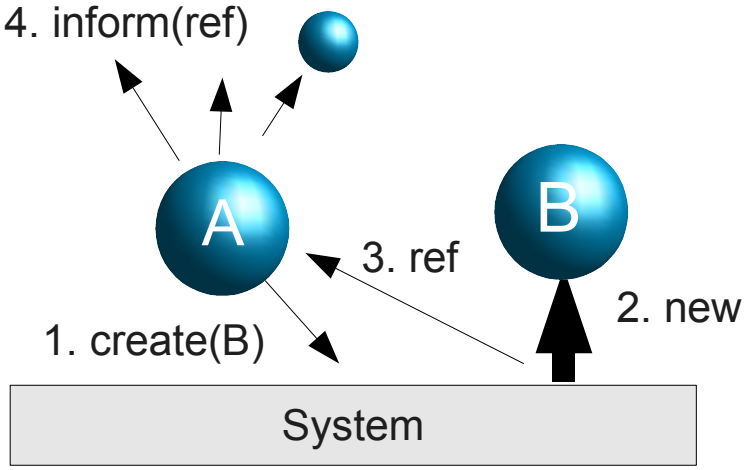
Agentenmanagement

Typisierung und Vermittlung

16.05.2007

Vermittlung

- Lokalisierung
  - Problem, einen speziellen Agenten wiederzufinden
  - Notwendig: eindeutige Referenz
- Woher bekommt man diese Referenz?
  - MAFAgentSystem:
    name = create_agent(...)
  - AMETAS:
    puid = m_Driver.requestPUStartup(...)
- Verbreitung der Referenz
  über die Anwendung



Eigenschaften

- Und wenn man nicht informiert wurde?
  - z.B. in offenen Agentenanwendungen

➡ Finden eines Agenten anhand einer bestimmten *Eigenschaft*

Ein Agent, der die geforderten Eigenschaften erfüllt, ist ein möglicher Auswahlkandidat.

Die Eigenschaften müssen **beständig** sein – zumindest für einen zugesicherten Zeitraum

Wofür braucht man Typen?

- Implementierung
  - Neue Version eines Agenten erstellen, die alles das kann, was der bisherige Agent konnte, aber etwas mehr
  - Nutzung eines bestehenden Agenten für eine neue Anwendung
- Verarbeitung
  - Selektieren von bestimmten Agenten für eine Aufgabe / für den Zugriff auf bestimmte Dienste
- Nutzer
  - Suchen eines Agenten für einen bestimmten Zweck (Kinokarten besorgen und Blumen kaufen)

Eigenschaften

- A hat die Eigenschaft E
  - A erfüllt die Aussage „X hat die Eigenschaft E“ für freie X
  - A erfüllt das Prädikat „hat die Eigenschaft E“:
    - Notation: E(A)
    - Einstelliges Prädikat
- Beispiel
  - ist_Schreibgerät(Bleistift), ist_Schreibgerät(Kugelschreiber)

Extensionalitätsprinzip

- Extensionalität
  - Jedes Prädikat definiert eindeutig eine Menge von Objekten, die es erfüllen.
  - $M := \{ x \mid P_M(x) \}$
  - Jede Menge definiert mindestens ein Prädikat, das von jedem seiner Elemente erfüllt wird.
    - trivial: $P_M(x) = x \in M$
    - $M=\{1,2,3,4,5,6\}$; $P_M(x) = (x \in \mathbf{N}) \wedge (0 < x < 7)$ oder $P_M(x) =$ „x ist eine Zahl auf dem Würfel“

Mengen und Prädikate

- Suchen einer Instanz = Suchen eines Vertreters aus der Menge
- „Ich suche eine Augenzahl“: 3 möglich, 7 oder π nicht
- „x ist eine gerade Augenzahl“ =
  „x ist eine Augenzahl“ UND „x ist eine gerade Zahl“

  GA(x) := A(x) ^ G(x)

  Verschärfung des Prädikats mit UND / ^
  Verkleinerung der Treffermenge

Teilmengen und Prädikate

- Verschärfung des Prädikats → Bildung von Teilmengen

$$\{ x \mid (P \wedge Q)(x) \} = \{ x \mid P(x) \wedge Q(x) \} = \{ x \mid P(x) \} \cap \{ x \mid Q(x) \}$$

Alle Elemente einer Teilmenge erfüllen immer das Prädikat, das ihre Obermenge definiert.

Ein gegebenes Prädikat wird als „verträglich“ zu einem gesuchten Prädikat erachtet, wenn die definierte Menge an Instanzen eine Teilmenge der erfüllenden Instanzen der Obermenge darstellt.
Jede Instanz der Teilmenge ist dann ein geeigneter Vertreter der gesuchten Obermenge.

Typen und Klassen

- Datentyp
  - In Java und generell bekannt als primitive Typen (int, boolean, ...) oder komplexe Typen (String, Object)
  - Festlegung, welche Operationen mit den Vertretern des Typs erlaubt sind
    - Numerisch: Addition, Multiplikation usw.
    - Strings: Anhängen, Teilstring, Zeichen an Position usw.
- Klassen
  - Typkonstrukt der objektorientierten Sprachen
  - Instanziierung liefert einen Vertreter mit definiertem Verhalten (Methoden, Felder, Verhalten auf Interaktionen...)

Typen

- Subtypen
  - Bei primitiven Typen: Einschränkung des Bereichs „intGerade“ ist ein Subtyp von „int“
    - gleiche Operationen, Vertreter nur die geraden int-Werte
    - Verschärfung des Prädikats
- Subklassen
  - Hinzufügen weiterer Operationen
  - Prinzipiell jedoch das Gleiche: Bildung einer Teilmenge von gültigen Instanzen
    - class OrderedSet extends Set
      ... Element getNextElement();

Typen

- Semantisches Problem
  - Jeder Vertreter von „OrderedSet“ ist auch ein „Set“
  - Aber „Set“ ist ungeordnet – Widerspruch?
    - Nur dann, wenn „ungeordnet sein“ ein Prädikat ist, das „Set“ definiert (wie realisieren?)
  - Die Eigenschaft, etwas „nicht erfüllen zu müssen“, ist nicht gleich der Eigenschaft, etwas „nicht erfüllen zu dürfen“
    - Mathematisches Beispiel: $2 < 3$, aber $2 \leq 3$ ist auch korrekt.

Typen und Vermittlung

- Instanzvermittlung
  - Gesucht ist eine Instanz eines bestimmten Typs
  - Prüfung: Erfüllt die Instanz alle Bedingungen des Typs? Wenn ja, als Ergebnis liefern
- Typvermittlung
  - Gesucht ist ein Typ, der „verträglich“ mit einem bestimmten Typ ist (*konform*). Alle Instanzen dieses Typs sind dann passend.
  - Prüfung: Ist das Prädikat eines Typs T eine Verschärfung des Prädikats des gesuchten Typs? Wenn ja, T als Ergebnis liefern
  - Schwierig für den Fall, dass die Mengen über nicht zueinander konforme Prädikate definiert wurden.

