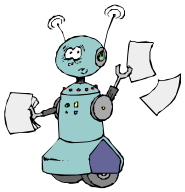
Praktikum Agententechnologie

Karl-Heinz Krempels
RWTH Aachen

Inhalt

Teil 1 - Einführung

Was sind Agenten?



Eigenschaften von Agenten

AgentCities

Was sind Agenten?

Agenten
- sind Programme:
  - die ihre Umwelt wahrnehmen und verändern,
  - mit anderen Agenten kommunizieren,
  - auf einer Agentenplattform existieren.
- sind Vertreter für Personen, Systeme oder Teilsysteme.

Definition des Agentenbegriffs über Eigenschaften der Agenten:
- autonom,
- reaktiv, proaktiv,
- mobil, stationär,
- kommunikativ,
- lernfähig, intelligent,
- interaktiv.

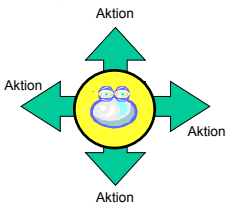
Eigenschaften von Agenten
Autonomie

Autonomie
- Der Agent entscheidet selbstständig anhand bestimmter Kriterien über seine nächste Aktion.
- Beispiele:
  - Orientierung auf einer Landkarte:
    Der Agent entscheidet anhand von bestimmten Kriterien (Zeit, Position, Helligkeit,..) welchen Weg er nutzt.

  - Buchung einer Reise:
    Der Agent entscheidet anhand der Präferenzen seines Prinzipals (Reiseziel, Transportmittel, ...) welche Reise er buchen soll.

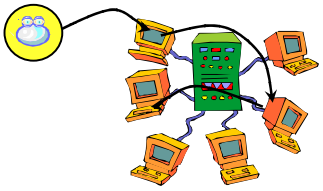
Eigenschaften von Agenten
Reaktivität / Proaktivität

Reaktivität / Proaktivität
- Der Agent *reagiert* auf Änderungen seiner Umgebung.
- Der Agent *verändert* seine Umgebung aufgrund von internen Parameter oder seines aktuellen Zustandes.
- Voraussetzungen:
  - Existenz von Sensoren und Regeln (Reaktivität),
  - Existenz eines Zieles für den Agenten (Proaktivität).



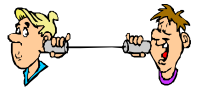
Mobilität

- Der Agent kann von einer Plattform auf eine andere Plattform migrieren.
- Voraussetzungen:
  - Serialisierbarkeit des Agentencodes,
  - Die Zielplattform kann den Code des Agenten interpretieren.



Kommunikation

- Der Agent kommuniziert mit seiner Umgebung (Dienste, andere Agenten, Agentensystem(e)).



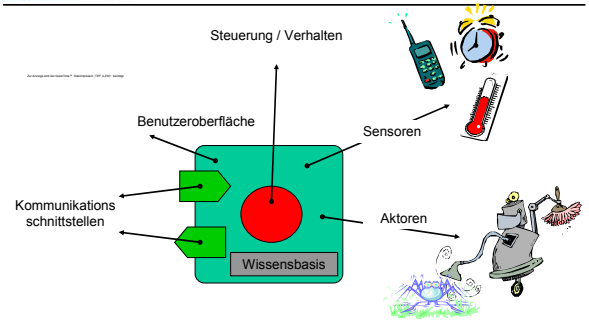
- Voraussetzungen:
  - gemeinsame Sprache (ACL),
  - gemeinsame Kommunikationsprotokolle,
  - gemeinsame Interaktionsprotokolle,
  - gemeinsame Weltsicht (Ontologie).

Lernfähigkeit / Intelligenz

- Der Agent kann auf verschieden Weisen lernen:
  - Übernahme von Wissen,
  - Anweisungen, Beispielen,
  - Bewertung des Erfolges / der Güte einer ausgeführten Aktion
    - Intern - durch Regeln / Funktionen,
    - Extern - durch einen Benutzer / System.
- Voraussetzung:
  - Existenz eines Bausteins im Agenten zur Wissensrepräsentation und Auswertung.

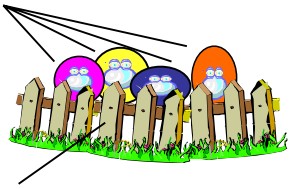
Interaktivität

- Mit einem Benutzer durch Benutzerschnittstelle.
- Beispiele:
  - SMS, WAP, eMail, GUI, Spracherkennung, usw.



Woraus besteht AgentCities.NET

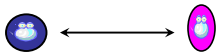
- Agenten



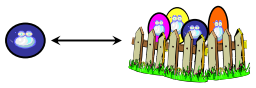
- Agentenplattformen

Angebotene Dienste in AgentCities.NET

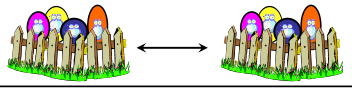
- Kommunikationsdienst - ermöglicht folgende Arten von Kommunikation:
  - Agent - Agent



  - Agent - Plattform

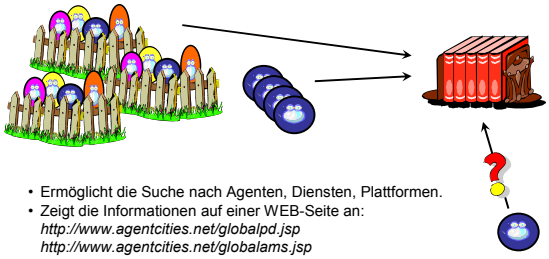


  - Plattform - Plattform



Angebotene Dienste in AgentCities.NET

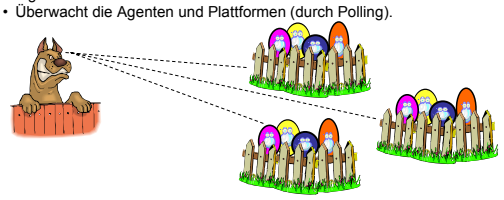
- Verzeichnisdienst
  - ermöglicht die Registrierung von Agenten, Diensten, Plattformen.



  - Ermöglicht die Suche nach Agenten, Diensten, Plattformen.
  - Zeigt die Informationen auf einer WEB-Seite an:
    http://www.agentcities.net/globalpd.jsp
    http://www.agentcities.net/globalams.jsp

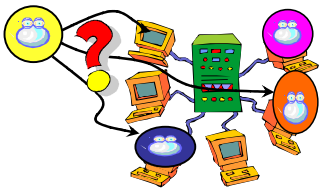
Angebotene Dienste in AgentCities.NET

- Managementdienst
  - Überwacht die Agenten und Plattformen (durch Polling).



  - Zeigt die Informationen auf einer WEB-Seite an:
    http://www.agentcities.net/globaldf.jsp

- Wie können **Kommunikationspartner gefunden werden?**
  - Anfrage bei einem zentralen Dienst,
  - Anfrage bei einem lokalen Dienst,
  - Feste Namensgebung (pizza@agent.org:7778/JADE).



