

# ***Applications of Artificial Intelligence***

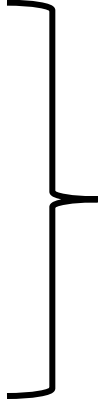
Sebastian Iwanowski  
FH Wedel

## **Chapter 6: Ontology Management**

### **6.1 Motivation and Examples with the Tourist Information System**

# Base Technology: Semantic Network

- **ontology management**
- **description language**
- **description logics**



developed in the 1990s based  
on AI syntax standards of the  
1980s

## Modern adaptation (2001): *Semantic Web standards*

Initiator: Tim Berners-Lee

Ontology management, description language and description logics  
in XML or comparable standards

### **Common feature:**

Universally valid definitions in a syntax readable by engines and browsers

# Ontologies

## What is an Ontology ?

- Term „Ontology“ was used in (AI) agent-oriented programming
- „Ontology“ means „vocabulary of concepts“
- Ontologies serve for description of semantics
- New revival : Semantic Web

## Syntax for describing ontologies

- Object-oriented classification trees (UML)
- KQML (Knowledge Query Manipulation Language)
- XML

# Ontologies

**What purpose do we want to use ontologies for ?**

- **Unifying the semantics between heterogenous service and information providers**
- **Opportunity of connecting new providers into an existing service system**

# Roots of the Tourist Information System

Daimler Telematic Research (Berlin) 1999 - 2001

## Organisational and strategic setup:

- Department was a spin-off of an agent-based technology department.
- Daimler was planning to be a trendsetter for telematic added-value services.
- The Berlin group had a Smart fleet available for telematic experiments.
- Main idea was to provide location-based services  
(first in the world: a combination of location-based service and context-dependent planning).
- Since Smart was designed to be a city car, multi-modal routing was also an option.
- **Most important:**  
**Daimler did not want to be involved in maintenance of tourist information infos.**

# Berlin Companion

**Daimler Telematic Research (Berlin) 1999 - 2001**

## **Prototype on a Smart fleet with the following principles:**

- Tourist has always final decision and control.
- The content is managed by on-line providers, not by this system.
- Several providers are consulted for any service (achieving fault tolerance and more objective information).
- System generates added-value information that is not given by any single provider.
- New providers may enter the system whenever they want (also for new categories of interest).

# Tourist Information System as an Example for a Server Platform with Advanced Goals

## Goals:

- Support of platform and **software** independent services

→ Distributed Systems

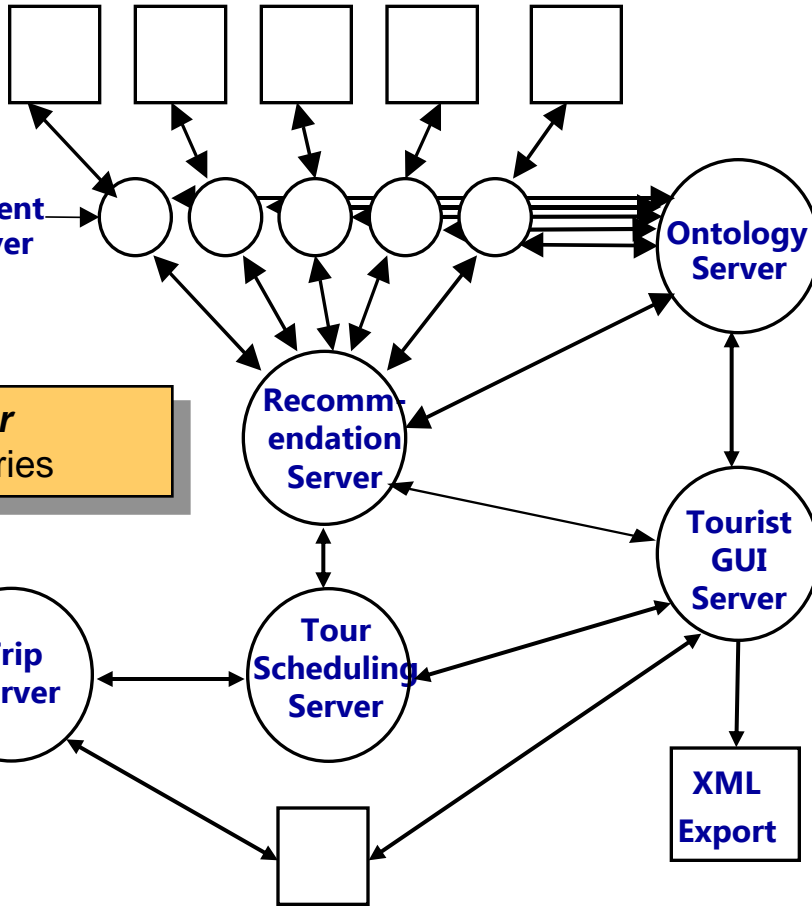
- Support of integration of new service types
- Support of enabling added-value services

