffentlichungsdatum
1. Februar 2007 (01.02.2007)

PCT

(10) Internationale Veröffentlichungsnummer
WO 2007/012499 A2

(51) Internationale Patentklassifikation:

Nicht klassifiziert

(21) Internationales Aktenzeichen: PCT/EP2006/007527

(22) Internationales Anmeldedatum:

28. Juli 2006 (28.07.2006)

(25) Einreichungssprache: Deutsch

(26) Veröffentlichungssprache: Deutsch

(30) Angaben zur Priorität:

10 2005 035 903.5 28. Juli 2005 (28.07.2005) DE

(71) Anmelder (für alle Bestimmungsstaaten mit Ausnahme von US): **X-AITMENT GMBH** [DE/DE]; Science Park 2, 66123 Saarbrücken (DE).

(72) Erfinder; und

(75) Erfinder/Anmelder (nur für US): **SCHILD, Tobias** [DE/DE]; Jasperstrasse 5, 66333 Völklingen (DE). **GERBER, Andreas** [DE/DE]; Dresdner Str. 1, 66955

Pirmasens (DE). **VIERKE, Gero** [DE/DE]; Meissenwies 10, 66123 Saarbrücken (DE). **GERBER, Christian** [CH/CH]; Fliederweg 18, 67677 Enkenbach-Alsenborn (CH). **WILHELM, Markus** [DE/DE]; Karl-Marx-Strasse 136, 66280 Sulzbach/Saar (DE).

(74) **Anwalt: KÖLLNER, Malte**; Vogelweidstrasse 8, 60596 Frankfurt (DE).

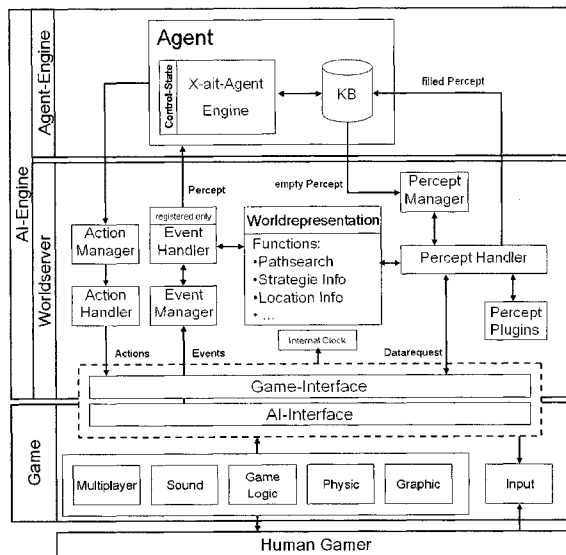
(81) **Bestimmungsstaaten** (soweit nicht anders angegeben, für jede verfügbare nationale Schutzrechtsart): AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LV, LY, MA, MD, MG, MK, MN, MW, MX, MZ, NA, NG, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RS, RU, SC, SD, SE, SG, SK, SL, SM, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) **Bestimmungsstaaten** (soweit nicht anders angegeben, für jede verfügbare regionale Schutzrechtsart): ARIPO (BW,

[Fortsetzung auf der nächsten Seite]

(54) Title: GENERIC KI-ARCHITECTURE FOR A MULTI-AGENT-SYSTEM

(54) Bezeichnung: GENERISCHE KI-ARCHITEKTUR FÜR EIN MULTIAGENTEN-SYSTEM



(57) **Abstract:** The invention relates to architecture of a computer program in order to implement a multi-agent-system. The architecture enables the agents to interact with a simulation or game world on a first plane and/or with robots in the real world_[GV1]. Said architecture has a second and third plane. Said second plane contains an abstract representation of the simulation world on the first plane which reduces on concepts. Said third plane implements the agents of the multi-agent-system. Interfaces are only arranged between the first and the second plane, and between the second and the third plane, not, however, between the first and the third plane. The artificial intelligence of the agents is implemented on the second and third planes such that the simulation world of the first plane can be widened, which leads to artificial intelligence. As a result, the architecture provides a KI-middleware for, for example, computer games.

[Fortsetzung auf der nächsten Seite]

WO 2007/012499 A2



GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), eurasisches (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), europäisches (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IS, IT, LT, LU, LV, MC, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Zur Erklärung der Zweibuchstaben-Codes und der anderen Abkürzungen wird auf die Erklärungen ("Guidance Notes on Codes and Abbreviations") am Anfang jeder regulären Ausgabe der PCT-Gazette verwiesen.

Veröffentlicht:

- *ohne internationalen Recherchenbericht und erneut zu veröffentlichen nach Erhalt des Berichts*

(57) Zusammenfassung: Es wird eine Architektur eines Computerprogramms zum Implementieren eines Multiagenten-Systems vorgeschlagen. Die Architektur erlaubt die Interaktion von Agenten mit einer Simulations- oder Spielwelt in einer ersten Ebene bzw. von Robotern mit der realen Welt_[GV1]. Dazu hat die Architektur eine zweite und eine dritte Ebene. Die zweite Ebene enthält eine abstrakte, auf Konzepte reduzierte Repräsentation der Simulationswelt der ersten Ebene. Die dritte Ebene implementiert die Agenten des Multiagenten-Systems. Schnittstellen bestehen lediglich zwischen der ersten und zweiten Ebene, sowie zwischen der zweiten und dritten Ebene, nicht jedoch zwischen der ersten und dritten Ebene. Die zweite und dritte Ebene implementieren Künstliche Intelligenz der Agenten, so dass die Simulationswelt der ersten Ebene um Agenten ergänzt werden kann, die über künstliche Intelligenz verfügen. Auf diese Weise stellt die Architektur eine KI-Middleware für z. B. Computerspiele zur Verfügung.

Generische KI-Architektur für ein Multiagenten-System

Beschreibung

Gebiet der Erfindung

Die gängigen Simulationsprogramme oder Computerspiele weisen eine sehr gute Grafik und Simulation der physikalisch relevanten Effekte auf, jedoch nur eine begrenzte KI (KI = Künstliche Intelligenz, im Englischen: AI = artificial intelligence) der vom Computer gesteuerten Spieler (Agenten). Während die Grafik und die Physik in Spielen über die Jahre hinweg stetig an Professionalität zugenommen haben, wurde die Implementierung der KI als notwendiges Übel hingenommen.

Stand der Technik

Das Forschungsgebiet der Multiagenten Systeme und Architekturen ist breit. Jörg Müller beschreibt in "The Design of Intelligent Agents" [JPM99] eine Reihe von unterschiedlichen Design- und Architekturansätzen. Insbesondere wird die Agentenarchitektur InteRRaP vorgestellt, in der die Fähigkeiten eines Agenten in die drei Bereiche "Reaktives Verhalten", "Rationales Planen" und "Kooperation" durch Schichten modelliert werden. In InteRRaP verfügt jede Schicht über eine eigene Wissensbasis. Der Agent verfügt über ein Interface, mit dem er auf seine virtuelle oder reale Umwelt einwirken kann. Nur die unterste Schicht hat dabei Zugriff auf das Interface. Die anderen Schichten müssen sich der Steuerung durch die Verhaltensschicht bedienen, um auf die Umgebung einzuwirken.

Aufgabe

Aufgabe der Erfindung ist es, den Einsatz von Künstlicher Intelligenz bei Simulationen und Computerspielen zu erleichtern.

5

Lösung

Diese Aufgabe wird durch die Erfindungen mit den Merkmalen der unabhängigen Ansprüche gelöst. Vorteilhafte Weiterbildungen der Erfindungen sind in den Unteransprüchen gekennzeichnet. Der Wortlaut sämtlicher Ansprüche wird hiermit durch Bezugnahme zum Inhalt dieser Beschreibung gemacht. Die Erfindung umfasst auch alle sinnvollen und insbesondere alle erwähnten Kombinationen von unabhängigen und/oder abhängigen Ansprüchen.

Die Entwicklung einer generischen KI Architektur (KI = Künstliche Intelligenz, im Englischen: AI = artificial intelligence) wurde durch die Misere aktueller Computerspiele in Bezug auf deren Künstliche Intelligenz für die vom Computer gesteuerten Spieler (Agenten) motiviert. Während die Grafik und die Physik in Spielen über die Jahre hinweg stetig an Professionalität zugenommen haben, wurde die Implementierung der KI als notwendiges Übel hingenommen.

In den Bereichen Grafik und Physik sind die Spielentwickler bereits soweit, dass eine Grafik- oder Physik-Middleware zum guten Ton gehört. Im Bereich KI gibt es hingegen keine Gesamtkonzepte, welche dem Spielentwickler die Programmierung einer eigenen KI abnehmen.

Mit der hier vorgestellten generischen KI-Architektur soll eine Plattform bereitgestellt werden, welche es dem Spielentwickler sehr leicht macht, eine KI zu integrieren, ohne sich mit dem Thema KI im Detail auseinander setzen zu müssen. Es soll eine KI-Engine geschaffen werden, die für einen Spielentwickler wie ein weiterer menschlicher Spieler wirkt. Wir haben uns deshalb als Ziel gesetzt, die KI-Architektur (Engine) so

generisch wie möglich zu halten. Die Architektur soll eine Umgebung für die Agenten bereitstellen, innerhalb deren sie interagieren können. Wie wir später sehen werden, erzeugt die generische Architektur ein Metalevel für die Spielumgebung, so
5 dass ein Agent in verschiedenen Spielen eingesetzt werden kann, ohne die genaue Umsetzung im Spiel berücksichtigen zu müssen.

Erfindungsgemäß wird zur Lösung der Aufgabe eine Recheneinheit zum Implementieren von Multiagenten-Systemen vorgeschlagen
10 mit Mitteln zum Implementieren einer ersten Ebene, die ein System darstellt. Diese Ebene definiert z. B. das zu implementierende Computerspiel und wird deshalb häufig auch als Game-Engine bezeichnet. Etwas allgemeiner gesprochen stellt die erste Ebene in der Regel eine Simulationswelt dar. Sie kann aber
15 auch die reale Welt sein, wenn die Agenten Roboter steuern. Allgemein gesprochen handelt es sich bei der ersten Ebene um ein System, das sich mit der Zeit verändert. Es kann durch Aktionen verändert werden. Ferner können sein Zustand bzw. Veränderungen seines Zustands wahrgenommen werden.

20 Ferner enthält die Recheneinheit Mitteln zum Implementieren einer zweiten Ebene, die einen Weltserver aufweist, der eine abstrakte, auf Konzepte reduzierte Repräsentation des Systems der ersten Ebene aufweist. Der Weltserver reduziert z. B. Objekte des Systems der ersten Ebene auf wenige Parameter.

25 Zwischen der ersten Ebene (Game-Engine) und der zweiten Ebene (Weltserver) gibt es eine Schnittstelle.

Schließlich enthält die Recheneinheit Mitteln zum Implementieren einer dritten Ebene, die Mittel zum Implementieren der Agenten des Multiagentensystems aufweist (Agent-Engine).

30 Zwischen der zweiten Ebene (Weltserver) und der dritten Ebene (Agent-Engine) gibt es ebenfalls eine Schnittstelle, nicht jedoch zwischen der ersten und dritten Ebene. Die zweite Ebene

(der Weltserver) trennt die erste Ebene (das Spiel bzw. die Game-Engine) von den Agenten.

Dieser Aufbau erlaubt eine heuristische Behandlung virtueller physikalischer Effekte, durch den Weltserver, wodurch eine wesentlich komplexere Welt mit höherer Recheneffizienz simuliert werden kann. Die Architektur erlaubt also eine wesentlich gesteigerte Verarbeitungsgeschwindigkeit von Simulationsproblemen.

Als Recheneinheit kommen z. B. ein Computer oder Computersystem, ein Mikrocontroller, ein DSP, ein FPGA oder ähnliches oder eine Mehrzahl davon in einem Netzwerk in Frage. Als Computersystem kommen sowohl ein Stand-alone Computer in Betracht, als auch ein Netzwerk von Rechnern, beispielsweise ein hausinternes, geschlossenes Netz, oder auch Rechner, die über das Internet miteinander verbunden sind. Ferner kann das Computersystem durch eine Client-Server-Konstellation realisiert sein, wobei Teile der Erfindung auf dem Server, andere auf einem Client ablaufen.

Vorzugsweise weist die Schnittstelle zur Kommunikation zwischen der ersten Ebene (Game-Engine) und der zweiten Ebene (Weltserver) eine Struktur auf, die der Struktur einer Schnittstelle zwischen der ersten Ebene und einem menschlichen Bediener oder Spieler entspricht. D. h. für eine Simulation oder ein Spiel wirkt die KI-Middleware (also der Weltserver und die Agenten) bzw. wirken die erfindungsgemäßen Agenten wie weitere menschliche Spieler. Es bedarf keiner aufwändigen Anpassung des Spiels und keiner gesonderten Programmierung der KI.

Zur Vereinfachung der Struktur und Erhöhung der Verarbeitungsgeschwindigkeit erfolgt die Kommunikation zwischen der zweiten und der dritten Ebene mittels der Schnittstelle über Aktionen und Perzepte. Eines darüber hinaus gehenden Austau-

sches bedarf es nicht. Mit anderen Worten: die zweite Ebene (der Weltserver) kapselt aus Sicht der Agenten die Daten des Systems der ersten Ebene (der Game-Engine) mithilfe von Aktionen und Perzepten.

5 Unter einer Aktion wird dabei eine Interaktion verstanden, die in das System der ersten Ebene eingeleitet wird und dieses verändern kann.

 Durch ein Perzept wird die Welt wahrgenommen. Durch ein Perzept kann ein Agent Zustände der Welt abfragen bzw. wahrnehmen.
10 Perzepte simulieren u. a. die Sinne eines Menschen. Perzepte haben den Vorteil, dass sie in sich abgeschlossene Datenpakete sind, welche nicht an Schnittstellendefinitionen gebunden sind.

 Damit ein Agent ein Perzept empfangen bzw. wahrnehmen kann,
15 muss er einen geeigneten Sensor aufweisen. Dieser simuliert z. B. ein Sinnesorgan, also z. B. die Fähigkeit zu Sehen oder zu Hören. Daher wird jedem Agent in der Regel mindestens ein vorgegebener virtueller Sensor zugeordnet, und die Perzepte werden passend zu den jeweiligen Sensoren des Agenten gefiltert.
20 Mit anderen Worten: es wird eine Filterung der den Agenten zur Verfügung stehenden Informationen danach vorgenommen, was die jeweiligen Agenten wahrnehmen können. Dadurch können sehr realistische und flexible Multitagenten-Systeme implementiert werden.

25 Die Aufgabe wird ferner durch eine Recheneinheit gelöst, die lediglich aus der zweiten und dritten Ebene besteht, sowie aus den Schnittstellen zwischen der ersten und zweiten Ebene und der zweiten und dritten Ebene. In der ersten Ebene ist ein System bereits vorhanden, was von der Recheneinheit selbst nicht
30 zur Verfügung gestellt wird; an das System der ersten Ebene wird lediglich angedockt. Eine derartige Konstellation kann zum Beispiel auftreten, wenn die Agenten der dritten Ebene Roboter

in der realen Welt steuern. Ebenso tritt diese Konstellation auf, wenn die zweite und dritte Ebene Künstliche Intelligenz für ein bereits vorhandenes Computerspiel in der ersten Ebene bereitstellen.

5

Die Aufgabe der Erfindung wird ferner durch einen Simulator für ein Multiagenten-Systems gelöst. Der Simulator hat Mittel zum Implementieren einer ersten Ebene, die ein System darstellt. Ferner hat er Mittel zum Implementieren einer zweiten
10 Ebene, die eine abstrakte, auf Konzepte reduzierte Repräsentation des Systems der ersten Ebene aufweist. Schließlich hat er noch Mittel zum Implementieren einer dritten Ebene, die Mittel zum Implementieren der Agenten des Multiagenten-Systems aufweist. Der Simulator hat eine Schnittstelle zur Kommunikation
15 zwischen der ersten und der zweiten Ebene, sowie eine Schnittstelle zur Kommunikation zwischen der zweiten und der dritten Ebene. Der Simulator kann sowohl in Hardware als auch in Software ausgeführt sein.

20 Die Aufgabe der Erfindung wird ferner durch eine Architektur bzw. Software-Architektur zum Implementieren eines Multiagenten-Systems gelöst. Die Architektur weist eine ersten Ebene auf, die ein System darstellt. Ferner eine zweite Ebene, die eine abstrakte, auf Konzepte reduzierte Repräsentation des Sys-
25 tems der ersten Ebene aufweist. Schließlich noch eine dritte Ebene, die Mittel zum Implementieren der Agenten des Multiagenten-Systems aufweist. Eine Schnittstelle sorgt für die Kommunikation zwischen der ersten und der zweiten Ebene. Eine weitere Schnittstelle für die Kommunikation zwischen der zweiten und
30 der dritten Ebene.

Die Aufgabe der Erfindung wird ferner durch eine Architektur eines Computerprogramms bzw. Software-Architektur zum Implemen-

tieren eines Multiagenten-Systems gelöst. Diese besteht lediglich aus der zweiten und dritten Ebene, sowie aus den Schnittstellen zwischen der ersten und zweiten Ebene und der zweiten und dritten Ebene. In der ersten Ebene ist ein System bereits
5 vorhanden, was von der Architektur selbst nicht zur Verfügung gestellt wird; an das System der ersten Ebene wird lediglich angedockt.

Die Aufgabe der Erfindung wird ferner durch eine Middleware
10 für die Entwicklung eines Computerprogramms zum Implementieren eines Multiagenten-Systems gelöst. Die Middleware beruht auf der beschriebenen Software-Architektur, enthält jedoch als Middleware folglich nicht die komplette Implementation oder das ganze Spiel. Insbesondere ist das System der ersten Ebene nicht
15 Teil der Middleware. Die Middleware ist so aufgebaut, dass sie diese ergänzt. Daher stellt die Middleware lediglich die zweite und dritte Ebene zur Verfügung, die die Fähigkeiten der Agenten bereitstellt. Dadurch können Simulationen oder Computerspiele um die KI-Fähigkeiten von Agenten erweitert werden bzw. diese
20 können bei der Entwicklung leicht integriert werden. Die Middleware ist damit ein Werkzeug oder Software-Tool für die Entwicklung von Simulationen oder Computerspielen.