Abfrage aller registrierten Plattformen - *request*

```
(request
 :sender(agent-identifier
        :name InterestedAgent@aachen.agentcities.net
        :addresses(sequence
            http://aachen.agentcities.net:7778/acc))
 :receiver(set(agent-identifier
        :name globalapd@root.agentcities.net
        :addresses(sequence
            http://root.agentcities.net:7778/acc))
        )
 :content"((action(agent-identifier
            :name globalapd@root.agentcities.net
            :addresses(sequence
              http://root.agentcities.net:7778/acc))
          (get-all-platforms))
         )"
 :reply-with rw2347058927340878234
 :language FIPA-SL0:encoding fipa.sl.rep.string.std
 :conversation-id ci234578629347652
 :protocol fipa-request
 :ontology FIPA-Agent-Management)
```


Antwort - *agree*

```
(agree
 :receiver(set(agent-identifier
            :name InterestedAgent@aachen.agentcities.net
            :addresses (sequence
                  http://aachen.agentcities.net:7778/acc)))
 :in-reply-to rw2347058927340878234
 :language FIPA-SL0
 :encoding string
 :ontology FIPA-Agent-Management
 :protocol fipa-request
 :conversation-id ci234578629347652
 :content"((action (agent-identifier
            :name globalapd@root.agentcities.net
            :addresses(sequence
                  http://root.agentcities.net:7778/acc))
        (get-all-platforms)))"
 :sender(agent-identifier
        :name globalapd@root.agentcities.net
        :addresses(sequence
            http://root.agentcities.net:7750/acc)))
```


Antwort - *inform*

```
(inform
 :receiver(set(agent-identifier
            :name InterestedAgent@aachen.agentcities.net
            :addresses (sequence
                  http://aachen.agentcities.net:7778/acc)))
 :in-reply-to rw2347058927340878234
 :language FIPA-SL0
 :encoding string
 :ontology FIPA-Agent-Management
 :protocol fipa-request
 :conversation-id ci234578629347652
 :content"((result
            (action (agent-identifier
                :name globalapd@root.agentcities.net
                :addresses(sequence
                    http://root.agentcities.net:7778/acc))
            (get-all-platforms))

            ... < Liste von AgentPlatformDescriptions > ...
 )"
 :sender(agent-identifier ...
```

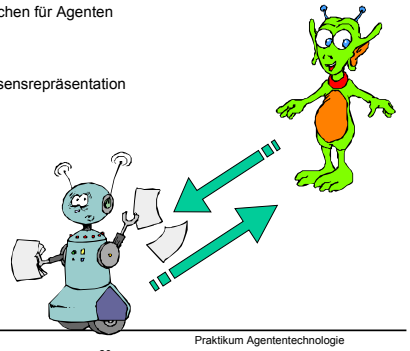
AgentCities
Erstellung einer ACL Nachricht in JADE

Erstellung einer *ping*-Antwort

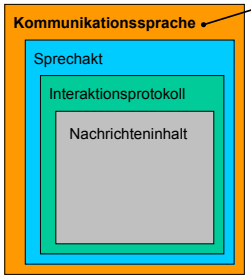
```
ACLMessage reply = msg.createReply();

if(msg.getPerformative()== ACLMessage.QUERY_REF){
    String content = msg.getContent();

   if ((content != null) && (content.indexOf("ping") != -1)) {
     //received a QUERY_REF with correct content.
     reply.setPerformative(ACLMessage.INFORM );
     reply.setContent("alive");
   } else {
     //received a QUERY_REF with uncorrect content.
     reply.setPerformative(ACLMessage.NOT_UNDERSTOOD);
     reply.setContent("( UnexpectedContent (expected ping))");
   }
}
```


Teil 2 - Kommunikation in Agentensystemen

- Kommunikationssprachen für Agenten
- Sprechakte
- Interaktionsprotokolle
- Sprachen für die Wissensrepräsentation (Content Languages)
- Ontologien



Kommunikationssprachen für Agenten



- Bietet Agenten die Möglichkeit Informationen untereinander auszutauschen.
- Legt die Struktur einer Nachricht fest.

Beispiele: FIPA ACL, KQML

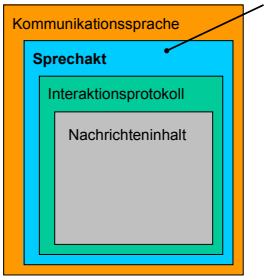
Kommunikationssprachen für Agenten

• *performative*	Typ der Nachricht.	**Typ**
• *sender*	Absender(agent).	**Teilnehmer**
• *receiver*	Empfänger(agent).	
• *reply-to*	Empfänger(agent).	
• *content*	Inhalt der Nachricht.	**Inhaltsbeschreibung**
• *language*	Sprache für den Nachrichteninhalt.	
• *encoding*	Codierung des Nachrichteninhalts.	
• *ontology*	Ontologie für die Wissensinterpretation.	
• *protocol*	Interaktionsprotokoll.	**Interaktionskontrolle**
• *conversation-id*	Kennung der Interaktion.	
• *reply-with*	Kennung für eine Antwort.	
• *in-reply-to*	Bezug auf eine vorhergehende Nachricht.	
• *reply-by*	Zeitangabe bis wann eine Antwort gewünscht wird.	

Kommunikationssprachen für Agenten
Aufbau einer Nachricht in ACL

```
(AGREE
 :sender (agent-identifier
          :name ams@igel:7778/JADE
          :addresses (sequence http://igel:7779/acc )
         )
 :receiver (set (agent-identifier
                 :name sniffer0@igel:7778/JADE
                 :addresses (sequence http://igel:7779/acc )
                )
           )
 :content
 :reply-with sniffer0@igel:7778/JADE1035972898799
 :language FIPA-SL0
 :ontology JADE-Agent-Management
 :protocol fipa-request
 :conversation-id C1220976_1035972898740
)
```


Sprechakte



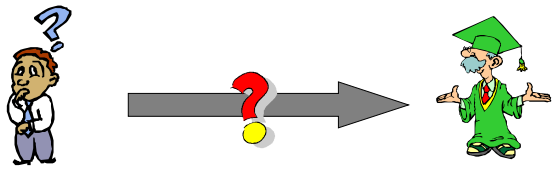
- Stellt eine Aussageform innerhalb einer Sprache dar.
- Semantische und syntaktische Einschränkung des Nachrichteninhalts.

Beispiele:

• *REQUEST*	Aufforderung
• *QUERY*	Anfrage
• *INFORM*	Benachrichtigung
• *PROPOSE*	Angebot
• *AGREE*	Annahme eines Angebots
• *REFUSE*	Ablehnung eines Angebots

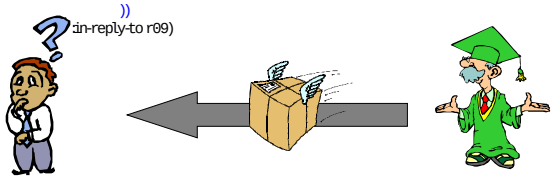
Sprechakte
Beispiel 1: *Anfrage - query-if*

```
(query-if
   :sender (agent-identifier :name i)
   :receiver (set (agent-identitifier :name j))
   :content
     ((registered (server d1) (agent j)))
   :reply-with r09
   ...)
```



Sprechakte
Beispiel 2: *Benachrichtigung - inform*

```
(inform
   :sender (agent-identifier :name j)
   :receiver (set (agent-identifier :name i))
   :content
          ((not (registered (server d1)
                                (agent j)
                )
          ))
   :in-reply-to r09)
```



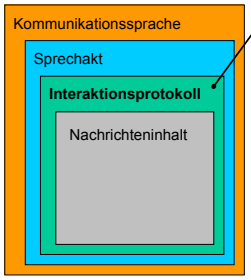
Sprechakte
FIPA ACL Sprechakte (1)

- *accept proposal* Annahme eines existierenden Angebots.
- *agree* Zustimmung zu einer vorgeschlagenen Aktion.
- *cancel* Absage einer schon akzeptierten Aktion.
- *cfp (call for proposal)* Anforderung eines Angebots.
- *confirm* Bestätigung einer Aussage (positiv).
- *disconfirm* Bestätigung einer Aussage (negativ).
- *failure* Fehlerhafte Ausführung einer Aktion.
- *inform* Information (positive Aussage).
- *inform-if* Information (bedingte Aussage).
- *inform-ref* Information (referenzierte Aussage).
- *not-understood* Nachricht nicht verstanden.
- *propagate* Nachricht an mehrere Agenten verteilen.