→ Artificial Intelligence

# Tourist Information System

System architecture was later be called **Service-oriented (SOA)**

## Service and Content Providers



**Content Server**  
connect providers to system

**Providers** are autonomous and have own ontologies

**Recommendation Server**  
manage added-value queries

**Ontology Server** connect different ontologies of providers with system ontology

**Trip Server**  
compute trips between two locations

**Tourist GUI Server**  
adapts to individual tourist preferences

**Tour Scheduling Server**  
arrange tour order

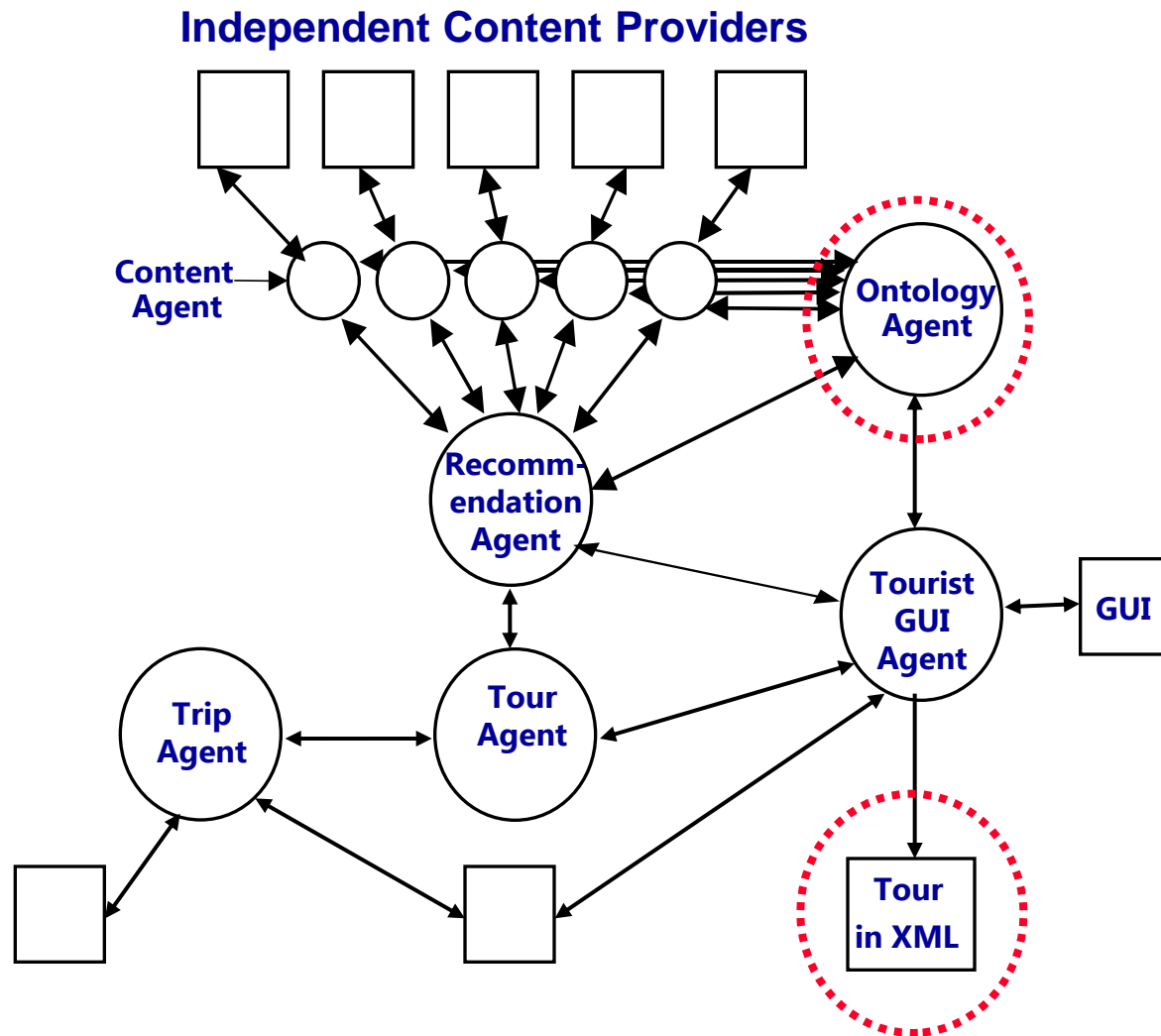
Public Transport Router

Digital Map

for mobile use



# Tourist Information System



Public Transport Router

Digital Map

Interface for a mobile

# Ontologies: Examples for a trip description

## KQML (AI description syntax of the 1990ies):

```
<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>)
```

# Ontologies: Examples for a trip description

## XML equivalent: (informal)

```
⟨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>  
    <geo-place> ⟨GeoPlace⟩ </geo-place>  
    <date> ⟨Date⟩ </date>  
    <time> ⟨Time⟩ </time>  
  </starting>  
  <ending>  
    <station-name> ⟨String⟩ </station-name>  
    <geo-place> ⟨GeoPlace⟩ </geo-place>  
    <date> ⟨Date⟩ </date>  
    <time> ⟨Time⟩ </time>  
  </ending>  
</public-ride>
```

```
⟨PrivateRide⟩ ::= <private-ride orderNr="⟨Integer⟩">  
  <transport-medium> ⟨String⟩ </transport-medium>  
  <starting> ⟨...same as with PublicRides...⟩ </starting>  
  <ending> ⟨...same as with PublicRides...⟩ </ending>  
</private-ride>
```

# Software independent service mediation

## Problem:

- How does a client know which questions a server understands?
- How does a server know which answer a client expects when the client asks a query?

