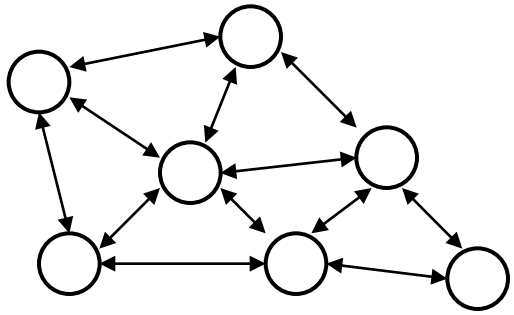
Grundlagen der Künstlichen Intelligenz

Sebastian Iwanowski
FH Wedel

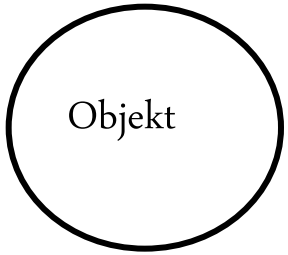
Kap. 6:
Agentenorientierte Systeme
(mit Anwendungsbeispiel Touristeninformationssystem)

Basistechnologie: Agentenorientierte Software

Multiagentensystem:



Softwareagent:



- autonom
- soziale Kompetenz
- reaktiv
- proaktiv

Aufgaben für einen Agenten im Multiagentenkontext:

- Planung
- Kommunikation
- Entscheidungsfindung
  - für individuellen Vorteil → Auktionen, Verhandlungen
  - für Gruppenvorteil → Schwarmintelligenz

wurde vertieft im Informatik-Seminar SS 2010

Kapitel 7

Touristeninformationssystem als Beispiel für eine Diensteplattform mit weiter gehenden Ansprüchen

Ziele:

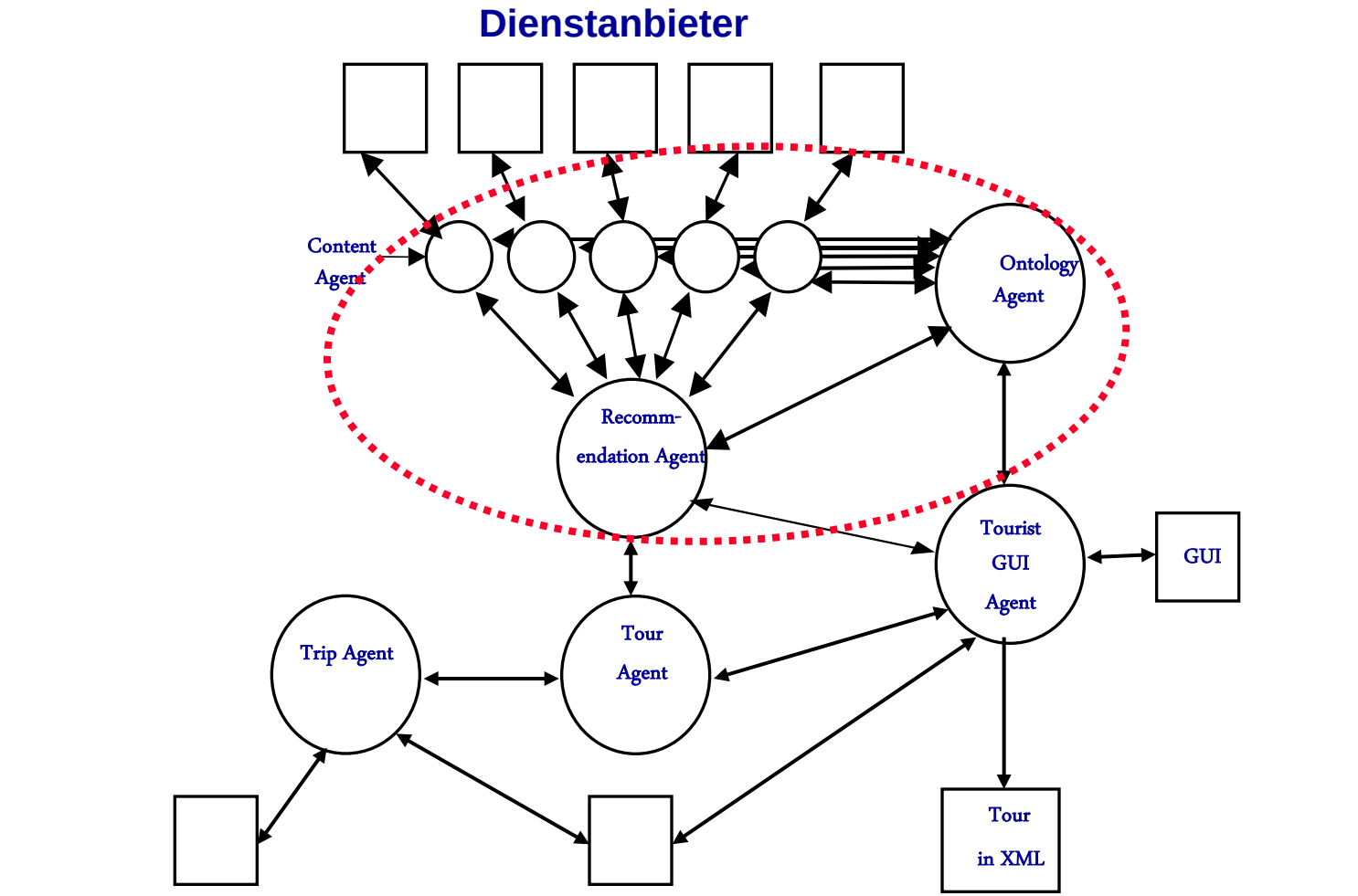
- **Unterstützung plattform- und softwareunabhängiger Dienste**