Typen und Vermittlung

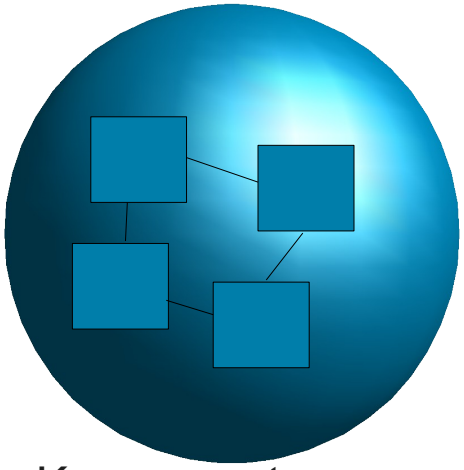
- Fazit
  - Typen sind geeignet, eine flexible Vermittlung zu realisieren
  - Jede Sache, die über beständige Eigenschaften verfügt, kann typisiert werden
    - Seifenblasen eher nicht, außer für kurze Momente
- Es müssen beständige Eigenschaften von Agenten identifiziert werden
  - Zwei Fälle
    - Laufende Instanzen (Instanzvermittlung)
    - Vorlagen zur Erzeugung von Agenteninstanzen (Typvermittlung)

Typen und Agenten

- Typüberprüfung bislang meist zur Kompilierungszeit
  - Vorab-Prüfung, ob Datentypen kompatibel sind
  - Frühzeitiges Entdecken von Programmierfehlern
- Hier aber nicht möglich
  - Agententypen sind in offenen Systemen a priori nicht bekannt
  - Agenten sind keine von einer Programmiersprache repräsentierten und vom Compiler überprüfbaren Konstrukte
  - Agenten sind aktive Objekte: Typbeschreibung sollte die Aktivität einbeziehen
    - bisherige Typen: nur statische Eigenschaften

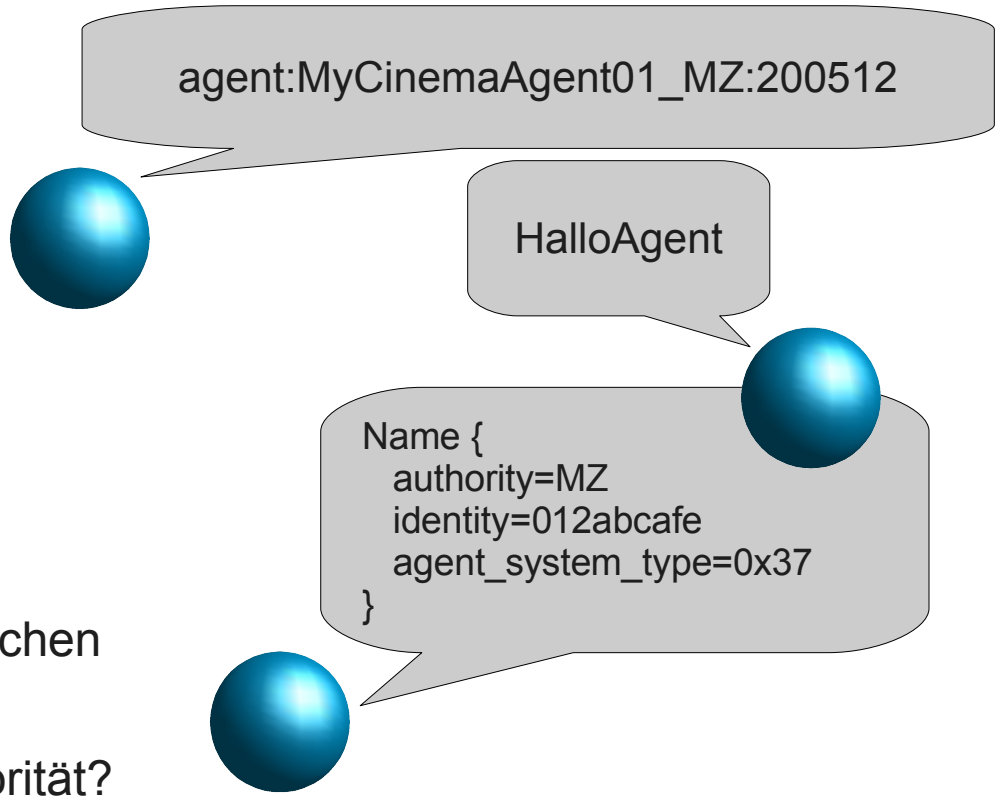
Typisierung

- Objekte
  - Klassen, Schnittstellen
- Aber...
  - wie definiert man einen Agententyp?
- Ansatz über Implementierung
  - Agententyp = Verbund aller Typen aller enthaltenen Komponenten
  - Problem: Zu einschränkend; man muss alles über den inneren Aufbau wissen
  - „Need to know“: Wieso sollte der innere Aufbau relevant sein?



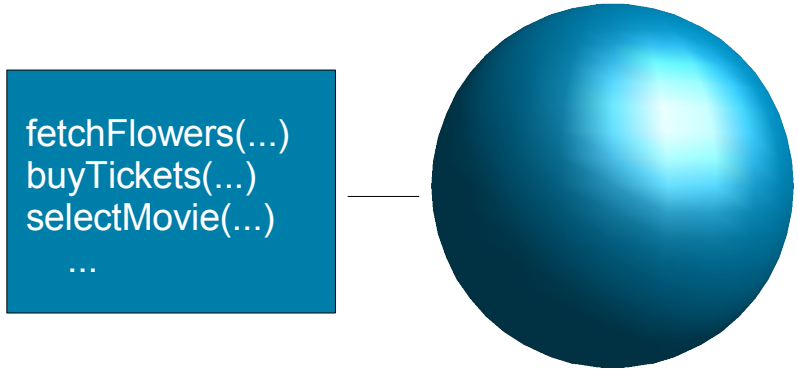
Typisierung

- Ansatz: Suchen nach Namen
- Vorteil
  - Bezeichnet den Agenten eindeutig und sicher
  - Meist in Stringform, lesbar
- Nachteil
  - Name muss bekannt sein
  - Willkürliche Zuordnung zwischen Eigenschaften und Name
  - Wer besitzt die Namensautorität?
  - Abbildung einer komplexen Beschreibung auf einen einfachen (lesbaren!) String bringt meist Informationsverlust



Typisierung

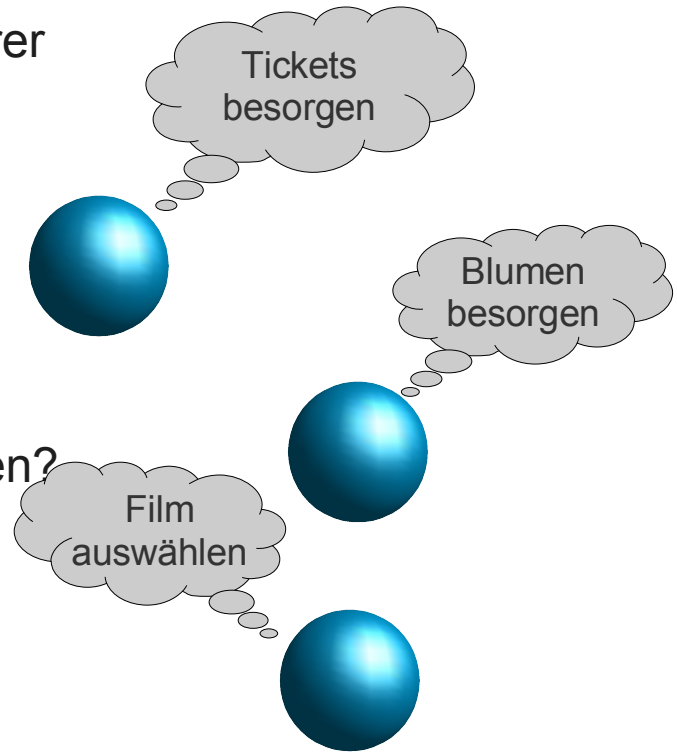
- Nächster Ansatz: Schnittstelle zum Agenten
  - ähnlich wie bei CORBA
- Nachteil
  - Methodennamen sagen nicht viel über Bedeutung aus („selectMovie“?)
  - Keine Beschreibung von Interaktionen: zu sehr Klient-Server-bezogen, keine komplexen Dialoge darstellbar
  - Woher bekomme ich die Schnittstellenbeschreibung?