## Solution:

1. Server publishes its own ontology
2. Client asks his questions giving an incomplete expression of the server ontologies
3. Agents completes the query with own data and sends that back.
  - Client may specify in his query how many answers should be given and how detailed they should be.

# Software independent service mediation

Example for the TripAgent with the ontology just defined (KQML):

## Query of TourAgent:

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

## Answer of TripAgent:

```
(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"))
```

## More detailed query of TourAgent:

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

## More detailed answer of TripAgent:

(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"))

# Connecting a new service provider into an existing distributed information system

## Problem:

- How can a new provider get linked into an existing system such that the provider is found by existing clients ?
- How can a new provider manage that an existing client understands his new services ?

## Solution:

1. New provider registers at central service point which is looked up by many clients.
2. New provider uses an ontology that already exists in the current information system.
3. New provider defines the top class(es) of his own ontology as subclass(es) of existing class(es) of the current ontology.

# Connecting a new service provider into an existing distributed information system

## Example:

- The system already knows some subclasses for POIs that are supported by different providers. **But no shopping!**
- A new ShoppingAgent wants to connect with the system.
- The Ontology of the ShoppingAgent has got ShoppingPOI as top class with subclasses and properties.
- ShoppingAgent decides to define ShoppingPOI as a direct subclass of the top class POI.



