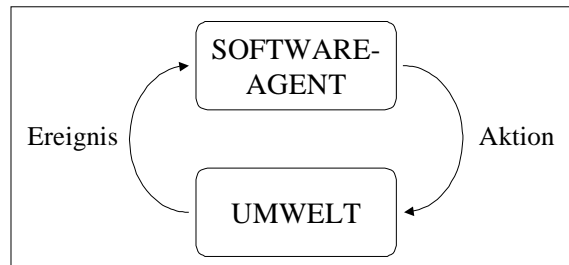


1.1 Softwareagenten

Ein **Softwareagent** ist ein Programmobjekt (siehe objektorientierte Programmierung) innerhalb eines Softwaresystems (Agentenplattform), welches **autonom**, d. h. ohne die direkte Intervention von Menschen oder anderen Softwareagenten, mit seiner Umwelt interagiert, um die von einem Anwender spezifizierten Ziele zu verfolgen. Der Softwareagent besitzt die Kontrolle über seinen inneren Zustand und seine Aktionen.



Softwareagent in Wechselwirkung mit seiner Umwelt

Softwareagenten besitzen die Grundeigenschaften:

- **Reaktivität**, d. h. die Fähigkeit, die Umwelt wahrzunehmen und zeitnah auf die Veränderung dieser zu reagieren, ohne dabei die Zielvorgaben zu verlassen und/oder
- **Proaktivität**, d. h. das Ausführen von zielgerichteten Aktionen aus eigener Initiative.

Diese Eigenschaften bestimmen ihre Verhaltensweisen und werden durch unterschiedliche interne Softwarestrukturen (Agentenarchitekturen) generiert. Es werden dabei meist drei grundlegende **Agentenarchitekturtypen** unterschieden.

Deliberative Softwareagenten

- **Deliberative Softwareagenten** besitzen ein explizites **Modell der Umwelt**, welches in symbolischer Repräsentation im Agenten gespeichert ist. Sie treffen mit Hilfe von **logischen Schlussfolgerungstechniken** die Entscheidung darüber, welche Handlungen A als nächste durchzuführen sind. Deliberative Softwarearchitekturen ermöglichen proaktives Verhalten der Softwareagenten.
- Ein einfaches **Modell eines deliberativen Softwareagenten** kann wie folgt aussehen:
 - ▷ Die Funktion *see* repräsentiert die Fähigkeit eines Softwareagenten, seine Umwelt wahrzunehmen. Die Umwelt eines Softwareagenten ist charakterisiert durch ihren Zustand S . Wenn mit P die Menge der **Wahrnehmungen** bezeichnet wird, ergibt sich für die Funktion *see*:

$$see : S \rightarrow P$$
 - Die wahrgenommenen Umwelteigenschaften werden als logische Ausdrücke in einer **internen Datenbank des Agenten** Δ gespeichert. Beispiele für solche Wahrnehmungen sind:
 $Temperature(reactor4726,321)$
 $Pressure(tank776,28)$
 - ▷ Außerdem führen wir die Funktion *next* ein, die den Inhalt einer Datenbank Δ mittels einer Wahrnehmung auf eine neue agenteninterne Datenbank abbildet. Die Gesamtmenge der internen Zustände bzw. Datenbanken Δ der Agenten wird mit D bezeichnet.

$$next : D \times P \rightarrow D$$
 - ▷ Das Verhalten des Agenten wird bestimmt durch eine Menge von **Aktionsregeln** ρ , welche in klassischer Prädikatenlogik als **Inferenzregeln** definiert sind.
 - ▷ Der Entscheidungsprozess eines Softwareagenten wird durch die Aktionsregeln ρ und **Aktionsbeschreibungen** ϕ modelliert. Wir schreiben $\Delta \vdash_{\rho} \phi$, wenn die Aktionsbeschreibung ϕ mittels der Inferenzregel ρ auf Basis der Datenbank Δ verifiziert werden kann.