Die Aufgabe der Erfindung wird ferner durch ein Verfahren
25 gelöst. Im Folgenden werden einzelne Verfahrensschritte näher beschrieben. Die Schritte müssen nicht notwendigerweise in der angegebenen Reihenfolge durchgeführt werden, und das zu schildernde Verfahren kann auch weitere, nicht genannte Schritte aufweisen.

30 Gemäß dem vorgeschlagenen Verfahren zum Implementieren eines Multiagenten-Systems wird zunächst eine erste Ebene implementiert, die ein System darstellt. Sodann wird eine zweite Ebene implementiert, die eine abstrakte, auf Konzepte reduzierte Rep-

räsentation des Systems der ersten Ebene aufweist. Schließlich wird eine dritte Ebene implementiert, die Mittel zum Implementieren der Agenten des Multiagenten-Systems aufweist.

Informationen werden zwischen der ersten und der zweiten Ebene, sowie zwischen der zweiten und der dritten Ebene ausgetauscht, nicht jedoch zwischen der ersten und der dritten Ebene.

Ferner wird die Aufgabe gelöst durch ein Computerprogramm, das bei Ablauf auf einer Recheneinheit, einem Mikrocontroller, DSP, FPGA oder Computer oder auf einer Mehrzahl davon in einem Netzwerk das erfindungsgemäße Verfahren in einer seiner Ausgestaltungen ausführt.

Weiterhin wird die Aufgabe gelöst durch ein Computerprogramm mit Programmcode-Mitteln, um das erfindungsgemäße Verfahren in einer seiner Ausgestaltungen durchzuführen, wenn das Programm auf einer Recheneinheit, einem Mikrocontroller, DSP, FPGA oder Computer oder auf einer Mehrzahl davon in einem Netzwerk ausgeführt wird. Insbesondere können die Programmcode-Mittel auf einem computerlesbaren Datenträger gespeicherte Instruktionen sein.

Außerdem wird die Aufgabe gelöst durch einen Datenträger, auf dem eine Datenstruktur gespeichert ist, die nach einem Laden in einen Arbeits- und/oder Hauptspeicher einer Recheneinheit, eines Mikrocontrollers, DSPs, FPGAs oder Computers oder einer Mehrzahl davon in einem Netzwerk das erfindungsgemäße Verfahren in einer seiner Ausgestaltungen ausführen kann.

Auch wird die Aufgabe gelöst durch ein Computerprogramm-Produkt mit auf einem maschinenlesbaren Träger gespeicherten Programmcode-Mitteln, um das erfindungsgemäße Verfahren in einer seiner Ausgestaltungen durchzuführen, wenn das Programm auf einer Recheneinheit, einem Mikrocontroller, DSP, FPGA oder Computer oder auf einer Mehrzahl davon in einem Netzwerk ausge-

führt wird. Dabei wird unter einem Computer-Programmprodukt das Programm als handelbares Produkt verstanden. Es kann grundsätzlich in beliebiger Form vorliegen, so zum Beispiel auf Papier oder einem computerlesbaren Datenträger und kann insbesondere
5 über ein Datenübertragungsnetz verteilt werden.

Schließlich wird die Aufgabe gelöst durch ein moduliertes Datensignal, welches von einer Recheneinheit, einem Mikrocontroller, DSP, FPGA oder Computer oder von einer Mehrzahl davon in einem Netzwerk ausführbare Instruktionen zum Ausführen des
10 erfindungsgemäßen Verfahrens in einer seiner Ausgestaltungen enthält.

Als Computersystem zum Ausführen des Verfahrens kommen sowohl ein Stand-alone Computer oder Mikrocontroller, DSPs oder FPGAs in Betracht, als auch ein Netzwerk von Mikrocontrollern,
15 DSPs, FPGAs oder Rechnern, beispielsweise ein hausinternes, geschlossenes Netz, oder auch Rechner, die über das Internet miteinander verbunden sind. Ferner kann das Computersystem durch eine Client-Server-Konstellation realisiert sein, wobei Teile der Erfindung auf dem Server, andere auf einem Client ablaufen.

Weitere Einzelheiten und Merkmale ergeben sich aus der nachfolgenden Beschreibung von bevorzugten Ausführungsbeispielen in Verbindung mit den Unteransprüchen. Hierbei können die jeweiligen Merkmale für sich alleine oder zu mehreren in Kombination
25 miteinander verwirklicht sein. Die Möglichkeiten, die Aufgabe zu lösen, sind nicht auf die Ausführungsbeispiele beschränkt.

Die Ausführungsbeispiele sind in den Figuren schematisch dargestellt. Gleiche Bezugsziffern in den einzelnen Figuren bezeichnen dabei gleiche oder funktionsgleiche bzw. hinsichtlich
30 ihrer Funktionen einander entsprechende Elemente. Im Einzelnen zeigt:

Fig. 2-1 einen Überblick der Komponenten in der KI Engine;

- Fig. 3-1 ein schematisches Modell eines X-Ait-Agenten;
- Fig. 4-1 eine schematische Darstellung des Ablaufs, wenn eine Aktion ins Spiel gesendet wird;
- Fig. 4-2 eine schematische Darstellung des Ablaufs, wenn ein Sprechakt von Agent A zu Agent B gesendet wird;
- Fig. 4-3 eine schematische Darstellung des Ablaufs, wenn ein Event vom Spiel an mehrere Agenten in der KI-Engine gesendet wird;
- Fig. 4-4 eine schematische Darstellung des Ablaufs, wenn eine Perzeptanfrage von Agent A gestellt wird, die durch den Cache in der Weltrepräsentation befriedigt werden kann;
- Fig. 4-5 eine schematische Darstellung des Ablaufs, wenn eine Perzeptanfrage von Agent A gestellt wird, wobei die Informationen in der Weltrepräsentation nicht aktuell genug sind, und eine Anfrage an das Spiel gestellt werden muss;
- Fig. 4-6 eine schematische Darstellung des Ablaufs, wenn eine Perzeptanfrage von Agent A gestellt wird, wenn die angeforderten Daten immer so aktuell sein müssen, dass es sich nicht lohnt, Daten in der Weltrepräsentation zwischenzuspeichern;
- Fig. 5-1 eine schematische Darstellung der KI-Engine in einer Singlethread Umgebung; und
- Fig. 5-2 eine schematische Darstellung der KI-Engine in einer Multithread Umgebung.

1 INHALTSVERZEICHNIS DER AUSFÜHRLICHEN FIGURENBESCHREIBUNG

2 ÜBERBLICK DER KOMPONENTEN

2.1 DER MENSCHLICHE SPIELER

2.2 SPIEL

2.2.1 Schnittstelle zwischen Spiel und KI Engine

2.3 WELTSERVER

2.3.1 Action-Manager

2.3.2 Action-Handler

5 2.3.3 Event-Manager

2.3.4 Event-Handler

2.3.5 Perzept-Manager

2.3.6 Perzept-Handler

2.3.7 Perzept-Plugins

10 2.3.8 Interne Uhr

2.3.9 Weltrepräsentation

2.4 AGENT ENGINE

2.4.1 Agent

2.4.1.1 X-ait-Agent Engine

15 2.4.1.2 Wissensbasis

3 DER X-AIT-AGENT

3.1 Die Architektur der X-Ait-Agenten

3.2 DAS BEVORZUGTE MODIFIZIERTE INTERRAP MODEL DES X-AIT-AGENTEN

20 3.2.1 Aktionen

3.2.2 Sprechakte

3.2.3 Perzepte

3.2.4 Perzept InQueue

3.2.5 Perzept Verteilung

25 3.2.6 Kooperationsebene

3.2.7 Lokale Planungsebene

3.2.8 Reaktivebene

3.2.9 Lokale Wissensbasis

4 DATENFLUSS

30 4.1 AKTIONEN INS SPIEL SENDEN

4.2 SPRECHAKTE VON EINEM AGENTEN ZUM ANDEREN VERSENDEN

4.3 SENDEN EINES EVENTS VOM SPIEL IN DIE KI-ENGINE

4.4 PERZEPTANFRAGE AN DAS SPIEL (PULL) - FALL 1

4.5 PERZEPTANFRAGE AN DAS SPIEL (PULL) - FALL 2

4.6 PERZEPTANFRAGE AN DAS SPIEL (PULL) - FALL 3

5 RECHENZEITKONTROLLE

5.1 GLOBALE RECHENZEITKONTROLLE

5 5.1.1 Singlethread Umgebung

5.1.2 Multithread Umgebung

5.2 LOKALE RECHENZEITKONTROLLE

5.2.1 Komponenten-Scheduler

5.2.2 Scheduler-FSM

10

2 Überblick der Komponenten

Fig. 2-1 zeigt den allgemeinen Datenfluss der KI Komponenten und des Spiels (Game). Um Daten nicht redundant zu halten, trennt der Weltserver (Worldserver) das Spiel von den Agenten und kapselt aus Sicht der Agenten (Agents) die Daten der Spiel-Engine mithilfe von Aktionen (Actions), Perzepten (Percepts). So kann die Entwicklung der KI-Engine getrennt von der Implementierung der Spiel-Engine gehalten werden. Der Weltserver stellt den Agenten also alle Informationen über das Spiel zur Verfügung und hält die Datenquellen transparent für die Agenten, denn die von den Agenten benötigten Daten können entweder schon in der globalen Weltrepräsentation (Worldrepresentation) gepuffert sein, oder müssen direkt aus der Spiel-Engine ausgelesen werden. Ziel ist es hierdurch eine Trennung der Logik in Agenten und Daten im Spiel zu ermöglichen. Lediglich die Action-Handler und Perzept-Handler greifen auf Interface-Methoden zu, die in der Spiel-Engine bereitgestellt werden müssen. Die Aktionen und Perzepte bieten, zusammen mit den jeweiligen Handlern, den Agenten eine standardisierte Schnittstelle für Daten und Funktionsaufrufe zur Spiel-Engine. So können bei einem äquivalenten Spiel (gleiches Genre) die Implementierung der Agenten zu 100% wieder verwendet werden und lediglich die Interface-Methoden innerhalb der Spiel-Engine müssen implementiert werden, damit sie die sehr einfach gestrickten Aktionen und die Datenanforderungen (Datarequest) ausführen können. Die Datenanforderungen sind eine weitere Form der Perzepte.

Die Schnittstelle zwischen dem Spiel und Weltserver bzw. Agent wurde so konzipiert, dass sie vom Prinzip her mit den Ein- und Ausgaben eines menschlichen Spielers übereinstimmt. Dies kann man in Fig. 2-1 am Datenfluss des KI-Interfaces (AI-Interface) in Richtung Spiel und dem eines menschlichen Spielers erkennen.

2.1 Der menschliche Spieler

Betrachtet man den menschlichen Spieler (Human Gamer), so hat dieser ein sehr begrenztes Interface um den Spielablauf zu manipulieren. Dies beschränkt sich im Wesentlichen auf die Ihm zur Verfügung stehenden Eingabegeräte (Tastatur, Maus, Joystick). Damit das Spiel dem Spieler Feedback über seine Aktionen mitteilen kann, gibt es Ausgabegeräte wie Monitor und Soundausgabe. Diese Dinge reichen, damit der Spieler genügend Informationen aus dem Spiel entnehmen kann, um den Spielablauf erfolgreich zu kontrollieren. Diesen Ansatz werden wir auch in der KI - Engine verwenden, wie man später sieht.

2.2 Spiel

Das Spiel, bestehend aus verschiedensten Komponenten wie Grafik, Sound, Physik usw., wird hier nicht näher erläutert. Der entscheidende Teil innerhalb des Spiels stellt das KI-Interface dar. Dieses erlaubt die Kommunikation der KI-Engine mit dem eigentlichen Spiel und den darin enthaltenen Komponenten.

2.2.1 Schnittstelle zwischen Spiel und KI-Engine

Die Schnittstelle zwischen Spiel und KI-Engine wurde nach dem Vorbild des menschlichen Spielers modelliert. Wir wollen dazu die auszutauschenden Daten auf ein Minimum reduzieren und beziehen uns nur auf Informationen die auch ein menschlicher Spieler erhalten kann, oder an das Spiel übergeben kann.

Die gesamte Schnittstelle zwischen Spiel und KI-Engine ist in zwei Teile unterteilt. Zum einen gibt es das KI-Interface und zum anderen das Game-Interface. Eine Aufgabe, die in beiden Schnittstellenteilen vorgenommen wird, ist die Konvertierung von IDs (Identifikatoren). Das Spiel sowie die KI-Engine vergeben verschiedene IDs für das gleiche Spielobjekt. Innerhalb des Spiels gibt es IDs für bewegliche Spielobjekte, die eine künst-

liche Intelligenz durch den Agenten in der KI-Engine erhalten sollen, welcher wiederum eine andere ID hat. Die verschiedenen IDs ermöglichen eine bessere Anordnung der Agentendatenstrukturen. Damit dies transparent für das Spiel und die KI-Engine vonstatten geht, werden die gängigen IDs innerhalb der Aktionen, Perzepte und Events in der Schnittstelle automatisch konvertiert. Für die Spezialfälle sind die Handler verantwortlich (siehe unten).

10 2.2.1.1 KI - Interface

Das KI-Interface (AI-Interface) ist Teil des Spiels und gibt dem Spiel die Möglichkeit, Informationen an die KI-Engine zu senden. Dazu gehört das Senden von Events und das Initialisieren sowie das manipulieren von Parametern der KI-Engine. Events werden von der Spiel-Engine erzeugt, wenn z. B. ein Sound abgespielt wird. In einem Event sind nur rudimentäre Informationen enthalten, wie z. B. wer das Event ausgelöst hat und um welches Event es sich handelt. Damit die KI-Engine Rechenzeit bekommen kann, gibt es eine Update Methode, welche den Datenfluss der KI-Engine in das Spiel startet und ebenfalls die KI interne Spieluhr synchronisiert. Die komplette Funktionsreferenz zum KI-Interface findet sich in Kapitel 6 (siehe unten).

2.2.1.2 Game - Interface

25 Das Spiel-Interface (Game-Interface) befindet sich in der KI-Engine und ermöglicht die Kommunikation der KI-Engine in Richtung des Spiels. Dabei gibt es zwei Hauptfunktionen mit der die Kommunikation zum Spiel stattfindet:

30 PerformAction:

Aktionen, welche in der KI-Engine generiert wurden, werden in das Spiel gesendet. Eine Aktion ist eine Nachricht in Form eines C++ Structs. Diese enthält elementare Befehle

wie "bewege links", "bewege rechts", "nimm", usw.. Diese Befehle entsprechen den Aktionen, die ein menschlicher Nutzer oder Spieler erzeugen kann, der z. B. Eingaben über eine Tastatur tätigt. Sie müssen in gleicher Weise wie Eingaben eines menschlichen Spielers durch eine Spiel-Engine umgesetzt werden können. Dies erleichtert oder ermöglicht die Verarbeitung durch die Game-Engine. Die PerformAction Methode ruft dazu eine zur Initialisierung der KI-Engine definierte Callback-Funktion im Spiel auf. Diese Callback-Funktion muss nun die Aktionen in der Spielwelt für einen bestimmten Agenten umsetzen. Die Aufrufe der Callback-Funktion werden durch das Spiel zu einem vom Spiel definierten Punkt getriggert. Beispiele von Aktionen und einer entsprechenden Callback-Funktion gibt es in Abschnitt 6.5.

RequestData:

Die RequestData Funktion verwendet ebenfalls, wie die PerformAction Methode, eine Callback-Funktion um Daten aus dem Spiel abzurufen. Die Aufgabe der Funktion ist das Beantworten bzw. Befüllen einer Datenanforderung (DataRequest) der KI-Engine an das Spiel. Dies soll die im Vergleich zum menschlichen Spieler fehlende Monitorausgabe ersetzen. Die abgefragten Daten orientieren sich aber an den in Spielen vorhandenen Datentypen, damit keine komplizierten Konvertierungen vorgenommen werden müssen.

2.3 Weltserver

Der Weltserver (Worldserver) innerhalb der KI-Engine stellt einen gewissen Abstraktionslevel zwischen dem eigentlichen Spiel und den darin agierenden Agenten zur Verfügung. Die verschiedenen Agenten können mit dem Weltserver über Aktionen und Perzepte interagieren. Der Weltserver nimmt Aktionen und Perzepte von den Agenten entgegen und wandelt sie so um, so dass

das Spiel die Daten richtig interpretieren kann. Dadurch müssen für verschiedene Spiele nur das KI-Interface und evtl. Teile des Weltservers angepasst werden, damit ein unveränderter Agent mit dem Spiel interagieren kann. Der Weltserver besteht aus
5 verschiedenen Komponenten, welche den Datenfluss vom Agenten zur Spiel-Engine und zurück leiten.

