

Seminar zum Thema „Künstliche Intelligenz“
Software-Agenten

Kay Hasselbach · mi5482

- 1. Einführung**
- 2. Grundlagen**
- 3. Agenten-Strukturen**
- 4. Architekturen**
- 5. Beispiel: „RoboCup“**
- 6. Quellen**

1. Einführung

- Definition Agent
- Einführende Beispiele

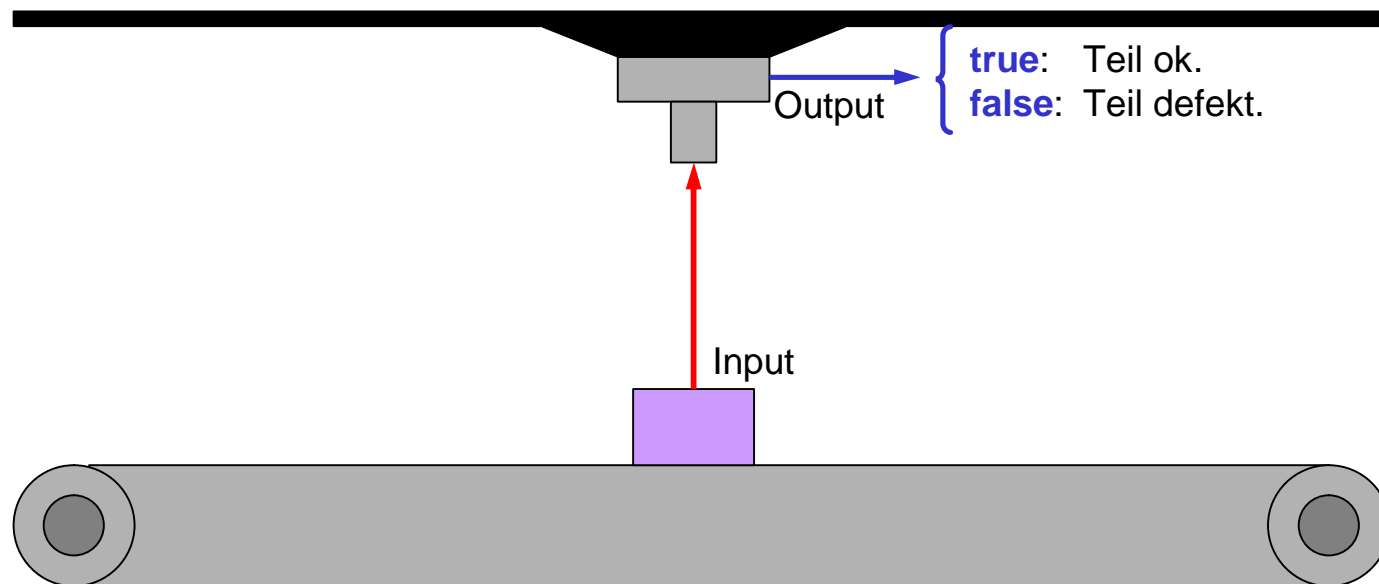
Definition Agent:

- Arbeitet autonom
- Interagiert mit einer Umwelt
- Kann Aufträge entgegennehmen oder Ziele verfolgen
- Andauernde Verfügbarkeit

...aber wozu braucht man nun Agenten?

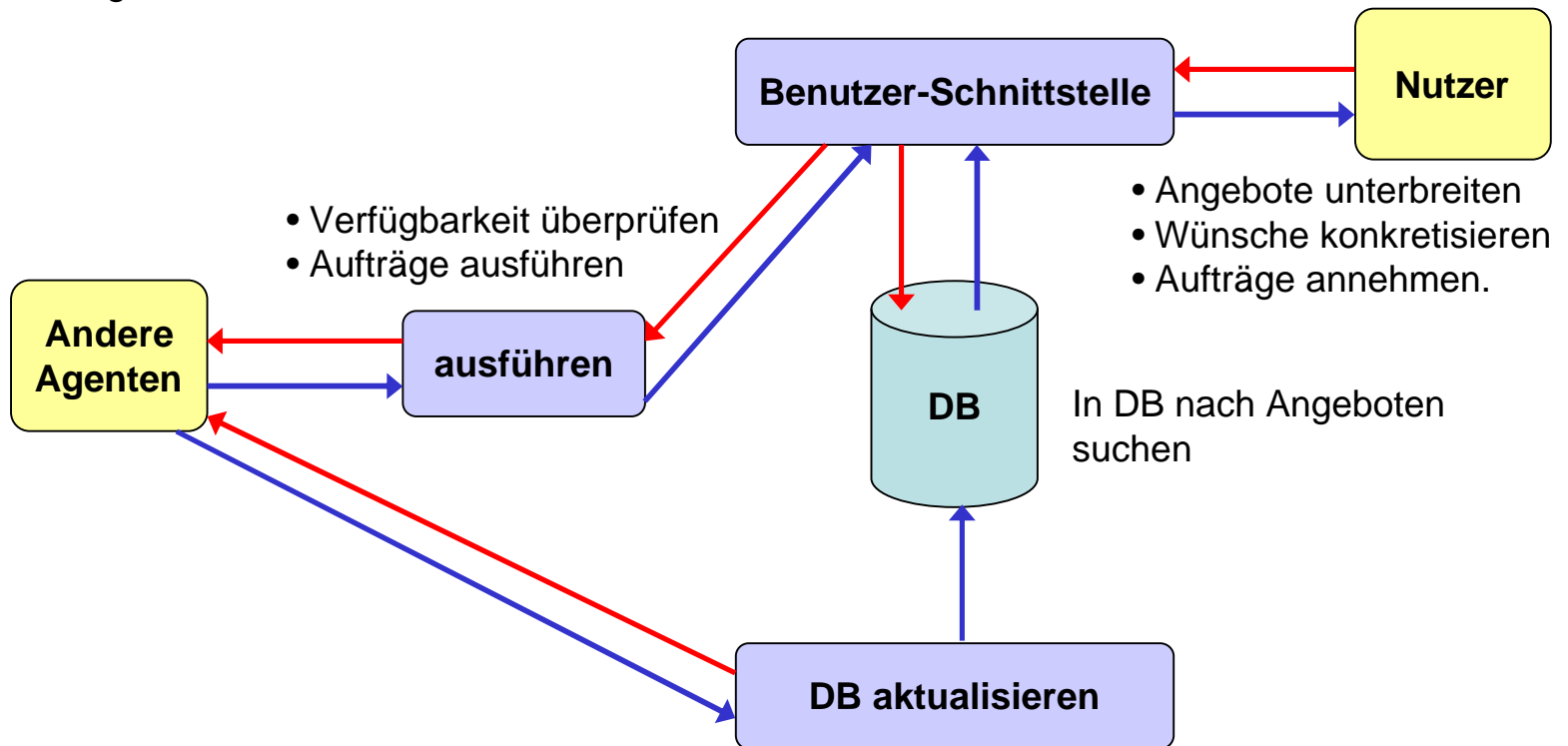
Einführende Beispiele:

Klassifizierung von Teilen:



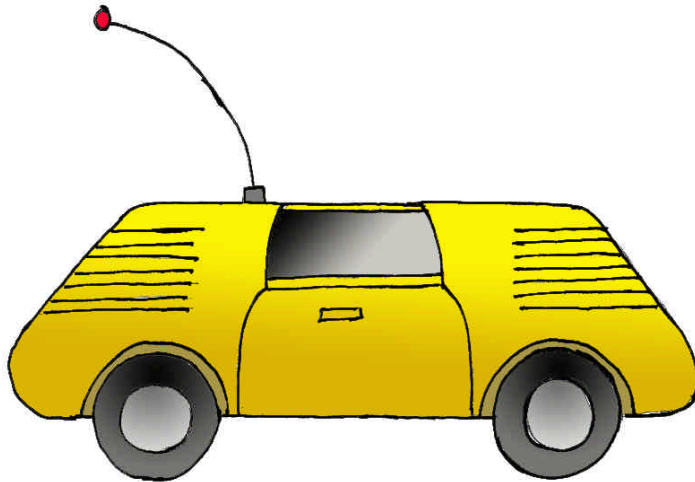
Einführende Beispiele:

Verkaufsagent:



Einführende Beispiele:

Etwas Sci-Fi: Elektronischer Taxifahrer:



- Interaktion mit Fahrgast (Ziel bestimmen etc.)
- Eigenständiges Erreichen des Fahrziels.
- Maximierung des Gewinns.
- Gleichzeitig: Zufriedenstellung des Kunden.
- Minimierung von Risiken und Kosten.

2. Grundlagen

- Umwelt des Agenten
- Rationalität
- Parallelität und Nebenläufigkeit

Umwelt des Agenten:

- Vollständig beobachtbar vs. Teilweise beobachtbar
- Deterministisch vs. Nichtdeterministisch
- Episodisch vs. Sequenziell
- Statisch vs. Dynamisch
- Diskret vs. Stetig
(Anwendbar auf: Die Zeit, die Umgebung, die Wahrnehmung, die Aktionen des Agenten)
- Einzelagent vs. Multiagent

Unterschiedliche Betrachtungsweisen:

- Sicht des Entwicklers (Sicht zur Entwurfszeit)
- Sicht des Agenten (Sicht zur Laufzeit)

Rationalität:

Ausgangslage:

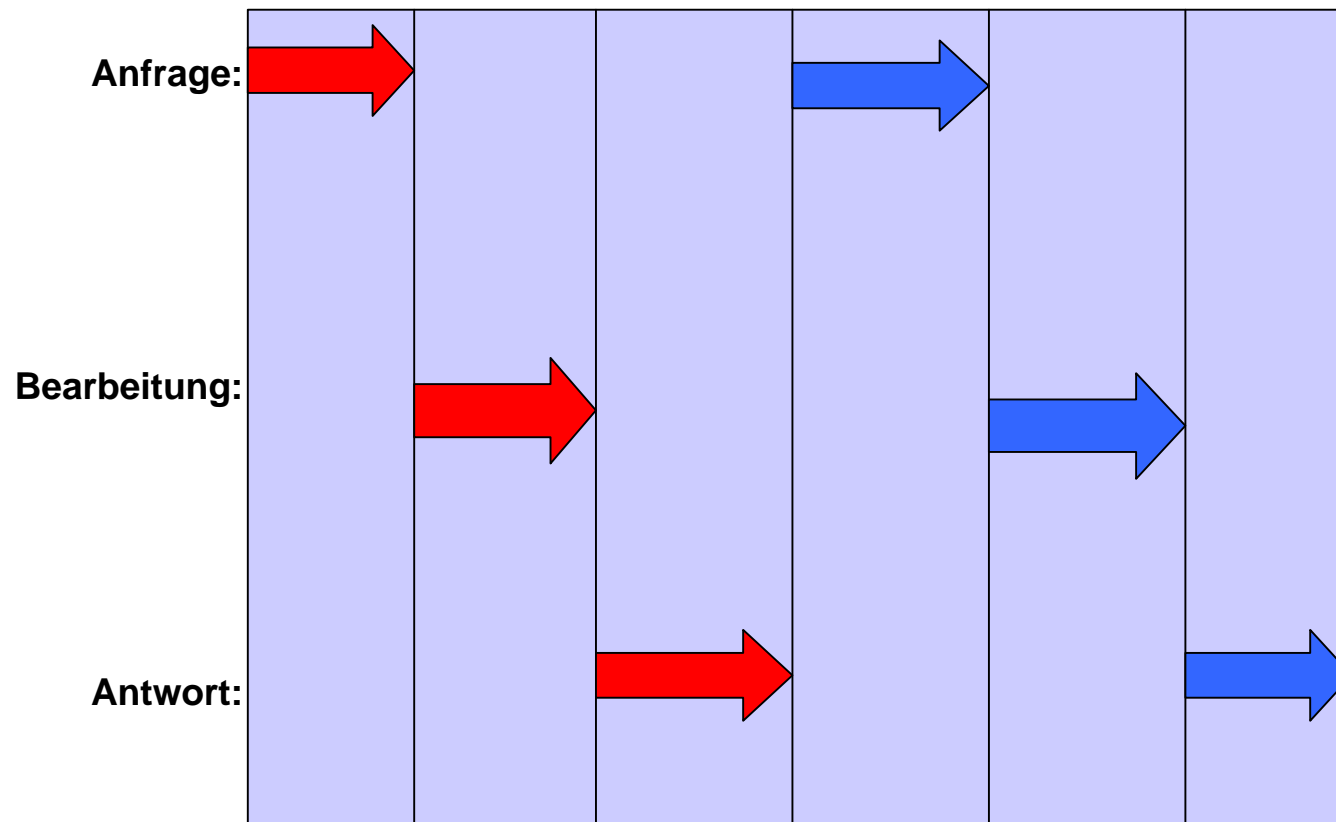
- Agent verfügt nur über eine eingeschränkte Sichtweise (ist also nicht allwissend)
- Kann aus Effizienzgründen eine bewusst getroffene Entscheidung sein

⇒ Das optimale (beste) Verhalten ist für den Agenten nicht voraussagbar

Rationaler Agent: Wählt auf effiziente Weise auf Grundlage seines Wissens über die Umwelt immer die Aktionen aus, von denen er erwarten kann, dass sie seine *Leistungsbewertung* maximieren.