Sprechakte
FIPA ACL Sprechakte (2)

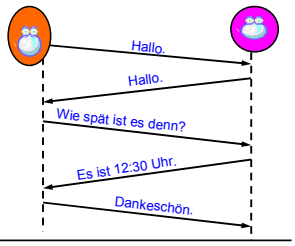
- *propose* Erstellung eines Angebots.
- *proxy* Weiterleiten von Nachrichten.
- *query-if* Anfrage ob eine Aussage wahr ist.
- *query-ref* Anfrage ob eine referenzierte Aussage wahr ist.
- *reject proposal* Zurückweisung eines Angebots.
- *request* Anforderung einer Aktionsausführung.
- *request-when* Anforderung einer Aktionsausführung, wenn eine Bedingung erfüllt wird (temporal).
- *request-whenever* Anforderung einer Aktionsausführung, immer wenn eine Bedingung erfüllt wird (temporal).
- *subscribe* Interesse an bestimmten Ereignissen anmelden.

Interaktionsprotokolle

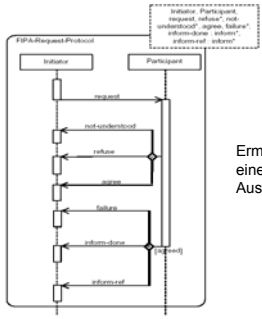


- Legt eine Reihenfolge für Sprechakte fest, s.d. Interaktionsmuster wiederholt werden können.

Beispiel: Frage nach der aktuellen Uhrzeit



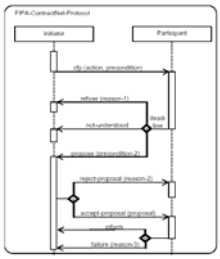
Interaktionsprotokolle
Beispiel 1: *FIPA Request Interaction Protocol*

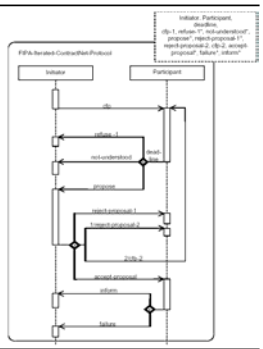


Ermöglicht einem Agenten *(Initiator)* von einem anderen Agenten *(Participant)* die Ausführung einer Aktion zu fordern.

Interaktionsprotokolle

Beispiel 2: *FIPA (Iterated) Contract Net Protocol*

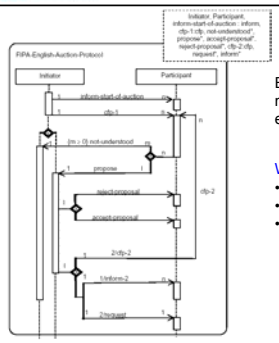




Ermöglicht es einem Agenten *(Initiator)* mit anderen Agenten *(Participants)* einen Vertrag abzuschließen.

Interaktionsprotokolle

Beispiel 3: *FIPA English Auction Protocol*

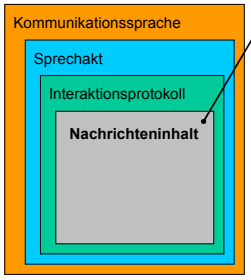


Ermöglicht es einem Agenten *(Initiator)* mit anderen Agenten *(Participants)* eine Auktion durchzuführen.

Weitere Protokolle:
- *FIPA Dutch Auction Protocol*
- *FIPA Brokering Protokol*
- *FIPA Recruiting Protokol*

FIPA Spezifikationen der Protokolle:
- http://www.fipa.org/

Nachrichteninhalt / Information



- Enthält die Information der Nachricht.
- Ist mit Hilfe einer Wissensrepräsentationssprache formuliert (KIF, CCL, SL).

Beispiel: *FIPA-SL*

```
(propose
:sender (agent-identifier :name j)
:receiver (set (agent-identifier :name i))
:content
  ((action j (sell plum 50))
   (= (any ?x
        (and (= (price plum) ?x)
             (< ?x 10)
        )
      )
     5)
   )
:ontology fruit-market
:in-reply-to proposal2
:language FIPA-SL)
```


Nachrichteninhalt - syntaktische und semantische Formulierung

FIPA SL - Syntax

FIPA *Semantic Language (SL)*
- Stellt eine konkrete Syntax und Semantik bereit, die in Verbindung mit der Agentenkommunikationssprache *FIPA ACL* verwendet werden kann.
- Syntax von FIPA SL

```
Content = "(" ContentExpression+ ")".
ContentExpression = IdentifyingExpression
                  | ActionExpression
                  | Proposition.
Proposition = Wff.
Wff = AtomicFormula
                  | "(" UnaryLogicalOp Wff ")"
                  | "(" BinaryLogicalOp Wff Wff ")"
                  | "(" Quantifier Variable Wff ")"
                  | "(" ModalOp Agent Wff ")"
                  | "(" ActionOp ActionExpression ")"
                  | "(" ActionOp ActionExpression Wff ")".
UnaryLogicalOp = "not".
BinaryLogicalOp = "and"
                  | "or"
                  | "implies"
                  | "equiv".
```


Nachrichteninhalt - syntaktische und semantische Formulierung

FIPA SL - Syntax

```
AtomicFormula = PropositionSymbol
                  | "(" BinaryTermOp Term Term ")"
                  | "(" PredicateSymbol Term* ")"
                  | "true"
                  | "false".
BinaryTermOp = "="
                  | "<"
                  | ">"
                  | ">="
                  | "<"
                  | "=<"
                  | "member"
                  | "contains"
                  | "result".
Quantifier = "forall"
                  | "exists".
ModalOp = "B"
                  | "U"
                  | "PG"
                  | "I".
ActionOp = "feasible"
                  | "done".
```


Nachrichteninhalt - syntaktische und semantische Formulierung

FIPA SL - Syntax

```
Term = Variable
                  | FunctionalTerm
                  | ActionExpression
                  | IdentifyingExpression
                  | Constant
                  | Sequence
                  | Set.
IdentifyingExpression = "(" ReferentialOperator Term Wff ")".
ReferentialOperator = "iota"
                  | "any"
                  | "all".
FunctionalTerm = "(" "cons" Term Term ")"
                  | "(" "first" Term ")"
                  | "(" "rest" Term ")"
                  | "(" "nth" Term Term ")"
                  | "(" "append" Term Term ")"
                  | "(" "union" Term Term ")"
                  | "(" "intersection" Term Term ")"
                  | "(" "difference" Term Term ")"
                  | "(" ArithmeticOp Term Term ")"
                  | "(" FunctionSymbol Term* ")"
                  | "(" FunctionSymbol Parameter* ")".
```

Nachrichteninhalt - syntaktische und semantische Formulierung
FIPA SL - Syntax

```
Constant = NumericalConstant
                    | String
                    | DateTime.
NumericalConstant = Integer
                    | Float.
Variable = VariableIdentifier.
ActionExpression = "(" "action" Agent Term ")"
                    | "(" "|" ActionExpression ActionExpression ")"
                    | "(" "," ActionExpression ActionExpression ")".
PropositionSymbol = String.
PredicateSymbol = String.
FunctionSymbol = String.
Agent = Term.
Sequence = "(" "sequence" Term* ")".
Set = "(" "set" Term* ")".
Parameter = ParameterName ParameterValue.
ParameterValue = Term.
ArithmeticOp = "+"
                    | "-"
                    | "*"
                    | "/"
                    | "%".
```


Nachrichteninhalt - syntaktische und semantische Formulierung
FIPA SL - Beispiel

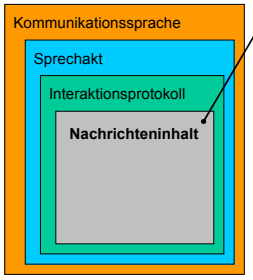
- Anfrage mit dem Allquantor *all*.

```
(query-ref
 :sender (agent-identifier :name B)
 :receiver (set (agent-identifier :name A))
 :content
     ((all ?x (q ?x c)))
 :language FIPA-SL
 :reply-with query2)
```

- Antwort mit einer leeren Menge.

```
(inform
 :sender (agent-identifier :name A)
 :receiver (set (agent-identifier :name B))
 :content
     ((= (all ?x (q ?x c))(set)))
 :language FIPA-SL
 :in-reply-to query2)
```


Nachrichteninhalt - Ontologien



- Dienen der Interpretation der Begriffe.
- Werden für bestimmte Domänen definiert.

