S

CORPORATE TECHNOLOGY

Vorlesung Multiagentensysteme

7. ENTWICKLUNG AGENTENBASIERTER SOFTWARESYSTEME

S

CORPORATE TECHNOLOGY

Überblick

- In diesem Kapitel befassen wir uns mit der Frage, welche Sprachen und Tools heute zur Programmierung agentenbasierter Systeme verfügbar sind.
- Wir werden die Features heute verfügbarer Systeme am Beispiel der Agentenplattform Jade illustrieren

S

CORPORATE TECHNOLOGY

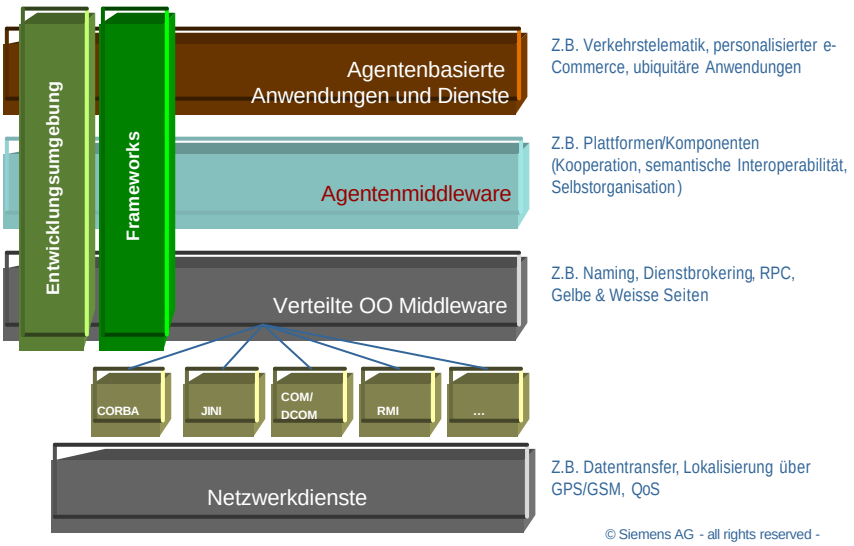
Plattformen für Agenten und Multiagentensysteme

- Ziel: Unterstützung des Designs und der Implementierung von Agenten und Multiagentensystemen durch:
  - Programmiersprachen
  - Ablaufumgebungen für Agentensysteme (Agentenplattformen)
  - Softwareentwicklungsmethodologien
  - Entwicklungs-, Simulations- und Testtools
- FIPA beschreibt eine Referenzarchitektur für Agentenplattformen im Sinne einer Menge von Schnittstellen, die eine „FIPA-kompatible Agentenplattform“ implementieren muss.
- In den letzten Jahren haben sich einige FIPA-kompatible Agentenplattformen als Open-Source-Plattformen etabliert, insbesondere Jade, Jade/Leap, FIPA-OS und COUGAAR.

S

CORPORATE TECHNOLOGY

Generische skalierbare Middleware



Z.B. Verkehrstelematik, personalisierter e-Commerce, ubiquitäre Anwendungen

Z.B. Plattformen/Komponenten (Kooperation, semantische Interoperabilität, Selbstorganisation)

Z.B. Naming, Dienstbrokering, RPC, Gelbe & Weisse Seiten

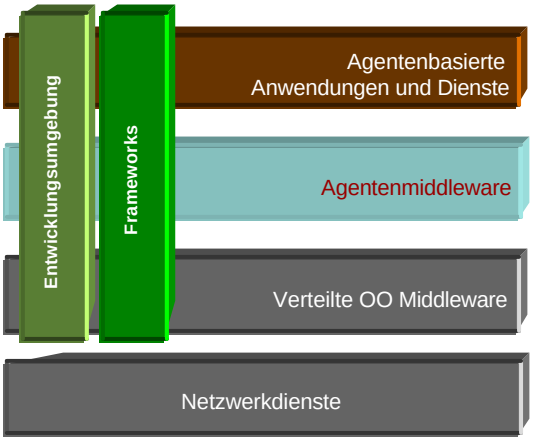
Z.B. Datentransfer, Lokalisierung über GPS/GSM, QoS

S

CORPORATE TECHNOLOGY

Generische skalierbare Middleware

Eine generische Agenteninfrastruktur ist Schlüssel, um nächste Generation von Diensten und Anwendungen zu bauen.