Nur tauglich in geschlossenen Systemen mit vorherrschender Anfrage-Antwort-Kommunikation

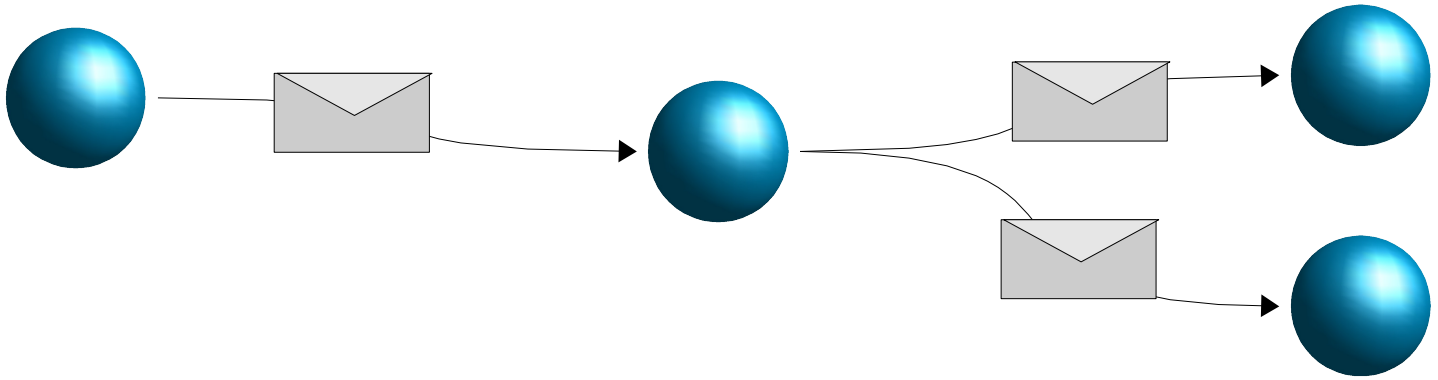
Typisierung

- Semantische Eigenschaften
  - Agenten anhand ihres Einsatzes oder anderer abstrakter Eigenschaften beschreiben
- Vorteil
  - sehr flexibel, nahe an Nutzersicht
- Nachteil
  - Wie formalisiert man abstrakte Eigenschaften?
  - Kommunikation egal?



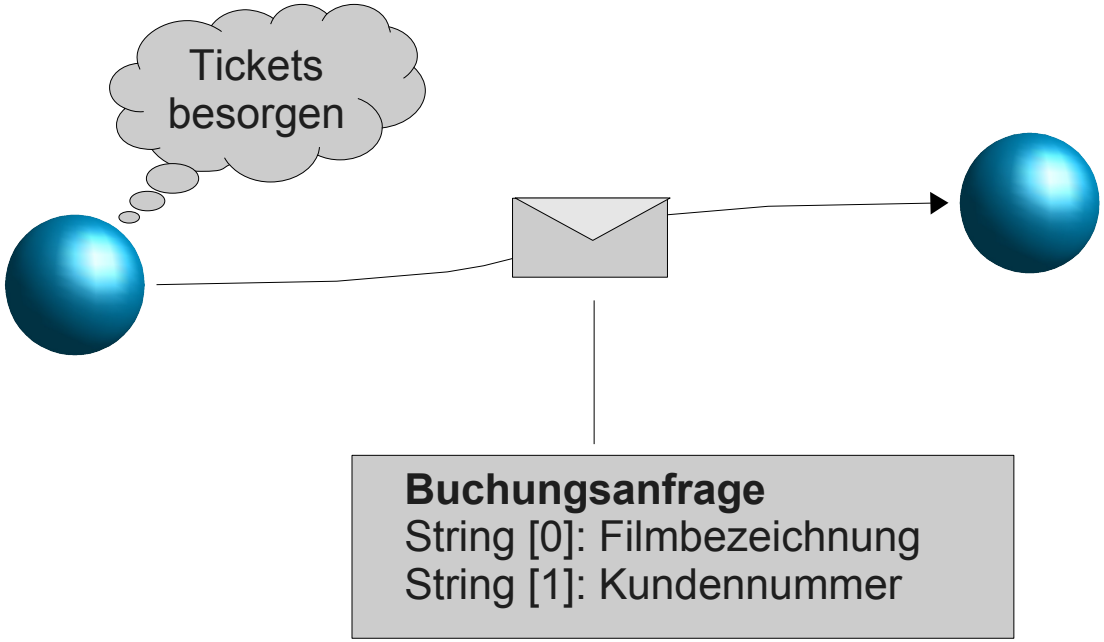
Typen und Agenten

- Möglicher Ansatz für Agententypen
  - Agent interagiert mittels Nachrichten mit seiner Umgebung
    - asynchron oder synchron
    - kann Nachrichtenaustausch sein, aber auch Methodenaufruf
  - Typ muss Nachrichten und Interaktionsmuster beachten



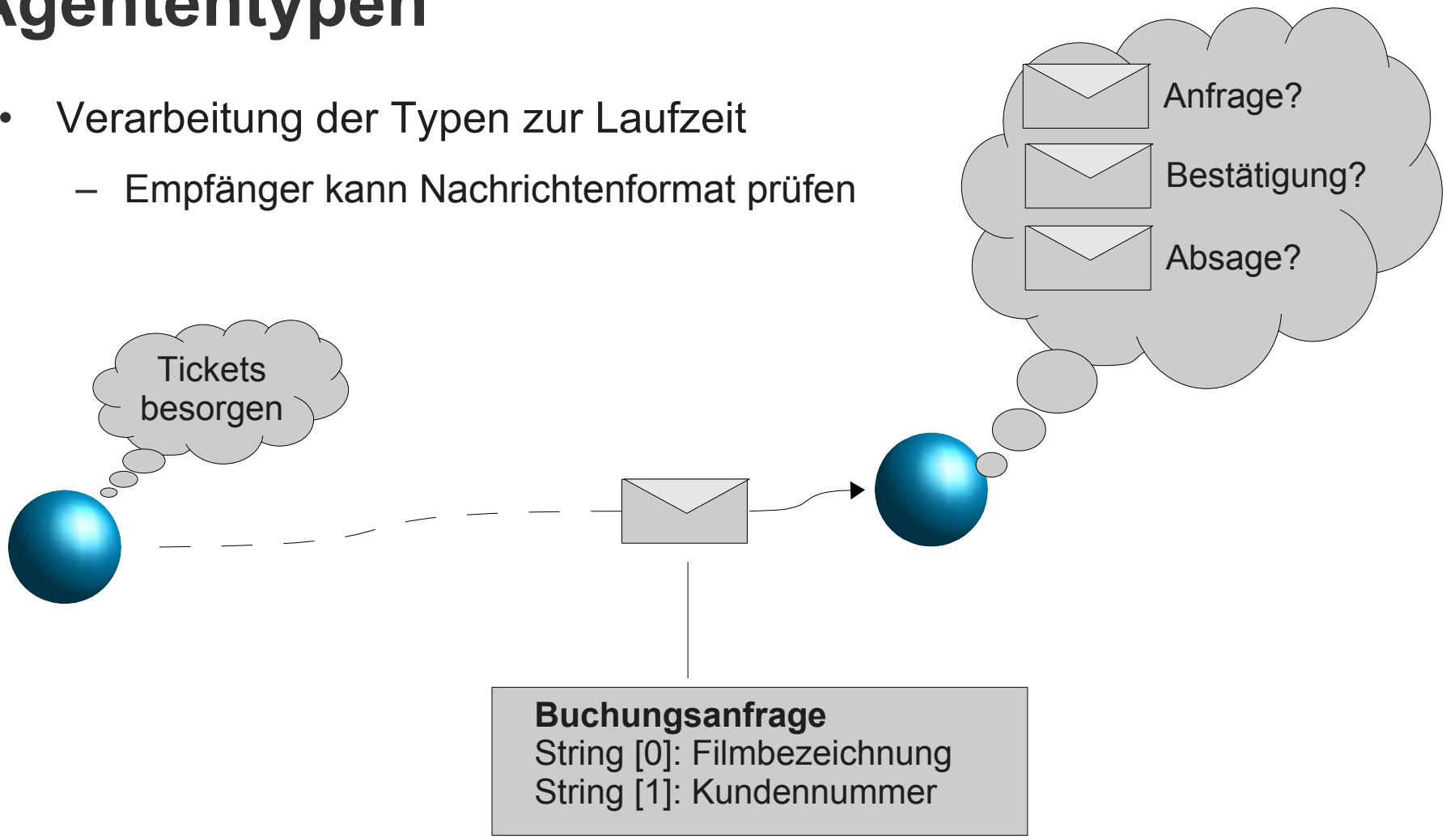
Agententypen

- Außerdem Semantik beachten
  - Nachrichten semantisch annotieren (Anmerkungen hinzufügen)
  - Auch Agent kann semantisch annotiert werden