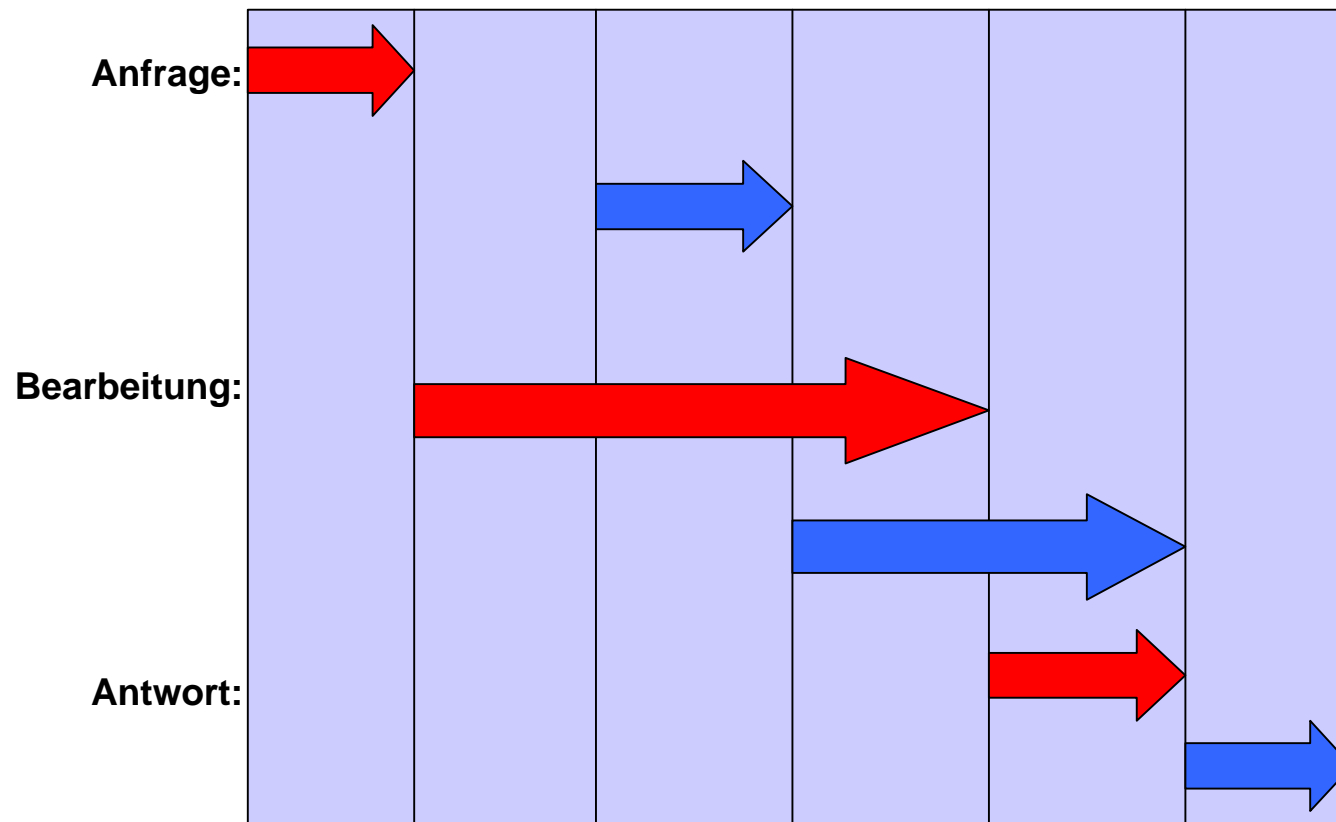
Parallelität und Nebenläufigkeit:

Beispiel: Sequenzielle Abarbeitung von Aufträgen



Parallelität und Nebenläufigkeit:

Beispiel: Parallele Abarbeitung von Aufträgen



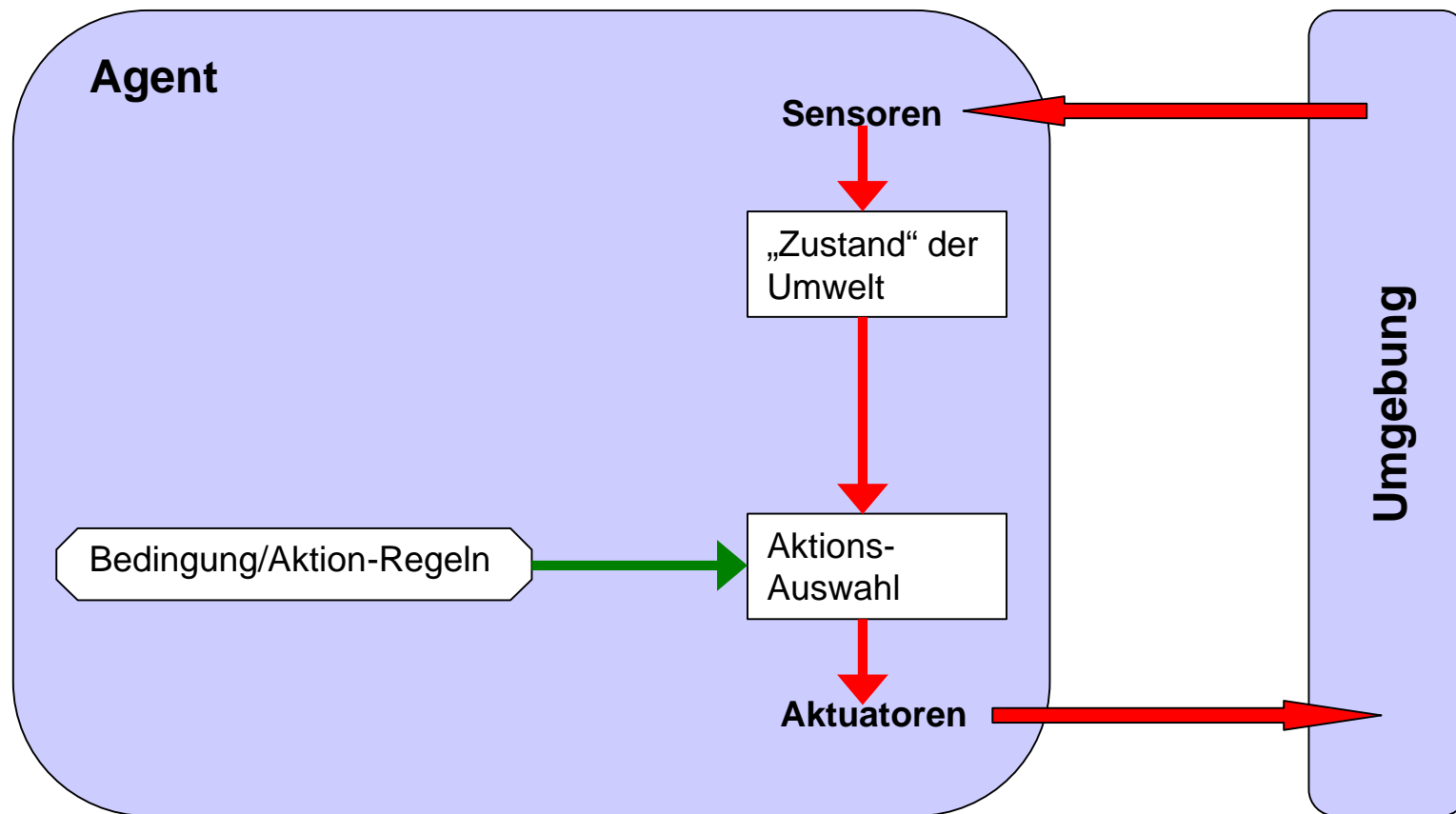
3. Agenten-Strukturen

- Komponenten eines (Software-)Agenten
- Einfacher Reflex-Agent
- Modelbasierter Reflex-Agent
- Zielbasierter Agent
- Nutzenbasierter Agent
- Stimulus Response und andauernde Tätigkeit
- Lernende Agenten

Komponenten eines (Software-)Agenten:

- Umweltkopplung
- Umweltmodell
- Steuerung und Management
- Fähigkeiten
- Entscheidungskomponente

Einfacher Reflex-Agent:



Einfacher Reflex-Agent:

Pseudocode:

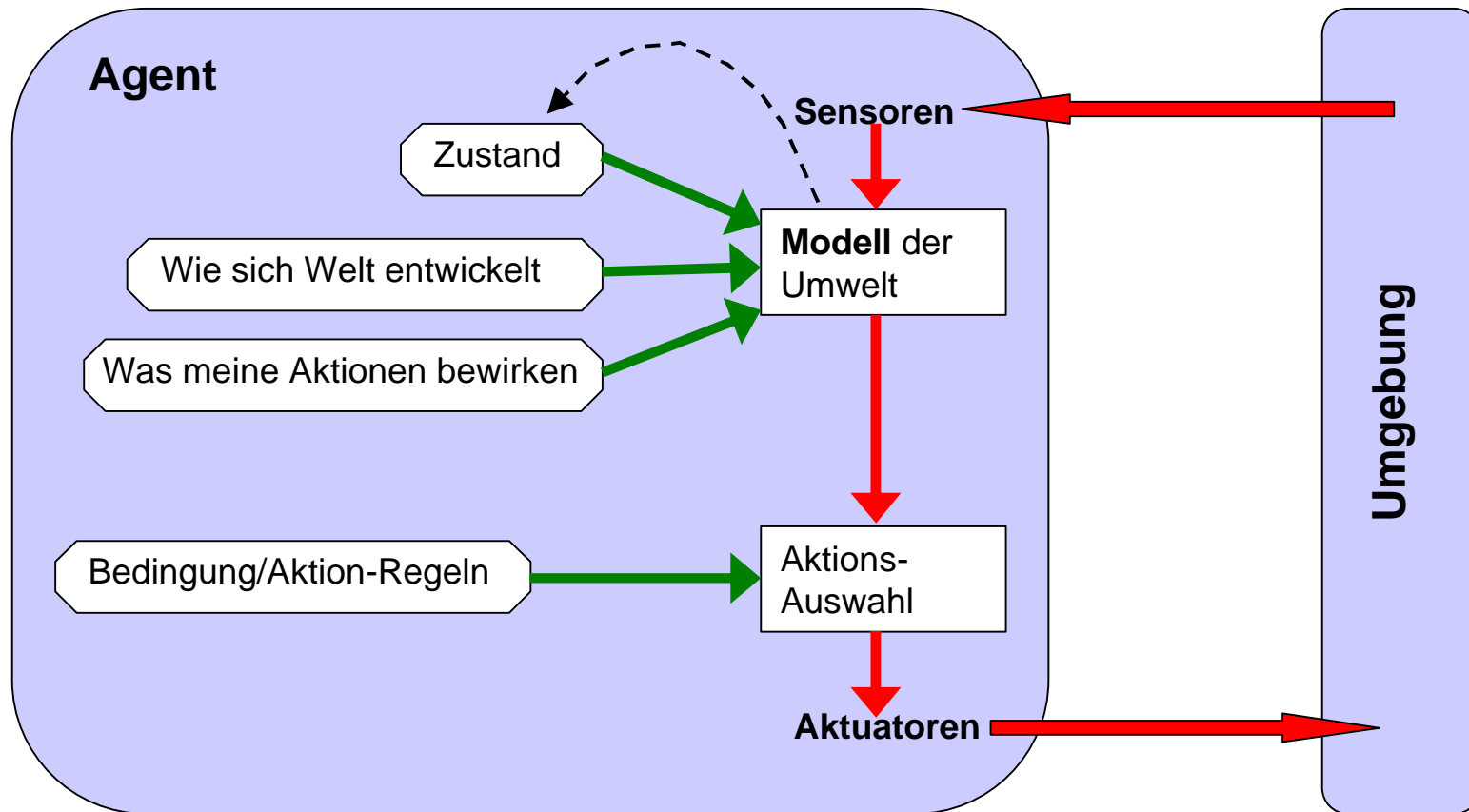
```
var
  Rules rules; /* Eine Menge von Bedingung/Aktion-Regeln (Fähigkeiten). */

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

Einfacher Reflex-Agent:

- Funktioniert nur, wenn Entscheidung **ausschließlich** aufgrund des aktuellen Zustands getroffen werden kann (episodisch).
 - ⇒ Funktioniert nur in **vollständig beobachtbaren** Umgebungen.
- Endlosschleifen in teilweise beobachtbaren Umgebungen sind häufig unvermeidbar.
 - ⇒ Für teilweise beobachtbare Umgebungen besser komplexere Agenten verwenden.

Modelbasierter Reflex-Agent:



Modelbasierter Reflex-Agent:

Pseudocode:

var

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

repeat

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

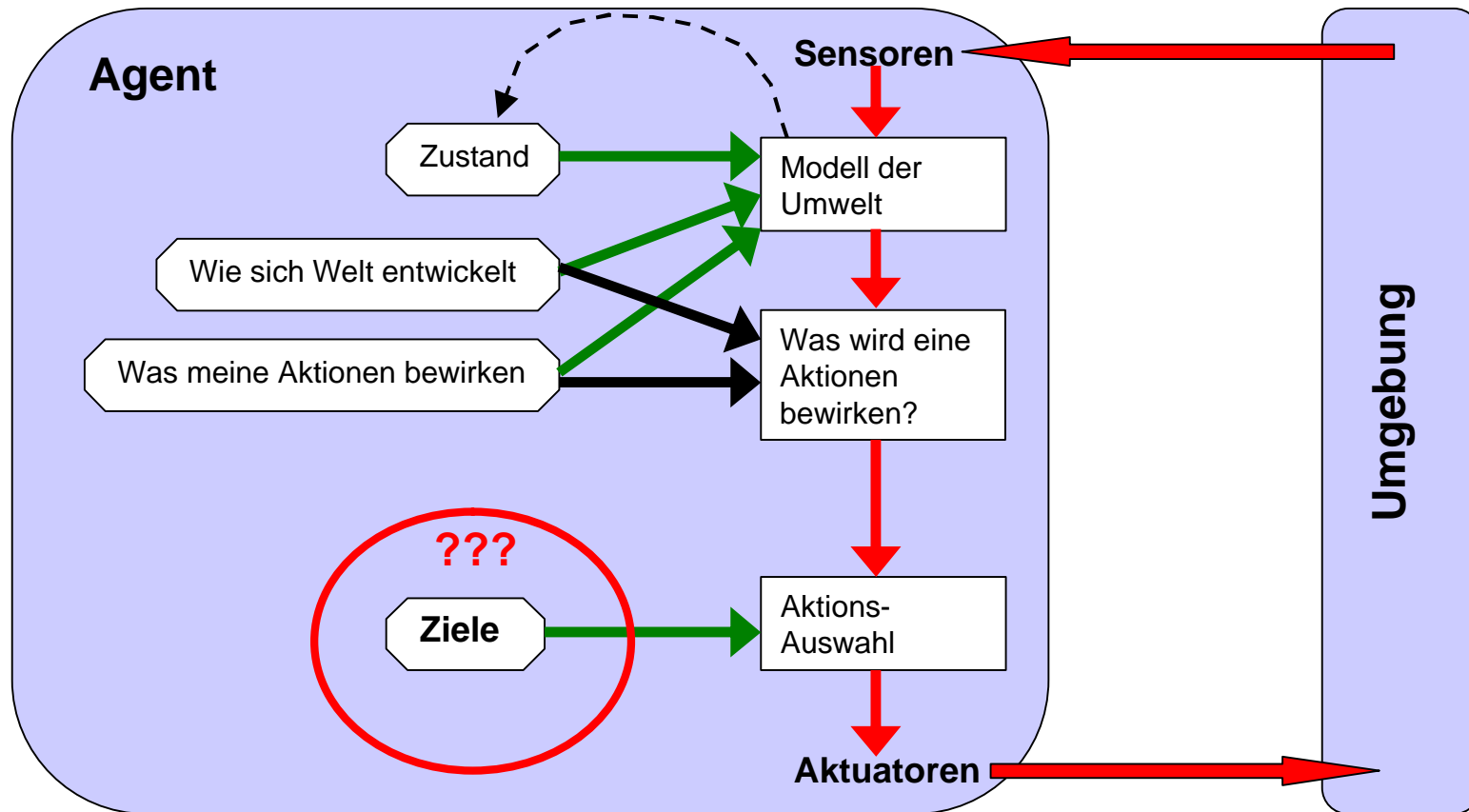
forever

Modelbasierter Reflex-Agent:

- Wissen um aktuellen Zustand der Welt ist nicht immer ausreichend um richtige Entscheidung zu treffen.
 - Das Verfolgen längerfristiger Ziele ist so nicht möglich.
- ® **Zielbasierte Agenten.**

3. Agenten-Strukturen

Zielbasierter Agent:



Ziele:

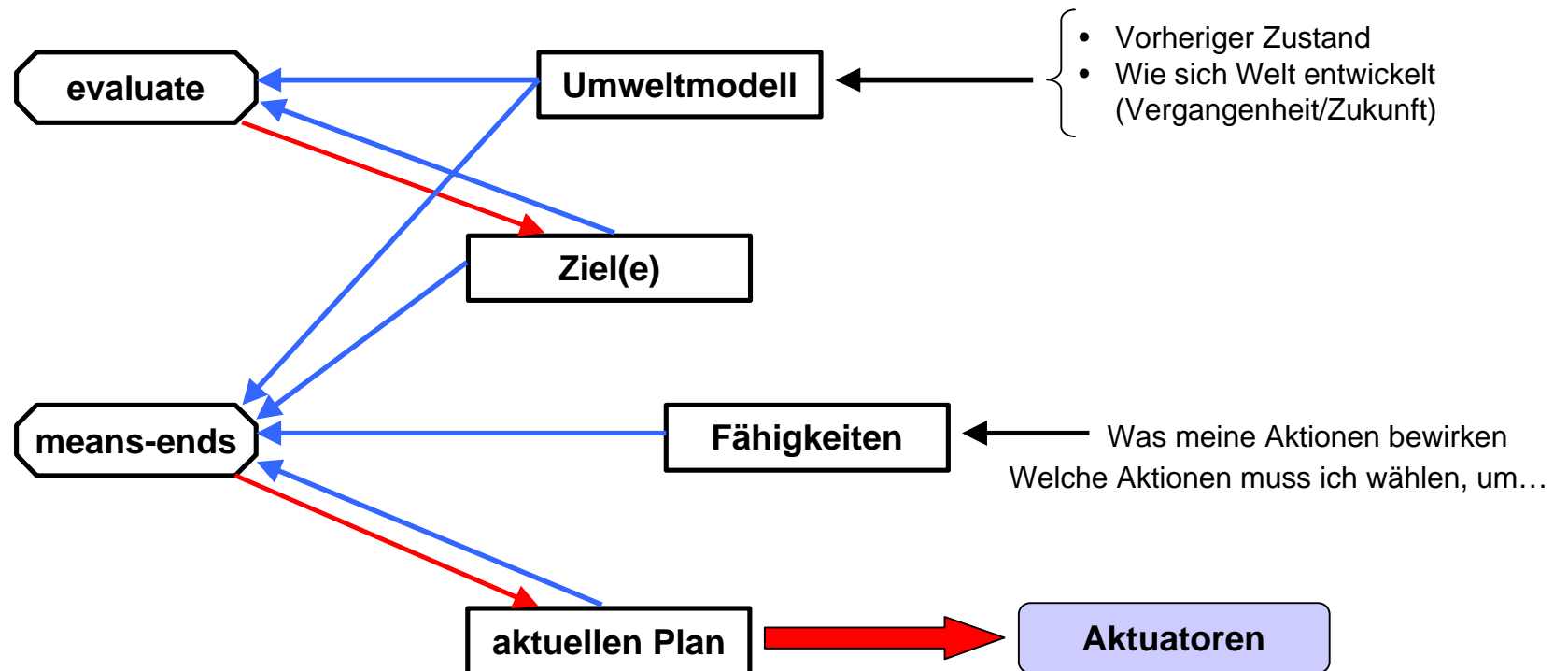
Allgemein:

- Agent bestimmt ein erstrebenswertes Ziel, dass er zu erreichen versucht.
- Es gibt kurz, mittel oder langfristige Ziele (Einige sind schnell erreichbar, andere nicht)
⇒ Ziele können wiederum aus Teilzielen bestehen.
- Prinzipiell könnte ein Agent mehrere Ziele gleichzeitig verfolgen.
- Zunächst nur Betrachtungsweise für ein Ziel gleichzeitig.
→ Zum nebenläufigen Verfolgen mehrerer Ziele später mehr (BDI-Architektur)

3. Agenten-Strukturen

Zielbasierter Agent:

Deliberation und Means-ends-reasining:



3. Agenten-Strukturen

Zielbasierter Agent:

Pseudocode:

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

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

Zielbasierter Agent:

Deliberation: Festlegen eines Ziels.

Means-ends-reasoning: Auswahl von Fähigkeiten/Plänen zur Erreichung des Ziels.

- Es dürfen in der ersten Phase keine Ziele festgelegt werden, für die es keine Fähigkeiten gibt → Realismus-Forderung.
- Zielrevision, falls ausgewähltes Ziel nicht erreichbar ist.
- Ziele können wieder aus Teilzielen bestehen (Mögliche Datenstruktur: Keller).