- Beispiel für eine Aktionsbeschreibung:

Open(valve221)

- Ein Aktionsregel könnte lauten:

Pressure(tank776,28) ∧ Temperature(reactor4726,321) → Open(valve221)

- ▷ Der Entscheidungsprozess eines Softwareagenten wird durch die Funktion *action* repräsentiert:

action : D → A

Der Pseudocode dieser Funktion ist der folgende:

```

1 function action ( $\Delta : D$ ):A
2 begin
3     for each  $a \in A$  do
4         if  $\Delta \vdash_{\rho} Do(a)$  then
5             return  $a$ 
6         end-if
7     end-for
8     for each  $a \in A$  do
9         if  $\Delta \not\vdash_{\rho} \neg Do(a)$  then
10            return  $a$ 
11        end-if
12    end-for
13    return null
14 end function action

```

- Im ersten Teil der Funktion *action* (Zeilen 3-7) versucht der Softwareagent, nacheinander für jede mögliche Handlungsalternative die Aktionsbeschreibung *Do(a)* (mit $Do(a) \in \Phi$) der Datenbank mit Hilfe der Inferenzregeln ρ zu verifizieren und führt im Falle eines Erfolges die entsprechende Handlungsalternative *a* durch. Die Aktionsregeln ρ in der Datenbank Δ sind dabei so definiert, dass für den Fall einer Verifizierung von *Do(a)*, *a* die optimale Handlungsalternative ist.
- Kann der Softwareagent den Ausdruck *Do(a)* für keine der Handlungsalternativen *a* beweisen, so versucht er, eine Handlungsalternative zu finden, die mit den Inferenzregeln und der Datenbank konsistent bzw. nicht explizit verboten ist, d. h. für die $\neg Do(a)$ nicht abgeleitet werden kann (Zeilen 8-12) und führt diese durch.
- Falls keine verifizierbare Handlungsalternative gefunden werden kann, wird die Aktion *null* ausgegeben und damit keine Aktion durchgeführt (Zeile 13).

Reaktive Softwareagenten

- **Reaktive Softwareagenten** besitzen kein internes, symbolisches Modell ihrer Umwelt und verzichten aus Komplexitätsgründen auf Schlussfolgerungstechniken. Ihr Verhalten wird durch einfache **Reiz-Antwort-Schemata** beschrieben, welche vordefinierte Verhaltensweisen für bestimmte Umweltzustände vorsehen. Komplexes Verhalten entsteht bei reaktiven Agentenarchitekturen nur als Ergebnis der Interaktion vieler Agenten in einem **Multiagentensystem**. Ein solches Systemverhalten wird auch als **Emergenz** bezeichnet.
- Eine typische **reaktive Agentenarchitektur** ist die **Subsumptions-Architektur** von Rodney Brooks:
 - ▷ Ein Softwareagent besteht aus einer Reihe von **Verhaltensmodulen**, in denen die **Verhaltensweisen des Agenten** gespeichert sind. Die Verhaltensweisen werden durch Regeln repräsentiert, welche aus einem wahrgenommenen Input direkt eine Aktion ableiten: *situation* \rightarrow *action*
 - ▷ Eine **Verhaltensregel** ist definiert als ein Paar (c, a) , bei dem *c* eine so genannte *condition* und $a \in A$ eine Handlungsalternative darstellt, wobei $c \in P$ aus der Menge der möglichen Wahrnehmungen stammt. Eine Verhaltensregel (c, a) wird aktiviert, wenn die Bedingung $see(s) \in c$ für diesen Umweltzustand

$s \in S$ erfüllt ist. Mit $Beh = \{(c, a) \mid c \subseteq P \text{ und } a \in A\}$ bezeichnen wir die Menge aller möglichen Verhaltensregeln.

