# D7001D Network programming and distributed applications

## Agents

Evgeny Osipov

Communication Networks Group
Luleå Tekniska Universitet

# What is Agent

- Agent terms raised from several areas such as blend of the following technologies:
  - network communication, artificial intelligence, multi-agent-Systems, distributed problem solving, distributed systems, parallel processing, knowledge engineering, distributed artificial intelligence and object-oriented technology
  - Each group has its own definitions and its terminology

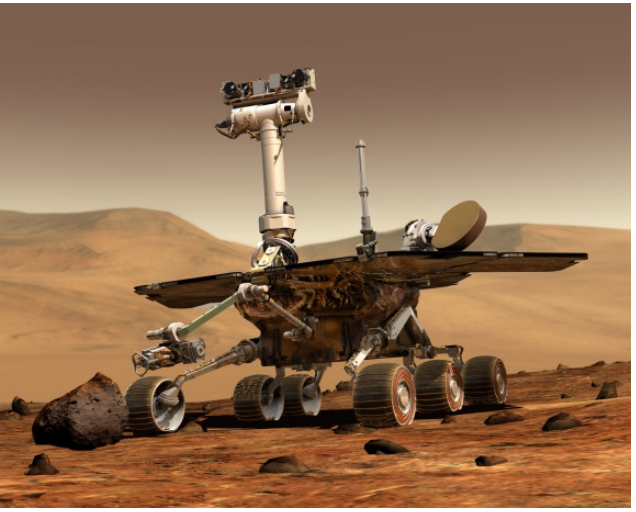
# Open Distributed Systems

- **Open System**
  - Capable of dynamically changing.
  - Its components are not known in advance, can change over time,
  - Highly heterogeneous.

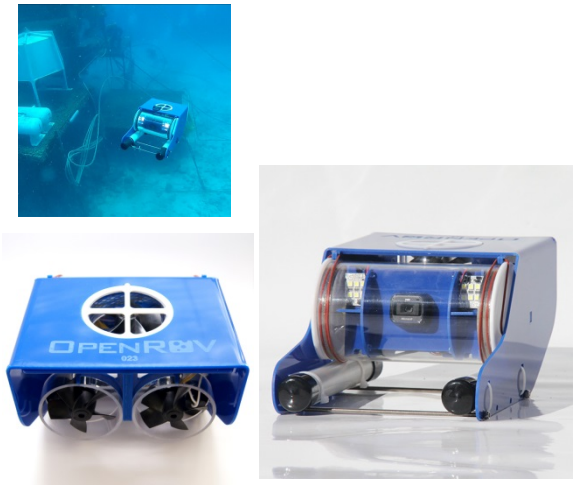
**We need special software architecture to work with Open Distributed Systems**

# Special type of distributed applications

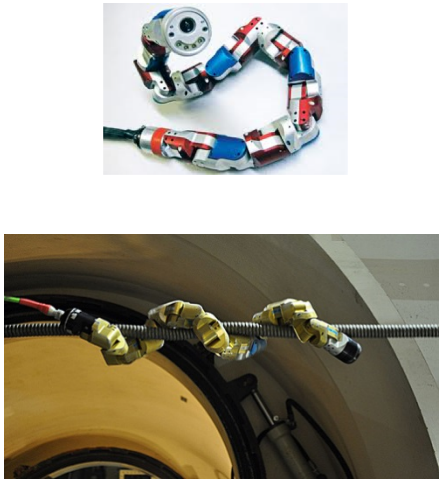
- Agents (and their physical instantiation in robots) have a role to play in high-risk situations, unsuitable or impossible for humans
- The degree of autonomy will differ depending on the situation (remote human control may be an alternative, but not always)