2.3.1 Action-Manager

Der Actionmanager übernimmt das Timing von Aktionen, die ins
10 Spiel geleitet werden müssen. Dabei besteht der Actionmanager hauptsächlich aus einer Queue, welche Aktionen entgegen nehmen kann und einer 1-zu-1 Zuweisung, die einem bestimmten Typ von Aktion genau einen Actionhandler zuweist. Aktionen können von dem Actionmanager jeder Zeit von den Agenten entgegen genommen
15 werden. Diese werden dann zuerst in eine FIFO-Queue gesteckt. In regelmäßigen Abständen wird dem Actionmanager Zeit zugewiesen, damit dieser die in seiner Queue befindlichen Daten abarbeiten kann. Dieser Zeitpunkt wird durch das Spiel bestimmt, da es bei der Abarbeitung der Aktionen zu Callback-Aufrufen innerhalb des Spiels kommt. Nachdem der Actionmanager nun Zeit be-
20 kommen hat, nimmt er eine Aktion nach der anderen aus der Queue und ruft für den jeweiligen Typ der Aktion den passenden Actionhandler auf, wobei er dem Actionhandler die Aktion übergibt. Danach wird die Referenz auf die Aktion aus der Queue
25 entfernt und mit der nächsten Aktion fortgefahren. Dieser Vorgang endet erst, wenn die komplette Queue abgearbeitet ist. Hier gibt es keine Unterteilung in Rechenschritte oder Rechenzeit. Dies würde dazu führen, dass ein stetig zu geringer Anteil an freier Rechenzeit, die Queue überlaufen lassen würde.
30 Durch das komplette Abarbeiten der Queue wird das Spiel soweit ausgebremst, dass es mit den Eingaben der KI-Engine zurechtkommt. Dies ähnelt dem Prinzip einer Multiplayer Engine, welche alle Daten der Klienten bearbeiten muss, damit ein korrektes

Gameplay zustande kommt. Da wir nicht immer für alle möglichen Aktionen die entsprechenden Actionhandler laden möchten, gibt es bei dem Actionmanager die Möglichkeit einzelne Actionhandler für eine gewisse Aktion zu registrieren. Will ein Agent eine
5 gewisse Aktion an das Spiel senden, so muss er sich zu Beginn (während seiner Initialisierung) am Weltserver für eine bestimmte Aktion registrieren. Der Weltserver sorgt dann dafür, dass ein entsprechender Actionhandler angelegt wird, und am Actionmanager für die angeforderte Aktion registriert wird.

10

2.3.2 Action-Handler

Ein Actionhandler führt Operationen für einem bestimmten Typ von Aktionen aus. Es kann zu jedem Actionhandler nur genau eine Aktion geben, die er bearbeiten kann, aber es kann für eine Ak-
15 tion mehr als einen Actionhandler geben. Es existiert somit eine 1-zu-x Beziehung zwischen Aktion und Actionhandler. Dies hat den Vorteil, dass wir in einem Actionhandler verschiedene Nachbearbeitungsoperationen durchführen können, welche eine Aktion auf das jeweilige Spiel anpassen. Dabei können keine neuen In-
20 formationen in die Aktion eingeführt werden, aber die in der Aktion enthalten Daten können konvertiert werden, wenn dies nötig ist. Der Agent kann somit weiterhin die ihm bekannten Aktionen schicken, ohne die Variation im Spiel beachten zu müssen. Eine der häufigsten Anwendungen für den Actionhandler ist die
25 Konvertierungen von IDs. So können Aktionen, zu den bereits standardmäßig vorhandenen IDs, welche in der Schnittstelle konvertiert werden, weitere IDs über Spielobjekte enthalten. Diese müssen ebenfalls konvertiert werden, damit das Spiel die Aktion richtig verarbeiten kann.

30

2.3.3 Event-Manager

Der Eventmanager übernimmt das Timing für das Versenden von Events in die KI-Engine. Er besteht, ebenso wie der Actionmana-

ger, aus einer Queue welche neue Events entgegen nehmen kann und besitzt eine Zuweisung von Events zu Eventhandlern. Hier haben wir wieder eine 1-zu-x Beziehung, da es zu jedem Event mehr als einen Eventhandler geben kann. Diese Mehrfachbeziehung
5 wird zur Initialisierungsphase der KI-Engine auf eine 1-zu-1 Beziehung festgelegt.

Ähnlich wie beim Actionmanager wollen wir nicht immer für alle möglichen Events die Eventhandler laden. Damit ein bestimmter Agent Events aus dem Spiel empfangen kann, muss er
10 dies dem Eventmanager mitteilen. Dazu registriert er sich am Weltserver für ein gewisses Event. Dies führt dazu, dass der Weltserver einen neuen Eventhandler anlegt und diesen beim Eventmanager registriert. Zusätzlich wird nun noch der Agent mit seiner ID sowie seiner SubID beim Eventmanager für ein gewisses
15 Event registriert. Die Registrierungsinformation des Agenten wird dabei direkt im entsprechenden Eventhandler hinterlegt. Diese Information benötigt der Eventhandler, damit er bei einem eingehenden Event, das daraus resultierende Perzept an den korrekten Empfänger senden kann. Die bereits o. g. SubID ist eine
20 ID, welche einzelne Komponenten innerhalb des Agenten adressiert. Damit umgeht man einen weiteren Verwaltungsaufwand für das Empfangen von Perzepten innerhalb des Agenten.

Der Eventmanager kann zu jedem Zeitpunkt neue Events aus dem Spiel aufnehmen. Diese werden aber solange in der FIFO-Queue
25 gespeichert, bis der Weltserver dem Eventmanager Rechenzeit zuteilt, so dass dieser die Events weiterverarbeiten kann. Hat der Eventmanager einmal die Rechenkontrolle, so arbeitet er sämtliche Events in seiner Queue ab. Dies verhindert ein Überlaufen der Queue in dem Fall dass immer zu wenig Rechenzeit zur
30 Verfügung steht. Dazu ruft er zu jedem Event den passenden Eventhandler auf, und übergibt das Event an den Eventhandler. Im Gegensatz zum Actionmanager wird das Event nicht in jedem Fall aus der Queue gelöscht. Sollte der Eventhandler mit der Bear-

beitung des Events nicht fertig werden, so wird die weitere Bearbeitung des Events bis zum nächsten Aufruf verschoben, indem das Event in der Queue verbleibt.

5 2.3.4 Event-Handler

Ein Eventhandler führt Operation auf einem bestimmten Event aus. Die Aufgabe der Eventhandler ist die Transformation eines Events in ein Perzept und das Update der globalen Weltrepräsentation (Worldrepresentation) wenn dies für ein Event notwendig ist. Jeder Eventhandler hat eine Liste von Agenten IDs und zugehörigen SubIDs für die er Perzepte aus dem Event erzeugen soll. Dazu nimmt er das Event und führt im ersten Schritt ein Update der globalen Weltrepräsentation durch, wenn dies für das Event notwendig ist. Dies kann z. B. ein Soundevent sein, welches zuerst durch die Welt propagiert werden muss, damit man feststellen kann, welchen Agent dieses Soundevent überhaupt erreicht. Da diese Berechnungen teilweise sehr lange dauern können, länger als der KI-Engine Rechenzeit pro Frame zusteht, kann ein komplexer Algorithmus an einer definierten Stelle die Verarbeitung unterbrechen und dem Eventmanager mitteilen, dass er für genau dieses Event wieder aufgerufen werden will. Der Eventmanager verschiebt dann die weitere Ausführung der Eventverarbeitung auf den nächsten Frame.

Ein Frame ist ein Einzelbild einer Simulation oder eines Computerspiels. Typischerweise möchte man davon ca. 25, 50 oder mehr pro Sekunde dem menschlichen Nutzer darbieten, damit Bewegungen fließend wirken und nicht ruckeln. Daher stehen für die Berechnung eines Frames in der Regel etwa 10-40 ms zur Verfügung.

30 Wurde nun ein Update der Weltrepräsentation durchgeführt kann die Eventbearbeitung fortgeführt werden. Es wird nun für jede beim Eventhandler registrierte Agentenkomponente ein Per-

zept erzeugt und dieses an den jeweiligen Agenten weitergeleitet.

Neben dem Umkopieren der Daten aus dem Event-Struct in ein Perzept-Struct kann der Eventhandler weitere Operationen durchführen. So kann er z. B. bei einem Soundevent weitere Informationen für den Agenten erzeugen, wie die Lautstärke des bei ihm empfangen Sounds, wenn dies nicht bereits durch die Spiele-Engine passiert ist. Desweiteren fällt die Aufgabe der Konvertierung von IDs an, für IDs die über den Standard im Event-Struct hinausgehen. Man kann also das Event im Eventhandler beliebig aufbereiten, da man eine andere Datenstruktur, das Perzept, weiterleitet.

2.3.5 Perzept-Manager

Der Perzeptmanager übernimmt das Timing zum Versenden von Perzepten an die Spiel-Engine. Wie jeder andere bisher vorgestellte Manager besitzt auch der Perzeptmanager eine FIFO-Queue, welche neue Perzepte vom Agenten entgegen nimmt. Damit die Perzepte bearbeitet werden können gibt es wieder eine Zuweisungstabelle von Perzepten zu Perzepthandlern. Zwischen Perzepten und Perzepthandlern gibt es eine 1-zu-x Beziehung, damit man auf verschiedene Arten und Weisen an die Informationen für ein Perzept herankommen kann. Zur Initialisierungsphase werden die entsprechenden Perzepthandler festgelegt, damit man auf eine 1-zu-1 Beziehung kommt. Da wir auch hier nicht jedes Mal für alle möglichen Perzepte die entsprechenden Perzepthandler laden wollen, benutzen wir auch hier das Prinzip des Registrierens. Will nun ein Agent eine Perzeptanfrage durchführen, so muss er sich beim Weltserver für die jeweiligen Perzepte registrieren. Falls nicht bereits ein Perzepthandler für das jeweilige Perzept existiert, so wird im Weltserver ein neuer Perzepthandler angelegt und dieser dann beim Perzeptmanager registriert. Nach

der Registrierung kann der Agent Perzepte an den Perzeptmanager schicken.

Für die Bearbeitung der Perzepte in der FIFO-Queue unterscheiden wir zwei Bearbeitungsschritte. Zum einen haben wir den
5 FetchData Schritt und zum anderen den ProcessData Schritt. Diese Unterteilung haben wir vorgenommen, um die Zeit während der Synchronisation zwischen dem Spiel und KI-Thread zu minimieren. Im FetchData Schritt soll der Perzepthandler lediglich Daten für die Bearbeitung des Perzepts beim Spiel auslesen. Die ausgelesenen Daten werden zum Perzeptmanager durchgereicht, so
10 dass dieser sie für den ProcessData Schritt zwischenspeichern kann. Diesen Vorgang führt der Perzeptmanager für alle Perzepte in seiner Queue durch, ohne die Queue dabei zu löschen. Nachdem man für alle Perzepte einmal den FetchData Schritt ausgeführt
15 hat, ist dieser Schritt abgeschlossen. Der Perzeptmanager gibt dann die Rechenkontrolle zurück an den Weltserver. Der zweite Schritt ist ein ProcessData Schritt. Während dieses Schritts, nimmt der Perzeptmanager jeweils ein Perzept mit dem zugehörigen DataRequest aus der Queue und führt damit den passenden
20 Perzepthandler mit dem ProcessData Schritt aus. In diesem Schritt kann der Perzepthandler die zuvor gesammelten Daten bearbeiten und in einem Perzept zurück an den Agenten schicken. Zusätzlich kann der Perzepthandler dem Perzeptmanager mitteilen, dass er die Bearbeitung des Perzepts nicht in angemessener
25 Zeit abschließen konnte, und beim nächsten ProcessData wieder aufgerufen werden will. Der Perzeptmanager merkt sich diese Perzepte und führt für diese beim nächsten Mal keinen FetchData Schritt mehr durch. Im ProcessData Schritt werden alle Perzepte abgearbeitet bis die Queue leer ist, damit es nicht zu einem
30 Überlauf in der Queue kommt, wenn zu wenig Rechenzeit für die KI-Engine zur Verfügung steht.

2.3.6 Perzept-Handler

Ein Perzepthandler wird dazu benötigt, ein gewisses Perzept mit Informationen aus dem Spiel zu füllen. Er stellt damit eine generelle Schnittstelle zwischen den Datenformaten der Agenten
5 und dem Spiel dar.

Wie bereits im Perzeptmanager erklärt, gibt es zwei verschiedene Bearbeitungsschritte, die jeder Perzepthandler beherrschen muss. Zum einen ist dies der FetchData Schritt. Hier sammelt der Perzepthandler die Informationen aus dem Spiel, die
10 er benötigt um das Perzept mit Informationen zu füllen. Die Daten bekommt er über die so genannten DataRequest Structs. Diese werden durch das Game-Interface an das Spiel gesendet und dort mit einer Callback Funktion befüllt. Nach dem Rückkehren der Callback Funktion ist das DataRequest mit gültigen Daten ge-
15 füllt. Die gefüllten DataRequest werden an den Perzeptmanager gereicht. Dies hat den Vorteil, dass man keine perzeptspezifischen Informationen im Perzepthandler speichern muss. Der Perzepthandler ist lediglich für die Bearbeitung der Perzepte verantwortlich, und deshalb existiert für alle Perzepte eines Typs
20 nur ein Perzepthandler. Nachdem alle Daten vom Spiel angefordert wurden, ist der Schritt FetchData abgeschlossen.

Im Bearbeitungsschritt ProcessData bekommt der Perzepthandler zum einen das zu füllende Perzept, und zum anderen eine Menge von DataRequests, die er zuvor in FetchData gesammelt
25 hat. Mit diesen Daten füllt nun der Perzepthandler das Perzept. Sollte er für die Arbeit zu lange benötigen, so kann er dem Perzeptmanager mitteilen, dass er beim nächsten Durchgang erneut aufgerufen werden will. Für die Bearbeitung der Daten kann der Perzepthandler zusätzliche Perzept Plugins (Percept Plugins)
30 gins) sowie die Weltrepräsentation hinzuziehen. Während der Berechnung unterliegt der Perzepthandler einem Level of Detail (LOD) Mechanismus. Der LOD entspricht dabei der Priorität des jeweiligen Agenten, welcher das Perzept angefordert hat. Sollte

der Agent eine geringe Priorität haben, so sollte der Perzepthandler intensive Rechenoperationen vermeiden. Dies kann man am Beispiel von Synthetic Vision (die synthetische Sicht eines Agenten auf die virtuelle Welt) während einer Perzeptanfrage erklären. Das Perzept Plugin Synthetic Vision verbraucht sehr viel Rechenzeit. Dabei stellt das Plugin die sichtbaren Objekte innerhalb eines Bildes fest. Man kann dies aber auch dadurch approximieren, dass man direkt bei der Spiel-Engine anfragt, welche Objekte innerhalb eines definierten einsehbaren Bereichs liegen.

Ist das Perzept nach den Berechnungen komplett gefüllt, wird es an den Agenten zurückgesendet, welcher als Absender im Perzept genannt ist.

2.3.7 Perzept-Plugins

Ein Perzept Plugin unterstützt den Perzepthandler bei der Berechnung der Perzepte. Ein Perzept Plugin kann z.B. Synthetic Vision sein, das aus einem Bild die sichtbaren Objekte extrahiert.

2.3.8 Interne Uhr

Die interne Uhr speichert die aktuelle Spieluhr. Diese ist nicht zu verwechseln mit der Systemuhr. Es gibt viele Spiele die ihre eigene Spieluhr haben, damit sie Zeitlupen realisieren können.

2.3.9 Weltrepräsentation

Die Weltrepräsentation dient dazu, die Spielwelt für den Agenten zu abstrahieren. Die Daten in der Weltrepräsentation können in einem Offline Prozess gewonnen werden, oder als Online Prozess während des Spiels. Eine der Hauptaufgaben der Weltrepräsentation ist die Speicherung eines Navigationsgraphen, damit die Agenten in der virtuellen Welt navigieren können. Für

die verschiedenen Spielgenres gibt es verschiedene Formen der Weltrepräsentation, welche über ein einheitliches Interface angesteuert werden. Beispiele wären hier zum einen der Navigations-Mesh-Graph, welcher häufig Verwendung in First-Person-Shootern findet, und zum anderen eine Heightmap zur Navigation in Echtzeit Strategie Spielen.

Die Weltrepräsentation kann auch als Cache für Informationen dienen, die man nicht immer für alle Agenten aus dem Spiel entnehmen will. Dies kann zum Beispiel die Position von gewissen Objekten sein. Zudem hat man die Möglichkeit effizient Nachbarschaftstests auszuführen, um z. B. Kollisionen zu vermeiden, wenn dies das Spiel nicht bereits zur Verfügung stellt.

2.4 Agent-Engine

Die Agent-Engine symbolisiert alle Agenten die sich in der KI-Engine befinden. Die Agentenverwaltung sowie die Rechenzeitverwaltung der Agenten werden vom Weltserver durchgeführt. An dieser Stelle werden nur kurz die groben Bestandteile eines Agenten erklärt (siehe Fig. 3-1). Die genaue Beschreibung folgt in Kapitel 3.

2.4.1 Agent

Der Agent besteht aus zwei Hauptbestandteilen. Dies ist zum einen die X-ait-Agent Engine, und zum anderen die Wissensbasis (Knowledge Base - KB).

2.4.1.1 X-ait-Agent Engine

Die Agentenarchitektur modelliert den Datenfluss in einer kognitiven Umgebung. Eine genaue Beschreibung der einzelnen Komponenten der Architektur folgt im Kapitel 3.1.

2.4.1.2 Wissensbasis

Die Wissensbasis (Knowledge Base - KB) stellt jedem Agenten sein individuelles Wissen zur Verfügung. Wie bereits erwähnt findet sich hier auch wieder ein Teil der Weltrepräsentation.

- 5 Die X-ait-Agent Engine bekommt den Großteil ihres Wissens direkt aus der Wissensbasis. Ausgeschlossen davon sind z. B. einige Events, die der Agent nicht wahrnehmen konnte. Der genaue Aufbau der Wissensbasis wird ebenfalls in Abschnitt Agent erklärt.

10

3. Der X-ait-Agent

3.1 Die Architektur der X-Ait-Agenten

- 15 Die X-ait-Agenten (siehe Fig. 3-1) sind eine Weiterentwicklung der InterRRaP-Architektur, die aus drei Ebenen besteht, in der jede höhere Ebene eine höhere Abstraktion als die untere aufweist [JPM99]. Zusätzlich werden die drei Ebenen vertikal in zwei Module geteilt: Ein Modul enthält die Wissensbasis jeder Ebene und das andere enthält verschiedene Kontrollkomponenten, die mit den korrespondierenden Ebenen der Wissensbasis interagieren. Vertikal über alle Ebenen verteilt existiert die lokale Weltrepräsentation, die alle 3D-Informationen (Position von Gegenständen und Agenten, Raumaufteilung etc.) enthält.

- 20 Die InterRRaP-Architektur hat ein Welt-Interface, d. h. eine Schnittstelle zur Welt. Technisch besteht das Welt-Interface aus mehreren Komponenten. Der Agent interagiert mit der Welt ausschließlich über Aktionen und Perzepte.