→ Verteilte Systeme

- **Unterstützung der Generierung neuer Diensttypen**
- **Unterstützung der Generierung von Mehrwertdiensten**

→ Künstliche Intelligenz

Touristeninformationssystem



Ontologien

Was ist eine Ontologie ?

- Begriff „Ontologie“ stammt aus agentenorientierter Programmierung
- Ontologie bezeichnet Begriffswelt
- Ontologien dienen zur Beschreibung von Semantik
- Neueste Entwicklung: Semantic Web

Syntax zur Beschreibung von Ontologien

- Objektorientierte Klassifizierungsbäume (UML)
- KQML (Knowledge Query Manipulation Language)
- XML

Ontologien

Was bezweckt man mit der Definition von Ontologien ?

- **Vereinheitlichung der Begriffswelt zwischen autonomen Dienst- oder Informationsanbietern**
- **Möglichkeit der Einbindung neuer Anbieter in ein bestehendes verteiltes Dienst- oder Informationssystem**

Ontologien: Beispiel einer Tripbeschreibung

KQML:

```
<Trip> ::= (Trip
            :startingPlace <GeoPlace>      :departure <TimeStamp>
            :destinationPlace <GeoPlace>   :arrival <TimeStamp>          :numberOfChanges <number>
            :rides (list {<Ride>}+))

<Ride> ::= <PublicRide> | <PrivateRide>

<PublicRide> ::= (PublicRide
                        :lineName <string>
                        :startingPlace <StopPlace>      :departure <TimeStamp>
                        :destinationPlace <StopPlace>   :arrival <TimeStamp>
                        :directionPlace <StopPlace>     :rideTime <number>)

<PrivateRide> ::= (PrivateRide
                        :transportMedium <string>
                        :startingPlace <Place>          :departure <TimeStamp>
                        :destinationPlace <Place>       :arrival <TimeStamp>
                        :rideLength <number>            :rideTime <number>)
```