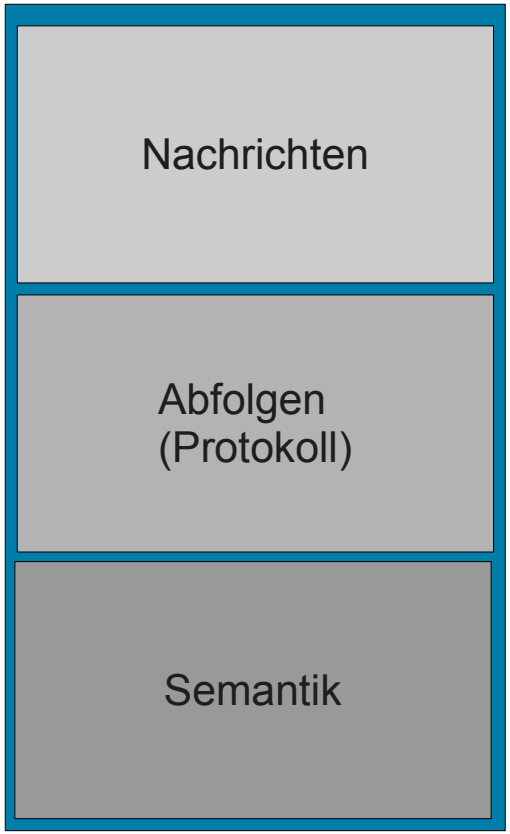


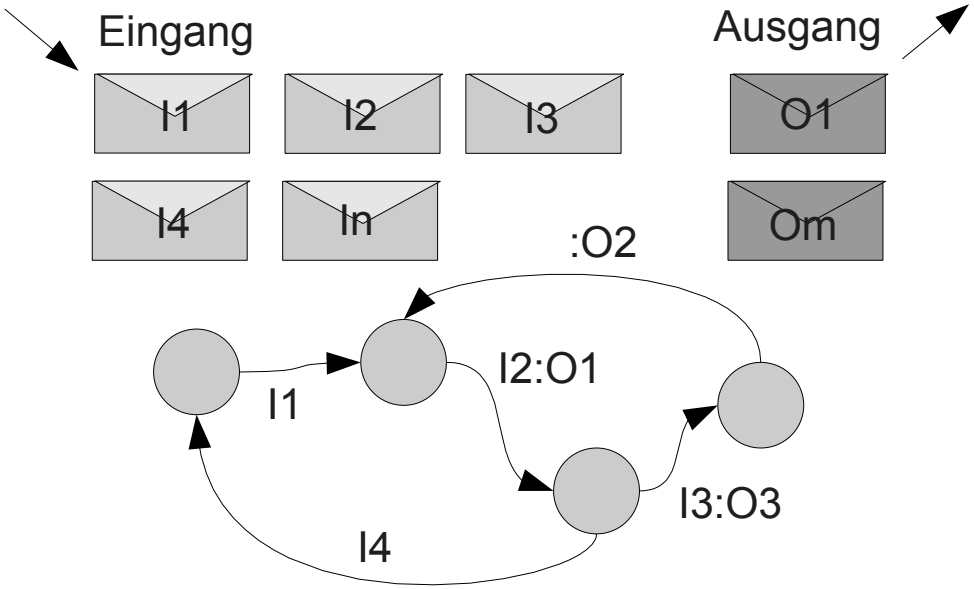
Agententypen

- Verarbeitung der Typen zur Laufzeit
  - Empfänger kann Nachrichtenformat prüfen



Hybridtypen





I1: Buchungsanfrage

I2: Bestellbestätigung

O1: Buchungsbestätigung

Selbst: Kinoschalter-Agent

Nachrichten

- Auflistung aller gültigen Nachrichtenformate
  - Eingehend: Welche Nachrichten werden akzeptiert?
  - Ausgehend: Welche Nachrichten werden versendet?
- Ähnlich einer Methodensignatur
  - Keine Aussage über Reihenfolge oder Abhängigkeit
  - Kein „Methodenname“
  - damit geringere Aussagekraft als eine Signatur

Nachrichtentyp

Beispiel

```
messages {
   in {
      I1: { java.lang.String, AMETAS.data.ALong };
      I2: { java.lang.Boolean };
   }
   out {
      O1: { java.net.INetAddress };
      Migrate: { migration };
   }
}
```

Pseudonachricht
Ich will migrieren
(an die Stelle gerichtet)

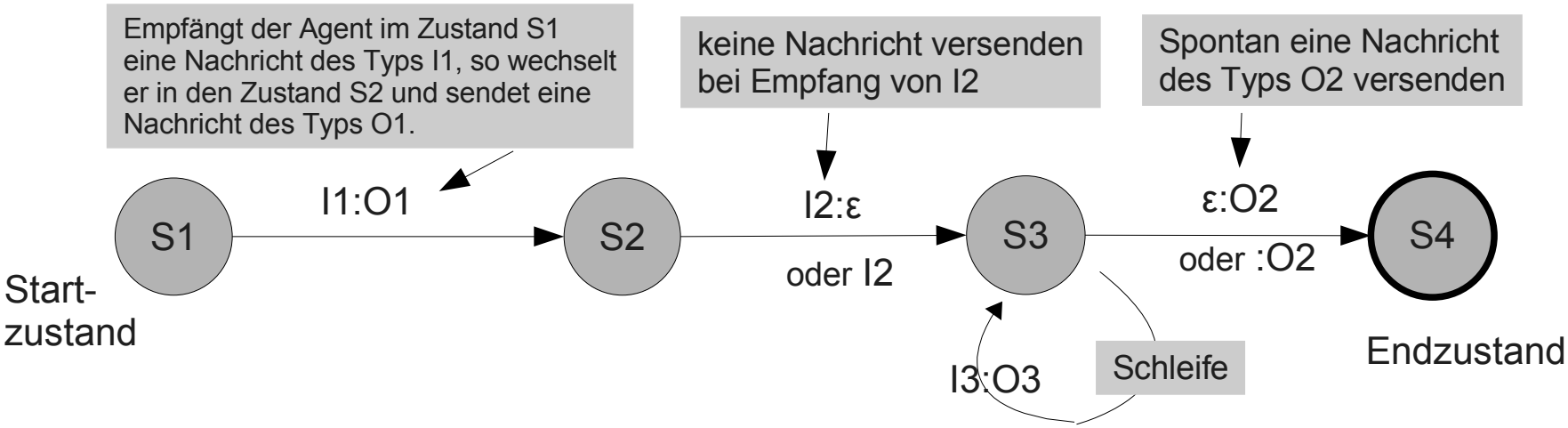
Mögliche Kombinationen:

I1 empfangen: O1 senden

I1 empfangen, danach I2 empfangen:
Migrate senden

Reihenfolgen

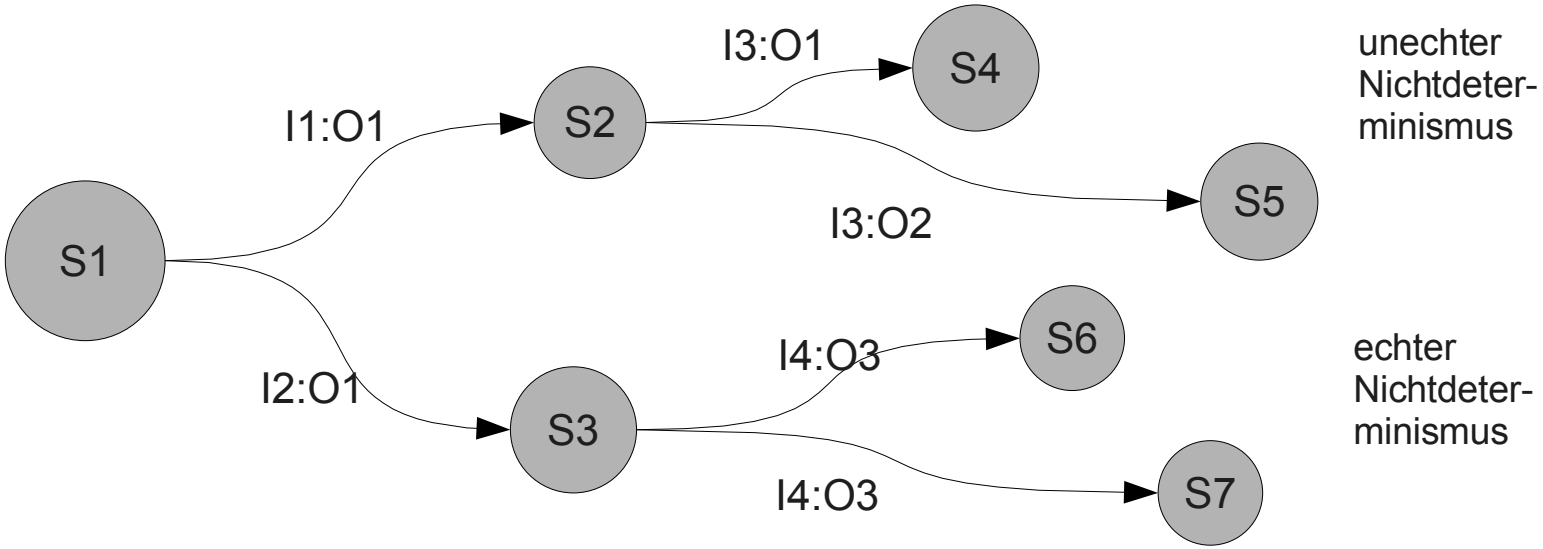
- Festlegen, welche Nachrichen hintereinander folgen
- Endlicher Automat mit Ausgabe



Notation
Ausgangszustand = Nachricht > Endzustand (: Nachricht (, Nachricht)*)
(+ Endzustand (: Nachricht(, Nachricht)*)*);

als Pfeil lesen

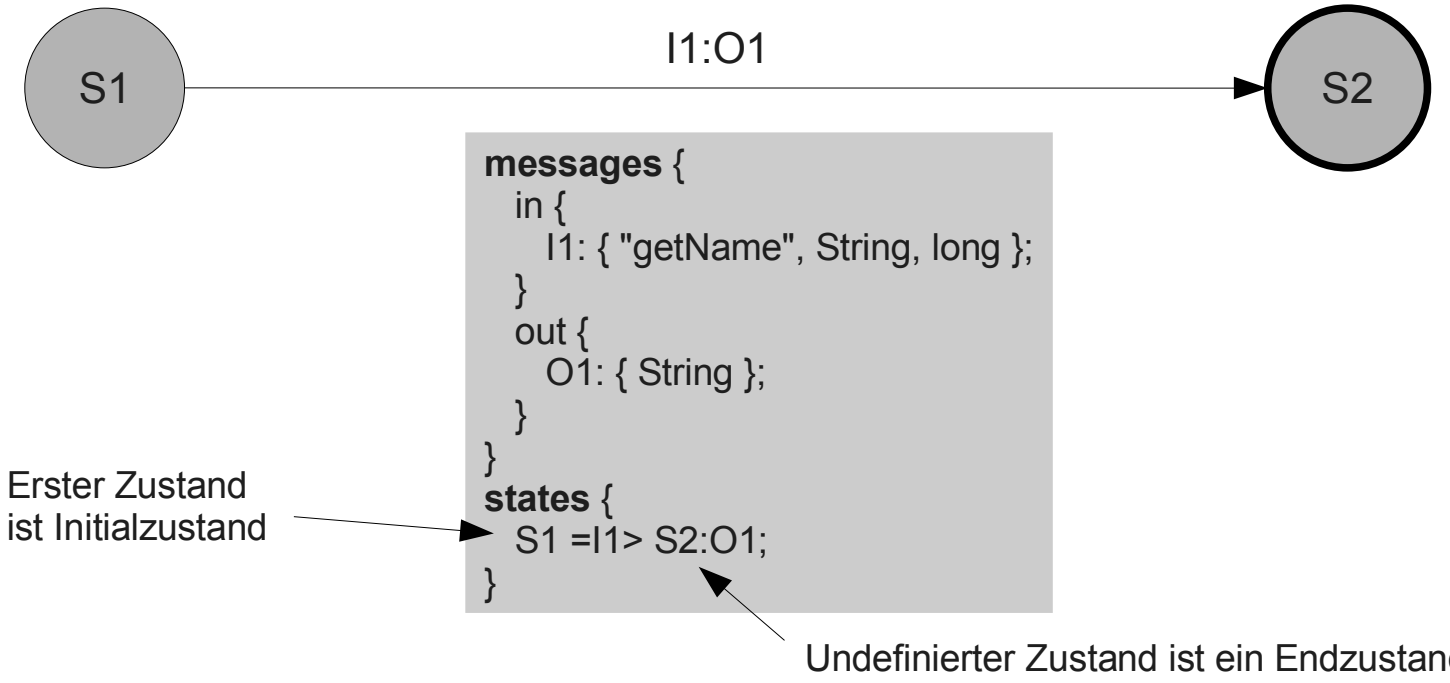
Reihenfolgen und Nichtdeterminismus



- Autonomie des Agenten
  - Der Agent kann zwischen Alternativen wählen, wie eine Nachricht zu behandeln ist
  - Annahme: Es gibt nur endlich viele Entscheidungspfade -> alle möglichen Entscheidungen sind in der Beschreibung aufführbar (NEA: nichtdeterministischer endlicher Automat)
  - Nichtdeterministische Alternative durch "+" gekennzeichnet

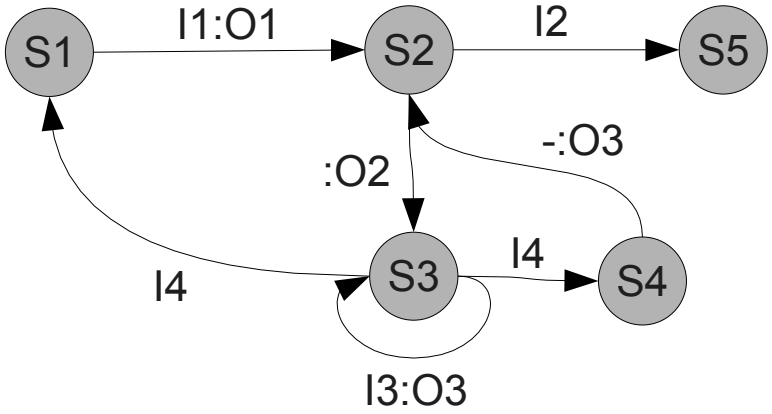
Reihenfolgen

- Vergleich mit Signaturen
  - *public String getName(String sUserID, long nKey);*