- 25 In der Reaktivebene (Behaviour) der InterRRaP-Architektur werden die so genannten "Behaviours" programmiert und abgelegt. Das sind eingekapselte Verhaltensweisen und Aktionen des Agenten. Am Beispiel Computerfußball wären dies elementare Verhaltensweisen (atomare Aktionen) eines Agenten bzw. Spielers, wie z. B. "Vorlaufen", "Drehen", "Dribbeln mit Ball" und "Tor-

schuss". Die Abfolge solcher Einzelverhaltensweisen wird durch spontane Reaktion im Spielgeschehen angestoßen, oder eine Abfolge der atomaren Aktionen wird in so genannten Skripten fest definiert oder durch die nächst höhere Ebene geplant.

5 In der Planungsebene wird für jeden einzelnen Spieler ein typischer Funktionsplan aus den Elementar-Behaviours (atomaren Aktionen) zusammengestellt, d. h. ein Plan ist eine Abfolge von "Elementar-Behaviours". Ein Beispiel für einen Plan könnte z. B. lauten: <"Vorlaufen" "Dribbeln mit Ball" "Torschuss">, was
10 einem Skript entspricht.

In der Kooperationsebene (Cooperation) werden soziale Pläne erstellt, am Beispiel Fußball wäre das das Zusammenspiel der elf Spieler einer Fußballmannschaft. Im Allgemeinen werden dies feste Pläne sein, wie das Zusammenspiel der Spieler durch koor-
15 diniertes Laufverhalten und Passspiel, mit dem Ziel zum Torerfolg zu kommen. So kann der Agent anderen Agenten Anweisungen geben, in dem er die Ausführung von Elementar-Behaviours anweist.

Als Grundlage für die Beschreibung der Aktionen z. B. wäh-
20 rend der Kommunikation für die Kooperation dient die vom Planer für die entsprechende Aktion gespeicherte ID. Als Beschreibung für die Welt und die darin ausführbaren Aktionen dient dem Planer der PDDL-Standard (PDDL = Planning Domain Definition Language) (Typen, Prädikate, Aktionen).

25

3.2 Das bevorzugte modifizierte InteRRaP Model des X-Ait-Agent

Das modifizierte InteRRaP Model hat einen optimierten Datenfluss, der es erleichtert, Informationen zwischen den verschiedenen Ebenen des InteRRaP Models auszutauschen. Im Vordergrund
30 stand, das Model so flexibel wie möglich zu gestalten, damit man den X-Ait-Agenten modular aufbauen kann und in den neu implementierten Modulen keine Verwaltung de Datenflusses mehr vornehmen muss. In dem ursprünglichen InteRRaP Model muss jedes

Modul verwalten, ob es auf eine Aktion oder Perzept reagiert. Diese Entscheidung ist entweder sehr leicht zu treffen, wenn man davon ausgeht, dass gewisse Aktion und Perzepte nur von einzelnen Ebenen bearbeitet werden können, oder sehr schwer, wenn es darum geht, bereits erzeugte Aktionen in ihrem Inhalt zu verändern, wenn eine tiefere Ebene etwas ausführen will. Der neue Ansatz orientiert sich an den zielgerichteten Aktionen und Perzepten. Wenn zwei Ebenen miteinander kommunizieren, so tun sie dies über Perzepte. Perzepte haben den Vorteil, dass es in sich abgeschlossene Datenpakete sind, welche nicht an Schnittstellendefinitionen gebunden sind. Dies sind sehr grundlegende Perzepte, welche für eine gewisse Ebene innerhalb des InterRaP selbstverständlich sind. So muss z.B. die Reaktivebene ein Perzept mit einem Plan verstehen können, da es die Definition der Reaktivebene ist, dass sie Pläne ausführt. Ein Grund warum wir zur internen Kommunikation keine konkrete Schnittstelle verwenden, ist die Tatsache, dass man mit Perzepten die Rechenzeit besser kontrollieren kann. Wird ein Perzept von einer Ebene zu einer weiteren geschickt, so muss dieses Perzept durch die Perzept-Queue. Diese wird aber nur zu jedem Frame geleert. So gibt es automatisch einen Suspend zwischen dem Auffüllen des Perzepts in der Absender-Ebene und dem Empfang in der Empfänger-ebene.

Eine weitere offensichtliche Änderung im Design der einzelnen Komponenten im Vergleich zum ursprünglichen InterRaP Model ist die Wissensbasis (Knowledge Base). Die vom Framework (also der generischen KI-Architektur) bereitgestellte gemeinsame Wissensbasis besteht nur noch aus der lokalen Weltrepräsentation (Local Worldrepresentation) und dem Perzept Speicher (Percept Memory). Die lokale Weltrepräsentation ist dabei eine Kopie der globalen Weltrepräsentation mit individuellen Änderungen. Ferner gibt es den Perzept Speicher. Dieser speichert Perzepte zwischen und stellt auch Perzept-Anfragen an den KI Weltserver,

wenn Informationen im Speicher zu alt sind oder fehlen. Die in dem ursprünglichen InterRaP dargestellten Wissensbasen für Kooperation, Lokale Planung und Reaktivebene existieren immer noch, aber werden nicht vom Framework zur Verfügung gestellt.

5 Die Wissensbasen sind spezielle Implementierungen, die abhängig von dem jeweiligen Modul sind.

3.2.1 Aktionen

10 Eine der elementaren Datenstrukturen, mit denen der Agent mit seiner Umgebung interagieren kann, sind die Aktionen (Actions). Eine Aktion beschreibt ein elementares Steuerungsereignis, ähnlich einem Tastendruck auf der Tastatur durch einen Menschen. Eine Aktion enthält immer folgende Informationen:

- 15 • Typ der Aktion (Tastendruck, Mausbewegung, Texteingabe, usw.) und
- ID des Agenten, der die Aktion ausführen will.

Zu den in jeder Aktion vorhandenen Informationen kommen noch Aktionstyp-spezifische Informationen hinzu. Dies könnte im Fall einer Aktion "Tastendruck" die gedrückte Taste sein.

20 Ein weiterer Hauptverwendungszweck von Aktionen ist die Übermittlung von Sprechakten (Speechact). Die Sprechakte dienen zur Kommunikation zwischen den Agenten. Diese werden aber immer über das Spiel gesendet, damit das Spiel die volle Kontrolle über die Kommunikation zwischen den Agenten hat. Damit man
25 nicht eine weitere Schnittstelle ins Spiel benötigt, werden die Sprechakte als Aktion ins Spiel gesendet und vom Spiel als Event zum Zieladressat in die KI-Engine gesendet. In Fig. 3-1 wird dies durch die Sprechakte (Speechacts), welche ebenfalls zum Actionmanager gesendet werden, dargestellt.

30 Aktionen werden, wie im Schaubild gezeigt, nur von der Kooperations- und Reaktivebene losgeschickt.

3.2.2 Sprechakte

Sprechakte sind sehr kompakte Datenstrukturen, die zwischen zwei Kommunikationspartnern ausgetauscht werden. Ein Sprechakt soll die Kommunikation zwischen zwei Menschen nachmodellieren.

5 Dazu legt man sich zuerst nicht auf ein spezielles Medium fest, sondern analysiert den reinen Sprechverkehr zwischen zwei Menschen. Aus den Analysen dieses Gesprächsverkehrs und der zusätzlichen Notwendigkeit von Information zur Verwaltung durch das Framework ergeben sich folgende Grunddaten, die in jedem
10 Sprechakt vorzugsweise vorhanden sind:

- Typ des Sprechakts (Anweisung, Information, usw.),
- ID des Agenten, der diesen Sprechakt initiiert hat (Absender) und
- ID des Agenten, der den Sprechakt empfangen soll (Empfänger).
- 15 • Eine Erkennungsmarke, mit der der Empfänger auf diese Anfrage antworten soll, wenn dies nötig ist.
- Die Erkennungsmarke mit der der Absender rechnet, wenn man auf eine Anfrage antwortet.

20 Diese Informationen werden für die Kommunikation immer benötigt. Zusätzlich werden noch Nutzdaten benötigt.

3.2.3 Perzepte

Die Perzepte sind ebenfalls sehr elementare Datenstrukturen.
25 Diese ermöglichen dem Agenten, Informationen aus der Spielwelt über den KI-Weltserver zu entnehmen, oder damit er innerhalb des Agenten kommunizieren kann. Perzepte haben - ebenso wie Aktion - vordefinierte Grundinformationen. Diese sind vorzugsweise:

- 30
- Typ des Perzepts (Hören, Sehen, Sprechakt usw.),
 - ID des Agenten, der das Perzept empfangen soll, und

- SubID der Komponente innerhalb eines Agenten, für den das Perzept bestimmt ist.

Zusätzlich zu den Grundinformationen, die für die Vermittlung und Verteilung der Perzepte durch das Framework benötigt werden, gibt es die Nutzinformationen die je nach Perzept verschieden sind.

3.2.4 Perzept InQueue

Die Perzept InQueue nimmt alle eingehenden Perzepte entgegen. Diese können sowohl extern als auch intern erzeugt worden sein. Die Perzept InQueue ist eine einfache FIFO-Queue. Diese wird jedes Mal, wenn ein Agent die Rechenkontrolle bekommt, komplett abgearbeitet. So soll verhindert werden, dass es zu einem Überlauf in der Queue kommt, wenn zu wenig Rechenzeit zur Verfügung steht.

3.2.5 Perzept Verteilung

Die Perzept Verteilung (Percept Distribution) ist eine Erweiterung der Perzept InQueue. Diese ist dafür verantwortlich, dass die Perzepte in der Queue an die entsprechenden Komponenten (Reagieren, Planen, Kooperieren) weitergeleitet werden können. Dieser Vorgang wird jedes Mal getriggert, wenn der Agent Rechenzeit bekommt. Dabei wird mit der SubID innerhalb des Perzepts die entsprechende Komponente des Agenten ermittelt und dann die entsprechende Handle Methode (Bearbeitungsmethode) der Komponente mit dem Perzept aufgerufen. Die Handle Methode der Komponente verarbeitet dann das Perzept. Dies wird so lange durchgeführt, bis die Queue leer ist. Dabei sollte in keiner der Handle Methoden neue Perzepte angelegt werden, welche dann wiederum an den aktuellen Agenten gesendet werden. Sollte dies dennoch vorkommen, so werden neue Perzepte, die während der Arbeitung der Queue eingehen, in diesem Schritt nicht betrachtet. Dies wird dadurch erreicht, dass man nicht die Queue

abarbeitet bis sie leer ist, sondern über die Queue iteriert bis zu einem am Anfang des Vorgangs festgelegten Ende.

Die SubIDs innerhalb eines Agenten werden in einen statischen und in einen dynamischen Bereich aufgeteilt. Der statische Bereich ist schnell festgelegt. Dies sind die drei Hauptkomponenten des Agenten, die Kooperativeebene, Planungsebene und die Reaktivebene. Dazu ist die Einteilung wie folgt:

- 0 : Reaktivebene
- 1 : Planungsebene
- 2 : Kooperativeebene

Diese SubIDs sind statisch, damit eine interne Kommunikation überhaupt möglich wird. Zudem sind sie über alle Agenten hinweg gleich, so dass eine Kooperationsebene eines Agenten mit der Kooperationsebene eines anderen Agenten kommunizieren kann.

Alle weiteren SubIDs werden dynamisch während der Laufzeit vergeben. Mögliche Komponenten sind z. B. weitere Teilkomponenten innerhalb der Reaktivebene, oder Teile des Perzept Speichers, z. B. Sehen, Hören, also die 5 Sinne. Diese Komponenten kommen mit einer dynamisch zugewiesenen SubID aus, weil sie diese nur dazu benötigen, um neue Daten vom KI-Weltserver anzufordern. Diese Komponenten kommunizieren nicht mit anderen Komponenten innerhalb eines Agenten oder den Komponenten eines anderen Agenten.

3.2.6 Kooperationsebene

Die Kooperationsebene ist die höchste Ebene in einem durch InteRRaP modellierten Agenten. In ihr findet die Kommunikation zwischen zwei Agenten statt. Dazu können in der Kommunikationsebene verschiedene Kommunikationsprotokolle zum Einsatz kommen. Eines der am häufigsten verwendeten Protokolle in dieser Ebene ist das bekannte Contract Net Protocol. Die Kooperationskomponente kann Sprechakte in Form von Aktionen an andere Agenten schicken. Dazu muss die Kooperationskomponente nur die ID des

anderen Agenten wissen, da sich die Kooperationskomponente eines jeden Agenten für Sprechakte am Weltserver registrieren muss, und damit dort die entsprechende SubID, die als Empfänger-ID verwendet wird, hinterlegt.

5 Durch die Änderung des Datenflusses im bevorzugten modifizierten X-Ait-Agenten kann nun auch die Kooperationsebene direkt über die Percept InQueue Pläne an die Reaktivebene senden. Dies hat den Vorteil, dass man keinen großen Kommunikationsoverhead für sehr simple Agenten hat, welche nur Kommandos entgegen nehmen und diese Ausführen. Der bevorzugte modifizierte X-Ait-Agent ist in dieser Hinsicht weniger rechenintensiv.

10 Die normale Aufgabe der Kooperationsebene ist das kommunizieren mit anderen Agenten und schließlich, oder während der Kommunikation, das Senden von Aufträgen an die lokale Planungsebene.

15 Die Kommunikation innerhalb des Agenten funktioniert über Perzepte. Dazu stehen verschiedene Perzepte zur Kommunikation zwischen Kommunikationsebene und lokaler Planung sowie Kommunikationsebene und Reaktivebenen zur Verfügung. Dies sind:

- 20 • Kommunikationsebene <-> Planungsebene:
 - o PerceptGoal: Die Planung soll einen Plan für ein gewisses Ziel (Goal) erzeugen.
 - o PerceptGoalFB: Für die Antwort auf ein zuvor gesendetes PerceptGoal. Darin ist enthalten ob das Ziel ausgeführt werden konnte oder nicht und eine Begründung
 - 25 im Falle des Scheiterns.
- Kommunikationsebene <-> Reaktivebene:
 - o PerceptPlan: Enthält einen Plan, der von der Reaktivebene ausgeführt werden soll.
 - 30 o PerceptPlanFB: Damit wird aus der Reaktivebene die Antwort auf einen zuvor übertragenen PerceptPlan an die Kommunikationsebene übermittelt. In diesem Percept ist enthalten, ob der Plan erfolgreich war oder

gescheitert ist, und wenn er gescheitert ist, in welchem Planschritt.

3.2.7 Lokale Planungsebene

5 Die Lokale Planung (Local Planning) nimmt Ziele / Aufgaben aus der Kommunikationsebene entgegen und erzeugt daraus einen möglichen Plan. Dieser Plan wird dann zur Ausführung an die Reaktivebene gesendet. Für die Planungsebene ist es nicht vorgesehen, dass sie Aktionen versenden kann. Sie dient als Planungswerkzeug zwischen Kommunikationsebene und Reaktivebene. 10 Dazu bedient sie sich denselben Perzepte wie die Kommunikationsebene.

- Planungsebene <-> Kommunikationsebene:

- o PerceptGoal: Enthält ein Ziel für das die Planungsebenen einen Plan erzeugen soll. 15
- o PerceptGoalFB: Für die Antwort auf ein zuvor gesendetes PerceptGoal. Darin ist enthalten, ob das Ziel ausgeführt werden konnte oder nicht und ggf. die Begründung für das Scheitern.

- 20 • Planungsebene <-> Reaktivebene:

- o PerceptPlan: Enthält einen Plan, der von der Reaktivebene ausgeführt werden soll.
- o PerceptPlanFB: Damit wird aus der Reaktivebene die Antwort auf einen zuvor übertragenen PerceptPlan an 25 die Planungsebene übermittelt. In diesem Perzept ist enthalten, ob der Plan erfolgreich war oder gescheitert ist, und wenn er gescheitert ist, in welchem Planschritt.

30 3.2.8 Reaktivebene

In der Reaktivebene passiert die eigentliche Steuerung des Agenten. Hier werden einzelne Behaviours ausgeführt, welche wiederum Steuerinputs für das Spiel erzeugen. Die Reaktivebene

kann keine eigenen Entscheidungen treffen. Sie kann aber auf gewisse Ereignisse mit vordefinierten Behaviours reagieren. Dies wäre z. B. der Fall, wenn ein Gegner gesichtet wird und ein Behaviour dann automatisch auf diesen zielt und schießt.

5 Die Reaktivebene kann sowohl mit der Kommunikations- als auch mit der Planungsebene kommunizieren. Dies ist davon abhängig, wer den aktuellen Plan an die Reaktivebene gesendet hat. Die Annahme von neuen Plänen läuft wie bereits bei den beiden anderen Ebenen beschrieben über Perzepte. Die Reaktivebene muss da-
10 zu die beiden folgenden Perzepte verstehen:

- PerceptPlan: Enthält einen Plan, der von der Reaktivebene ausgeführt werden soll.
- PerceptPlanFB: Damit wird aus der Reaktivebene die Antwort auf einen zuvor übertragenen PerceptPlan an die Planungs-
15 oder Kommunikationsebene übermittelt. In diesem Perzept ist enthalten, ob der Plan erfolgreich war oder gescheitert ist, und wenn er gescheitert ist, in welchem Planschritt.

20 3.2.9 Lokale Wissensbasis

Die lokale Wissensbasis (Local Knowledge Base) speichert das Wissen, welches übergreifend für alle Komponenten des X-Ait-Agenten zur Verfügung steht. Dazu wird versucht, die Daten in eine allgemeine Form zu bringen, damit das Wissen in jeder Kom-
25 ponente verwendet werden kann, ohne komplizierte Wissensrepräsentationen verstehen zu müssen. Sollte eine Komponente spezielles Wissen benötigen, so wird dieses innerhalb der Komponente angelegt und verwaltet. Die allgemeine Wissensbasis kann aufgrund ihrer Generalität in zwei Teile unterteilt werden.