**The Spirit Rover explores Gusev Crater on Mars, image credit NASA/JPL**



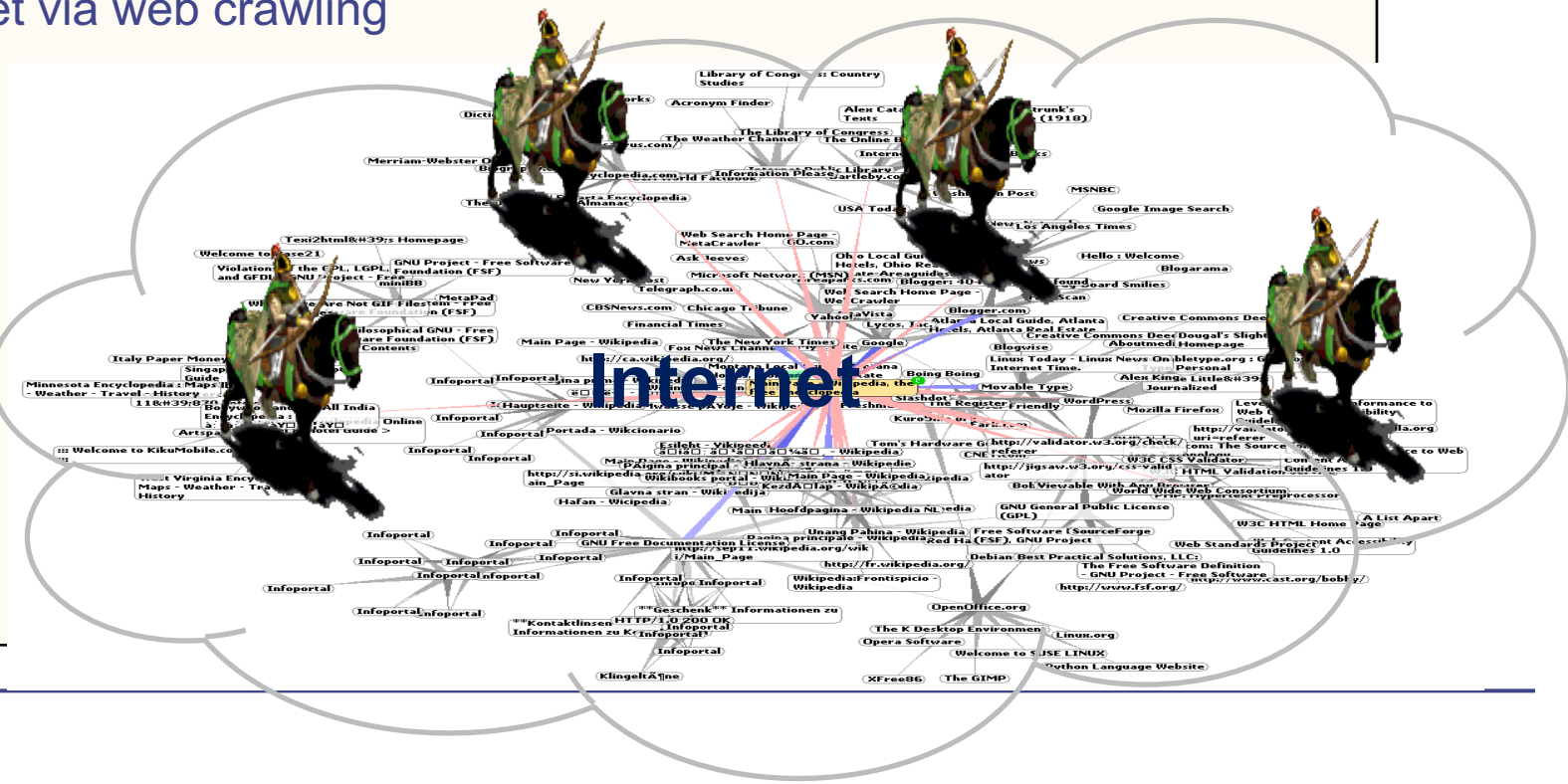

**OpenROV is an open-source underwater robot for exploration and education**



**CMU Snake Robot Trial at Nuclear Power Plant**

# Special type of distributed applications

- A **Web crawler** is an Internet bot that systematically browses the World Wide Web, typically for the purpose of Web indexing.
- **Distributed web crawling** is a distributed computing technique whereby Internet search engines employ many computers to index the Internet via web crawling