- Standardisierung
- Offen, skalierbar
- Generische Komponenten
- Methodologien und Tools
- Frei verfügbar
- Internet / WWW tauglich

S

CORPORATE TECHNOLOGY

Bedeutung von Standardisierung

- Schafft Interoperabilität von Agenten, die von unterschiedlichen Designern geschrieben wurden
- Reduziert Entwicklungsaufwand (keine proprietären Schnittstellen)
- Ermöglicht kontrollierte Änderungen von Schnittstellen
- Führt zu Code, der leichter verwaltet werden kann
- Erzeugt Communities (open source)
- Aber: Standardisierung ist mühsam und braucht viel Zeit

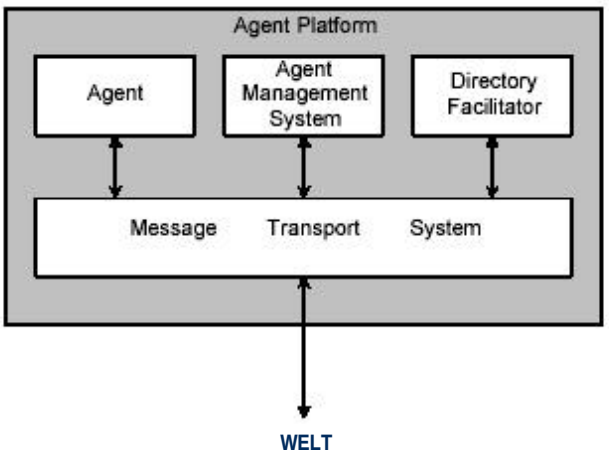
➔ **Nur die essentiellen Elemente eines Systems sollten standardisiert werden**

- FIPA: Fokus auf Kommunikation und Ablaufumgebung

S

CORPORATE TECHNOLOGY

FIPA Referenz-Architektur



S

CORPORATE TECHNOLOGY

Bestandteile des FIPA-Referenzmodells

- **Agent Management System (AMS)**
  - Agent, der Kontrolle über Zugriff auf und Benutzung der Agentenplattform inne hat
  - Erzeugen, Initialisieren, Ausführen, Migrieren von Agenten
- **Directory Facilitator (DF)**
  - "Gelbe-Seiten"-Mechanismus (Agent)
  - Angebotene Funktionen: register, deregister, modify, search
- **Message Transport System (auch Agent Communication Channel)**
  - Führt Transport von Nachrichten innerhalb der Plattform und mit anderen Plattformen durch

Kommunikation mit AMS und DF

- AMS und DF sind wiederum FIPA-kompatible Agenten
- AMS und DF bieten Kommunikationsschnittstelle basierend auf:
  - Logische Wissensrepräsentationssprache FIPA-SL0
  - FIPA Agent Management Ontologie
  - FIPA Request Interaktionsprotokoll

S

Beispiel: Jade Agenten-Plattform

- Java-basierte FIPA-kompatible Plattform
  - Implementiert FIPA Referenzarchitektur
    - Agent Management System
    - Directory facilitator („Gelbe Seiten“)
    - Agent communication channel
  - Liefert Java-APIs für FIPA-Standardkomponenten
    - Agenten-Kommunikationssprache
    - Agent Management Ontologie
    - Standard-Interaktionsprotokolle
- Open-Source-Implementierung unter LGPL Lizenzmodell (Lesser GNU Public License)
- Gemeinsame Entwicklung von CSELT und Uni Parma.
- Projekt-Homepage: http://sharon.cselt.it/projects/jade