- Beispiele für Verhaltensregeln sind:
 - (1.1) *if Alert(reactor4726,On) then Open(valve334)*
 - (1.2) *if Temperature(reactor4726,321) and Pressure(tank776,28) then Open(valve221)*
 - (1.3) *if Flowrate(tube211,588) and Level(tank776,28) then Close(valve221)*
- ▷ Die Verhaltensmodule sind hierarchisch in Ebenen angeordnet, wobei die am unteren Ende der **Hierarchie** angesiedelten Module für grundlegende Aufgaben vorgesehen sind. Diese besitzen die Möglichkeit, die Ausführung von Aktionen auf höheren Ebenen bei Interessenkollision zu verhindern. Mit fallendem Rang in der Hierarchiestufe wächst die Priorität, mit der Aktionen weiter oben liegender Module unterbunden werden können.
- ▷ Zur Realisierung der Hierarchie der Verhaltensmodule ist auf die Menge der Verhaltensregeln $R \subseteq Beh$ eine **Inhibitionsrelation** definiert: $\prec \subseteq R \times R$. Wir schreiben $b_1 \prec b_2$, wenn $(b_1, b_2) \in \prec$, was bedeutet, dass die Verhaltensweise b_1 sich auf einer niedrigeren Hierarchieebene als b_2 befindet bzw. eine höhere Priorität als b_2 hat.
- In unserem Fall lautet die Inhibitionsrelation: $(1.1) \prec (1.2) \prec (1.3)$
- ▷ Die **Entscheidungsfunktion** *action* des Agenten wird nun durch die Menge der Verhaltensregeln dargestellt, die bedingt durch die Inhibitionsrelation unterschiedliche Prioritäten haben.
- ▷ Die Funktion *action* ist wie folgt definiert:


```

1 function action(p:P):A
2 var fired: ϕ(R)
3 var selected:A
4 begin
5     fired := {(c,a)|(c,a) ∈ R and p ∈ c}
6     for each (c,a) ∈ fired do
7         if ¬(∃(c',a') ∈ fired such that (c',a') ≺ (c,a)) then
8             return a
9         end-if
10    end-for
11    return null
12 end function action
```
- ▷ Zur Auswahl der auszuführenden Handlungsalternative werden zunächst alle erfüllten (*fired*) Verhaltensregeln bestimmt (Zeile 5). Anschließend wird aus dieser Menge diejenige Verhaltensregel (c, a) ausgewählt, welche die höchste Priorität hat und deren zugehörige Handlungsalternative a ausgeführt (Zeile 8). Falls keine Verhaltensregel erfüllt ist, wird die Aktion *null* ausgegeben bzw. keine Handlungsalternative durchgeführt.
- Sind beispielsweise folgende Verhaltensregeln erfüllt:
 - (1.1) *if Alert(reactor4726,On) then Open(valve334)*
 - (1.3) *if Temperature(reactor4726,321) and Pressure(tank776,28) then Open(valve221)*
 so wird *Open(valve334)* ausgewählt da: $(1.1) \prec (1.3)$.

Hybride Softwareagenten

- **Hybride Softwareagenten** vereinigen deliberative und reaktive Ansätze auf Basis **mehrschichtiger Architekturen**. Während mit Hilfe der unteren Schichten grundlegende Verhaltensmuster eines Agenten in Form von Reiz-Antwort-Schemata implementiert sind und somit Elemente der reaktiven Architektur eingeführt werden, wird der deliberative Prozess der Planung und Entscheidungsfindung in höher gelegenen Schichten bearbeitet.

- Es können folgende Typen von **Schichtenarchitekturen** unterschieden werden:

▷ **Horizontale Schichtenarchitektur:**

- Alle Schichten arbeiten parallel, haben Zugriff auf die Wahrnehmung des Agenten und können Aktionen vorschlagen.
- Ein zentrales Kontrollsystem entscheidet, welche Schicht zu einem bestimmten Zeitpunkt die Kontrolle über den Agenten besitzt.