Beispiel: *fruit-market*

```
(propose
:sender (agent-identifier :name j)
:receiver (set (agent-identifier :name i))
:content
  ((action j (sell plum 50))
   (= (any ?x
        (and (= (price plum) ?x)
             (< ?x 10)
        )
      )
    5)
  )
:ontology fruit-market
:in-reply-to proposal2
:language FIPA-SL)
```


Nachrichteninhalt - Ontologien
Definition

Was ist eine Ontologie (aus unserer Sicht)?

- Eine Dokumentation einer bestimmten Weltsicht.
- Eine Konzeptualisierung.
- Definiert eine Menge von Termen - das Vokabular.
- Definiert die Interpretation der Terme.
- Definiert die Struktur der Statements in dem Modell.

Was ist mit Ontologie (für uns!) nicht gemeint?

- Die philosophische Interpretation: „Lehre vom Seienden“, ...

Nachrichteninhalt - Ontologien
Zweck

Wozu wollen wir Ontologien (in unserem Sinne) verwenden?

- Um Softwaresystemen das gleiche Verstehen einer Darstellung/eines Konzeptes zu ermöglichen.
- Um die Interoperabilität und die Wiederverwendbarkeit von Softwarekomponenten, Modellen und Spezifikationen zu verbessern.
- Um das Domänenwissen von dem Anwendungs-/Lösungswissen zu trennen.

Nachrichteninhalt - Ontologien
Typen von Ontologien

Domain-Ontologien

- Stellen Konzeptualisierungen und Gerüst für Wissensbasen bestimmter Domänen dar (z.B. Medizin, Öl-Produktion, Flugzeugbau usw.).
- Entsprechen dem Standard ISO 10303 - Standard of Product Data Modelling, Series 101-105. (Schiffsbau, Elektrotechnische Verkabelung und andere Bereiche)

Task-Ontologien

- Enthalten die Methode/das Vorgehen zur einer Aufgabenlösung.
- Sind für die Lösung einer bestimmten Aufgabe vorgesehen (z.B. propose-and-revise - Methode für Konfigurationsaufgaben).
- Stellt keine Struktur einer Wissensbasis dar!

Generische Ontologien

- Definieren allgemeine Konzepte, die in mehreren Domänen vorkommen (z.B. Ontologien für Ereignisse, Zustände, Zeit, geometrische und topologisch Darstellungen).

Inspirative Methode

- Basiert auf den Antworten zur Frage, warum die Ontologie benötigt wird.
- Führt zu einer Ontologie, die eine individuelle Sicht einer Domäne enthält.

Induktive Methode

- Baut auf der Analyse von Szenarien aus der entsprechenden Domäne, auf.
- Führt zu einer Ontologie, die für die meisten Szenarien in einer Domäne verwendet werden kann.

Deduktive Methode

- Baut auf allgemeinen Prinzipien für ein bestimmtes Szenario auf.
- Anpassung der Prinzipien zur Integration in die Ontologie

Synthetische Methode

- Es werden mehrere „kleine“ Ontologien erstellt, s.d. keine eine Teilmenge einer anderen ist (Überschneidungen sind möglich).
- Aus den „kleinen“ Ontologien wird eine „große“ Ontologie erstellt, in welche alle vorhandenen Konzepte einfließen.

Kollaborative Methode

- Die Sichtweise von allen Entwicklern der Ontologie werden berücksichtigt.
- Die Ontologie wird schrittweise verfeinert, wobei z.B. in jedem Schritt die Änderung eines Entwicklers durch die restlichen Entwickler angepasst wird.
- Führt meistens zu einer Ontologie mit größerer Akzeptanz.

Bestimmung des Zwecks der Ontologie

- Wofür soll die Ontologie verwendet werden?
- Wie soll die Ontologie verwendet werden?

Bestimmung des Formalitätsgrades der Ontologie

- Je mehr eine Ontologie für automatisierte Aufgaben vorgesehen ist, desto formaler sollte sie sein.

Abgrenzung der Ontologie

- Hilfsmittel - beispielhafte Anwendungsszenarien.
- Bestimmung des Aufgabenbereichs - eine Menge von Termen und Konzepten werden in die Ontologie aufgenommen.

Erstellung von formalen Definitionen und Axiomen

- Verwendung einer formalen Sprache.
- Spezifikation und Implementation der Ontologie.
- Verbreitete Sprache für diesen Zweck: Knowledge Interchange Format.

Formale Bewertung

- Kann aus den Axiomen und einer Menge von Instanzen der Objekte und Relationen der Ontologie ein Satz erster Ordnung hergeleitet werden?
- Prüfung, ob Sätze erster Ordnung, die nur Terme der Ontologie verwenden, mit den Axiomen der Ontologie konsistent sind.

Existierende Werkzeuge:

- *Ontolingua*, *Chimæra*, *OILed*, *Protégé 2000*

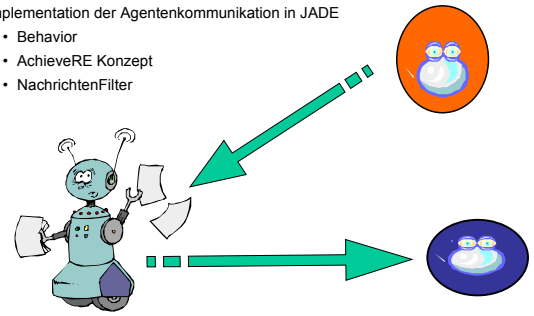
Funktionalität der Werkzeuge:

- Unterstützung beim Design von Ontologien:
  - Modellierung, Implementierung, Test.
- Unterstützung bei der Pflege von Ontologien:
  - Zusammenführung, Extraktion, Versionsverwaltung
- Schnittstellen zu anderen Systemen:
  - Agentensysteme, Expertensysteme, logische Programmiersprachen.
  - Export in XML oder in relationale Datenbanken mit SQL-Schnittstelle.

Vorteile:

- Schnelleres Design von Ontologien möglich.

- Implementation der Agentenkommunikation in JADE
  - Behavior
  - AchieveRE Konzept
  - NachrichtenFilter

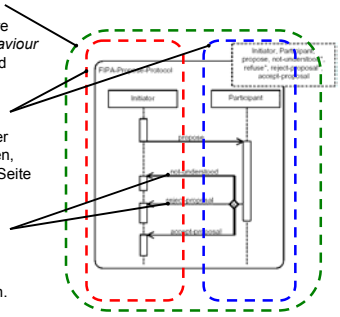


JADE
Implementierung der Kommunikation in JADE

Interaktionsprotokoll
- wird durch zwei oder mehrere Automaten umgesetzt - *Behaviour*
- es wird zwischen *Initiator* und *Responder* unterschieden.

Behaviour
- Verkapselt eine Aufgabe oder eine Teilaufgabe des Agenten,
- Automat zur Kontrolle einer Seite des Interaktionsprotokolls.

Sprechakt
- Kann mit Hilfe von bereitgestellten Methoden in JADE einfach erstellt werden.