CORPORATE TECHNOLOGY

S

CORPORATE TECHNOLOGY

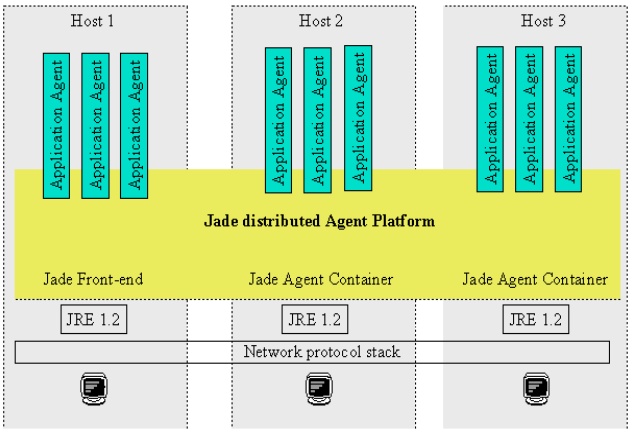
Jade-Plattform

- Agentenplattform auf mehrere Hosts verteilbar, maximal eine Plattform (= 1 Prozess) pro Host
- Jede Plattform besteht aus sogenannten Containern. Container stellen die Laufzeitumgebung für Jade-Agenten dar
  - Main Container: Dort laufen AMS, DF und die von Jade verwendete RMI-Registry
  - Agent Container: Können auf unterschiedlichen Hosts gestartet werden und kommunizieren mit dem Main Container
- Graphische Benutzerschnittstelle
- Diverse Entwicklungs- und Debuggingtools

S

CORPORATE TECHNOLOGY

JADE-Agentenplattform: Architektur



S

CORPORATE TECHNOLOGY

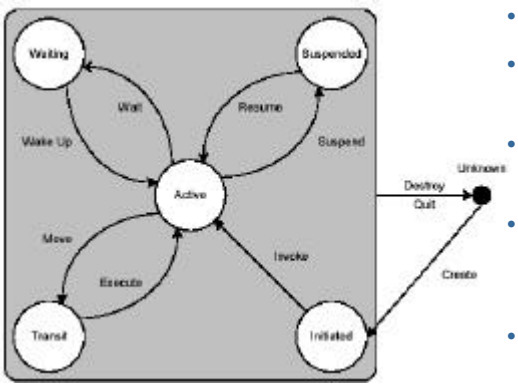
Jade-Agenten

- Jeder Jade-Agent ist einem Container in einer Plattform zugeordnet
  - Agent = Thread
  - Agent verfügt über eine Reihe sogenannter Behaviours, die sein Verhalten definieren
  - Mehrere Behaviours können gleichzeitig aktiv sein; sie werden durch einen internen nicht-preemptiven Behaviour Scheduler nach einer Round Robin Strategie geschedult.
- Jade-Agent: Instanz einer Subklasse von jade.core.Agent
- Jade unterstützt FIPA-Agentenlebenszyklus, d.h. mögliche Zustände eines Agenten; d.h. es bietet public Methoden für Transitionen zwischen Zuständen, z.B. doWait()