Ontologien: Beispiel einer Tripbeschreibung

XML: (informell)

```
⟨Trip⟩ ::= <trip language="⟨String⟩" orderNr="⟨Integer⟩">
        <starting> ⟨GeoPlace⟩ ⟨Date⟩ ⟨Time⟩ </starting>
        <ending> ⟨GeoPlace⟩ ⟨Date⟩ ⟨Time⟩ </ending>
        <rides> {⟨Ride⟩}+ </rides>
    </trip>

⟨Ride⟩ ::= ⟨PublicRide⟩ | ⟨PrivateRide⟩

⟨PublicRide⟩ ::=   <public-ride orderNr="⟨Integer⟩">
            <line> ⟨String⟩ </line>
            <starting>
                <station-name> ⟨String⟩ </station-name>
                ⟨GeoPlace⟩
                ⟨Date⟩
                ⟨Time⟩
            </starting>
            <ending>
                <station-name> ⟨String⟩ </station-name>
                ⟨GeoPlace⟩
                 ⟨Date⟩
                ⟨Time⟩
            </ending>
        </public-ride>

⟨PrivateRide⟩ ::=  <private-ride orderNr="⟨Integer⟩">
            <starting> ⟨GeoPlace⟩ ⟨Date⟩ ⟨Time⟩ </starting>
            <ending> ⟨GeoPlace⟩ Date⟩ ⟨Time⟩ </ending>
        </private-ride>
```

Softwareunabhängige Dienstevermittlung

Problem:

- **Woher weiß ein Client, welche Fragen der Agent versteht ?**
- **Woher weiß ein Agent, welche Antwort ein Client zu einer gegebenen Frage erwartet ?**

Lösung:

1. **Agent gibt seine Ontologie bekannt**
2. **Client stellt seine Frage als unvollständigen Begriff aus der Server-Ontologie**
3. **Agent vervollständigt den Begriff mit Daten, die er generiert, und sendet sie als Antwort zurück.**
    - Client kann in Frage angeben, wie viele Antworten er erwartet und wie detailliert die Antworten sein sollen.

Softwareunabhängige Dienstevermittlung

Beispiel für TripAgent als Agent mit der eben definierten Ontologie (KQML):

Frage des TourAgents:

```
(Trip       :startingPlace (GeoPlace :x 4591300 :y 5822100)
            :departure (TimeStamp :date “29.02.2000” :time “12:00”)
            :destinationPlace (GeoPlace :x 4593100 :y 5825850))
```

Antwort des TripAgents:

```
(Trip       :startingPlace (GeoPlace :x 4591300 :y 5822100)
            :departure (TimeStamp :date “29.02.2000” :time “12:05”)
            :destinationPlace (GeoPlace :x 4593100 :y 5825850)
            :arrival (TimeStamp :date “29.02.2000” :time “12:17”)
```

Detailliertere Frage des TourAgents:

```
(Trip       :startingPlace (GeoPlace :x 4591300 :y 5822100)
            :departure (TimeStamp :date “29.02.2000” :time “12:00”)
            :destinationPlace (GeoPlace :x 4593100 :y 5825850)
            :rides *)
```

Detailliertere Antwort des TripAgents:

```
(Trip
    :startingPlace (GeoPlace :x 4591300 :y 5822100)
    :departure (TimeStamp :date “29.02.2000” :time “12:05”)
    :destinationPlace (GeoPlace :x 4593100 :y 5825850)
    :arrival (TimeStamp :date “29.02.2000” :time “12:17”)
    :numberOfChanges 2
    :rides
        ((PrivateRide
            :transportMedium “footwalk”
            :startingPlace (GeoPlace :x 4591300 :y 5822100)          :departure (TimeStamp :date “29.02.2000” :time “12:05”)
            :destinationPlace (GeoPlace :x 4591207 :y 5822200)       :arrival (TimeStamp :date “29.02.2000” :time “12:07”)
            :rideLength 137                                          :rideTime “00:02”)
        (PublicRide
            :lineName “U9”
            :startingPlace (StopPlace :stopArea “U Turmstraße”)      :departure (TimeStamp :date “29.02.2000” :time “12:08”)
            :destinationPlace (StopPlace :stopArea “U Osloer Straße”) :arrival (TimeStamp :date “29.02.2000” :time “12:16”)
            :directionPlace (StopPlace :stopArea “U Osloer Straße”)  :rideTime “00:08”)
        (PrivateRide
            :transportMedium “footwalk”
            :startingPlace (GeoPlace :x 4593092 y: 5825754)          :departure (TimeStamp :date “29.02.2000” :time “12:16”)
            :destinationPlace (GeoPlace :x 4593100 :y 5825850)       :arrival (TimeStamp :date “29.02.2000” :time “12:17”)
            :rideLength 96                                           :rideTime “00:01”))
```