# Connecting a new service provider into an existing distributed information system

## Example:

### Ontology already existing:

(POI (DictEntry POI)

:subClasses (<SightseeingPOI> <DiningPOI> <EntertainmentPOI>)

:name (DictEntry Name) <string>

:address (DictEntry Address) <AddressPlace>

:description (DictEntry Description) <DictEntry>

:place (DictEntry Dummy) ({<GeoPlace>}+))

# Connecting a new service provider into an existing distributed information system

## Example:

### Ontology of the new ShoppingAgent:

```
(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>}}))
```

### Connecting into existing ontology:

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

# Providing an added-value service

## Problem:

- Individual providers may offer several services, but no one can combine them.
- Client wants to use several services but needs a combination.

## Solution:

- An added-value service offers coordination activities.
- The added-value service does not provide own services but rather asks several partial queries to original service providers.
- The added-value service knows all ontologies of original service providers and the relations in between.

# Providing an added-value service

## Example:

- **Added-Value Service Provider:**  
The **RecommendationAgent** offers to ask for restaurants of a special cuisine in which concerts of a special music group take place.
- **Original Service Provider:**  
The **RestaurantAgent** offers an ontology which classifies restaurants by their cuisine. It may also give the information if music is played at that restaurant but it may not have the current concert program.
- **Original Service Provider:**  
The **EntertainmentAgent** offers an ontology in which entertainments are classified and detailed concert programs are given. It may also describe the location and may even mention that it is restaurant, but it does not classify the cuisine.
- Der RecommendationAgent asks both agents as detailed as they understand this and computes the intersection set between both answers. This set is answered, but the individual parts are still tagged by the provider.

# Providing an added-value service

## Example:

### Ontology of RestaurantAgent (excerpt):

(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>

# Providing an added-value service

## Example:

### Ontology of EntertainmentAgent (excerpt):

```
(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 ())))
```

# Providing an added-value service

## Example:

### Query to RecommendationAgent:

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

### Query of RecommendationAgent to RestaurantAgent:

(POI (Restaurant :cuisine Italian))

### Query of RecommendationAgent to EntertainmentAgent:

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

# Providing an added-value service

## Example:

Answer of RecommendationAgent after receiving the answers to the partial questions:

```
((Restaurant :seats 40 :stars 3 :cuisine (DictEntry Italian) :name "Piazza del Jazz"  
:address (AddressPlace :postalCode 1040 :city "Berlin"  
:streetName "Platz des Kommunismus" :number 3)  
:description (DictEntry PiazzaJazzEntry)  
:place ((GeoPlace :x 4005 :y 5501))))
```

AND

```
(EntertainmentPOI :seats 100 :name "Piazza del Jazz"  
:address (AddressPlace :postalCode 10400 :city "Berlin"  
:streetName "Käthe-Kollwitz-Platz" :number 3)  
:description (DictEntry PiazzaJazzEntry)  
:place ((GeoPlace :x 4002 :y 4999))  
:events ((Jazz  
:description (DictEntry AckermanConcertEntry)  
:startingTime (TimeStamp :date „29.02.2000“ :time „20:00“)  
:endingTime (TimeStamp :date „29.02.2000“ :time „23:00“))))
```



# Important questions for practice

- **When are two instances considered equal?**

**Declare certain key properties and require them to be similar to a certain extent!**

- **What requirements do we need for a multilingual service?**

**A simple dictionary is not enough: Terms may depend on the context they are used. A class dependent dictionary may do so.**

- **How will we support new languages introduced by a new provider?**

**The tourist may choose any language in which he wants to be informed. If this language is supported by a POI / Event, the tourist is informed in this language, otherwise in English.**

- **How can we manage wrong-leading entries?**

**The information provider has to be shown together with the information. The tourist may disqualify certain providers (or preselect a positive list of the only providers he wants to get informed from).**