Nutzenbasierter Agent:

- Nur eine „Erweiterung“ des Zielbasierten Agenten.
- Nutzenfunktion bildet Zustand / Zustandsfolge auf reelle Zahl ab, die Grad des Nutzens beschreibt.
- Ziele / Teilziele mit hohem Nutzen sind solchen mit niedrigem Nutzen vorzuziehen.
- **Rationaler Zielbasierter Agent \hat{U} Nutzenbasierter Agent**

Stimulus Response oder andauernde Tätigkeit:

- Einfacher Reflex-Agent
 - Modelbasierter Reflex-Agent
 - Zielbasierter Agent
 - Nutzenbasierter Agent
- } **Stimulus Response**
- } **Andauernde Tätigkeit**

...aber auch „Mischformen“ sind denkbar.

Lernende Agenten:

- Es ist oft nicht möglich/sinnvoll komplettes Wissen über Umwelt statisch im Agenten abzulegen (Große Datenbestände, zu unflexibel)
- Lernender Agent kann sein Wissen selbst aktualisieren / erweitern.
 - Agent bewertet Erfolg seiner Aktionen (Feedback) und verbessert so seine Auswahlkriterien für Aktionen.
 - Das Ausführen zusätzlicher Aktionen zur Wissensgenerierung kann sinnvoll sein.

4. Architekturen

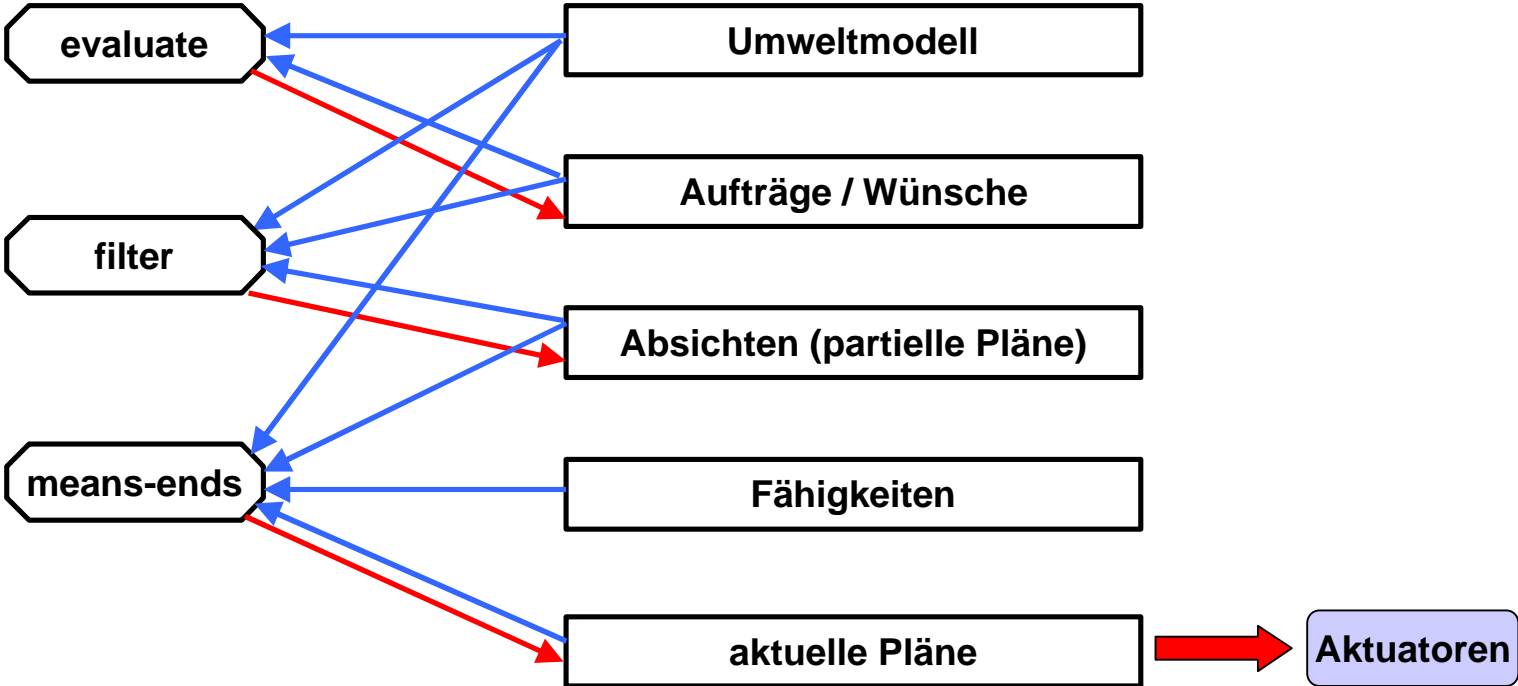
- BDI-Architektur (Belief – Desire – Intention)
- Schichtenmodel
- Hybride Architekturen

BDI-Architektur (Belief – Desire - Intention):

- Sinnvollerweise sollte ein Agent mehrere Ziele gleichzeitig verfolgen können.
 - Ziele dürfen untereinander nicht im Konflikt stehen (starke Realismus-Forderung).
 - Stehen Ziele im Konflikt, muss eine Auswahl getroffen werden.
 - Bei Auswahl:
Flexibilität (auf bessere Alternativen ausweichen) vs. **Stabilität** (alte Ziele zu ende verfolgen)
 - Ziele, die nicht mehr erreicht werden können, müssen ebenfalls entfernt werden.

 - Unterteilung der Ziele in:
 - Wünsche (desire) – Noch **keine Realismus-Forderung**.
 - Absichten / Verpflichtungen (intention) – Es gilt **starke Realismus-Forderung**
- ⇒ Wünsche können „im Hinterkopf“ behalten werden (Anlehnung an menschliches Verhalten).

BDI-Architektur (Belief – Desire - Intention):



BDI-Architektur (Belief – Desire - Intention):

Pseudocode:

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

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

Schichtenmodel:

Problem:

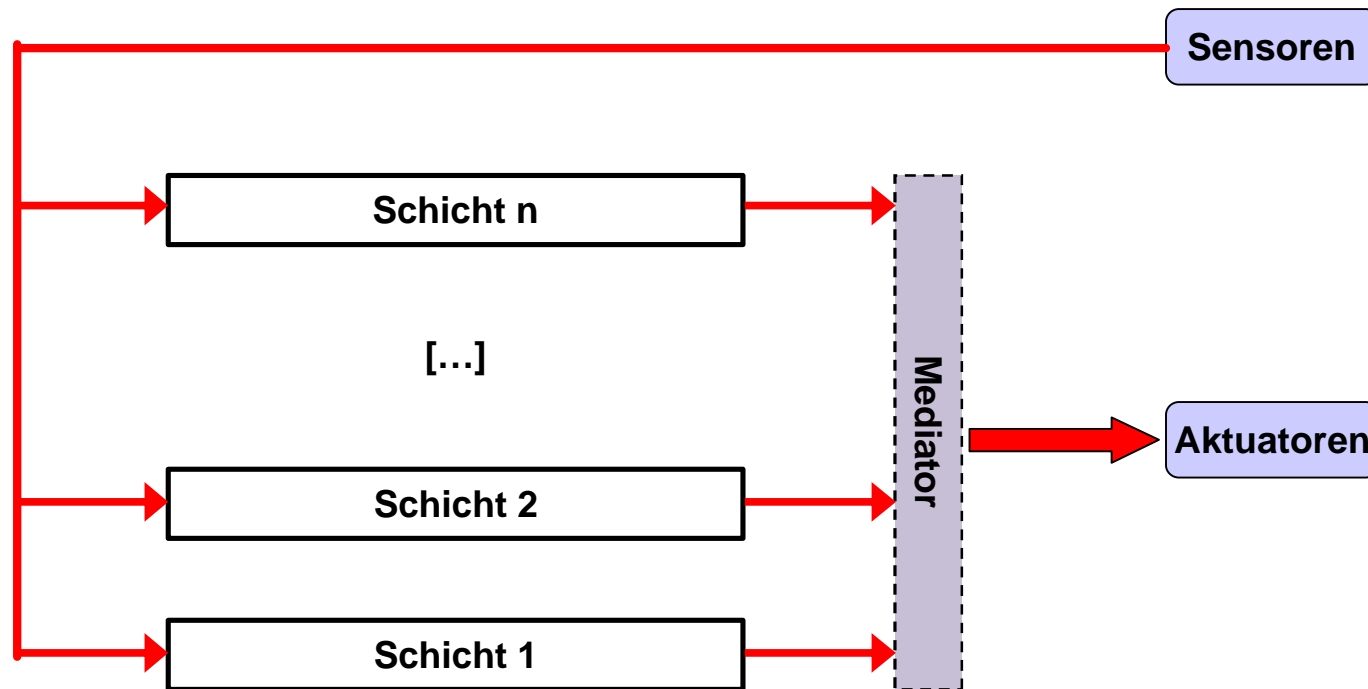