S

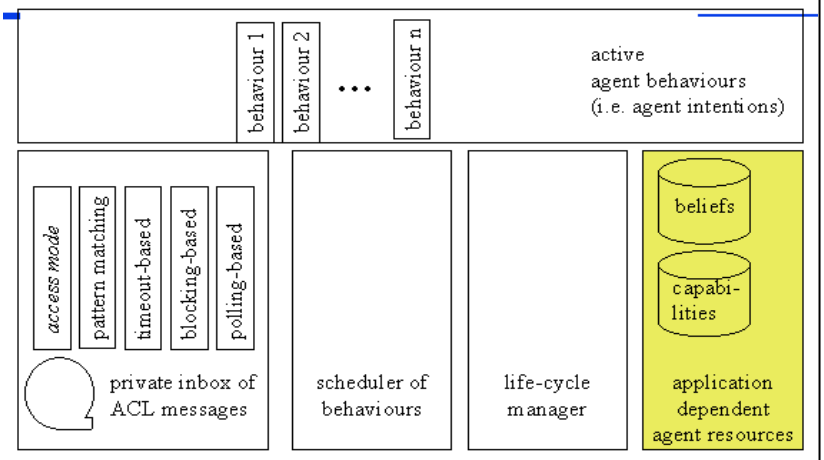
CORPORATE TECHNOLOGY

FIPA Agent Lifecycle

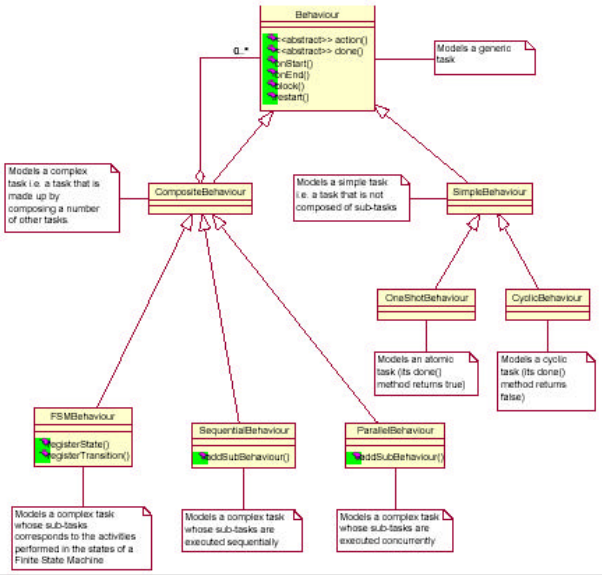


- INITIATED: Agent erzeugt
- ACTIVE: Agent bei AMS und DF angemeldet
- SUSPENDED: Gestoppt; kein Behaviour wird ausgeführt
- WAITING: Agent blockiert, sein Thread schläft (typisch: Warten auf Nachricht)
- DELETED: Agent ist terminiert, nicht mehr beim AMS registriert
- Daneben bietet JADE weitere Zustände für Mobile Agenten

JADE Agentenarchitektur

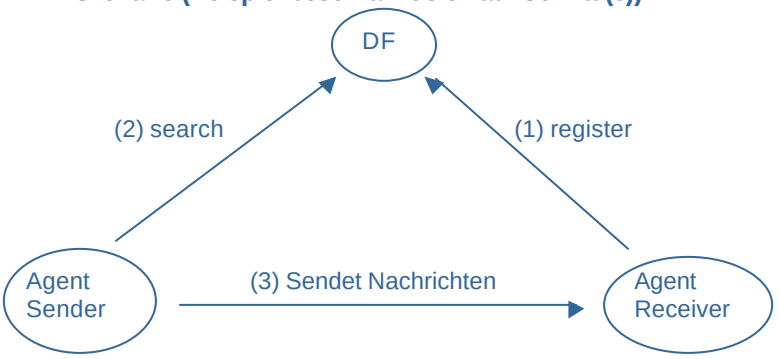


Behaviourtypen in Jade



Beispiel: Hello-World Agent

- **Szenario (Beispiel beschränkt sich auf Schritt (3)):**



- **Source Code von AgentReceiver und AgentSender am Ende des Kapitels**

Agentenprogrammierung

- **Agent = Instanz einer nutzerdefinierten Subklasse von `jade.core.Agent`**
- **Anwendungsentwickler implementiert folgende Methoden:**
  - **`setup()` zur Initialisierung fügt benötigte Aufgaben (Behaviours) hinzu (Methode addBehaviour(Behaviour))**
  - **`doDelete()` zum Stoppen des Agenten**
  - **ggf. `takeDown()`-Methode: wird bei doDelete() vom System aufgerufen, kann überschrieben werden. Dies erlaubt die flexible Spezifikation von Cleanup-Aktionen (z.B. De-registrieren des Agenten vom DF)**

S

CORPORATE TECHNOLOGY

Behaviour-Programmierung

- Behaviours dienen der Modellierung der Services/Aufgaben/Verhalten des Agenten
- Anwendungsentwickler implementiert zwei Methoden
  - `action()`: Ausführungslogik des Behaviours
  - `done()`: Rückgabewert des Behaviours