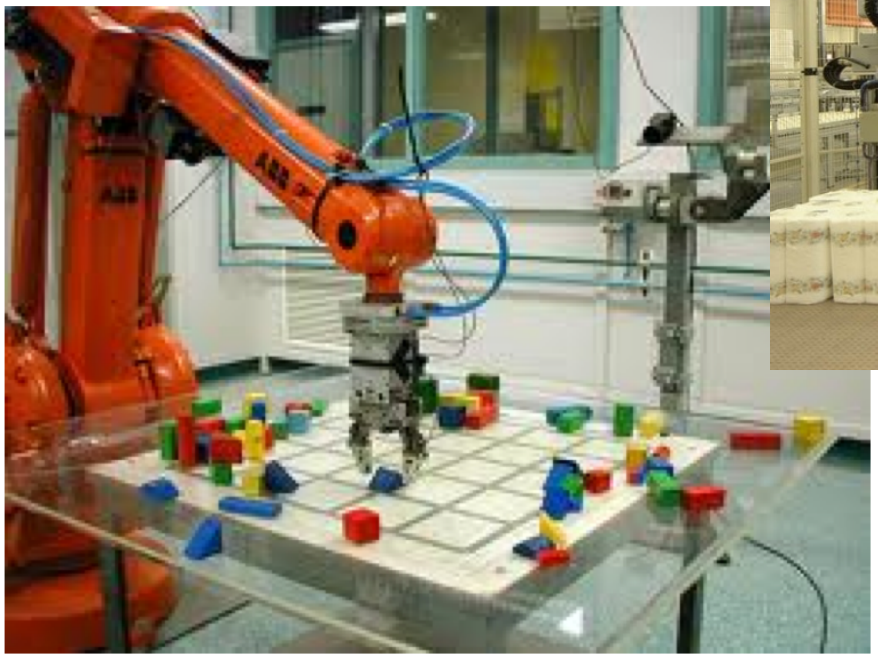


# Open Distributed Systems: Ubiquitous computing systems.

- In the era of Ubiquitous Computing. People are surrounded by lots of small invisible computers
- UC: offers an autonomous support for people's every day life
- Ubiquitous system needs context knowledge.
- Must dynamically adapt to the environment.
- Ubiquitous environment should know person's profiles, preferences, liking, habits etc…
- People often change their location and personal info.
- Environment is Highly Dynamic, Unpredictable etc…