Einklinken eines neuen Anbieters in ein bestehendes verteiltes Informationssystem

Problem:

- **Wie kann sich ein neuer Anbieter so einbinden, dass er von potentiellen Clients auch gefunden wird ?**
- **Wie kann der Anbieter erreichen, dass ein potentieller Client seine Dienste versteht ?**

Lösung:

1. **Neuer Anbieter meldet sich bei zentraler Stelle an, die von vielen Clients besucht wird.**
2. **Neuer Anbieter benutzt eine im bestehenden Informationssystem bereits verwendete Ontologie**
3. **Neuer Anbieter hängt von ihm benötigte neue Begriffe als Unterbegriffe an bestehende Begriffe an.**

Einklinken eines neuen Anbieters in ein bestehendes verteiltes Informationssystem

Beispiel:

- **Eine Menge von Ontologien für POIs ist dem System bereits bekannt**
- **Ein neuer ShoppingAgent für Einkaufsmöglichkeiten will sich einbinden**
- **Ontologie des ShoppingAgents besteht aus einem neuen Begriff ShoppingPOI mit seinen Unterbegriffen und Attributen**
- **ShoppingAgent beschließt, seinen Begriff als Unterbegriff des bereits bestehenden Begriffs POI einzuklinken**

Einklinken eines neuen Anbieters in ein bestehendes verteiltes Informationssystem

Beispiel:

Bereits bekannte Ontologie:

```
(POI (DictEntry POI)
        :subClasses (<SightseeingPOI> <DiningPOI> <EntertainmentPOI>)
        :name (DictEntry Name) <string>
        :address (DictEntry Address) <AddressPlace>
        :description (DictEntry Description) <DictEntry>
        :place (DictEntry Dummy) ({<GeoPlace>}+))
```

Einklinken eines neuen Anbieters in ein bestehendes verteiltes Informationssystem

Beispiel:

Ontologie des neuen ShoppingAgents:

```
(ShoppingPlace (DictEntry ShoppingPlace)
      :subClasses
         ((DepartmentStore (DictEntry DepartmentStore) :subClasses ())
          (GroceryStore (DictEntry GroceryStore) :subClasses ())
          (FashionShop (DictEntry FashionShop) :subClasses ())
          (SouvenirShop (DictEntry SouvenirShop) :subClasses ())
          (SpecialShop (DictEntry SpecialShop) :subClasses ()))
      :sellingItems (DictEntry SellingItems) ({<DictEntry>}))
```

Einklinken in bereits bekannte Ontologie:

```
(POI (DictEntry POI)
        :subClasses (<SightseeingPOI> <DiningPOI> <EntertainmentPOI> ShoppingPlace)
        :name (DictEntry Name) <string>
        :address (DictEntry Address) <AddressPlace>
        :description (DictEntry Description) <DictEntry>
        :place (DictEntry Dummy) ({<GeoPlace>}+))
```

Anbieten eines Mehrwertdienstes

Problem:

- **Einzelne Anbieter können verschiedene Dienste anbieten, aber keiner kann die Dienste kombinieren**
- **Client möchte mehrere Dienste in Anspruch nehmen, fordert aber, dass die Dienste koordiniert werden**

Lösung:

- **Ein Mehrwertanbieter bietet sich als Koordinator an.**
- **Der Mehrwertanbieter bietet keine originären Dienste an, sondern vermittelt Anfragen an die Teilanbieter.**
- **Der Mehrwertanbieter kennt alle Ontologien der Teilanbieter sowie die Beziehungen zwischen ihnen.**