S

CORPORATE TECHNOLOGY

Die Klasse AgentSender

- Der AgentSender im Beispiel verwendet ein von der Klasse SimpleBehaviour abgeleitetes Behaviour, das als anonyme Klasse innerhalb der `setup()` Methode erzeugt wird:
  - erzeugt ein ACLMessage-Objekt (FIPA Agent Communication Language)
  - sendet das Objekt viermal unter Verwendung der `send()` Methode der Agent-Klasse und verwendet dabei verschiedene Attribute der ACLMessage, z.B.
    - Receiver: Empfänger der Nachricht
    - Language: Typ des Nachrichteninhaltes
    - Content: Eigentlicher Inhalt
    - Ontology : Für die Interpretation des Inhalts zu verwendende Ontologie
  - terminiert danach unter Aufruf der `doDelete()` Methode

S

CORPORATE TECHNOLOGY

Die Klasse AgentReceiver

- Empfängt nacheinander die Nachrichten vom AgentSender
- Demonstriert dabei unterschiedliche Arten, eine Nachricht zu empfangen
  - op1(): Nicht-blockierende Methode `receive()`: solange die Nachricht nicht eintrifft, wird das Behaviour suspendiert
  - op2(): Blockierende Methode `blockingReceive(long)`: blockiert den kompletten Agenten für den in time angegebenen Zeitraum
  - op3(): `blockingReceive(MessageTemplate)`, wobei zusätzlich ein Filterausdruck (MessageTemplate) mitgegeben wird
- In der setup() Methode wird dieses Behaviour erzeugt und zur Behaviour-Liste des Agenten hinzugefügt

S

CORPORATE TECHNOLOGY

Ausführung

- Start der Agentenplattform
  - `java jade.Boot -gui Sender:demo.AgentSender`
  - allgemein:

```
java jade.Boot [-gui] {<AgentName>:<AgentClass>}
```

- Hinzufügen eines neuen Containers zur Plattform
  - `java jade.Boot -container Receiver:demo.AgentReceiver`
  - allgemein:

```
java jade.Boot -container [-gui] [-host]
  {<AgentName>:<AgentClass>}
```

CORPORATE TECHNOLOGY

Code für AgentSender

```
package examples.receivers;

import java.io.*;
import jade.core.*;
import jade.core.behaviours.*;
import jade.lang.acl.*;

public class AgentSender extends Agent {
  protected void setup() {
    addBehaviour(new SimpleBehaviour(this) {
      private boolean finished = false;
      public void action() {
        try{
          System.out.println("\nEnter responder agent name: ");
          BufferedReader buff = new BufferedReader(new
          InputStreamReader(System.in));
          String responder = buff.readLine();
          ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
           msg.addReceiver(new AID(responder));
           msg.setContent("FirstInform");
           send(msg);
           System.out.println("\nFirst INFORM sent");
           doWait(5000);
           msg.setLanguage("PlainText");
           msg.setContent("SecondInform");
           send(msg);
           System.out.println("\nSecond INFORM sent");
           doWait(5000);
           // same that second
           msg.setContent("\nThirdInform");
           send(msg);
           System.out.println("\nThird INFORM sent");
           doWait(1000);
           msg.setOntology("ReceiveTest");
           msg.setContent("FourthInform");
           send(msg);
           System.out.println("\nFourth INFORM sent");
           finished = true;
           myAgent.doDelete();
         }catch (IOException ioe){
           ioe.printStackTrace();
         }
       }

       public boolean done(){
         return finished;
       }
     }); // end of addBehaviour()
   } // end of setup()
} // end of AgentSender
```