JADE
Methoden zur Erstellung von Sprechakten

```
(AGREE
 :sender (ag
          n
          :
          )
 :receiver (
           )
 :content
 :reply-with
 :language
 :ontology
 :protocol
 :conversati
 )
```

setPerformative, getPerformative
setSender, getSender

addReceiver, getReceiver, removeReceiver,
clearAllReceiver, getAllReceiver

setContent, getContent,
setContentObject, getContentObject
setEncoding, getEncoding

setReplyWith, getReplyWith
setLanguage, getLanguage
setOntology, getOntology
setProtocol, getProtocol
setConversationId, getConversationId

JADE
Erstellung eines Sprechaktes

Erstellung einer Anforderung - 1

```
try {
  ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
  msg.setLanguage(codec.getName());
  msg.setOntology(NegotiationManagementOntology.ONTOLOGY_NAME);
  msg.addReceiver(a);
  msg.setSender(getAID());
  msg.setProtocol(FIPAProtocolNames.FIPA_REQUEST);

  PolicyAgents.negotiation.ontologies.management.negotiate i =
    new PolicyAgents.negotiation.ontologies.management.negotiate();
  i.setWeightVolume(weightVolume);
  i.setNegotiationService(getAID());
  i.setPreferences(adminPreferenceToManagement(prefList));
  jade.content.onto.basic.Action contentAction =
     new jade.content.onto.basic.Action();
  contentAction.setAction(i);
  contentAction.setActor(getAID());
```

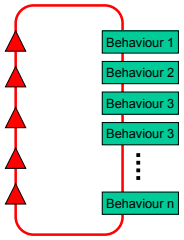

JADE
Erstellung eines Sprechaktes

Erstellung einer Anforderung - 2

```
  ContentManager manager = getContentManager();
  manager.setValidationMode(false);
  manager.fillContent(msg, (ContentElement) contentAction);

  myRequestBehaviour initializeBehaviour =
    new myRequestBehaviour(msg);
  addBehaviour(initializeBehaviour);
}
catch (CodecException fe) {
  System.err.println("initializeAgent: Codec exception" +
                                    fe.getMessage());
  fe.printStackTrace();
}
catch (OntologyException oe) {
  System.err.println("initializeAgent: Onto exception" +
                                    oe.getMessage());
  oe.printStackTrace();
}
```


JADE
Behaviours

Was ist ein Behaviour?
- Verkapselt eine Aufgabe oder eine Teilaufgabe des Agenten.
- Wird mit Hilfe eines Automaten implementiert.
- Wird mit den anderen Behaviours des Agenten im round-robin Verfahren abgearbeitet.



JADE
Methoden für Behaviours

Innerhalb eines Agenten
- addBehaviour()
- removeBehavior()

Innerhalb eines Behaviours
- action()
  Enthält den Code für die zu erledigende Aufgabe.
- done()
- onStart()
  Enthält den Code von Aktionen die vor dem Start ausgeführt werden müssen.
- onEnd()
  Enthält den Code von Aktionen die vor dem Ende ausgeführt werden müssen.
- block()
  Blockiert ein Behaviour.
- restart()
  Startet ein blockiertes Behaviour wieder.
- reset()
  Startet ein blockiertes Behaviour von vorne.

SimpleBehavior

- *OneShotBehavior* - *done()* liefert immer wahr,
  - *SenderBehavior* - sendet eine Nachricht,

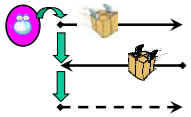
  - *ReceiverBehavior* - empfängt eine Nachricht,

  - *WakerBehavior* - führt ein SubBehavior oder ein Aktion nach einem vorgegebenem Zeitintervall aus.

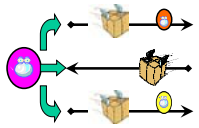
- *CyclicBehavior* - *done()* liefert immer falsch.

CompositeBehavior

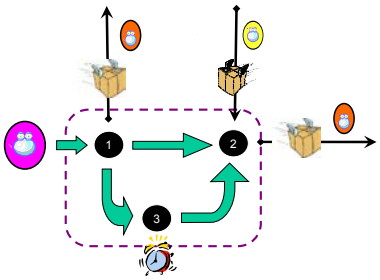
- *SequentialBehavior* - die registrierten SubBehaviors oder Aktionen werden sequentiell ausgeführt,



- *ParalelBehavior* - die registrierten SubBehaviors oder Aktionen werden parallel ausgeführt,



- *FSMBehavior* - die SubBehaviors oder Aktionen werden für die Zustände des Automaten registriert und von diesem dann ausgeführt.



Erstellung eines *PingBehaviors* - 1

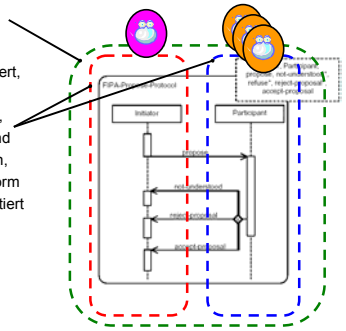
```
public class PingAgent extends Agent {
  class WaitPingAndReplyBehaviour extends SimpleBehaviour {
    private boolean finished = false;
    public WaitPingAndReplyBehaviour(Agent a) {
      super(a);
    }

    public void action() {
      ACLMessage msg = blockingReceive();
      if(msg != null){
          if(msg.getPerformative() == ACLMessage.NOT_UNDERSTOOD){
          //received a NOT-UNDERSTOOD message
            ...
      } else{
        //received a QUERY_REF with correct content.
      }
      else{
        //received a QUERY_REF with uncorrect content.
      }
    }
```


Erstellung eines *PingBehaviours* - 2

```
    public boolean done() {
      return finished;
    }
  } //End class WaitPingAndReplyBehaviour

    protected void setup() {
    /** Registration with the DF */
    ...
      try {
        DFService.register(this,dfd);
      } catch (FIPAException e) {
        System.err.println(getLocalName() +
              " registration with DF unsucceeded. Reason: "+e.getMessage());
        doDelete();
      }

    WaitPingAndReplyBehaviour PingBehaviour =
                              new WaitPingAndReplyBehaviour(this);
    addBehaviour(PingBehaviour);
    }
  } //end class PingAgent
```


Interaktionsprotokoll

- wird mit Hilfe von zwei oder mehreren Automaten realisiert,
- jeder Automat stellt einen Kommunikationspartner dar,
- es wird zwischen *Initiator* und *Responder(s)* unterschieden,
- die Automaten werden in Form eines Behaviours implementiert (*FSMBehaviour*).

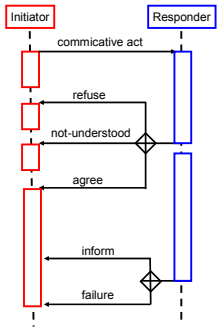


AchieveRE

- Implementiert ein generisches Interaktionsmuster für mehrere Interaktionsprotokolle (*query, propose, request, request-when, brokering, recruiting*),
- stellt Methoden zum Debuggen bereit,
- ist einfach zu verwenden.

Z.B.

```
ACLMessage request =
    new ACLMessage(ACLMessage.REQUEST);
request.setProtocol(
    FIPAProtocolNames.FIPA_REQUEST);
AchieveREInitiator(Agent myAgent,
    ACLMessage request)
```



Bereitgestellte Methoden des *AchieveREInitiator*:

- handleAllResponses();
- handleAllResultNotifications();
- handleAgree();
- handleRefuse();
- handleFailure();
- handleInform();
- handleNotUnderstood();
- prepareRequests();
- reset();

AchieveREInitiator Debugging

Bereitgestellte Methoden für den *AchieveREInitiator*:

- getDataStore().get(ALL_RESPONSES_KEY);
  gibt einen Vector mit der ersten Welle von Antworten zurück,
- getDataStore().get(ALL_RESULT_NOTIFICATIONS_KEY);
  gibt einen Vector mit der zweiten Welle von Antworten zurück,
- getDataStore().get(ALL_REQUESTS_KEY);
  gibt einen Vector mit allen gesendeten Nachrichten zurück,
- getDataStore().get(REQUEST_KEY);
  gibt die Initialisierungs-Nachricht zurück,
- getDataStore().get(REPLY_KEY);
  gibt die letzte empfangene Antwort zurück.