30 Dies ist zum einen das Wissen über die Landschaft bzw. Geometrie der Umgebung. Diese Information wird in der lokalen Weltrepräsentation (Local Worldrepresentation) abgespeichert. Dieser Teil der Wissensbasis liefert dem Agenten Informationen

darüber, wie seine Umgebung aufgebaut ist, und gibt ihm darüber hinaus die Möglichkeit, Pfadsuchen für die Umgebung zu starten. Die lokale Weltrepräsentation gleicht dem räumlichen Denkvermögen eines Menschen. Dieser kann sich damit auch in seiner Umgebung zurechtfinden. Er erkennt dadurch, ob er sich in einem Raum oder unter freiem Himmel befindet, und wie er von A nach B kommt. Die lokale Weltrepräsentation ist eine Kopie der globalen Weltrepräsentation, kann aber Abweichungen enthalten, die durch die Erlebnisse des Agenten entstanden sind.

Der zweite Teil der Wissensbasis ist der Perzept Speicher (Percept Memory). Der Perzept Speicher ist ein Cache, welcher Perzeptanfragen zwischenspeichert. Da Perzepte Faktenwissen enthalten, ist der Perzept Speicher eine Form des Faktenwissens. Jedes Perzept hat eine gewisse Lebenszeit, die Auskunft darüber gibt, wie lange sich das Perzept bereits im Perzept Speicher befindet. Zusätzlich zu der aktuellen Lebenszeit hat jedes Perzept im Perzept Speicher eine Lebensdauer, die, wenn sie überschritten wird, dem Speicher signalisiert, dass das Perzept aus dem Speicher entfernt werden soll. Da nun die Komponenten für eine Perzeptanfrage mit der Wissensbasis interagieren, können sie der Wissensbasis und damit dem Perzept Speicher angeben, wie alt die abgefragte Information sein darf. Ist die Information innerhalb des Perzept Speichers unterhalb dieser Grenze, so kann der Perzept Speicher die gewünschte Information direkt an die jeweilige Komponente zurückgeben. Sollte die Information zu alt sein oder die Lebensdauer des Perzepts überschritten sein, so kann der Perzept Speicher eine neue Perzeptanfrage an das Spiel starten. Dieser Caching Mechanismus ermöglicht es also einem Programmierer einer Komponente, den Agenten menschlicher zu gestalten, indem er sich nicht immer auf aktuelle Informationen aus dem Spiel stützt, sondern auch einmal Fehlinformationen unterliegt, wenn er ein gewisses Alter an Informationen zulässt.

Ein weiteres Anwendungsszenario ist die Unterteilung von Behaviours in Wissensaufbauende und Wissensabfragende Behaviours. So kann man ein Behaviour schreiben, welches Wissen in den Perzept Speicher schreibt, wenn es z. B. neue Objekte sieht. Zum
5 anderen kann man Behaviours schreiben, die genau dieses Wissen benutzen, egal wie alt es ist, und bei Nichtvorhandensein der Information keine neuen Perzeptanfragen an das Spiel senden.

4 Datenfluss

10 Der Kontrollfluss im Agent ist sowohl daten-, wie auch ziel-gesteuert. Der Eingang von Perzepten führt zu einer Veränderung des lokalen oder globalen Weltmodells. Als Reaktion auf Veränderungen des Weltmodells werden unterschiedliche Verhaltensmuster aktiviert, beendet oder ausgeführt. Die Ausführung eines
15 Verhaltensmusters wird durch die lokale Planungsebene und die Kooperationsebene angestoßen, wobei die Planungsebene dafür zuständig ist, neue eigene Pläne und daraus resultierend neue agentenübergreifende Pläne zu erzeugen, um die Ziele des Agenten wieder zu erreichen. Dies führt schließlich zur Ausführung von
20 Aktionen und zur Erzeugung von Nachrichten (Sprechakten).

Ziele werden von den Komponenten innerhalb eines Agenten vorgegeben, entweder statisch aus einer Datei, oder dynamisch als Reaktion auf eine Umweltveränderung. Diese Ziele initiieren einen Datenfluss. Diesen Datenfluss kann man innerhalb der KI-
25 Engine visualisieren. Es gibt verschiedene Datenflüsse, die innerhalb der KI-Engine ablaufen können. Dies sind zum einen Aktionen die gesendet werden, inklusive der darin verpackten Sprechakte. Daneben gibt es Events, die durch das Spiel erzeugt werden können, unter anderem als Ursache einer Aktion, die man
30 zuvor ins Spiel gesendet hat. Des Weiteren kann der Agent Perzeptanfragen an das Spiel stellen, die zu einem späteren Zeitpunkt durch das Spiel beantwortet werden. Im Folgenden soll der genaue Verlauf dieser Datenflüsse dargestellt werden.

Wir verwenden zur Darstellung des Datenflusses ein modifiziertes Interaktions-Diagramm (siehe Fig. 4-1). Dieses Diagramm hat in der vertikalen Achse die Zeit. Auf der horizontalen Achse sind die einzelnen Komponenten, die am Datenfluss teilhaben, dargestellt. Zusätzlich zu der normalen Markierung, dass in dieser Komponente Arbeit verrichtet wird, sind die Blöcke unterschiedlich schraffiert markiert, wobei

- ein  Block den Lebensstart einer Aktion / Event / Perzept markiert,
- ein  Block einen Bearbeitungsschritt für die Aktion / Event / Perzept markiert,
- ein  Block das Lebensende einer Aktion / Event / Perzept markiert, und wobei
- ein  Block zusätzliche Informationsabfragen signalisiert.

Eine weitere Änderung ist die Zusammenfassung von mehreren Datenflussschritten mithilfe einer dicken Umrandung. Dadurch wird symbolisiert, dass man diese Abfolge von Schritten extern nicht mehr zertrennen kann; trennen kann man den Datenfluss nur an Stellen, an denen eine neue Umrandung beginnt. Wird der erste Bearbeitungsschritt innerhalb einer Umrandung gestartet, so werden alle Schritte bearbeitet, bis die Umrandung in eine neue Umrandung übergeht. Beim Übergang von zwei unterschiedlichen Umrandungen kann, durch eine hier mögliche Trennung, eine sehr große Zeitverzögerung auftreten, je nachdem ob die KI-Engine für Multithreads oder Singlethreads optimiert ist. Genauer gesagt, wird der Datenfluss an den Übergängen angehalten, z. B. durch eine Queue, und kann später fortgesetzt werden.

4.1 Aktionen ins Spiel senden

Aktionen werden vom Agenten ins Spiel gesendet, damit diese den Zustand des Agenten verändern. In Fig. 4-1 sehen wir, welche Komponenten eine Aktion durchläuft, bis sie von Agent A bis zur Spiel-Engine (Game-Engine) gekommen ist. Der Ablauf lässt sich folgendermaßen beschreiben:

a) Der Agent A erstellt eine Aktion, entweder direkt in einer Komponente oder durch den ActionCreator und sendet diese an den im Worldserver befindlichen Actionmanager.

b) Wird dem Actionmanager die Rechenkontrolle übergeben, so führt der Actionmanager den entsprechenden ActionHandler auf der Aktion aus.

c) Der ActionHandler führt eine entsprechende Konvertierung der Daten in der Aktion durch, damit das Spiel die Aktion korrekt interpretieren kann. (z.B. Objekt-ID Konvertierung) und sendet die Aktion durch das KI-Interface an die Spiel-Engine.

d) Im KI-Interface werden die KI-Objekt-IDs einer Aktion in die Spiele-Objekt-ID konvertiert.

e) Die Game-Engine nimmt die Aktion in einer Callback Funktion entgegen und setzt die Aktion im Spiel um, oder verschiebt sie bis zu einem späteren Zeitpunkt.

4.2 Sprechakte von einem Agenten zum Anderen versenden

Damit ein Agent einen Sprechakt zu einem anderen Agenten senden kann, muss er diesen Sprechakt in einer Aktion kapseln und diese an das Spiel senden. Das Spiel hat dann die Aufgabe, diesen Sprechakt wieder in die KI-Engine zu senden. Dieser Mechanismus hat den Vorteil, dass die Spiellogik volle Kontrolle über die Kommunikation zwischen den Agenten hat. Sie kann somit Fehler in das Kommunikationsprotokoll einbauen, oder die Kommunikation vollständig unterdrücken, wenn z.B. die Spiellogik vorgibt, dass ein Agent zeitweise taub ist, oder der Funk in-

nerhalb eines gewissen Bereichs gestört ist. Die Daten laufen wie in Fig. 4-2 schematisch dargestellt:

- a) Agent A legt einen Sprechakt an, den er in eine Aktion kapselt, und sendet diese an den Actionmanager.
- 5 b) Wenn der Actionmanager die Rechenkontrolle erhält, nimmt er alle Aktionen aus seiner Queue und führt für jede Aktion den entsprechenden ActionHandler auf der Aktion aus.
- c) Der Actionhandler konvertiert die Empfänger- und Absender-IDs des Sprechakts und reicht die Aktion über das KI-Interface an das Spiel weiter.
- 10 d) Das KI-Interface konvertiert die KI-Absender-ID der Aktion in eine entsprechende Objekt-ID des Spiels.
- e) Im Spiel wird in einer Callback Funktion die Aktion geparkt und der Sprechakt, in einem Event gekapselt und zurück an das KI-Interface gesendet. Oder der Sprechakt wird
- 15 verworfen. Das Zurücksenden als Event kann auch durch die Spiel-Engine verzögert werden, z.B. wenn ein Agent zu viele Nachrichten versendet.
- f) Das KI-Interface konvertiert die Absender-ID des Events und reicht das Event an den Eventmanager weiter.
- 20 g) Bekommt der Eventmanager die Rechenkontrolle, dann arbeitet er seine Queue ab und ruft für das Sprechaktevent den passenden Eventhandler auf.
- h) Der Eventhandler wandelt das Event in ein Perzept und versendet es an den Agenten B, welcher als Empfänger im
- 25 Sprechakt genannt ist. Zusätzlich muss sich dieser Agent bei dem Eventhandler für Sprechakte registriert haben, sonst wird der Sprechakt verworfen.
- i) Im Agent B wird das Perzept an die Kommunikationskomponente weitergeleitet, die dann den Sprechakt aus dem Perzept
- 30 entnimmt.

4.3 Senden eines Events vom Spiel in die KI-Engine

Während des Spielablaufs werden ständig Events ausgelöst. Dies können zum einen Geräusche sein oder Ereignisse, die durch die Spiellogik ausgelöst werden, wie z. B. ein Rundenstart. Ereignisse werden normalerweise visuell auf dem Monitor ausgegeben, in Form von Text oder Bildern, oder in Form von Sound. Diese Ereignisse werden als Events an die KI-Engine gesendet, damit die Agenten den Spielablauf ebenfalls korrekt interpretieren können. Sendet das Spiel ein Event an die KI-Engine, gelangt der Event wie in Fig. 4-3 schematisch dargestellt an die Agenten:

- a) Das Spiel legt ein Event Struct für das entsprechende Ereignis an und gibt dieses an das KI-Interface.
- b) Das KI-Interface konvertiert die Absender-ID des Events und gibt das Event an den Eventmanager weiter.
- c) Wird dem Eventmanager die Rechenkontrolle übergeben, so ruft er für jedes Event in der Queue den passenden Eventhandler auf.
- d) Je nachdem, welcher Eventhandler ausgeführt wird, kann es dazu kommen, dass ein Update in der Weltrepräsentation durchgeführt wird. Dieser Schritt ist jedoch optional. Nachdem die Weltrepräsentation ein Update erhalten hat, wird aus dem Event ein Perzept erzeugt. Dieses Perzept kann nun mehr Informationen enthalten, als das Event. Im Falle eines Soundevents kann durch das Update in der Weltrepräsentation das Geräusch durch die Welt propagiert werden. Das Perzept welches an die Agenten geschickt wird enthält dann zusätzlich noch die Lautstärke der Geräuschquelle, oder sie erhalten das Perzept überhaupt nicht, wenn sie das Geräusch nicht mehr wahrnehmen können. Die erzeugten Perzepte werden dann an die Agenten gesendet.

- e) Der Agent nimmt das Perzept entgegen und leitet es, wenn er die Rechenkontrolle hat, an die entsprechende Komponente weiter.

5 4.4 Perzeptanfrage an das Spiel (Pull) - Fall 1

Will ein Agent Informationen über seine Umgebung abfragen, so muss er eine Perzeptanfrage starten. Die Anfrage wird im Agent immer damit gestartet, wenn er ein leeres Perzept an den Weltserver sendet, das dieser dann füllen soll. Der weitere Datenfluss kann in drei Fälle unterteilt werden, je nachdem wie aktuell die Informationen sein müssen und welcher Perzepthandler die Informationen besorgt.

Im ersten Fall wird ein Perzepthandler verwendet, welcher die Weltrepräsentation als Cache für die im Perzept gesuchten Informationen benutzt. Des Weiteren sind in diesem Fall die Informationen in der Weltrepräsentation gültig. Der Datenfluss läuft dann wie in Fig. 4-4 schematisch dargestellt ab:

- a) Der Agent A legt ein leeres Perzept an und sendet dies über den Weltserver an den Perzeptmanager.
- 20 b) Der Perzeptmanager bekommt nun die Rechenkontrolle für den Schritt FetchData. Er leitet das leere Perzept an den Perzepthandler weiter, indem er ebenfalls beim Perzepthandler die Methode FetchData ausführt.
- c) Der Perzepthandler stellt nun eine Anfrage (Request) an die Weltrepräsentation für die benötigten Daten im Perzept. Dabei dürfen die in der Weltrepräsentation gespeicherten Daten ein gewisses Alter nicht überschreiten.
- 25 d) Die Weltrepräsentation liefert in diesem Fall gültig Daten zurück (Reply).
- 30 e) Der Perzepthandler hat nun alle Informationen gesammelt. Da die Daten in der Weltrepräsentation den geforderten Daten in dem Perzepten entsprechen (KI interne Daten), werden diese Daten bereits jetzt in das Perzept geschrieben.

Der Perzepthandler hat dann den Schritt FetchData beendet und gibt das Perzept zurück an den Perzeptmanager.

f) Bekommt danach der Perzeptmanager die Rechenkontrolle für den Schritt ProcessData, dann führt er den Perzepthandler mit der Methode ProcessData aus.

g) Der Perzepthandler hat nun die Möglichkeit die zuvor hinterlegten Daten im Perzept nachzubearbeiten. Hat er dies erledigt, so sendet er das gefüllte Perzept zurück an den Agenten A

h) Nachdem der Agent A Rechenzeit zugewiesen bekommt, nimmt er das Perzept aus seiner InQueue und sendet es an die Komponente, welche zuvor die Anfrage gesendet hat.

4.5 Perzeptanfrage an das Spiel (Pull) - Fall 2

Der zweite Fall unterscheidet sich vom ersten Fall darin, dass die Weltrepräsentation in Punkt d) aus Fall 1 keine gültigen Daten zurückliefern kann. Der Ablauf ändert sich danach wie in Fig. 4-5 schematisch dargestellt:

a) Der Perzepthandler bekommt keine gültigen Daten von der Weltrepräsentation und stellt deshalb eine Anfrage (DataRequest) an die Spiel-Engine über das KI-Interface.

b) Die Spiel-Engine füllt das DataRequest mit den geforderten Daten und gibt das DataRequest über das KI-Interface zurück an den Perzepthandler.

c) Der Perzepthandler hat nun alle notwendigen Informationen gesammelt und gibt das Perzept zusammen mit dem DataRequest zurück an den Perzeptmanager. Er führt keine Bearbeitungsschritte auf dem DataRequest durch, da er sich im Schritt FetchData befindet.

d) Bekommt der Perzeptmanager dann irgendwann die Rechenkontrolle für den Schritt ProcessData, so führt er beim Perzepthandler die Methode ProcessData aus, und übergibt da-

bei das Perzept und die zuvor gesammelten Daten in Form der DataRequests.

e) Der Perzepthandler bearbeitet nun die Daten aus dem Data-request und füllt damit das Perzept. Zusätzlich führt er
5 ein Update bei der Weltrepräsentation aus, damit er beim nächsten mal wieder aktuelle Informationen der Weltrepräsentation hat. Danach sendet er das gefüllte Perzept zurück an den Agenten.

f) Nachdem der Agent A Rechenzeit zugewiesen bekommt, nimmt
10 er das Perzept aus seiner InQueue und sendet es an die Komponente, welche zuvor die Anfrage gesendet hat.

4.6 Perzeptanfrage an das Spiel (Pull) - Fall 3

Der dritte Fall tritt ein, wenn die angeforderten Daten immer so aktuell sein müssen (z.B.: die aktuelle Position des Agent für die Navigation), dass es sich nicht lohnt Daten in der Weltrepräsentation zwischenzuspeichern. Die Anfrage und das Update der Weltrepräsentation fallen deshalb in diesem Schritt
15 komplett weg. Der Datenfluss läuft dann wie in Fig. 4-6 schematisch dargestellt:
20

- a) Der Agent A legt ein leeres Perzept an und sendet dies über den Weltserver an den Perzeptmanager.
- b) Der Perzeptmanager bekommt nun die Rechenkontrolle für den Schritt FetchData. Er leitet das leere Perzept an den Perzepthandler weiter, indem er ebenfalls beim Perzepthandler
25 die Methode FetchData ausführt.
- c) Der Perzepthandler stellt über das KI-Interface direkt eine Anfrage (DataRequest) an die Spiel-Engine.
- d) Die Spiel Engine füllt das DataRequest mit den geforderten
30 Daten und gibt das DataRequest über das KI-Interface zurück an den Perzepthandler.
- e) Der Perzepthandler hat nun alle notwendigen Informationen gesammelt und gibt das Perzept zusammen mit dem DataRe-

quest zurück an den Perzeptmanager. Er führt keine Bearbeitungsschritte auf dem DataRequest durch, da er sich im Schritt FetchData befindet.

f) Bekommt der Perzeptmanager dann irgendwann die Rechenkontrolle für den Schritt ProcessData, so führt er beim Perzepthandler die Methode ProcessData aus, und übergibt dabei das Perzept und die zuvor gesammelten Daten in Form der DataRequests.

g) Der Perzepthandler bearbeitet nun die Daten aus dem Datarequest und füllt damit das Perzept. Danach sendet er das gefüllte Perzept zurück an den Agenten.

h) Nachdem der Agent A Rechenzeit zugewiesen bekommt, nimmt er das Perzept aus seiner InQueue und sendet es an die Komponente, welche zuvor die Anfrage gesendet hat.