- Es gibt lang-, mittel- und kurzfristige Ziele.
- Es gibt Einflüsse, die kurz-, mittel-, oder langfristige Folgen haben können.
 - ⇒ Sehr komplexe Problemstellung für Entscheidungskomponente

Lösung:

- Fähigkeitskomponente überprüft Bedingungen für Fortsetzung/Modifizierung selbst.
- Eine Unterteilung gemäß der Tragweite von Teilzielen und Plänen.
 - Trennung von kurz-, mittel und langfristigen Entscheidungen durch **Schichten**.
 - Jede Schicht trifft ihre Entscheidungen weitgehend selbstständig

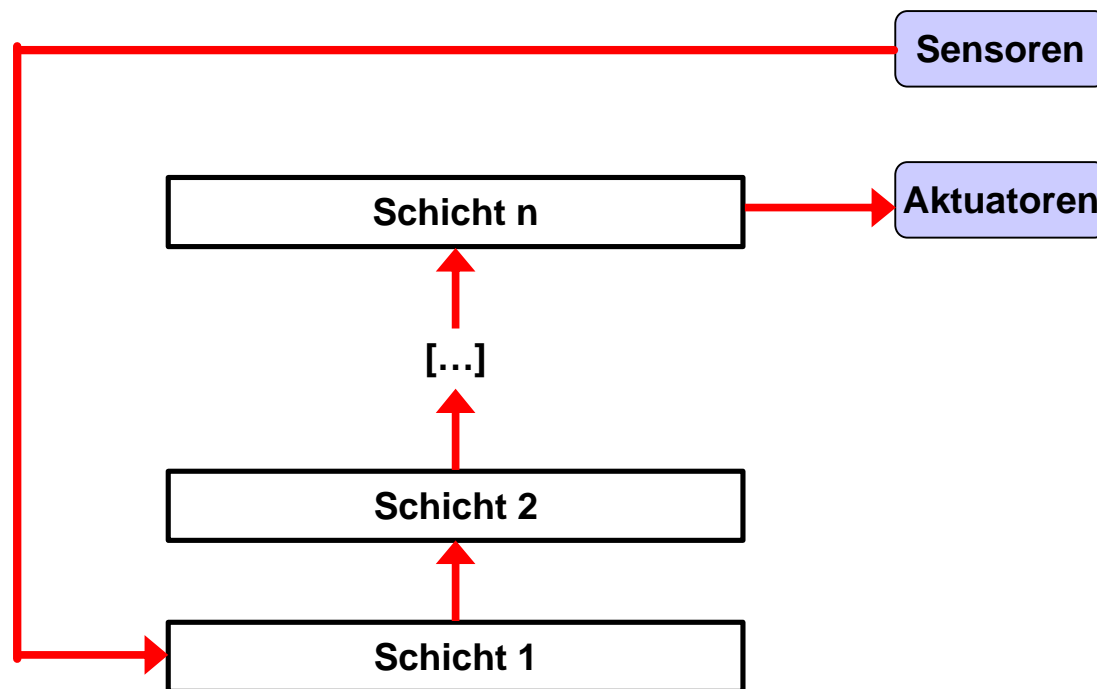
Schichtenmodel:

Horizontale Architektur:



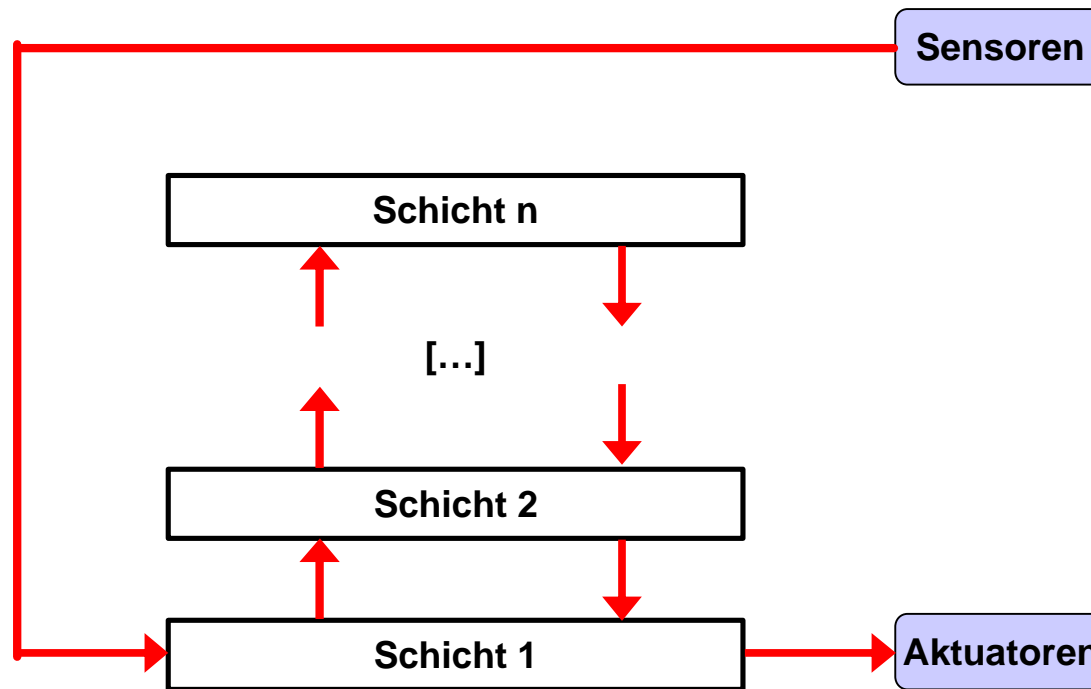
Schichtenmodell:

Vertikale Architektur – 1-Paß-Architektur:



Schichtenmodell:

Vertikale Architektur – 2-Paß-Architektur:



Schichtenmodel:

Gängige Schichten-Architekturen sehen 3 Schichten vor:

- oberste Schicht: **Kooperation** / Kooperative Planung.
- mittlere Schicht: Individuelle **Planung**.
- unterste Schicht: Reaktives **Verhalten**.

Hybride Architekturen:

- BDI strukturiert den Entscheidungsprozess.
 - Schichten strukturieren Verfeinerungsprozess von Absichten/Plänen.
 - Es bietet sich an beide zusammen einzusetzen.
 - Schichten für lang-, mittel-, und kurzfristige Planung und Entscheidungen.
 - BDI für Entscheidungsprozesse.
 - Auf oberster Schicht: Kooperative Planung. Erfordert **Stabilität im Verhalten**.
 - Unterste Schicht: Reaktives Verhalten: Erfordert **Flexibilität im Verhalten**.
- Bekanntestes Beispiel: InterRap.
Die erwähnten 3 Schichten: **Kooperation, Planung und Verhalten**.
Jede Schicht arbeitet nach dem BDI-Prinzip.
- Auch Verbindung von reaktivem Verhalten und komplexen Entscheidungsstrukturen ist möglich.
 - **Prinzipiell können alle vorgestellten Strukturen in einem Agenten vorkommen.**

5. Beispiel: „RoboCup“

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

Warum nicht auch in der KI?

RoboCup:

Internationale Initiative mit dem Ziel, die Forschung und Lehre in den Gebieten KI und Robotik zu fördern.

Fußball als Problemstellung:

Design autonomer Agenten, Kooperationsverhalten, Entscheidungsfindung in Echtzeitumgebungen, Verwerten unscharfer Informationen, Bildverarbeitung, Steuerung, usw.

- Ligen mit realen Robotern