Bereitgestellte Methoden des *AchieveREResponder*:

- createMessageTemplate();
- prepareResponse();
- registerPrepareResponse();
- prepareResultNotification();
- registerPrepareResultNotification();
- reset();

AchieveRE Debugging

Bereitgestellte Methoden für den *AchieveREResponder*:

- getDataStore().get(REQUEST_KEY);
  gibt die Initialisierungs-Nachricht zurück, die von dem *Initiator* gesendet wurde.
- getDataStore().get(RESPONSE_KEY);
  gibt die Nachricht zurück, die als Antwort an den Initiator gesendet wurde,
- getDataStore().get(RESULT_NOTIFICATION_KEY);
  gibt die Nachricht zurück, die als Benachrichtigung an den Initiator gesendet wurde *(inform, inform-ref, failure)*.


```
// Create a new ACL Message
ACLMessage request = new ACLMessage(ACLMessage.REQUEST);
request.setProtocol(FIPAProtocolNames.FIPA_REQUEST);
request.addReceiver(new AID("receiver", AID.ISLOCALNAME));

// Create a new AchieveREInitiator
myAgent.addBehaviour(
  new AchieveREInitiator(myAgent, request) {
        protected void handleInform(ACLMessage inform) {
              System.out.println(
              "Protocol finished. Rational Effect achieved." +
               "Received the following message: " + inform);
        }
  }
);
```

JADE
Erstellung eines *Request Interaction Protocol - Responders*

```
// Create a new ACL Message
MessageTemplate mt = AchieveREResponder.createMessageTemplate(
                                        FIPAProtocolNames.FIPAREQUEST);
// Create a new AchieveREResponder
myAgent.addBehaviour(
  new AchieveREResponder(myAgent, mt) {
      protected ACLMessage prepareResultNotification(
                                    ACLMessage request, ACLMessage response)
      {
          System.out.println("Responder has received the following" +
                                    "message: " + request);
          ACLMessage informDone = request.createReply();
          informDone.setPerformative(ACLMessage.INFORM);
          informDone.setContent("inform done");
          return informDone;
      }
});
```


JADE
Erstellung eines Nachrichtenfilters

Die *MessageTemplate* Klasse

- Ermöglicht das Erstellen von Mustern für Nachrichtenfilter.
- Bereitgestellte Methoden zum „matchen“:
  - `MatchAll();`
  - `MatchContent();`
  - `MatchConversationId();`
  - `MatchEncoding();`
  - `MatchReplyTo(); MatchReplyBy();`
    `MatchReplyWith(); MatchReplyByDate();`
  - `MatchLanguage(),`
  - `MatchOntology();`
  - `MatchSender();`
  - `MatchReceiver();`

  - and();
  - or();
  - not();

JADE
Erstellung eines Nachrichtenfilters - Beispiel

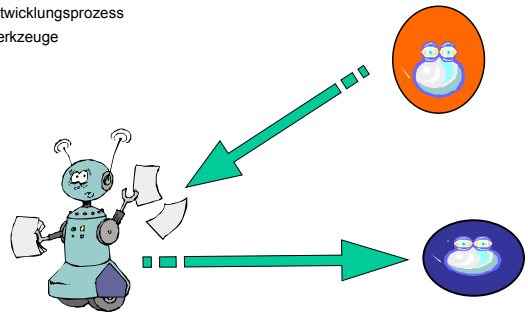
Erstellung eines Filters für Nachrichten:

- des Typs *QUERY*,
- mit der verwendeten Ontologie *OntHoS*,
- mit der Wissensrepräsentationssprache „FIPA-SL0“.

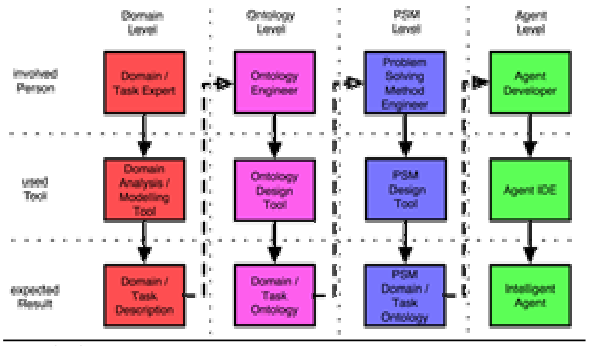
```
MessageTemplate m1 =
        MessageTemplate.MatchPerformative(ACLMessage.QUERY);
MessageTemplate m2 =
        MessageTemplate.MatchLanguage("FIPA-SL0");
MessageTemplate m3 =
        MessageTemplate.MatchOntology("OntHoS");
MessageTemplate m1andm2 =
        MessageTemplate.and(m1, m2);
MessageTemplate m1andm2_and_m3 =
        MessageTemplate.and(m1andm2, m3);
```


Teil 4 - Entwicklung von Agenten-basierten Lösungen

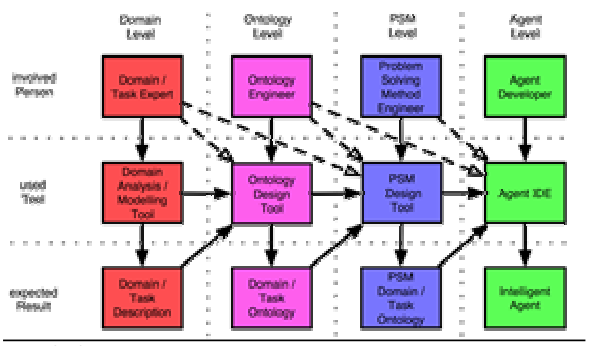
- Entwicklungsprozess
- Werkzeuge



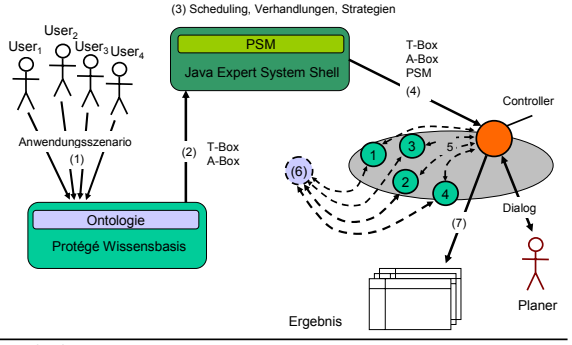
Entwicklungsprozess in JADE - aktuell



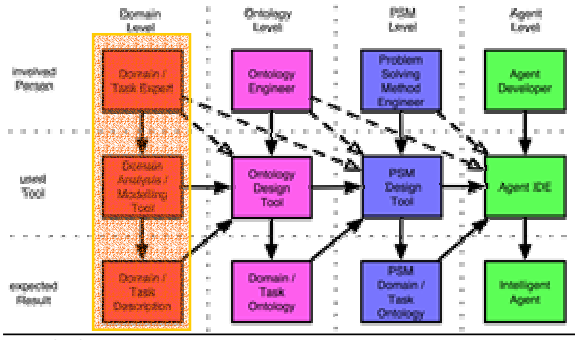
Entwicklungsprozess in JADE - Vision



Entwicklungsprozess



Entwicklungsprozess - Domain Level



Entwicklungsprozess - Domain Level

- Fragestellungen
  - Welches Wissen, welche Erkenntnisse sind für die Problemlösung wichtig?
    - Domänenexperte
    - Analytiker
  - Wie wird das Wissen oder die Erkentnisse modelliert?
  - Welche Werkzeuge werden verwendet?
  - Wie kann das Ergebnis - das Modell - weiter verarbeitet werden?

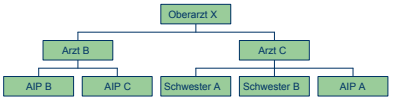
Was ist Wissen?

- Kenntnis von Fakten über Objekte einer Welt.
  - Beispiele:
    - Ein Auto hat vier Räder.
    - Ein Rad ist rund.
- Kenntnis von Zusammenhängen zwischen Ereignissen einer Welt.
  - Beispiel:
    - Wenn die Sonne scheint, ist es warm.
- Kenntnis wie Handlungen in einer Welt ausgeführt werden.
  - Beispiele:
    - Addition von natürlichen, reelen oder komplexen Zahlen.
    - Führung eines Bootes, Lenkung eines Fahrrads, Durchführung einer OP.
- Kenntnis wie Wissen angeeignet, verwaltet und angewendet wird.
  - Beispiel:
    - Neues Wissen kann durch das Lesen eines Buches erworben werden, d.h. die in dem Buch beschriebenen Kenntnisse werden (teilweise) übernommen.