Reihenfolgen

- Komplexeres Beispiel




```
states {
   S1 = I1 > S2:O1;
   S2 = I2 > S5;
   S2 = none > S3:O2;
   S3 = I3 > S3:O3;
   S3 = I4 > S4 + S1;
   S4 = none > S2:O3;
}
```

Sender und Empfänger

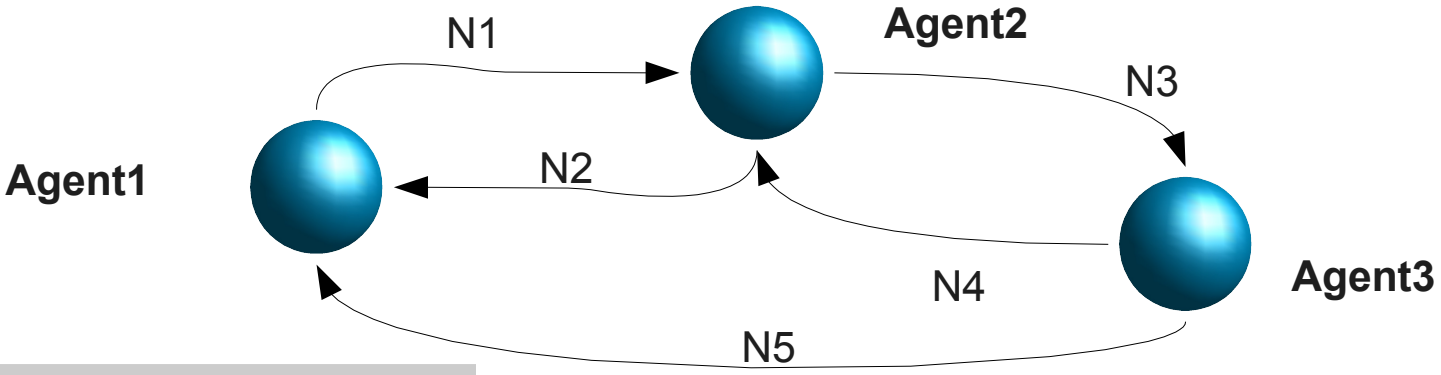
- Bisherige Schnittstellen
  - Sender bin ich
  - Empfänger ist das Objekt, das die Schnittstelle anbietet
- Agentenanwendungen
  - Multiagentenanwendung – viele Teilnehmer
  - (abstrakte) Klient-Server-Rollen ständig im Wechsel
  - Es kann relevant sein, dass ein bestimmter Agent in die Interaktion involviert wird

Man sollte Sender und Empfänger ebenfalls spezifizieren können. Standardvorgabe ist die konventionelle Kommunikation.

Sender und Empfänger

```
states {
   S1 = N1(Agent1) > S2:N3(Agent3);
   S2 = N4(Agent3) > S3:N2(Agent1);
}
```

Abfolge:
N1, N3, N4, N5, N2



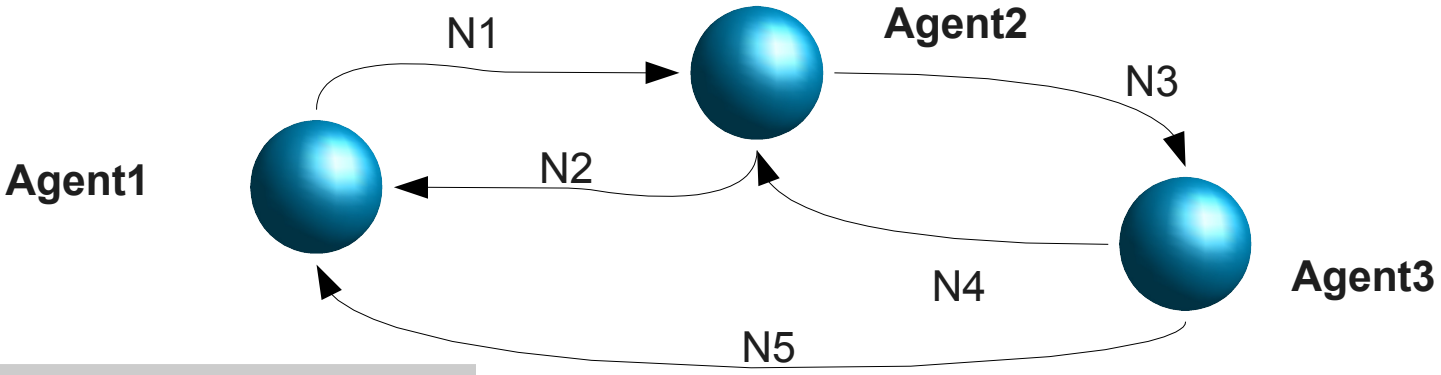

```
states {
  Start = none > S2:N1(Agent2);
   S2 = N5(Agent3) > S3;
   S2 = N2(Agent2) > S4;
   S3 = N2(Agent2) > End;
   S4 = N5(Agent3) > End;
}
```

```
states {
   S1 = N3(Agent2) > S2:N4(Agent2),N5(Agent1);
}
```

Primärer Partner

```
states {
   S1 = N1 > S2:N3(Agent3);
   S2 = N4(Agent3) > S3:N2;
}
```

Unbezeichneter Sender oder Empfänger = derjenige, der aufgrund dieser Beschreibung mit dem Agenten kommunizieren will (*primärer Partner*)




```
states {
  Start = none > S2:N1(Agent2);
   S2 = N5(Agent3) > S3;
   S2 = N2(Agent2) > S4;
   S3 = N2(Agent2) > End;
   S4 = N5(Agent3) > End;
}
```

```
states {
   S1 = N3 > S2:N4,N5(Agent1);
}
```

... bekommt N4 ...

Wer N3 schickt ...

... aber N5 geht immer an Agent1

Sender und Empfänger

- Pseudoangaben als Sender/Empfänger
  - **any**: Jeder kann Sender oder Empfänger sein (also nicht nur der primäre Kommunikationspartner)
  - **other**: Jeder außer dem primären Partner kann Sender oder Empfänger sein
  - **place**: Die Stelle ist Sender oder Empfänger

```
states {
   S1 = N1(any) > S2:N2;
}
```

ist nicht gleich

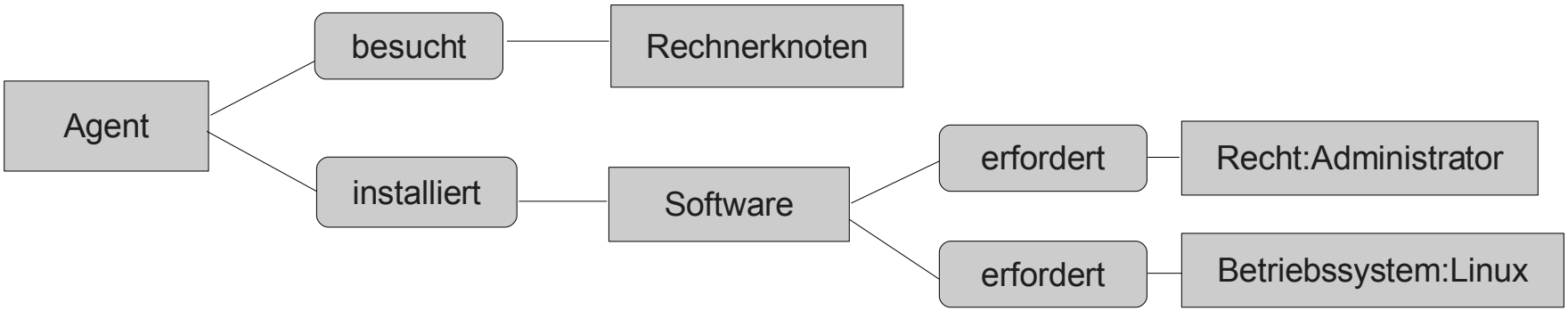
```
states {
   S1 = N1 > S2:N2;
}
```

- Migration als Pseudokommunikation
  - ist meist als Methodenaufruf realisiert
  - Ziel wird nicht angegeben

```
states {
   S1 = none > S2:Migrate(place);
}
```