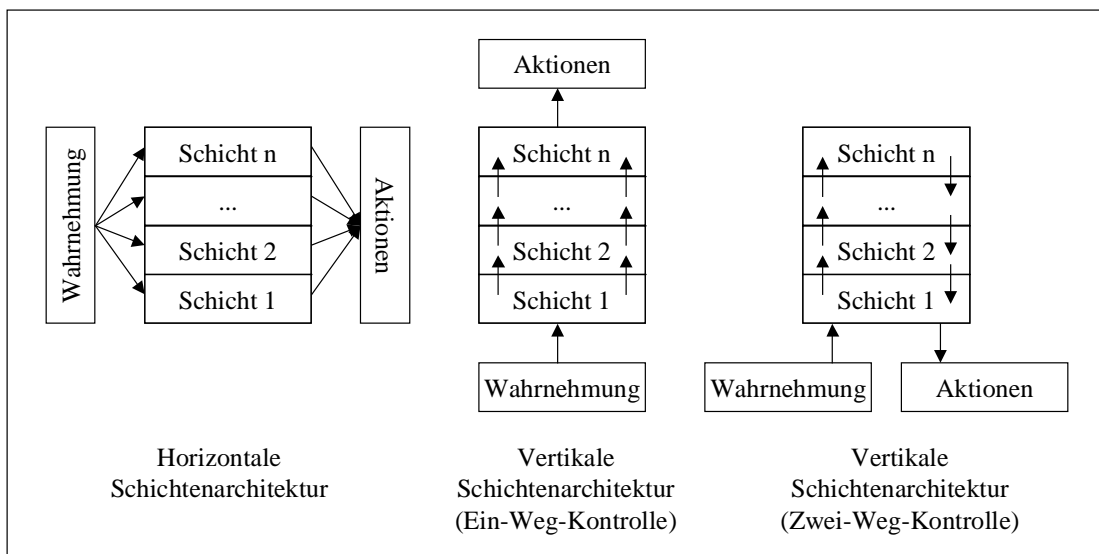
▷ **Vertikale Schichtenarchitektur:**

- Nur die unterste Schicht hat Zugriff auf die Wahrnehmung des Agenten. Kann eine Schicht den Input nicht verarbeiten, leitet sie ihn an die nächst höhere Schicht weiter. Je nach Gestaltung des Ausführungsprozesses werden zwei Subarchitekturtypen unterschieden:

Bei der *Ein-Weg-Kontrolle* führt diejenige Schicht eine Aktion aus, welche angemessen auf einen Input reagieren kann, während

bei der *Zwei-Weg-Kontrolle* nur die unterste Schicht eine Aktion ausführen kann und der Informationsfluss dementsprechend wieder in Richtung der untersten Schicht verläuft.

- ▷ Der Unterschied zwischen den Agentenarchitekturen liegt in der **Komplexität** und **Flexibilität** der Entscheidungsfindung. Während bei der horizontalen Schichtenarchitektur, bedingt durch das zentrale Kontrollsystem deutlich mehr Handlungsalternativen verglichen werden müssen und daher längere Verarbeitungszeiten die Folge sind, weisen vertikale Schichtenarchitekturen eine geringere Fehlertoleranz auf, da eine Entscheidung alle Schichten durchläuft und ein Fehler innerhalb einer Schicht ernsthafte Konsequenzen für die gesamte Entscheidungsfindung nach sich ziehen kann.

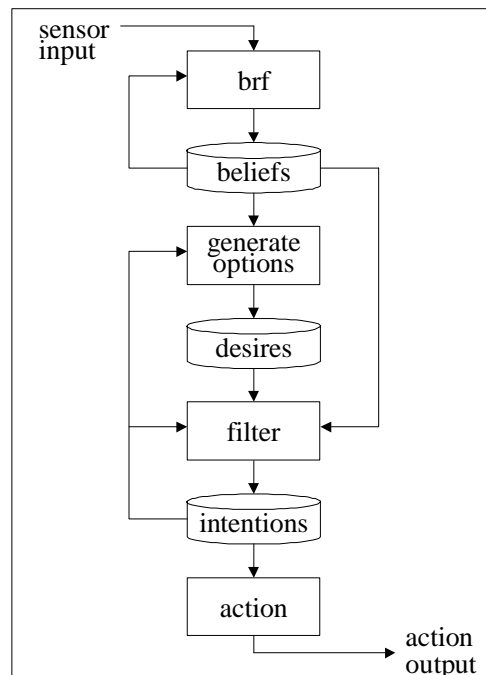


Schichtenarchitekturen von Softwareagenten

BDI-Softwareagenten

In einer **enger gefassten Definition** werden Softwareagenten **mentale Eigenschaften** zugeschrieben, die üblicherweise im Zusammenhang mit menschlichem Verhalten stehen, wie etwa **Überzeugungen, Wünsche** und **Absichten**. Solche Eigenschaften kommen in **Belief-Desire-Intention- (BDI-) Architekturen** zum Ausdruck. BDI-Architekturen können als rein deliberative oder hybride Softwarearchitekturen dargestellt werden:

- Wissenschaftliche Grundlage der BDI-Architektur ist die Theorie vom so genannten „**praktischen Denken**“, die sich an der Entscheidungsfindung des Menschen im täglichen Leben orientiert und daher durch kurze Entscheidungsprozesse über Aktionen zur Förderung langfristiger Ziele charakterisiert ist.