Wissensrepräsentation

	a	b	c
a	1	2	1
b	0	1	0
c	1	1	6

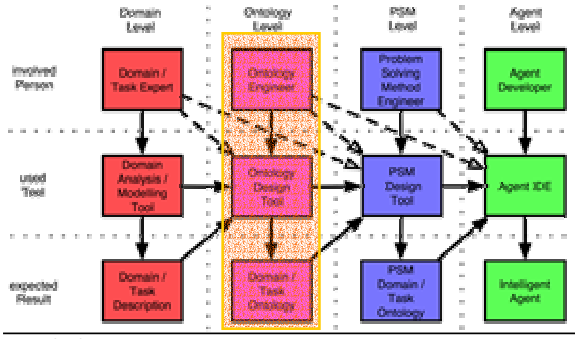
Dienst-Hierarchie



$$\sum_{i=0}^{k-1} 2^i = 2^k - 1$$

$$\forall x \left[Person(x) \supset x = jane \vee x = john \vee x = jim \ldots \right]$$

Entwicklungsprozess - Ontology Level



Konstruktion einer Ontologien - 1

Bestimmung des Zwecks der Ontologie

- Wofür soll die Ontologie verwendet werden?
- Wie soll die Ontologie verwendet werden?

Bestimmung des Formalitätsgrades der Ontologie

- Je mehr eine Ontologie für automatisierte Aufgaben vorgesehen ist, desto formaler sollte sie sein.

Abgrenzung der Ontologie

- Hilfsmittel - beispielhafte Anwendungsszenarien.
- Bestimmung des Aufgabenbereichs - eine Menge von Termen und Konzepten werden in die Ontologie aufgenommen.

Konstruktion einer Ontologien - 2

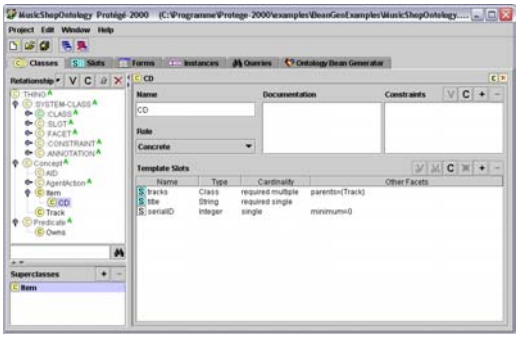
Erstellung von formalen Definitionen und Axiomen

- Verwendung einer formalen Sprache.
- Spezifikation und Implementation der Ontologie.
- Verbreitete Sprache für diesen Zweck: Knowledge Interchange Format.

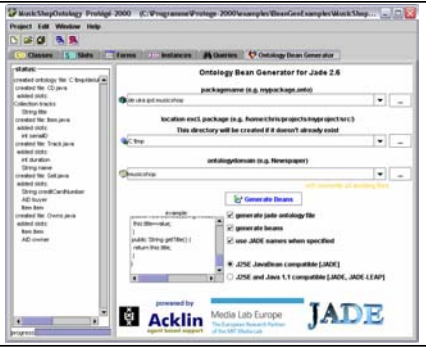
Formale Bewertung

- Kann aus den Axiomen und einer Menge von Instanzen der Objekte und Relationen der Ontologie ein Satz erster Ordnung hergeleitet werden?
- Prüfung, ob Sätze erster Ordnung, die nur Terme der Ontologie verwenden, mit den Axiomen der Ontologie konsistent sind.

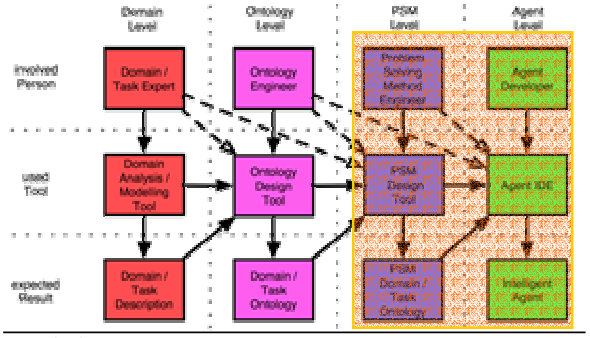
Modellierung der Ontologie und Wissensakquisition



Export des Szenarios - BeanGenerator/JessAgentTab



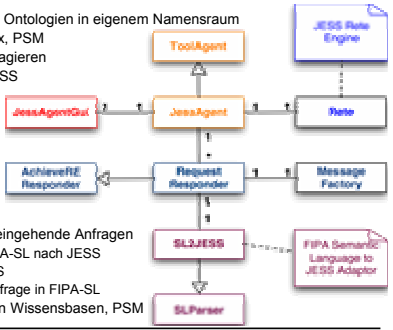
Entwicklungsprozess - PSM Level



JessAgent

Eigenschaften

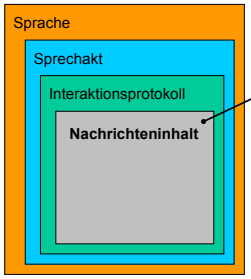
- Verwendung von mehreren Ontologien in eigenem Namensraum
- Trennung von A-Box, T-Box, PSM
- Regelbasiertes Agieren/Reagieren
- Benutzerschnittstelle zu JESS
- Unterstützt FIPA-konforme Kommunikation



Funktionsweise

- Laden der Wissenbasen
- Laden der PSM
- Verhalten des Agenten
  - reaktiv - basierend auf eingehende Anfragen
    - Übersetzung von FIPA-SL nach JESS
    - Interpretation in JESS
    - Beantwortung der Anfrage in FIPA-SL
  - aktiv - basierend auf den Wissensbasen, PSM

JessAgent - Nachrichtenverarbeitung
Empfang einer Nachricht




```
(propose
:sender (agent-identifier :name j)
:receiver (set (agent-identifier :name i))
:content
 ((action (agent-identifier
  :name JA@igel:1100/JADE
  :adresses (sequence
            http://igel:1099/acc))
            (addTask :aTask (Task
              :name OP                    :type
      operation                  :starting_time
              (Absolute_Timepoint
      ... ... ... ...                   :duration
      5))))
:ontology scheduler-ontology
:language FIPA-SL)
```

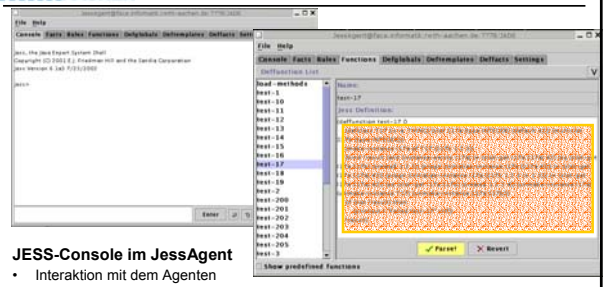

JessAgent - Nachrichtenverarbeitung
SL-Parser erstellt den Syntaxbaum der SL-Nachricht

```
Content
 ContentExpression
  ActionExpression
   Action
    Agent
     TermOrIE
     Term
      FunctionalTerm
       FunctionSymbol: agent-identifier
        Parameter
         ParameterName: :name
         ParameterValue
          TermOrIE
           Term
            Constant
             String: JA@igel:1100/JADE

                ... ... ... ...
```


JessAgent - Nachrichtenverarbeitung
SL2Jess Adapter übersetzt die Nachricht von SL nach JESS

```
(addTask
 (assert (Task
  (name OP)
  (type operation)
  (starting_time
   (assert (Absolute_Timepoint
    (daytime_of_timepoint
     (assert (DayTime
      (hour 12)
      (second 0)
      (minute 0)
     )

                ... ... ... ...

... ... ... ...
```