Semantik

- Formalisierung der Semantik über Konzeptgraphen
  - Graph mit zwei Klassen von Knoten: *Konzepte* und *Relationen*
  - hier eingeschränkt auf Konzeptbäume
  - Wurzel und Blätter sind Konzepte; Kindknoten sind stets aus der jeweils anderen Klasse

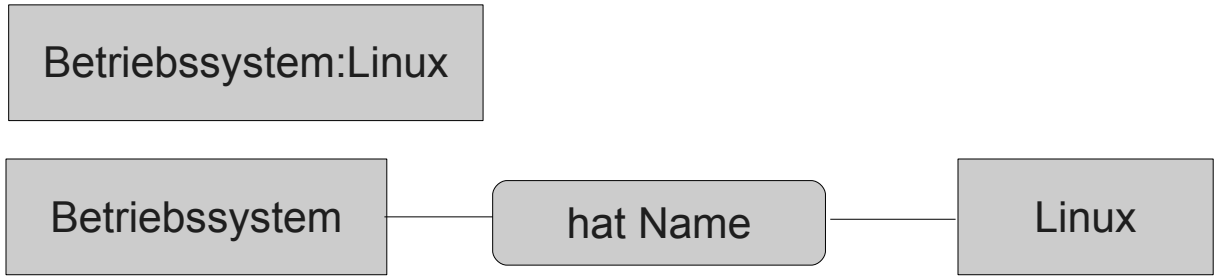


Konzeptgraphen

- Idee von John Sowa
- Erkenntnisse aus der Psychologie
  - Verstand arbeitet mit Konzepten, die er in Beziehung bringt
  - Assoziationsketten durch Verknüpfung von Konzepten
  - Im allgemeinen Fall ein gerichteter Graph (evtl. mit Zyklen)
- Konzept
  - Im alltäglichen Sprachgebrauch soviel wie „Subjekt“, „Objekt“
  - Kann aber (prinzipiell) auch das Prädikat sein
  - Instanzen als „Beispiele“ für das Konzept:
    - Konzept ist „Farbe“, Instanz ist „Rot“
    - Konzept ist „Betriebssystem“, Instanz ist „Windows“

Konzeptgraphen

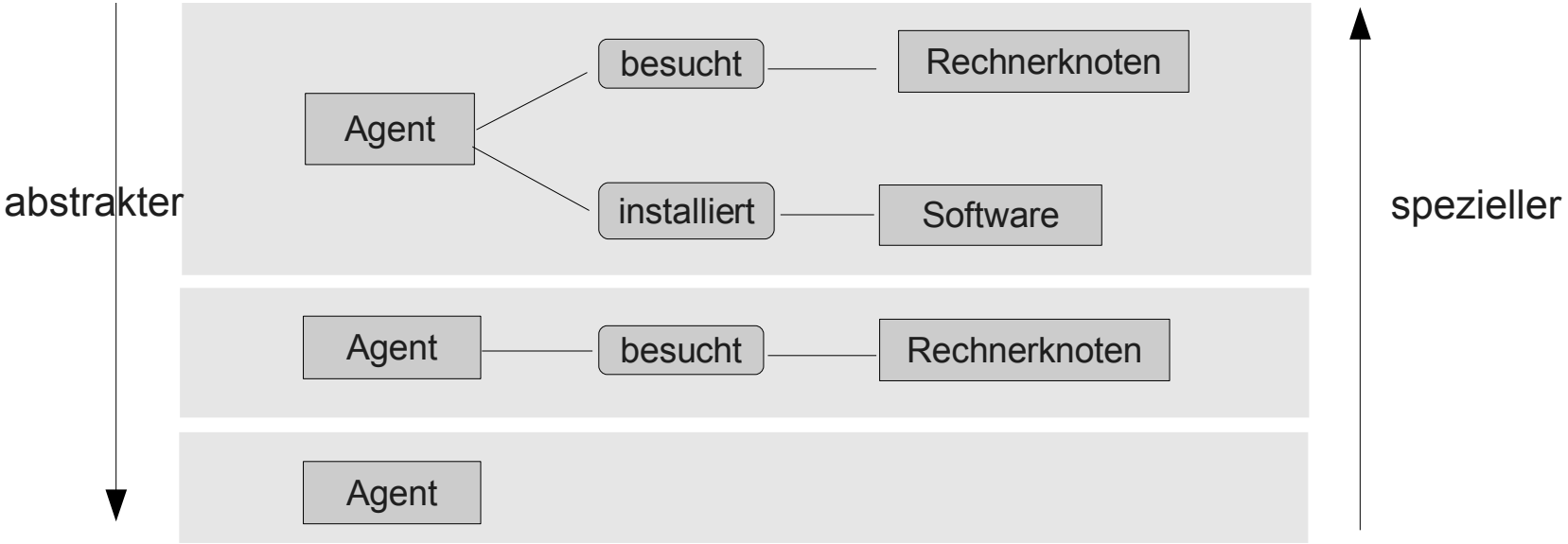
- Relation
  - Im alltäglichen Sprachgebrauch soviel wie ein Prädikat, Attribuierung
- Relationen haben keine Instanzen
- Jede Instantiierung kann auch als Relation geschrieben werden



Automatische Erzeugung von Konzeptgraphen nur in eingeschränkten Domänen möglich, etwa aus einer IDL-Beschreibung heraus

Semantik

- Nützliche Eigenschaften der Konzeptgraphen
  - leicht lesbar, intuitiv zu verstehen; einzelne Äste sind UND-verknüpft
  - spezialisierbar durch Anhängen weiterer Knoten
  - abstrahierbar durch Abschneiden von Knoten



Semantik

- Subtypen bei Konzeptgraphen
  - Detailliertere Konzeptgraphen => kleinere Mengen
  - Subtypen von Konzeptgraphen sind also Spezialisierungen
- Abfrage
  - Verwendung einer textuellen Repräsentation
    - [Konzept(:Instanz)] –> (Relation) – >[Konzept(:Instanz)] ...
- Beispiel
  - [Agent] -> (eingesetzt_in) -> [Netzmanagement]
    Suche alle Agenten, die im Netzmanagement eingesetzt werden

Semantik und Hybridtypen

- Dritter Teil der Hybridtypbeschreibung
- „Annotationen“ (semantische Anfügungen) kennzeichnen
  - Nachrichtenelemente (Daten), etwa „Name“, „Adresse“
  - ganze Nachrichten, etwa „Bestätigung“
  - Absender und Empfänger, wie „Ticket-Agent“, „Standardbenutzeradapter“
  - Zustände, so als „Wartezustand“, „Fehler“
  - den ganzen Agenten/Stellennutzer, so genannte Selbstannotation („Dieser Agent ist ein Kaufagent“)
- Vorteil
  - algorithmisch vergleichbar
  - jedoch keine Garantie, dass sinnvolle Treffer geliefert werden

Semantik

- Problem
  - Gleiches Phänomen wird häufig mit verschiedenen Worten beschrieben
  - Automatische Treffersuche daher sehr vage
- Teilweise Abhilfe
  - Verwendung eines „Wörterbuchs“ mit Begriffshierarchien: Ontologie (erklärt, was es in der Welt gibt)
  - lässt sich relativ einfach bereitstellen
  - Dennoch keine absolute Sicherheit
    - Ontologie nie umfassend genug
    - Sorgfältige Erfassung aller relevanten semantischen Eigenschaften erforderlich

Treffersuche (Matching)

- Aufgabe
  - Gegegen ist eine Typbeschreibung
  - Suche „verträgliche“ Typbeschreibungen (idealerweise die gleiche Beschreibung)
  - Liefere bei Bedarf Instanzen dieser Typbeschreibung
- Erfordernis
  - Beschreibung muss „abstrahierbar“ sein: Von einer gegebenen Beschreibung muss eine allgemeinere Beschreibung ableitbar sein
- Ist der Hybridtyp abstrahierbar?
  - Wenn ja, dann kann man einen Subtyp vermitteln, um die Anfrage zu befriedigen.