BDI-Architektur

- Die beiden Basiskomponenten von **BDI-Softwareagenten** sind:
 - ▷ **Datenstrukturen**, welche den mentalen Zustand eines Softwareagenten beschreiben, wobei die Inhalte häufig in Form von logischen Ausdrücken repräsentiert werden. Sie gliedern sich in:
 - **beliefs**, *Bel*, welche die vom Agenten wahrgenommenen Informationen über seine Umwelt repräsentieren,
 - **desires**, *Des*, die die zu einem bestimmten Zeitpunkt möglichen Handlungsalternativen darstellen und
 - **intentions**, *Int*, welche die Absichten, d. h. die aus den angestrebten Handlungsalternativen ausgewählten *desires*, charakterisieren.
- Der mentale Zustand eines BDI-Softwareagenten wird also dargestellt durch das Tripel (B, D, I) , mit $B \subseteq Bel$, $D \subseteq Des$ und $I \subseteq Int$.
- ▷ **Funktionen**, die definieren, welche Absichten auszuwählen sind (*deliberation*) und wie diese verfolgt werden sollen (*means-ends reasoning*).
 - Die **belief revision function**, *brf*, bestimmt eine neue Menge von beliefs auf Basis der aktuellen Menge der beliefs und des gegenwärtig wahrgenommenen Inputs:

$$brf: \wp(Bel) \times P \rightarrow \wp(Bel)$$

- Die **option generation function**, *options*, definiert auf Basis der gegenwärtigen Menge der beliefs und intentions die Menge der möglichen Handlungsalternativen (desires) hinsichtlich einer Verfolgung der intentions (means-ends reasoning):
 $options : \wp(Bel) \times \wp(Int) \rightarrow \wp(Des)$
- Die **filter function**, *filter*, aktualisiert die intentions des Softwareagenten auf Basis der vorhergehenden intentions und der gegenwärtigen beliefs und desires (deliberation):
 $filter : \wp(Bel) \times \wp(Des) \times \wp(Int) \rightarrow \wp(Int)$
 unter der Nebenbedingung:
 $\forall B \in \wp(Bel), \forall D \in \wp(Des), \forall I \in \wp(Int), filter(B, D, I) \subseteq I \cup D$
 Sie löscht bestehende intentions, falls sie nicht mehr erreichbar sind oder der Aufwand der Verfolgung höher als der erwartete Nutzen ist, behält sie bei, falls sie noch nicht erreicht sind, aber immer noch einen positiven Nutzen erwarten lassen oder adaptiert neue, um bestehende intentions zu erreichen oder neue Möglichkeiten zu erforschen.
- Die **execute function**, *execute*, wählt alle erreichbaren intentions aus und bestimmt die Handlungsalternativen, die zur Erfüllung dieser intentions durchgeführt werden müssen:
 $execute : \wp(Int) \rightarrow A$
- Die **action function**, *action*, ist die Entscheidungsfunktion eines BDI-Agenten die alle vorstehenden Funktionen zusammenfasst und letztendlich eine Handlungsalternative auswählt:
 $action : P \rightarrow A$
 Sie ist durch den folgenden Pseudocode definiert:


```

1 function action(p : P) : A
2 begin
3     B := brf(B, p)
4     D := options(D, I)
5     I := filter(B, D, I)
6     return execute(I)
7 end function action
      
```

Eigenschaften von Softwareagenten

Je nach Anwendungsdomäne können **weitere Eigenschaften** für Softwareagenten von Bedeutung sein:

- **Kooperationsfähigkeit** ist die Fähigkeit eines Softwareagenten, mit anderen Softwareagenten zusammenzuarbeiten und dabei deren Ziele zu berücksichtigen sowie Kompromissbereitschaft aufzuweisen. Basis jeglicher Kooperation ist **Kommunikation**, welche drei wichtige Aspekte beinhaltet:
 - ▷ Das **Interaktionsprotokoll** nimmt Bezug auf die Strategie eines Agenten, deren Verfolgung seine Interaktion mit anderen Agenten leitet. Dieses Protokoll kann ein Verhandlungsschema oder ein spieltheoretischer Mechanismus sein.
 - ▷ Die **Kommunikationssprache** ist das Medium, über welches die Inhalte zwischen Agenten kommuniziert werden. Gebräuchliche Agentenkommunikationssprachen sind KQML (Knowledge Query and Modification Language) und FIPA-ACL (FIPA-Agent Communication Language). Allen Agentenkommunikationssprachen ist gemein, dass ein Nachrichtenkopf mit Angaben zu Sender/Empfänger, Sprachdefinition (ACL, Knowledge Interchange Format), Wissensdomäne und Kommunikationstyp (Anfrage, Behauptung, Aufforderung usw.) dem eigentlich zu übertragenden Inhalt vorangestellt ist.
 - ▷ Das **Transportprotokoll** ist der gewählte (vom Kontext unabhängige) Transportmechanismus der Kommunikation, wie z. B. TCP-IP, SMTP, FTP.