CORPORATE TECHNOLOGY

Code für AgentReceiver

```
package examples.receivers;
import java.io.*;
import jade.core.*;
import jade.core.behaviours.*;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;

public class AgentReceiver extends Agent {
  class my3StepBehaviour extends SimpleBehaviour {
    final int FIRST = 1;
    final int SECOND = 2;
    final int THIRD = 3;
    private int state = FIRST;
    private boolean finished = false;

    public my3StepBehaviour(Agent a) { super(a); }

    public void action() {
      switch (state){
      case FIRST: {if (op1())
        state = SECOND;
        else
        state= FIRST;
        break;}
      case SECOND:{op2(); state = THIRD; break;}
      case THIRD:{op3(); state = FIRST; finished = true; break;}
      }
    }

  public boolean done() { return finished;}

  private boolean op1(){
     System.out.println( "\nAgent "+getLocalName()+" in state 1.1 is
       waiting for a message");
     MessageTemplate m1 =
       MessageTemplate.MatchPerformative(ACLMessage.INFORM);
     MessageTemplate m2 =
       MessageTemplate.MatchLanguage("PlainText");
     MessageTemplate m3 =
       MessageTemplate.MatchOntology("ReceiveTest");
     MessageTemplate m1andm2 = MessageTemplate.and(m1,m2);
     MessageTemplate notm3 = MessageTemplate.not(m3);
     MessageTemplate m1andm2_and_notm3 =
       MessageTemplate.and(m1andm2, notm3);
  // The agent waits for a specific message. If it doesn't arrive
  // the behaviour is suspended until a new message arrives.
     ACLMessage msg = receive(m1andm2_and_notm3);
     if (msg != null){
       System.out.println("\nAgent "+ getLocalName() +
         " received the following message in state 1.1: " +
         msg.toString());
       return true;
     }
     else {
       System.out.println("\nNo message received in state 1.1");
       block();
       return false;
     }
   }
```

CORPORATE TECHNOLOGY

Code für AgentReceiver (2)

```
private void op2(){
    System.out.println("\nAgent "+ getLocalName()
      + " in state 1.2 is waiting for a message");
 //Using a blocking receive causes the block
 // of all the behaviours
    ACLMessage msg = blockingReceive(5000);
    if (msg != null)
      System.out.println("\nAgent "+ getLocalName() +
        " received the following message in state 1.2: "
        +msg.toString());
    else
      System.out.println("\nNo message received
        in state 1.2");
  }
```

```
private void op3() {
    System.out.println("\nAgent: "+getLocalName()+
      " in state 1.3 is waiting for a message");
    MessageTemplate m1 =
      MessageTemplate.MatchPerformative(ACLMessage.INFORM);
    MessageTemplate m2 =
      MessageTemplate.MatchLanguage("PlainText");
    MessageTemplate m3 =
      MessageTemplate.MatchOntology("ReceiveTest");
    MessageTemplate m1andm2 = MessageTemplate.and(m1,m2);
    MessageTemplate m1andm2_and_m3 =
      MessageTemplate.and(m1andm2, m3);
// blockingReceive and template
    ACLMessage msg = blockingReceive(m1andm2_and_m3);
    if (msg!= null)
      System.out.println("\nAgent "+ getLocalName() +
        " received the following message in state 1.3: "
        + msg.toString());
    else
      System.out.println("\nNo message received in state 1.3");
  }
 } // End of my3StepBehaviour class

 protected voidsetup() {
   my3StepBehaviour mybehaviour = new my3StepBehaviour(this);
   addBehaviour(mybehaviour);
 }
} // end of AgentReceiver
```


CORPORATE TECHNOLOGY

Jade: Bewertung

- **Stärken**
  - Schnelle Entwicklung verteilter Anwendungen
  - Interaktionsprotokolle erlauben High-Level Beschreibungen
  - Tools für Verteiltes Debugging und Tracing
  - Lebhafte User Community bei freier Verfügbarkeit (Open Source)
  - Durch Ontologie-Konzept leichte Integration von Semantic-Web-Konzepten
  - Modular und erweiterbar (z.B. Transport-Ebene: auch TCP/IP, SOAP, HTTP)
- **Schwächen**
  - Mobilität und Sicherheit werden nur rudimentär unterstützt
  - Noch keine Unterstützung für komplexe Markt- und Verhandlungsmodelle
  - Gut geeignet für Rapid Prototyping, industrielle Einsetzbarkeit fraglich
  - Behaviour-Konzept für manche Anwendungen zu primitiv
  - Footprint für kleine Endgeräte zu groß (--> Jade/LEAP Erweiterung)

CORPORATE TECHNOLOGY

Lerninhalte dieses Kapitels

- Verständnis der Wichtigkeit von Standardisierung
- Vermitteln der grundlegenden Eigenschaften des FIPA-Standards für Agentensysteme
- Verständnis der wesentlichen Features von Agentenplattformen
- Kennenlernen einer speziellen Agentenplattform: Jade