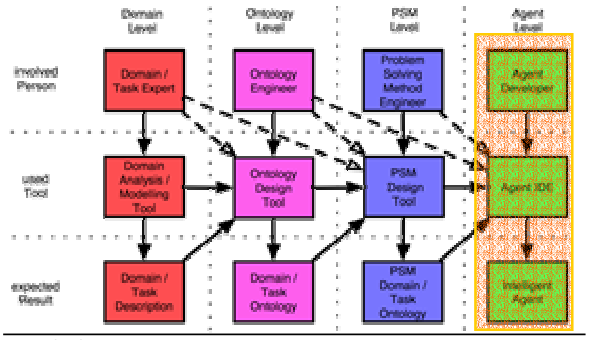

JessAgent - Nachrichtenverarbeitung
Auswertung des Nachrichteninhalts in JESS



JESS-Console im JessAgent

- Interaktion mit dem Agenten
- Zugriff auf alle Fakten, Regeln, Funktionien
- Ablaufverfolgung der Regelauswertung
- Änderung der PSM

Entwicklungsprozess - Agent Level



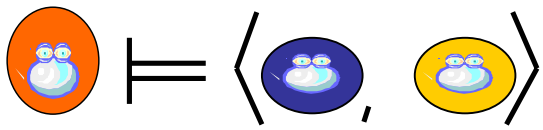
Einsatz des Agenten

JESS-Console im JessAgent

- Verwendung der JessAgent's ohne Console

Teil 5 - Regelbasierte Systeme

- Definition
- Expertensysteme
- Verwendung in MAS



Regeln

- Was sind Regeln?
  - Anweisungen für bestimmte Situationen.
  - Bedingungen des Typs
    wenn <*linke Seite*>
    dann <*rechte Seite*>
  - Bezeichnungen für die *linke Seite* der Bedingung:
    - LHS - *left-hand side*
    - Prädikat
    - Premisse
  - Bezeichnungen für die *rechte Seite* der Bedingung:
    - RHS - *right-hand side*
    - Aktion
    - Schlussfolgerung
  - LHS, RHS bestehen aus Fakten oder andere Regeln verwendet.

Regelbasierte Systeme

- Definition
  - Ein *regelbasiertes System (rule engine)* ist ein System, das aus Regeln die möglichen Schlussfolgerungen / Aktionen herleitet.
    if <LHS_1>
    then <RHS_1>
    if <LHS_2>
    then <RHS_2>
    if <LHS_3>
    then <RHS_3>
  - Regelbasierte Systeme
    - verfügen über Mechanismen um Regeln der Form
      if <*LHS*>
      then <*RHS*>
      zu folgen
    - enthalten kein zusätzliches Wissen.

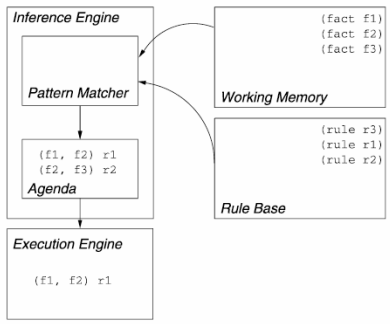
Expertensysteme

- Definition
  - *Expertensysteme (expert systems)*
    - sind *regelbasierte Systeme*
    - nehmen menschliches Wissen aus einem Anwendungsbereich in Form von Regeln und Fakten auf
  - Anwendungen
    - in Bereichen in denen auf Daten/Datenströme eine große Anzahl von Regeln angewendet wird:
    - Diagnosesysteme in der Medizin
    - Planung, Scheduling, Koordination
    - Ingenieurwissenschaften
    - eCommerce

Aufbau eines regelbasierten Systems (1)

- Komponenten eines regelbasierten Systems
  - Regelspeicher (rulebase)
    - enthält alle Regeln des Systems
  - *Arbeitsspeicher*
    - enthält die Fakten auf die die Regeln angewendet werden
  - *Regel-Interpreter*
    - wendet die Regeln der Reihe nach auf den Inhalt des *Arbeitsspeicher* (Fakten) an
    - besteht aus
      - Pattern Matcher
        - markiert eine Regel als aktiv, wenn die LHS mit den Fakten aus dem *Arbeitspeicher* „matcht"
      - Agenda
        - enthält die markierten Regeln, deren RHS ausgeführt wird (feuern)
      - Execution Engine
        - Laufzeitsystem zur Ausführung der Aktionen (RHS)

Aufbau eines regelbasierten Systems (2)



Auswertung der Regeln (1)

- Forward Chaining

  if <LHS>
  then <RHS>

  Matchen des <LHS> einer Regel führt zur Ausführung (Feuern) der <RHS> der gleichen Regel

  Suche des Zieles <RHS> ausgehend von der Ursache <LHS>

- Backward Chaining

  if <LHS>
  then <RHS>

  Teilweises Matchen des <LHS> einer Regel führt **nur dann** zur Ausführung (Feuern) der <RHS> der gleichen Regel **wenn** dies zum vollen Matchen der <LHS> führt.

  Suche der Ursache <LHS> ausgehend von dem Ziel <RHS>

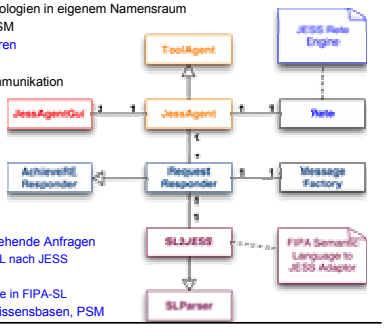
Auswertung der Regeln (2)

- Komplexität
  - einfacher Algorithmus: Testen alle Regeln bei jedem Durchlauf
    O((Regeln * Fakten)^m)
    m *- Mittlere Anzahl der Fakten pro Regel*
  - Verbesserter Algorithmus: Speicherung der Ergebnisse vorheriger Durchläufe
    O(Regeln x Fakten x m)

Verwendung von regelbasierten Systemen in MAS

Eigenschaften

- Verwendung von mehreren Ontologien in eigenem Namensraum
- Trennung von A Bx, T Bx, PSM
- Regelbasiertes Agieren/Reagieren
- Benutzerschnittstelle zu JESS
- Unterstützt FIPA konforme Kommunikation



Funktionsweise

- Laden der Wissenbasen
- Laden der PSM
- Verhalten des Agenten
  - reaktiv- basierend auf eingehende Anfragen
    - Übersetzung von FIPA-SL nach JESS
    - Interpretation in JESS
    - Beantwortung der Anfrage in FIPA-SL
  - aktiv- basierend auf den Wissensbasen, PSM

Programmierung von regelbasierten Systemen (1)

- Beispiel
  - if (< ?x 3)
    then (printout t "?x is less than three" crlf))
- Kommentare
  - ; This is a number
    1.2345
- Listen
  - (+ 3 2)
    (a b c)
    ("Hello, World")
    ()
    (1 2 3)
    (deftemplate foo (slot bar))

Programmierung von regelbasierten Systemen (2)

- Funktionsaufrufe
  - Jess> (+ (+ 2 3) (* 3 3))
    14
  - Jess> (printout t "The answer is " 42 "!" crlf)
    The answer is 42!
- Variablen
  - Jess> (bind ?a (+ 2 2))
    4
  - Jess> ?a
    4
  - Jess> (+ ?a 2)
    6

Programmierung von regelbasierten Systemen (4)

- Globale Variablen
  - Bezeichner fangen mit * an
  - Jess> (defglobal ?*x* = 3)
    TRUE
  - Jess> ?*x*
    3
  - (bind ?*x* 4)
    4
  - Jess> ?*x*
    4
  - Jess> (reset)
    TRUE
  - Jess> ?*x*
    3

Programmierung von regelbasierten Systemen (3)

- Funktionen, Regeln
  - Definition

```
(deffunction pottons-formula
    (?height ?a)
        (-(- ?height 100)(/ ?a(- ?height 100))))
        (defrule normal-weight
            (dimensions ?height ?a) =>
            (printout t "Normal: "
            (pottons-formula ?height ?a crlf)))
            (assert (dimensions 180 10))
        )
```


Ende

Vielen Dank für Ihre Aufmerksamkeit

```
catch(AgentDeathError ade) {
  // Do Nothing, since this is a killAgent from outside
}
```