# Special type of distributed applications

- Flexible Manufacturing



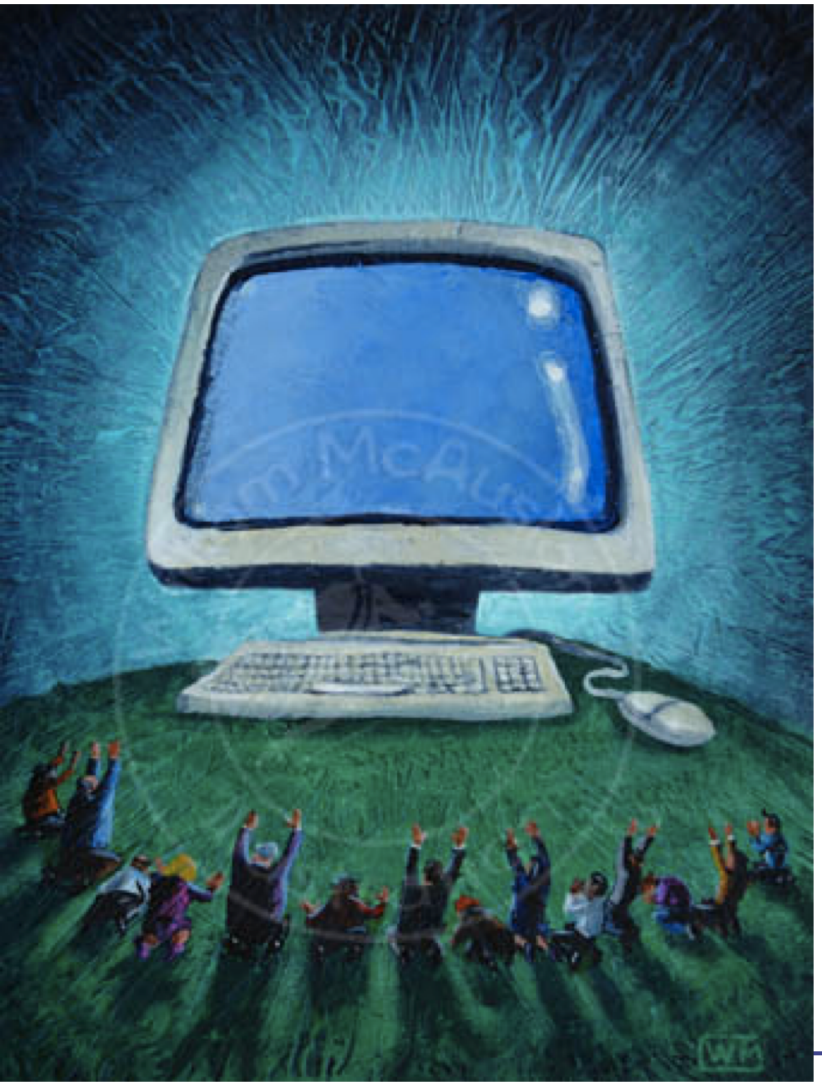
# Distributed software for Open Systems

- The most powerful tool for handling complexity in software development are ***modularity and abstraction***.

- ***We need a powerful tool for making systems*** **modular:**
  - Complex, large, or unpredictable
  - To develop a number of modular components
  - Each one which is specialized in a specific task and at solving a particular aspect of it.
  - To manage interdependent Activities: agents in the systems must cooperate and coordinate

  - **Agent-oriented software engineering**

# A shift of programming paradigm



- The main objective for the distributed computing system is its autonomity
- You set the goal the system should figure out how to achieve it.

# Our Focus is on (Intelligent) Software Agent

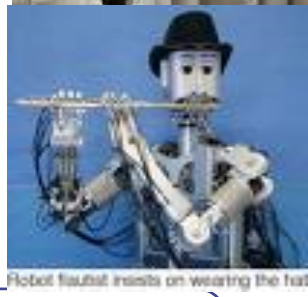
- An Agent:
  - **Is a computer system that is situated in some environment**
  - **Capable of autonomous action (Rather than Constantly being told)**
  - **To meet its design objective**

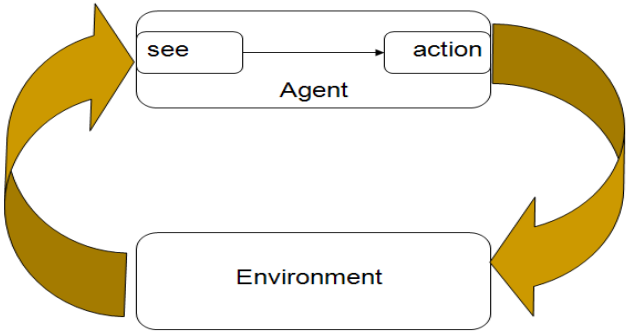
**Another Definition**

- Agent is either a software or hardware/combination of both that perform task with a certain degree of independency to achieve its goal.

## Every Agent System has:

- A performance measure
- An environment
- Actuators (to take action)
- Sensors (perception, see)




## Society of Agents (Multi-agent System)

- Agents Communicate using Agent Communication languages(ACL)

**Practical Reasoning Agents** | **Deductive Reasoning Agents** | **Reactive Agents** | **Hybrid Agents**

## Agent Characteristics

- Autonomous
- Reactive
- Proactive
- Situated
- Sociable
- Goal oriented

## Cooperative Problem Solving

- Enact Any Business Workflows

# Characterizing Agents

- Agent may have combination of the following properties:
  - ***Autonomous**: able to act without direct external intervention.*
  - ***Reactive**: Agents perceive their environment (which may be the physical world, a user via a* graphical user interface, a collection of other agents, the Internet, or perhaps all of these combined), and respond in a timely fashion to changes that occur in it
  - ***Interactive**: able to communicate with other agents and its environment.*
  - ***Sociable**: capable of two-way conversation, in which one party asks questions and the other* verifies or answers. The conversation allows inter-agent co-operation which enables agents to exchange their knowledge, beliefs and plans and to work together ***to solve large problems which are beyond an individual's capabilities***.
  - ***Proactive**: having a goal-oriented ability or a purpose. Find the way to reach the goal on its own*

# Characterizing Agents

- ***Proxy**: able to act on behalf of someone or something, that is, acting in the interest of, as a* representative of, or for benefit of some entity
- ***Co-ordinative**: able to perform some activity in a shared environment with other agents.*
- ***Co-operative**: able to co-ordinate with other agents to achieve a common purpose (some* scholars define this as collaboration)
- ***Intelligent**: able to state formalised knowledge such as beliefs, goals and assumptions and to* interact with other agents using symbolic language.
- ***Mobility** - agents can move around in their environment from one place to another and carry* data along with intelligent instructions and execute remotely.
- ***Risk and trust-** address the issue of delegation of tasks with at least a reasonable assurance that* the user wants to delegate the task, and that the agent can bring back the result the user wanted[Jenn+98].
- ***Rational** - able to choose an action based on internal goals and the knowledge that a particular* action will bring it closer to its goals. Previous and Current knowledge from the Environment.