Das Subtypprinzip

Ein Instanz eines Subtyps kann stets stellvertretend für eine Instanz des Supertyps verwendet werden, ohne dass es zu Einschränkungen oder Fehlverhalten kommt.

- Anwendung des Prinzips der geringsten Überraschung
  - Wenn schon nicht das Gefragte, dann wenigstens etwas Gleichwertiges
- Beispiel
  - Ich möchte einen Apfel essen
    - Dann bin ich mit einem Apfel der Sorte Granny Smith zufrieden
    - Umgekehrt gilt das i.A. nicht
  - Ich erwarte eine ganze Zahl
    - Dann bin ich mit einem int-Wert zufrieden
    - Ebenso: Umgekehrt gilt das i.A. nicht (z.B. int erwarten, aber long bekommen)

Subtyp bei Daten

- Einfachster Fall: Datentypvergleich
  - Definiert innerhalb der Programmiersprache
  - Komplexe, selbstgebaute Typen erfordern semantische Annotationen
    - Verzicht auf selbstgebaute Typen?
    - Alles als Text? Das verschiebt das Problem der Typüberprüfung nur.
- Java
  - Test über *boolean java.lang.Class.isAssignableFrom(Class)*
  - Zusätzlich Vergleich der Dimension: *String[]* Subtyp von *Object* -> *String[][]* Subtyp von *Object[]* Subtyp von *Object*

Subtyp bei Nachrichten

- Nachricht = Folge von Nachrichtenelementen
  - A = [A(i)], i=0...n
  - Jedes Nachrichtenelement A(i) hat einen Datentyp
  - Nachricht A ist Subtyp von Nachricht B, wenn jedes Element A(i) Subtyp von B(i)
  - Nachrichten dürfen länger sein als erwartet (Rest einfach abschneiden)
    - vorausgesetzt, die längere Nachricht passt in den bereitgehaltenen Speicher
- Beispiel

  A = [java.lang.String[], AMETAS.data.AInteger, java.net.InetAddress]
  B = [java.lang.Object, AMETAS.data.ALong]

  A ist Subtyp von B
  Probe: Wenn ich B erwarte, aber A erhalte, bin ich überrascht?

Kovarianz und Kontravarianz

- Subtypbeziehungen beim Nachrichtenaustausch
  - Der Subtyp soll höchstens jene Nachrichten schicken, die auch sein Supertyp schicken würde

  - Der Subtyp soll mindestens die Nachrichten akzeptieren, die auch sein Supertyp akzeptieren würde
- Typ der ausgegebenen Nachrichten verhält sich **kovariant** zum Objekttyp; Typ der akzeptierten Nachrichten verhält sich **kontravariant**
  - Dies gilt auch für die Datentypen der Nachrichten

Kovarianz und Kontravarianz

- Kontravarianz
  - Gesucht ist ein Agent, der A = [java.lang.String, AMETAS.data.AInteger] akzeptiert
  - Vorhanden ist B = [java.lang.String] und C = [java.lang.String, AMETAS.data.AShort]
  - Passend ist B, denn beim Senden von [String, AInteger] reagiert B fehlerfrei (B ignoriert allerdings möglicherweise AInteger)
  - C hingegen würde bei [String, 2^20] einen Fehler melden, da 2^20 kein short-Wert ist.
- Kovarianz
  - Obiges Beispiel, jedoch wird ein Agent gesucht, der A ausgibt
    - Dann passt C, aber nicht B.

Vergleichsalgorithmus

- Gegeben sei
  - ein Verzeichnis von Typbeschreibungen von Agenten (laufend oder startbar)
  - eine Anfrage in Form einer Typbeschreibung
- Aktion
  - Suche im Verzeichnis alle Beschreibungen, die einen Subtyp der angebotenen Beschreibung darstellen
  - Liefere diese Beschreibungen an den Anfrager
  - Falls der Anfrager Instanzen sucht, liefere die Identifikatoren der laufenden Agenten

Subtyp heißt ...

- bei den Nachrichten:
  - Die Menge der in-Nachrichten des Subtyps umfasst die Menge der in-Nachrichten des Supertyps (Kontravarianz)
  - entsprechend out-Nachrichten (Kovarianz)
  - Datentypen müssen ebenfalls verglichen werden
- beim Protokoll:
  - Jeder Protokollablauf des Supertyps muss im Subtyp möglich sein
    - d.h. man muss mit dem Subtypen in gleicher Weise eine Konversation führen können (aber ggf. auch andere, spezifische)
  - Protokoll schränkt Nachrichtenvergleich ein
    - Es werden nur Nachrichten miteinander verglichen, die bei entsprechenden Protokollabfolgen auftauchen

Subtyp heißt ...

- bei der Semantik:
  - Die Selbstannotation des Subtyps muss spezieller sein als die gefragte (des Supertyps)
  - Die semantischen Annotationen der Zustände und Nachrichten verhalten sich in der Typhierarchie wie die zugehörigen Zustände und Nachrichten (Nachrichtenelement = Subtyp -> zugehörige Annotation muss auch Subtyp sein)

Resultat

- Resultat
  - Ein System von Beschreibungen, anhand derer Agenten in einem Verzeichnis zu finden sind (aktuell laufend = Instanzvermittlung, startbar = Typvermittlung)
  - Eine Hierarchie von Beschreibungen aufgrund der „Subtypbeziehung“
  - Beschreibungen klären über legitime Operationen mit der Instanz auf: Typsystem
  - Mobile Agenten lassen sich von nichtmobilen unterscheiden (migration-Nachrichtenelement)
  - Multiagentenkommunikation lässt sich anhand der Beschreibungen repräsentieren

Hybridtypsystem

Kritik

- Hybridtypsystem sehr flexibel
  - möglicherweise zu flexibel, kann definierte Konversationen nur aufwändig überprüfen
  - Nichtdeterminismus zwar repräsentierbar, aber sehr aufwändig zu vergleichen (exponentieller Aufwand)
    - Ergebnisse sind hier immer noch von der eigentlichen Implementierung abhängig; d.h. wird der Nichtdeterminismus bei allen Partnern wirklich in der Implementierung beachtet?
- Nachrichtenvergleich nutzt Java-Typen
  - Viele Kommunikationssprachen rein stringbasiert
    - Man kann dann aber noch immer Vergleiche anstellen, etwa „123“ = „positive Ganzzahl“, „0.75“ = „rationale Zahl“ (Supertyp von positive Ganzzahl)
    - Prüfung semantisch, nicht mehr Datentypsystem-abhängig

Kritik

- Semantik
  - Erstellung obliegt dem Implementierer (oder Typ-Autor)
  - Festlegung auf eine Ontologie?
    - Menge der möglichen Begriffe eingrenzen
  - Ergebnisse unter Vorbehalt

Andere Ansätze

- Anderer Weg
  - Festlegung von Standard-Kommunikationsakten (geschlossene Welt!)
    - Auktion
    - Preisverhandlung
    - Bestätigte Bestellung ...
  - Vergleich der Verträglichkeit anhand dieser Akte
  - Vorteil: leichte Überprüfbarkeit
  - Nachteil: Problem ist eine Ebene verschoben worden
    - Man könnte die festgelegten Konversationen wiederum zu Bausteinen erklären und Protokolle darüber definieren => Geschäftsprozessebene
  - Allerdings können die Standardakte die meisten Fälle sinnvoll abdecken