5 Rechenzeitkontrolle

Die Rechenzeitkontrolle innerhalb der KI-Engine verteilt die zur Verfügung stehende Rechenzeit an die verschiedenen Komponenten. Dazu wird vom Weltserver die zur Verfügung stehende Rechenzeit an die verschiedenen Manager und an die Agenten verteilt. Des Weiteren hat jeder Agent eine rudimentäre Rechenzeitkontrolle für seine internen Komponenten. Wir unterscheiden deshalb die globale Rechenzeitkontrolle und die lokale Rechenzeitkontrolle innerhalb der Agenten.

5.1 Globale Rechenzeitkontrolle

Die globale Rechenzeitkontrolle verteilt die Rechenzeit an Hauptkomponenten, wie die Manager innerhalb des Weltservers und die einzelnen Agenten. Um der steigenden Parallelisierung von Rechenoperationen gerecht zu werden, haben wir zwei verschiedene Modelle für die Rechenzeitkontrolle. Ein Modell ist für die Verwendung in Singlethreaded Anwendungen ausgelegt und das andere Modell für die Verwendung in Multithreaded Anwendungen,

damit ein evtl. zur Verfügung stehender zweiter CPU-Kern genutzt werden kann.

5.1.1 Singlethread Umgebung

5 Wir haben zu Beginn gesehen, dass jeder Manager innerhalb des Weltservers mit Eingangsqueues ausgestattet ist, ebenso wie einzelne Agenten. In einer Singlethreaded Anwendung ist es für die gesamte Verzögerung, die durch die KI-Engine erzeugt wird, egal, in welcher Reihenfolge wir den Komponenten die Rechenkontrolle übergeben. Wir müssen immer alle Komponenten ausführen,
10 damit ein korrekter Datenfluss zustande kommt. Deshalb wollen wir die Rechenzeit so verteilen, dass die Latenzen zwischen Aktionen und Effekten minimiert werden.

Aus den Abhängigkeiten zwischen Aktionen, Events und Perzepte
15 te (z.B. Aktionen können Sprechakte enthalten und sind dann Events) ergibt sich die Rechenzeitverteilung wie in Fig. 5-1 gezeigt. Das Spiel ruft beim KI-Interface die Update Methode auf. In dieser Methode werden dann die verschiedenen Komponenten der KI-Engine in folgender Reihenfolge ausgeführt:

- 20 1. **Action-Manager**: Der Actionmanager sendet alle bei ihm eingetroffenen Aktionen über die Actionhandler in das Spiel.
2. **Event-Manager**: Der Eventmanager sendet bei ihm eingegangene Events an die Agenten weiter. Da wir vor dem Eventmanager bereits den Actionmanager ausgeführt haben, sind unter
25 den Events auch Sprechakte, die in diesem Frame erst in das Spiel gesendet wurden.
3. **Percept-Manager FetchData**: Der Perzeptmanager führt für alle bei ihm eingegangenen Perzepte den FetchData Schritt bei den jeweiligen Perzepthandlern durch.
- 30 4. **Percept-Manager ProcessData**: Da wir in einer Singlethreaded Anwendung sind, wird als nächster Schritt für alle Perzepte im Perzeptmanager der ProcessData Schritt ausgeführt. Damit sind dann die Perzepte abgearbeitet.

5. **KI-Scheduler + Agent-Scheduler:** Als nächstes gibt es einen iterierten Wechsel zwischen dem KI-Scheduler, der einem ganzen Agenten Rechenzeit zuweist, und dem Agent-Scheduler, welcher den einzelnen Komponenten innerhalb eines Agenten Rechenzeit zuweist. Zuerst bekommt der KI-Scheduler die restliche Zeit der KI-Engine und verteilt diese an die einzelnen Agenten. Hier können alle Agenten Rechenzeit bekommen oder nur ein Teil, wenn zu wenig Rechenzeit zur Verfügung steht. Durch ein Roundrobin Verfahren bekommen dann im nächsten Frame die anderen Agenten Rechenzeit. Der Agent-Scheduler verteilt die Gesamtzeit für einen Agenten auf die drei Hauptkomponenten im Agenten.

Es wird also die komplette KI im Game-Thread ausgeführt.

5.1.2 Multithread Umgebung

In modernen Rechensystemen ist ein starker Trend zur Parallelisierung der Verarbeitung zu erkennen. Die KI-Engine wurde deshalb so designed, dass sie auch in der Lage ist, parallel zum eigentlich Spiel Threads zu verarbeiten. Für Multithreading kommt man nicht ohne eine Synchronisierung der verschiedenen Threads aus. Damit durch das Synchronisieren möglichst wenig Rechenzeit verloren geht, werden die Komponenten in einer von Single-Threaded Anwendungen abweichender Reihenfolge aufgerufen, damit die Berechnungen während der Synchronisierung auf den reinen Datenaustausch minimiert werden.

In Fig. 5-2 kann man den Update Verlauf für die KI-Engine in einer Multithread Anwendung erkennen. Das Spiel führt über das KI-Interface die Sync (Update) Funktion der KI-Engine aus. Diese Sync Methode wartet nun bis der KI-Thread seine Berechnungen für einen Frame abgeschlossen hat. Danach werden, durch das Ausführen des Actionmanagers, die im Actionmanager gespeicherten Aktionen in das Spiel gesendet. Anschließend führt die Sync

Methode noch beim Perzept-Manager den FetchData Schritt aus, damit Anfragen von den Agenten an das Spiel befriedigt werden können. Damit sind alle Daten an das Spiel gesendet, bzw. aus dem Spiel entnommen. Die beiden Threads können sich nun trennen und getrennte Wege gehen bis zum nächsten Frame, wenn der Game-Thread erneut die Sync Methode des KI-Interface ausführt.

Es wird also nur die Synchronisation vom Game-Thread ausgeführt, während der Rest der Berechnung in einem extra KI-Thread ausgeführt werden.

Im KI-Thread bekommen nun noch die restlichen Komponenten Rechenzeit. Keine dieser Komponenten benötigt direkten Zugriff auf das Spiel, welches seinerseits Berechnungen für die Darstellung des Spiels ausführt. Der KI-Thread führt alle Updates aus, und vergibt ebenfalls, wie in der Singlethread Anwendung, Rechenzeit an die einzelnen Agenten. Wurde alle Rechenzeit aufgebraucht, so wartet der KI-Thread bis das Spiel wieder einen Sync ausführt.

Dem KI-Thread wird, wie in der Singlethread Anwendung, Rechenzeit zugewiesen. Dies bewirkt, dass das Spiel weiterhin die Kontrolle über die KI-Engine besitzt. Sollte z. B. der KI-Thread keine richtige Rechenleistung vom Betriebssystem zugeteilt bekommen, weil das Spiel auf einer Single-Core CPU oder einer CPU mit Hyperthreading läuft, dann muss es trotzdem möglich sein, dass die KI ausreichend Rechenzeit bekommt. Würde die KI-Engine nur solange Rechnen bis der Spielthread wieder einen Sync durchführt, kann es sein, dass die KI-Engine noch keine Berechnungen durchgeführt hat. Durch die zugewiesene Rechenzeit in der Sync Methode wird somit sichergestellt, dass die KI-Engine ein Minimum an Rechenzeit erhält. Der Spielthread muss auf den KI-Thread warten; damit steht alle Rechenleistung der CPU dem KI-Thread zur Verfügung.

Da während der Zustandsberechnung im Spiel weitere Events anfallen können, wurde die InQueue des Eventmanagers threadsafe

programmiert. So kann nun, während der Spielthread und der KI-Thread getrennte Wege gehen, der Spielthread Events in die KI-Engine senden, ohne einen Fehler zu erzeugen.

5 5.2 Lokale Rechenzeitkontrolle

Die lokale Rechenzeitkontrolle bezieht sich auf einzelne Agenten. Jeder Agent hat seine eigene kleine Rechenzeitkontrolle. Innerhalb des Agenten gibt es zwei Komponenten, die für die Rechenzeitverteilung verantwortlich sind. Dies ist zum einen
10 der Komponenten-Scheduler und zum andere die Scheduler-FSM.

5.2.1 Komponenten Scheduler

Da wir im Agenten nur drei verschiedene Komponenten (Koordination, Lokale Planung, Reaktivebene) haben, kann der Komponenten-Scheduler mit einer statischen Anzahl von Objekten arbeiten. Jede Komponente hat eine Priorität, anhand der festgelegt
15 wird, ob eine Komponente viel oder wenig Rechenzeit bekommt. Die Priorität ist dabei abhängig von der Situation, in der sich der Agent momentan befindet. Befindet sich der Agent momentan
20 in einer Planungsphase, so bekommt die Planungsebene mehr Rechenzeit als die Reaktiv- und Kooperationsebenen. Wird der Agent aber während der Planung unter Beschuss genommen, so muss der Reaktivebene schlagartig mehr Rechenzeit zugewiesen werden. Diese Aufgabe übernimmt die Scheduler-FSM.

25

5.2.2 Scheduler-FSM

Die Scheduler-FSM (Finite State Machine) ist eine einfache Zustandsmaschine, mit vordefinierten Zuständen. Diese Zustände beschreiben häufige Gedankenzustände wie:

- 30 • Neutral: Der Agent befindet sich in einem neutralen Zustand. Dies liegt vor, wenn keine Bedrohung vorliegt und der Agent einfach von A nach B geht.

- Think: Der Agent muss vermehrt nachdenken. Dies ist der Fall, wenn er einen neuen Plan erarbeitet.
- Panic: Dieser Zustand tritt ein, wenn sich der Agent in einer bedrohlichen Situation befindet, z. B. wenn er unter
5 Beschuss ist.
- Communicate: Dieser Zustand beschreibt den Agenten, wenn er kommuniziert.

Für jeden dieser Zustände kann der Designer festlegen, welche Prioritäten die Komponenten im Agenten erhalten. Tritt nun
10 ein Zustandswechsel ein, so setzt die Scheduler-FSM beim Agenten-Scheduler die neuen Prioritäten für diesen Zustand. In dem Zustand "Panic" könnte z. B. die Reaktivebene die größte Priorität erhalten. Die Reaktivebene erhält damit übermäßig viel Rechenzeit, und somit kann der Agent der Bedrohung entkommen.

15 Die einzelnen Zustände werden direkt von den jeweiligen Komponenten geändert. Damit verhindert wird, dass die Komponenten sich gegenseitig die Rechenzeit streitig machen, werden den jeweiligen Zuständen Prioritäten zugewiesen. Diese Priorität sollte man nicht mit den Prioritäten verwechseln, welche in einem Zustand den jeweiligen Komponenten im Agenten zugewiesen
20 werden. Die Scheduler-FSM kann immer nur von einem Zustand niedrigerer Priorität in einen Zustand höherer Priorität wechseln. Alleine die Komponente, die den höheren Zustand gewählt hat, kann diesen wieder in den Normalzustand zurücksetzen.

25

Liste der zitierten Literatur

- 5 [JPM99] Jörg P. Müller: "The Design of Intelligent Agents. A Layered Approach", Springer Verlag, Berlin (Februar 1999)

Patentansprüche

1. Recheneinheit zum Implementieren eines Multiagenten-
5 Systems mit:

a) Mitteln zum Implementieren einer ersten Ebene, die ein
System darstellt;

b) Mitteln zum Implementieren einer zweiten Ebene, die eine
abstrakte, auf Konzepte reduzierte Repräsentation des Systems
10 der ersten Ebene aufweist;

c) Mitteln zum Implementieren einer dritten Ebene, die Mit-
tel zum Implementieren der Agenten des Multiagenten-Systems
aufweist;

d) einer Schnittstelle zur Kommunikation zwischen der ersten
15 und der zweiten Ebene; und mit

e) einer Schnittstelle zur Kommunikation zwischen der zwei-
ten und der dritten Ebene.

2. Recheneinheit nach dem vorhergehenden Anspruch,
20 **dadurch gekennzeichnet,**

dass die Schnittstelle zur Kommunikation zwischen der ersten
und der zweiten Ebene eine Struktur aufweist, die der Struktur
einer Schnittstelle zwischen der ersten Ebene und einem mensch-
lichen Bediener entspricht.

25

3. Recheneinheit nach einem der vorhergehenden Ansprüche,
dadurch gekennzeichnet,

dass die Schnittstelle zur Kommunikation zwischen der zwei-
ten und der dritten Ebene über Aktionen und Perzepte erfolgt.

30

4. Recheneinheit nach dem vorhergehenden Anspruch,
dadurch gekennzeichnet,

dass einem Agent mindestens ein vorgegebener virtueller Sensor zugeordnet ist; und

dass die Perzepte dieses Agenten gemäß den jeweiligen Sensoren dieses Agenten gefiltert sind.

5

5. Recheneinheit zum Implementieren eines Multiagenten-Systems:

a) wobei in einer ersten Ebene ein System vorhanden ist; wobei die Recheneinheit folgendes aufweist:

10 b) Mittel zum Implementieren einer zweiten Ebene, die eine abstrakte, auf Konzepte reduzierte Repräsentation des Systems der ersten Ebene aufweist;

c) Mittel zum Implementieren einer dritten Ebene, die Mittel zum Implementieren der Agenten des Multiagenten-Systems auf-
15 weist;

d) eine Schnittstelle zur Kommunikation zwischen der ersten und der zweiten Ebene; und mit

e) eine Schnittstelle zur Kommunikation zwischen der zweiten und der dritten Ebene.

20

6. Simulator für ein Multiagenten-Systems mit:

a) Mitteln zum Implementieren einer ersten Ebene, die ein System darstellt;

25 b) Mitteln zum Implementieren einer zweiten Ebene, die eine abstrakte, auf Konzepte reduzierte Repräsentation des Systems der ersten Ebene aufweist;

c) Mitteln zum Implementieren einer dritten Ebene, die Mittel zum Implementieren der Agenten des Multiagenten-Systems aufweist;

30 d) einer Schnittstelle zur Kommunikation zwischen der ersten und der zweiten Ebene; und mit

e) einer Schnittstelle zur Kommunikation zwischen der zweiten und der dritten Ebene.

7. Architektur eines Computerprogramms zum Implementieren eines Multiagenten-Systems mit:

- a) einer ersten Ebene, die ein System darstellt;
- 5 b) einer zweiten Ebene, die eine abstrakte, auf Konzepte reduzierte Repräsentation des Systems der ersten Ebene aufweist;
- c) einer dritten Ebene, die Mittel zum Implementieren der Agenten des Multiagenten-Systems aufweist;
- d) einer Schnittstelle zur Kommunikation zwischen der ersten
- 10 und der zweiten Ebene; und mit
- e) einer Schnittstelle zur Kommunikation zwischen der zweiten und der dritten Ebene.

8. Architektur eines Computerprogramms zum Implementieren eines Multiagenten-Systems:

- a) wobei in einer ersten Ebene ein System vorhanden ist; wobei die Architektur folgendes aufweist:
- b) eine zweite Ebene, die eine abstrakte, auf Konzepte reduzierte Repräsentation des Systems der ersten Ebene aufweist;
- 20 c) eine dritte Ebene, die Mittel zum Implementieren der Agenten des Multiagenten-Systems aufweist;
- d) eine Schnittstelle zur Kommunikation zwischen der ersten und der zweiten Ebene; und mit
- e) eine Schnittstelle zur Kommunikation zwischen der zweiten
- 25 und der dritten Ebene.

9. Middleware für die Entwicklung eines Computerprogramms zum Implementieren eines Multiagenten-Systems:

- a) wobei das Computerprogramm in einer ersten Ebene ein System simuliert;
- 30 b) wobei die Middleware eine zweite Ebene aufweist, die eine abstrakte, auf Konzepte reduzierte Repräsentation des Systems der ersten Ebene aufweist;

c) wobei die Middleware eine dritte Ebene aufweist, die Mittel zum Implementieren der Agenten des Multiagenten-Systems aufweist;

d) wobei die Middleware eine Schnittstelle zur Kommunikation
5 zwischen der ersten und der zweiten Ebene aufweist; und

e) wobei die Middleware einer Schnittstelle zur Kommunikation zwischen der zweiten und der dritten Ebene aufweist.

10. Verfahren zum Implementieren eines Multiagenten-Systems:

10 a) wobei eine erste Ebene simuliert wird, die ein System darstellt;

b) wobei eine zweite Ebene simuliert wird, die eine abstrakte, auf Konzepte reduzierte Repräsentation des Systems der ersten Ebene aufweist;

15 c) wobei eine dritte Ebene simuliert wird, die Mittel zum Implementieren der Agenten des Multiagenten-Systems aufweist;

d) wobei Informationen zwischen der ersten und der zweiten Ebene ausgetauscht werden; und

e) wobei Informationen zwischen der zweiten und der dritten
20 Ebene ausgetauscht werden.

11. Computerprogramm, dadurch gekennzeichnet, dass es bei Ablauf auf einer Recheneinheit, einem Mikrocontroller, DSP, FPGA oder Computer oder auf einer Mehrzahl davon in einem Netzwerk das Verfahren nach dem Verfahrensanspruch ausführt.
25

12. Computerprogramm mit Programmcode-Mitteln, um ein Verfahren gemäß dem Verfahrensanspruch durchzuführen, wenn das Computerprogramm auf einer Recheneinheit, einem Mikrocontroller, DSP, FPGA oder Computer oder auf einer Mehrzahl davon in
30 einem Netzwerk ausgeführt wird.

13. Computerprogramm mit Programmcode-Mitteln gemäß dem vorhergehenden Anspruch, die auf einem computerlesbaren Datenträger gespeichert sind.