## Environments – *Accessible* vs. *inaccessible*

- An accessible environment is one in which the agent can obtain complete, accurate, up-to-date information about the environment's state
- Most moderately complex environments (including, for example, the everyday physical world and the Internet) are inaccessible
- The more accessible an environment is, the simpler it is to build agents to operate in it

## Environments – *Deterministic* vs. *non-deterministic*

- A deterministic environment is one in which any action has a single guaranteed effect — there is no uncertainty about the state that will result from performing an action
- The physical world can to all intents and purposes be regarded as non-deterministic
- Non-deterministic environments present greater problems for the agent designer

## Environments – *Episodic* vs. *non-episodic*

- In an episodic environment, the performance of an agent depends on a number of discrete episodes, with no link between the performance of an agent in different scenarios
- Episodic environments are simpler from the agent developer's perspective because the agent can decide what action to perform based only on the current episode — it need not reason about the interactions between this and future episodes

**Example: Current working shift in a Factory**

## Environments – *Static* vs. *dynamic*

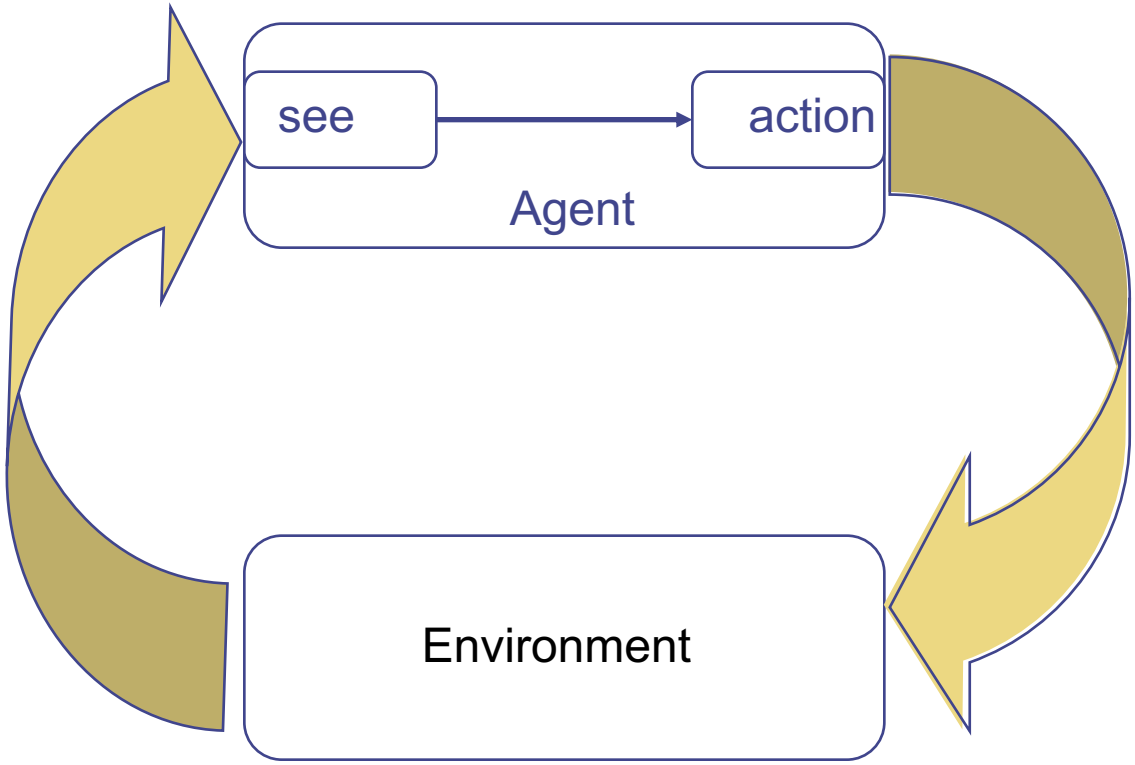
- A static environment is one that can be assumed to remain unchanged except by the performance of actions by the agent
- A dynamic environment is one that has other processes operating on it, and which hence changes in ways beyond the agent's control
- Other processes can interfere with the agent's actions (as in concurrent systems theory)
- The physical world is a highly dynamic environment