Anbieten eines Mehrwertdienstes

Beispiel:

- **Der RecommendationAgent bietet die Möglichkeit, nach Restaurants eines bestimmten Typs zu fragen, in denen ein Konzert eines bestimmten Typs stattfindet**
- **Ein RestaurantAgent bietet Ontologie an, in der Restaurants katalogisiert sind**
- **Ein EntertainmentAgent bietet Ontologie an, in der Entertainments katalogisiert sind**
- **Der RecommendationAgent fragt diese beiden Agent getrennt und übermittelt die zusammengesetzte Antwort, wobei er die jeweilige Herkunft nicht verbirgt**

Anbieten eines Mehrwertdienstes

Beispiel:

Ontologie des RestaurantAgents (Auszug):

```
(DiningPlace (DictEntry DiningPlace)
    :subClasses
        ((Bistro (DictEntry Bistro)
            :subClasses ()
            :seatingFacilities (DictEntry SeatingFacilitiesExist) <boolean>
        (Restaurant (DictEntry Restaurant)
            :subClasses ()
            :seats (DictEntry NumberOfSeats) <number>
            :stars (DictEntry CookingHats) <number>))
    :cuisine (DictEntry Cuisine) <DictEntry>
```

Anbieten eines Mehrwertdienstes

Beispiel:

Ontologie des EntertainmentAgents (Auszug):

```
(EntertainmentPOI (DictEntry EntertainmentPOI)
    :subClasses ()
    :events (DictEntry Dummy) ({<CulturalEvent>}+)

<CulturalEvent> ::=
(CulturalEvent (DictEntry CulturalEvent)
    :subClasses (<Theater> <Concert>)
    :description (DictEntry Description) <dictEntry>
    :admissionFee (DictEntry AdmissionFee) <float>
    :startingTime (DictEntry StartingTimeGeneral) <TimeStamp>
    :endingTime (DictEntry EndingTimeGeneral) <TimeStamp>)))

<Concert> ::=
(Concert (DictEntry Concert)
    :subClasses
        ((Classic (DictEntry Classic) :subClasses ())
         (Jazz (DictEntry Jazz) :subClasses ())
         (Folk (DictEntry Folk) :subClasses ())
         (Pop (DictEntry Pop) :subClasses ())
         (Rock (DictEntry Rock) :subClasses ())))
```

Anbieten eines Mehrwertdienstes

Beispiel:

Frage an den RecommendationAgent:

(POI ((Restaurant :cuisine Italian) AND
(EntertainmentPOI :events (Jazz :startingTime (TimeStamp :date „29.02.2000“)))))

Frage des RecommendationAgents an den RestaurantAgent:

(POI (Restaurant :cuisine Italian))

Frage des RecommendationAgent an den EntertainmentAgent:

(POI (EntertainmentPOI :events (Jazz :startingTime (TimeStamp :date „29.02.2000“))

Anbieten eines Mehrwertdienstes

Beispiel:

Antwort des RecommendationAgents nach Erhalt der Teilantworten:

```
((Restaurant    :seats 50 :stars 3 :cuisine (DictEntry Italian) :name “Piazza del Jazz”
                :address (AddressPlace :postalCode 10400 :city “Berlin”
                                       :streetName “Käthe-Kollwitz-Platz” :number 3)
                :description (DictEntry PiazzaJazzEntry)
                :place ((GeoPlace :x 4000 :y 5500))))
AND
(EntertainmentPOI     :seats 50 :name “Piazza del Jazz”
                      :address (AddressPlace :postalCode 10400 :city “Berlin”
                                             :streetName “Käthe-Kollwitz-Platz” :number 3)
                      :description (DictEntry PiazzaJazzEntry)
                      :place ((GeoPlace :x 4000 :y 5500))
                      :events ((Jazz
                                :description (DictEntry AckermanConcertEntry)
                                :startingTime (TimeStamp :date „29.02.2000“ :time „20:00“)
                                :endingTime (TimeStamp :date „29.02.2000“ :time „23:00“)))))
```