5 14. Datenträger, auf dem eine Datenstruktur gespeichert ist, die nach einem Laden in einen Arbeits- und/oder Hauptspeicher einer Recheneinheit, eines Mikrocontrollers, DSPs, FPGAs oder Computers oder einer Mehrzahl davon in einem Netzwerk das Verfahren nach dem Verfahrensanspruch ausführt.

10 15. Computerprogramm-Produkt mit auf einem maschinenlesbaren Träger gespeicherten Programmcode-Mitteln, um alle Schritte gemäß dem Verfahrensanspruch durchzuführen, wenn das Programm auf einer Recheneinheit, einem Mikrocontroller, DSP, FPGA oder Computer oder auf einer Mehrzahl davon in einem Netzwerk ausgeführt wird.

20 16. Moduliertes Datensignal, welches von einer Recheneinheit, einem Mikrocontroller, DSP, FPGA oder Computer oder von einer Mehrzahl davon in einem Netzwerk ausführbare Instruktionen zum Ausführen eines Verfahrens nach dem Verfahrensanspruch enthält.

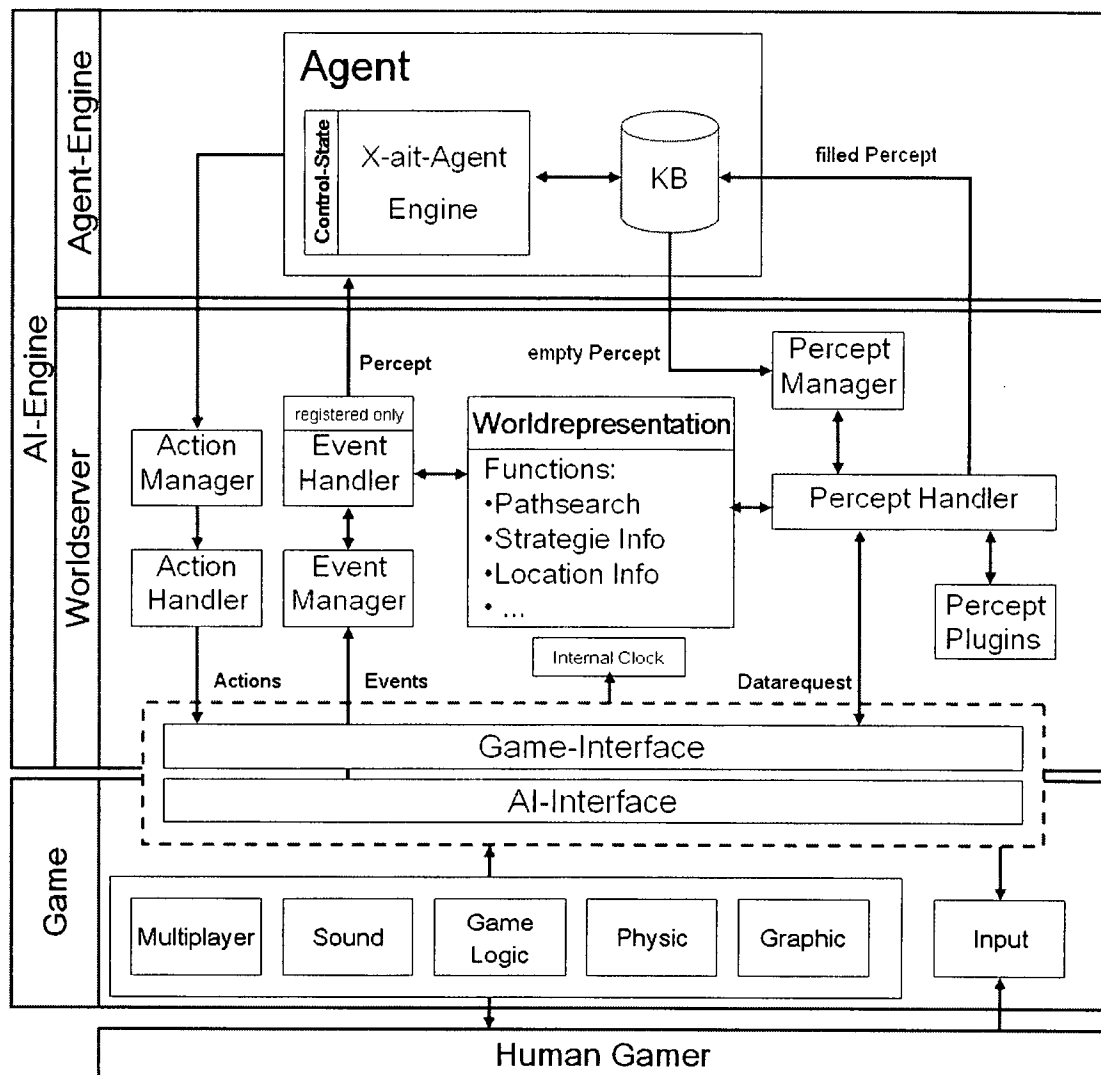


Fig. 2-1

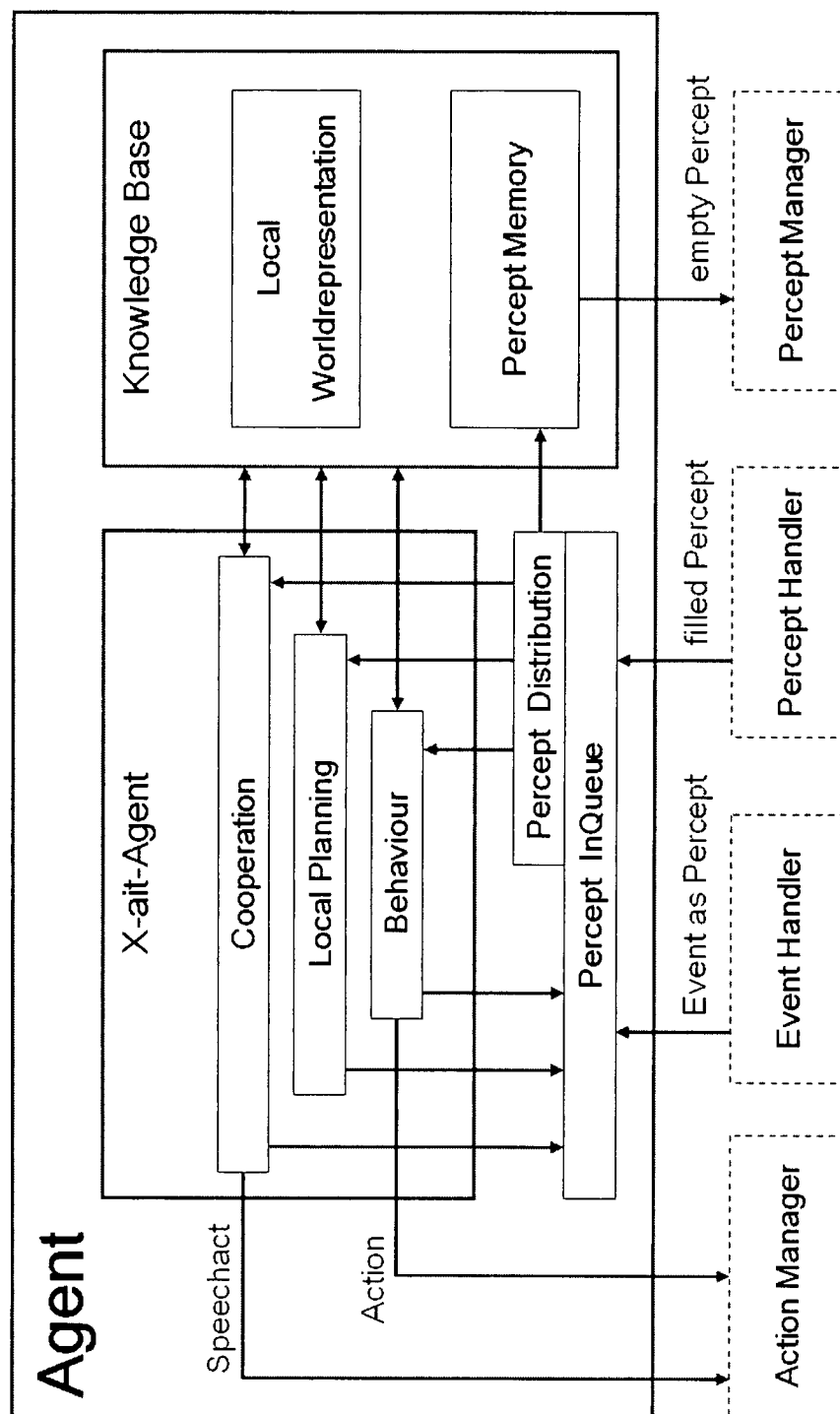


Fig. 3-1

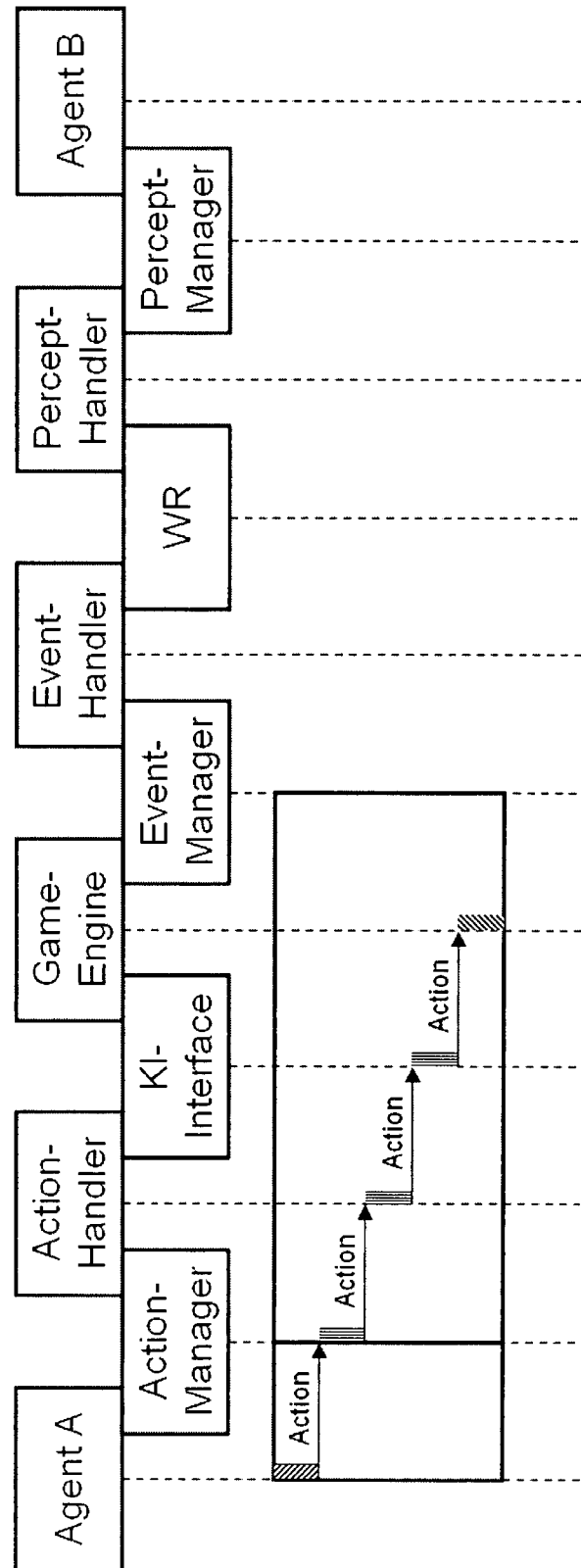


Fig. 4-1

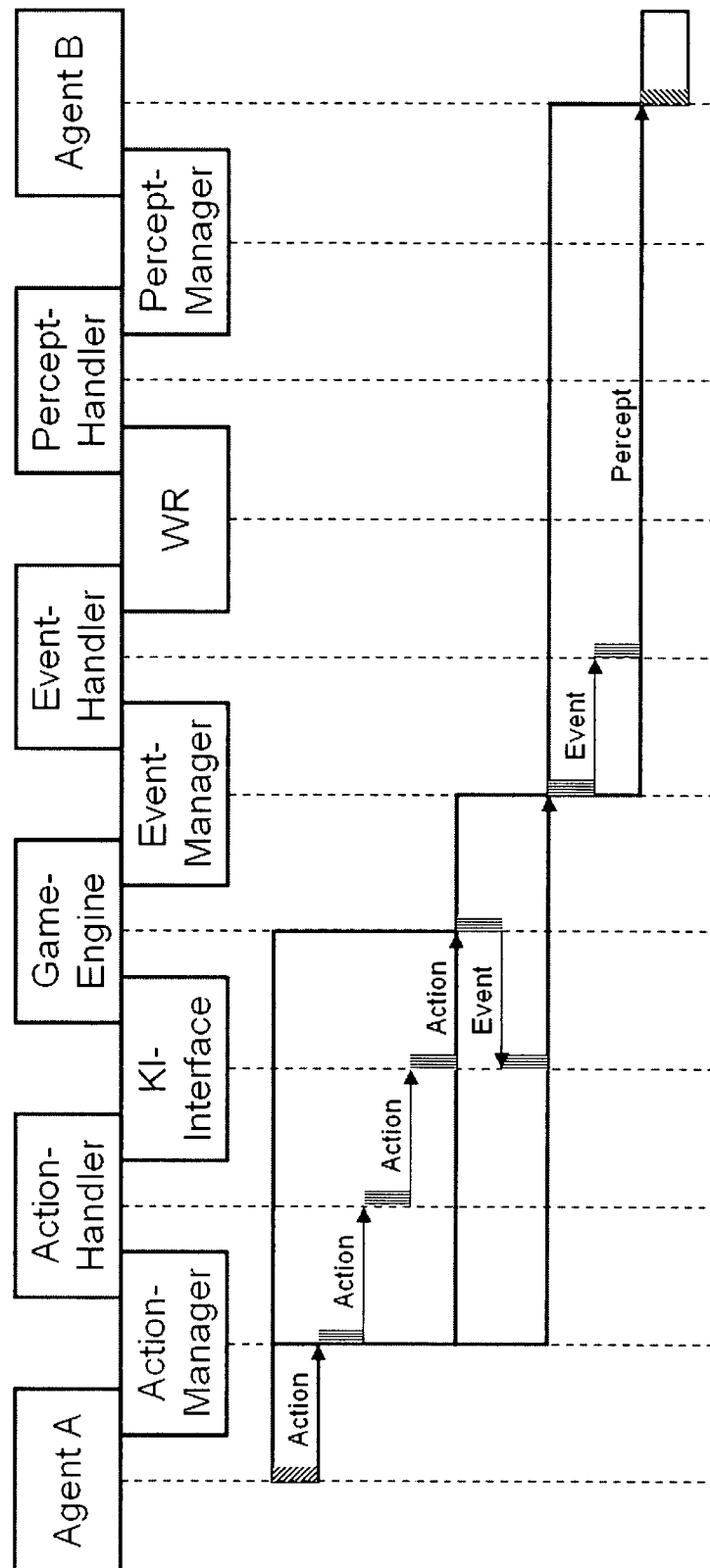


Fig. 4-2

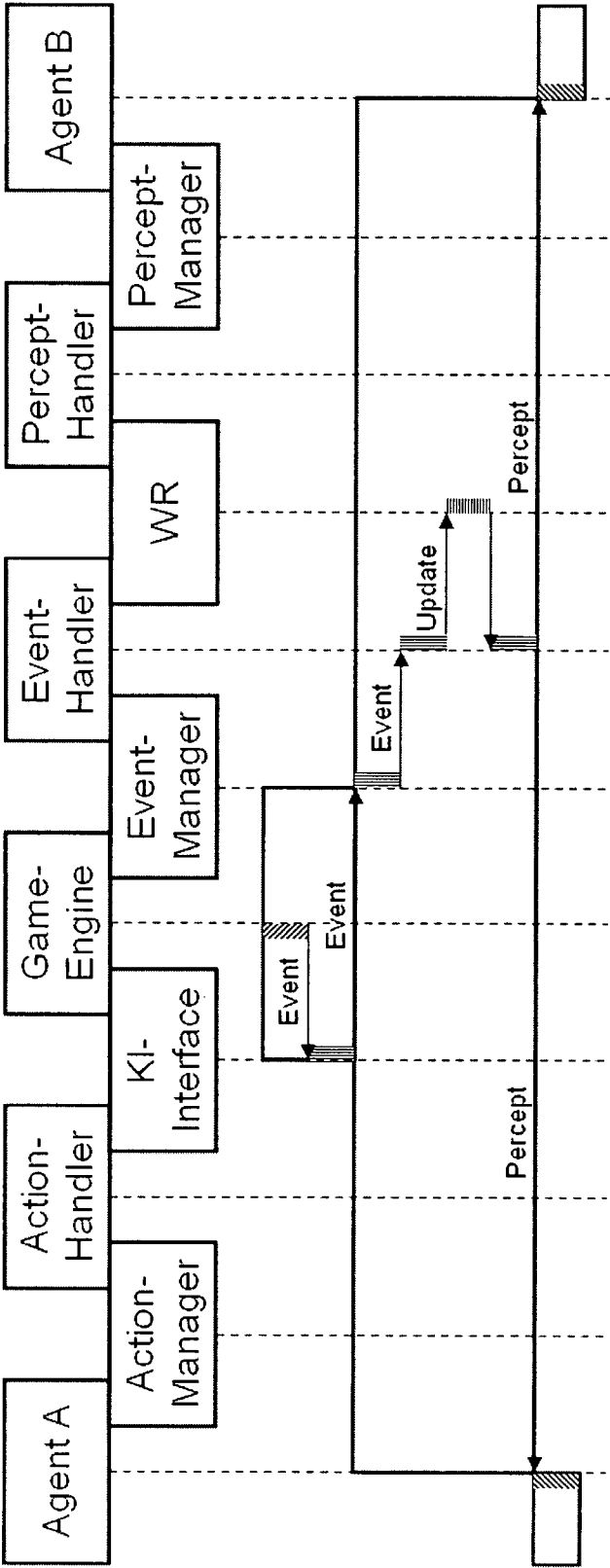


Fig. 4-3

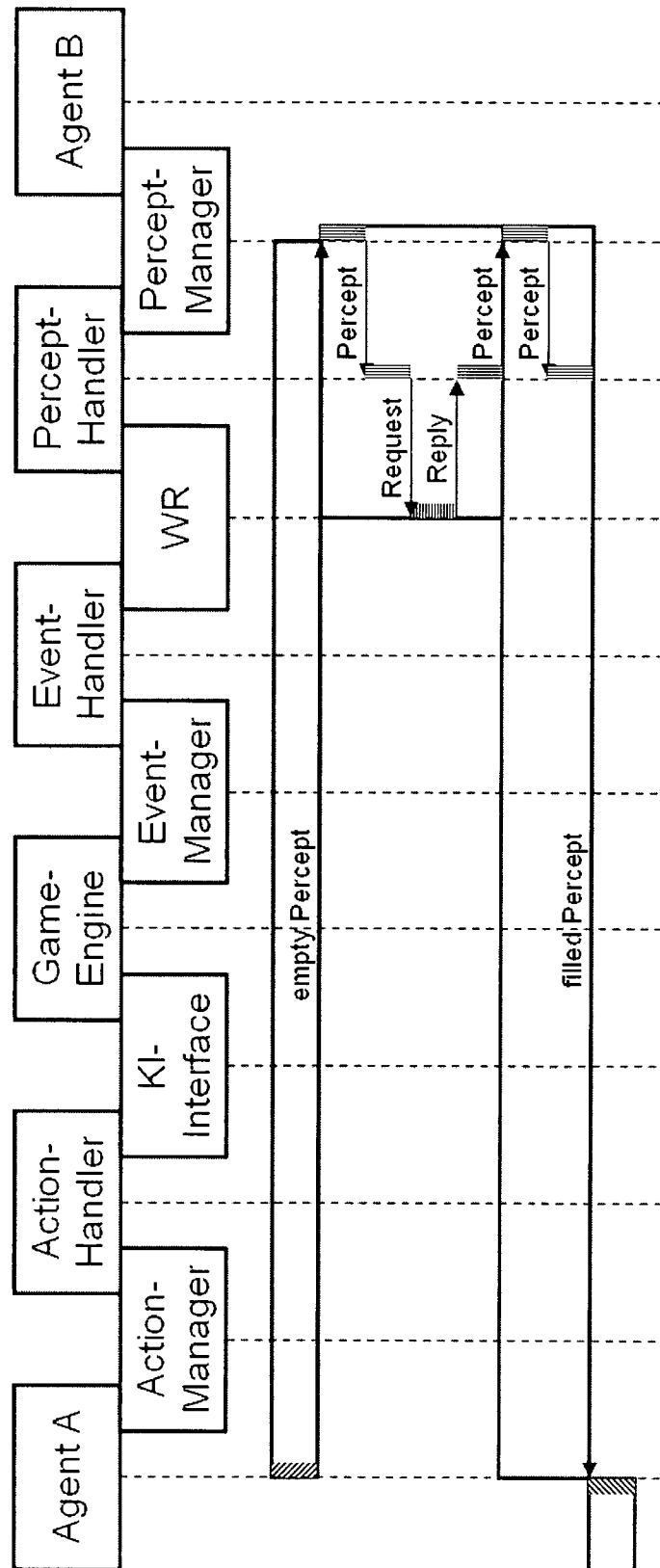


Fig. 4-4

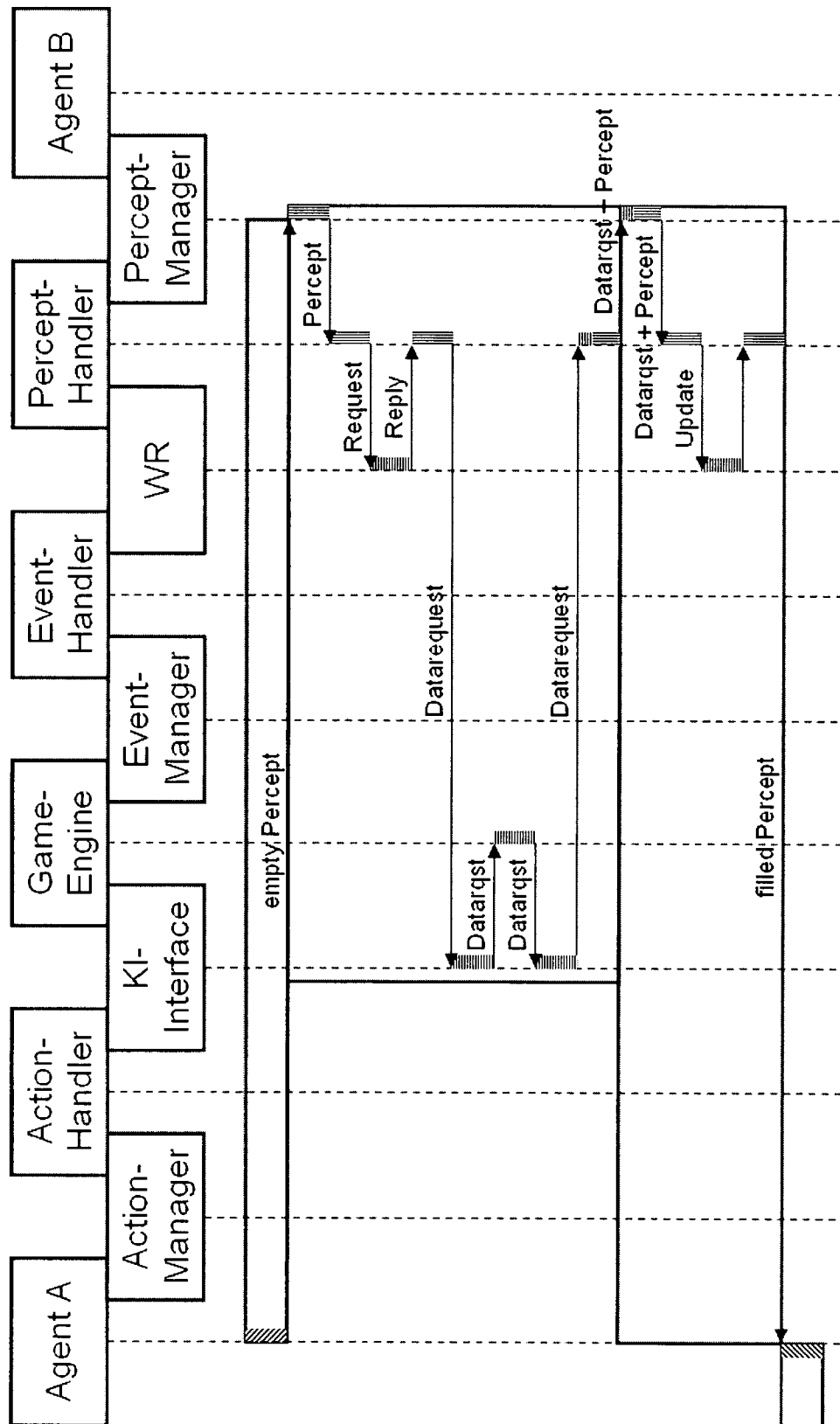


Fig. 4-4

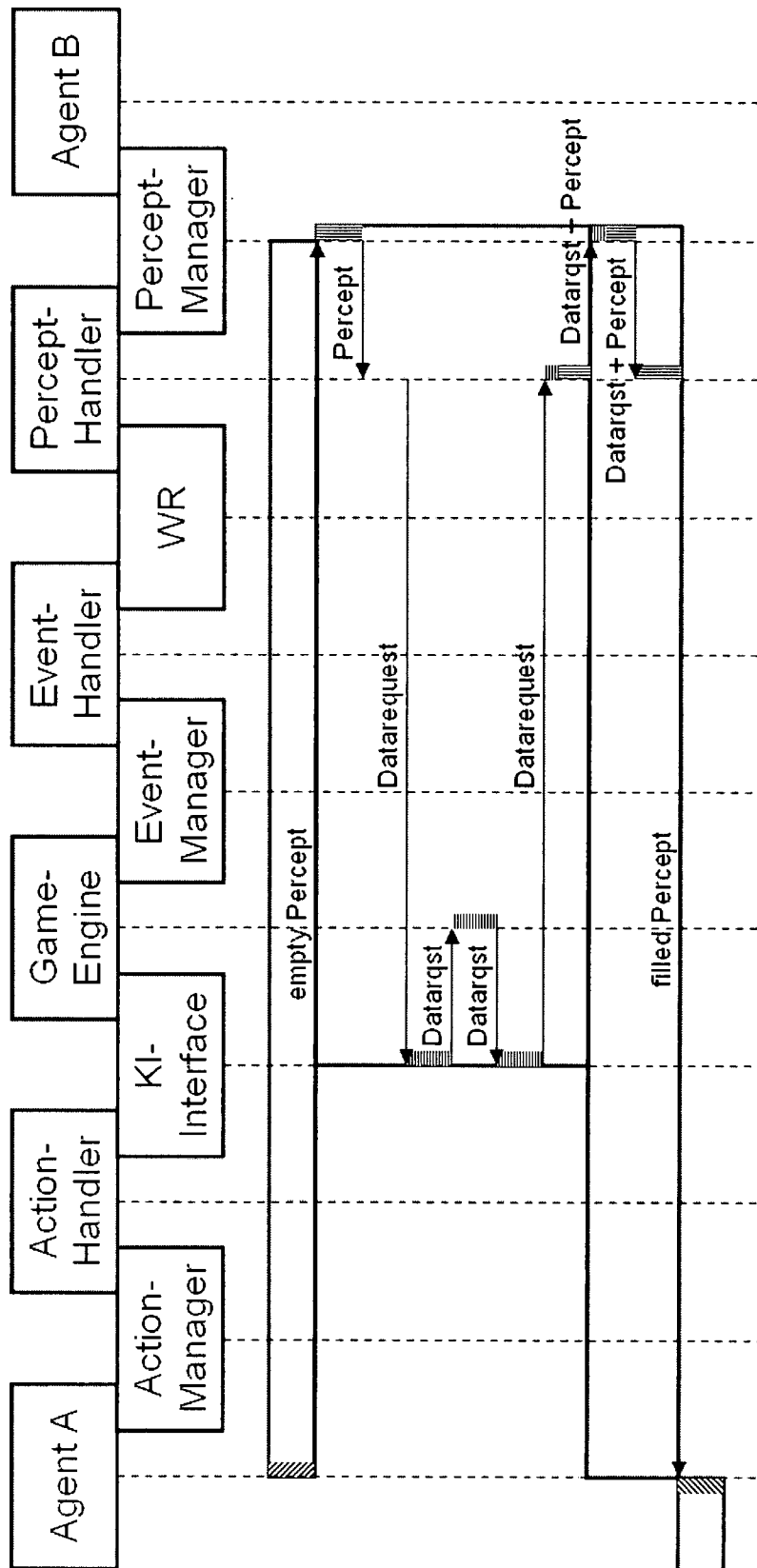


Fig. 4-5

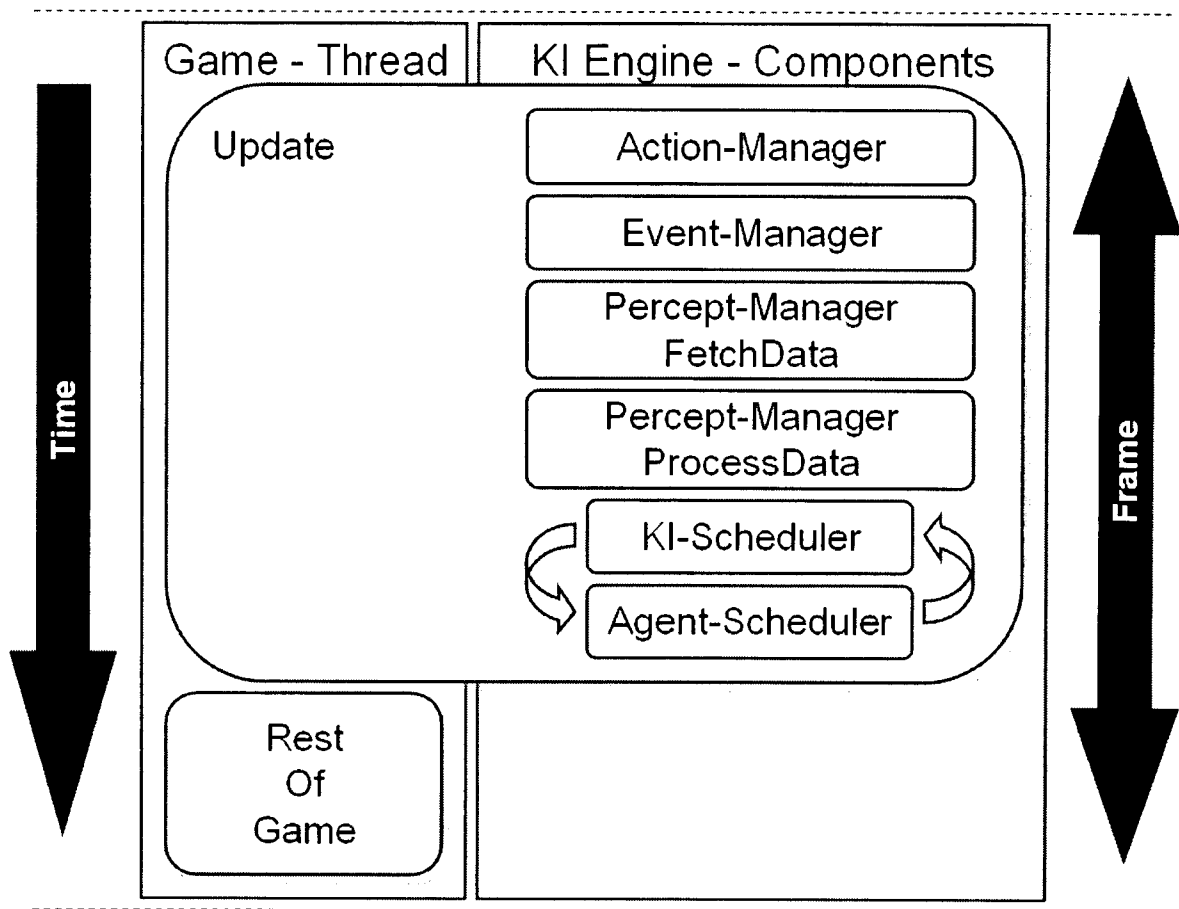


Fig. 5-1

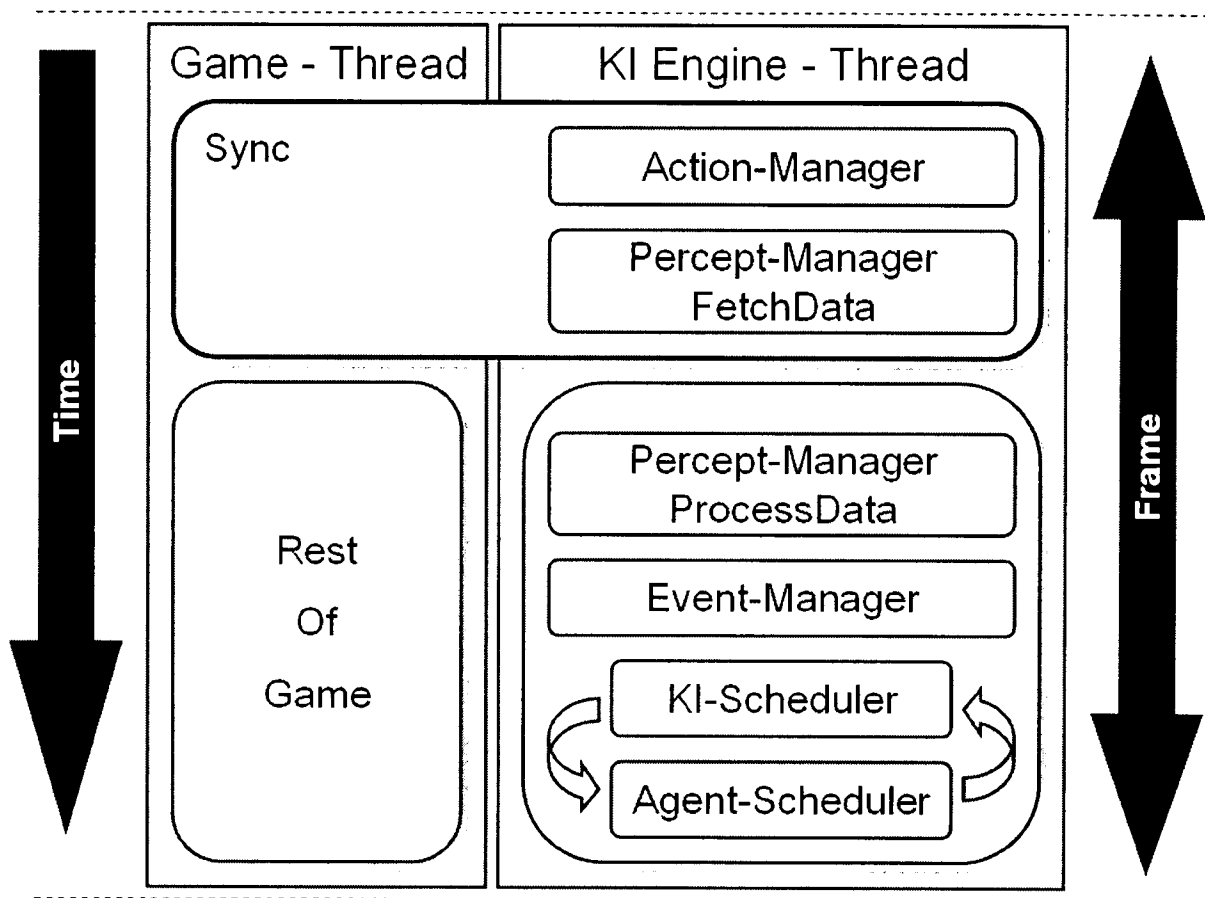


Fig. 5-2