- **Die Simulationsliga**

Die Simulationsliga:

- Zwei Mannschaften treten auf virtuellem Fußballfeld gegeneinander an.
- Simulation des Spiels erfolgt über „Soccerserver“.
 - Nimmt Kommandos von Spielern an und berechnet resultierende Bewegung von Spielern und Ball.
 - Spieler können: Alle 100ms eine Aktion ausführen (ein Kommando abschicken), alle 150ms eine Sichtinformation empfangen.
 - Zeit ist in 10ms-Zeitpunkte unterteilt / Darstellung des Raums erscheint kontinuierlich.
- Kommandos der Spieler: **turn** (drehen), **dash** (beschleunigen), **kick** (schießen), **turn-neck** (Kopf wenden), **say** (Rufen: Hören alle Spieler in der Nähe).
- Komplexere Handlungen müssen als Folgen solcher Aktionen geplant und ausgeführt werden.
- Informationen nicht ganz zuverlässig. Für weiter entfernte Objekte: zufällige Verfälschungen.
- Ballbewegungen sind schwach nicht-deterministisch: zufällige Verfälschungen.
- Spiel erfolgt unter Echtzeitbedingungen.
- Umwelt ist sehr dynamisch ⇒ Nur begrenzte Zeit für effiziente Entscheidungen.

5. Beispiel: „RoboCup“

Die Simulationsliga:

Der „Soccermonitor“ ermöglicht es, das Spiel am Monitor zu verfolgen:



Quelle: www.robocup.de/AT-Humboldt/

Das AT Humboldt:

- **Weltmeister 1997 in Nagoya.**
- **Vize-Weltmeister 1998 in Paris.**
- **Vize-Weltmeister(3D) 2004 in Lissabon.**
- Implementierung in C++
- BDI- und 2-Paß-Schichten-Architektur.
- Feingliedrige Fähigkeiten: Initialisierungsaktionen werden weggelassen, wenn sie nicht erforderlich sind, z.B. beim Kicken des Balls, wenn der Ball günstig kommt.
- Nur bei größeren Abweichungen Korrekturen vornehmen, geringfügige ignorieren (Effizienz).
- Pläne können **zeitweise** fanatisch verfolgt werden (Stabilitätsforderung).
- Kooperation ohne Kommunikation:
Durch beobachten der Mitspieler und vorgegebene Annahmen über deren Verhalten.

Für das Projektmanagement war eine klare Strukturierung nach Vorgaben der BDI-Architektur äußerst wichtig.

Schlussbetrachtungen:

- 2002/2003 wurde der 2D-Agent des AT Humboldt komplett neu entwickelt
- Seit 2004: 3D-Simulationsliga.

Literatur:

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

Links:

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