- **Kollaborative-Softwareagenten** sind ein Agententyp, dessen wesentliche Eigenschaft Kooperationsfähigkeit darstellt. Sie kommen zur Lösung von Problemen zum Einsatz, die z. B. zu komplex für die Bewältigung durch ein einzelnes zentrales System sind oder die eine natürliche Verteiltheit aufweisen, wie die Kontrolle von verteilten Produktionsstätten innerhalb eines Unternehmens. Die Zuweisung der Aufgaben wird meist mittels Verhandlungen zwischen den Agenten vorgenommen.
- **Lernfähigkeit** ist die Fähigkeit eines Softwareagenten, das eigene Verhalten durch Interaktion mit der Umwelt, dem Menschen oder anderen Softwareagenten längerfristig zu verändern. Lernen ist dabei nicht nur auf das Sammeln und Auswerten von Fakten innerhalb logikbasierter Systeme beschränkt, sondern es können auch andere Techniken des Maschinenlernens wie Neuronale-Netze, Bayes'sche Belief-Netze sowie Bekräftigungslernen zum Einsatz kommen.
 - **Bekräftigungslernende-Softwareagenten:** Agenten, die anhand von positiven Umweltreaktionen auf ihre Handlungen lernen und dabei eine Reduktion der dazu nötigen Bewertung auf häufig auftretende Aktionsmuster vornehmen. Sie werden z. B. in Multiagentensystemen zur Bewältigung von Koordinationsaufgaben im Supply Chain Management eingesetzt.
 - **Interface-Softwareagenten:** Persönliche Assistenten eines Benutzers, den sie im Umgang mit bestimmten Anwendungssystemen unterstützen. Der Interface-Softwareagent überwacht und imitiert die Aktionen des Benutzers und erlernt dabei, wie diese besser durchgeführt werden können.
- Softwareagenten, denen reaktives sowie proaktives Verhalten möglich ist (etwa auf Basis einer hybriden Agentenarchitektur bestehend aus reaktiven und deliberativen Komponenten) und die darüber hinaus kooperations- und lernfähig sind, werden als **intelligente Softwareagenten** bezeichnet.
- **Mobilität** ist die Fähigkeit eines Softwareagenten, sich (über eine Infrastruktur) selbstständig in einem elektronischen Netzwerk von einem Ort zu einem anderen zu bewegen. Vorreiter auf dem Gebiet der mobilen Agenten ist das Projekt *www.agentcities.net*, das sich mit der Realisierung einer weltweiten Plattform für mobile Agenten, basierend auf dem FIPA-Standard (Foundation for Intelligent Physical Agents), beschäftigt.

Neben einer Klassifizierung hinsichtlich der Eigenschaften von Softwareagenten werden sie häufig durch ihre Aufgabe, die sie zu erfüllen haben, bzw. ihre **Rolle** definiert. So lassen sich z. B. Planungs-, Kontroll- sowie Informationssoftwareagenten unterscheiden.

- **Informationssoftwareagenten** unterstützen einen Benutzer bei der Suche nach Informationen in Netzwerken wie dem Internet. Sie spüren potenzielle Informationsquellen auf, filtern diese entsprechend dem Interessenprofil ihrer Benutzer und präsentieren ihm das Ergebnis. Gemäß ihrer Anwendungsdomäne kann es notwendig sein, dass sie Lernfähigkeit besitzen, z. B. um im Zeitablauf die Präferenzen ihrer Benutzer besser zu erfassen. Solche Softwareagenten können gleichzeitig auf Grund ihrer Eigenschaften als **Interface-Softwareagenten** bezeichnet werden.