## Environments – *Discrete* vs. *continuous*

- An environment is discrete if there are a fixed, finite number of actions and percepts in it
- Chess game is an example of a discrete environment, and taxi driving is an example of a continuous one
- Continuous environments have a certain level of mismatch with computer systems
- Discrete environments could *in principle* be handled by a kind of "lookup table"

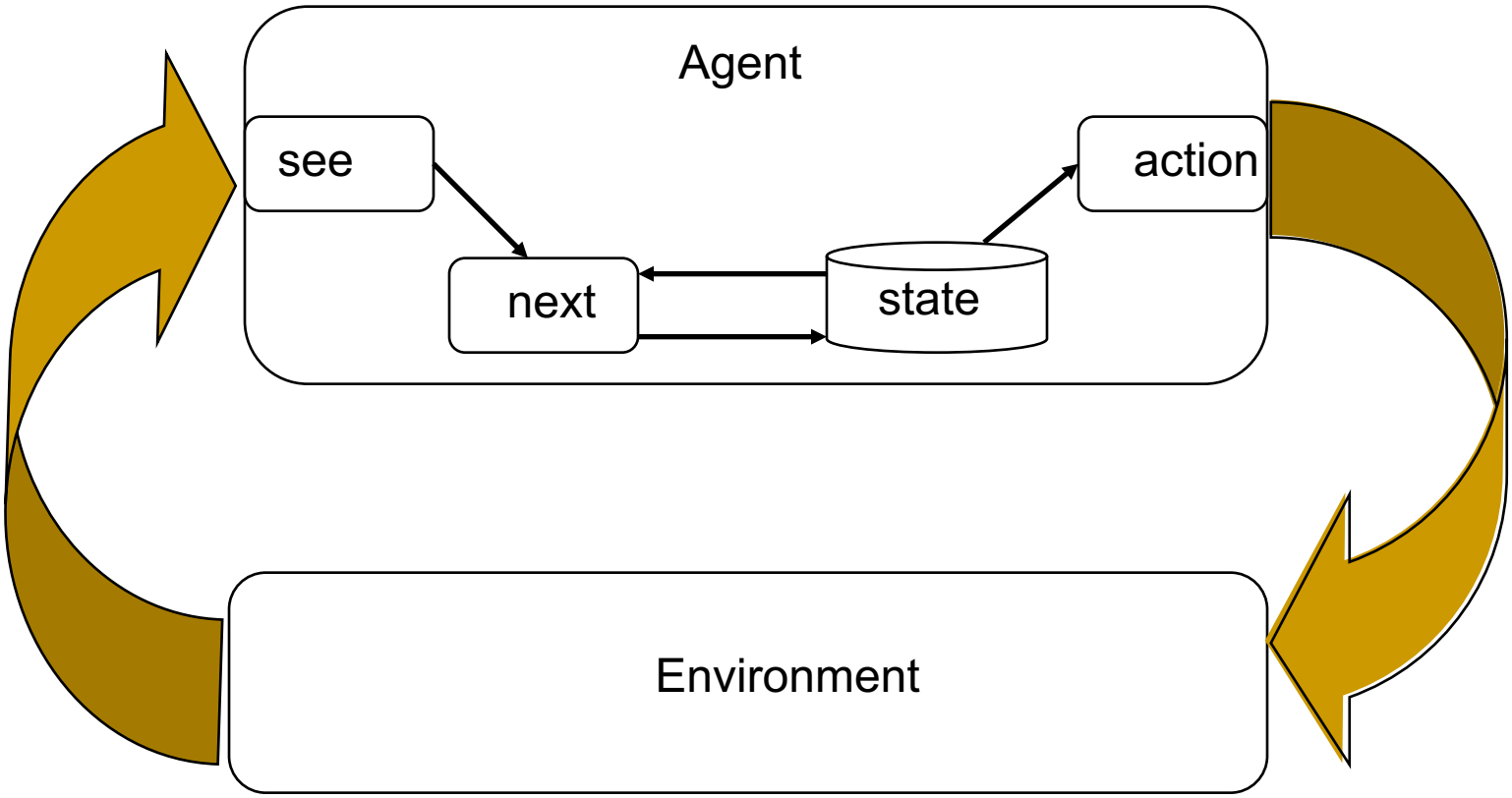
# Agents and Environment

*perception* system:



# Agents and Environment

agents that *maintain state*:



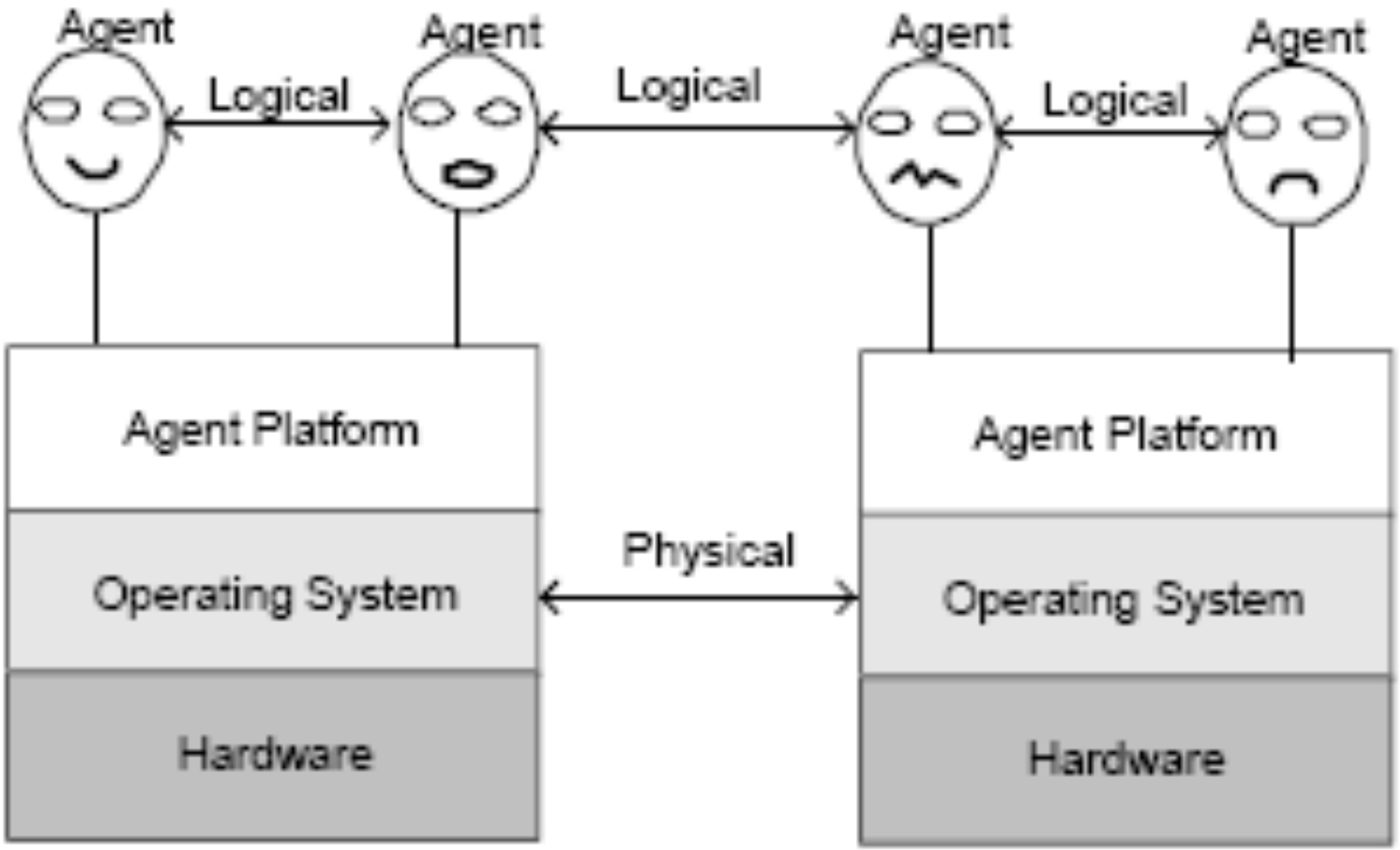
# Branches of Agents Application

- Intelligent Agents
- Mobile Agents
- Software Agent
- Multi-Agent System

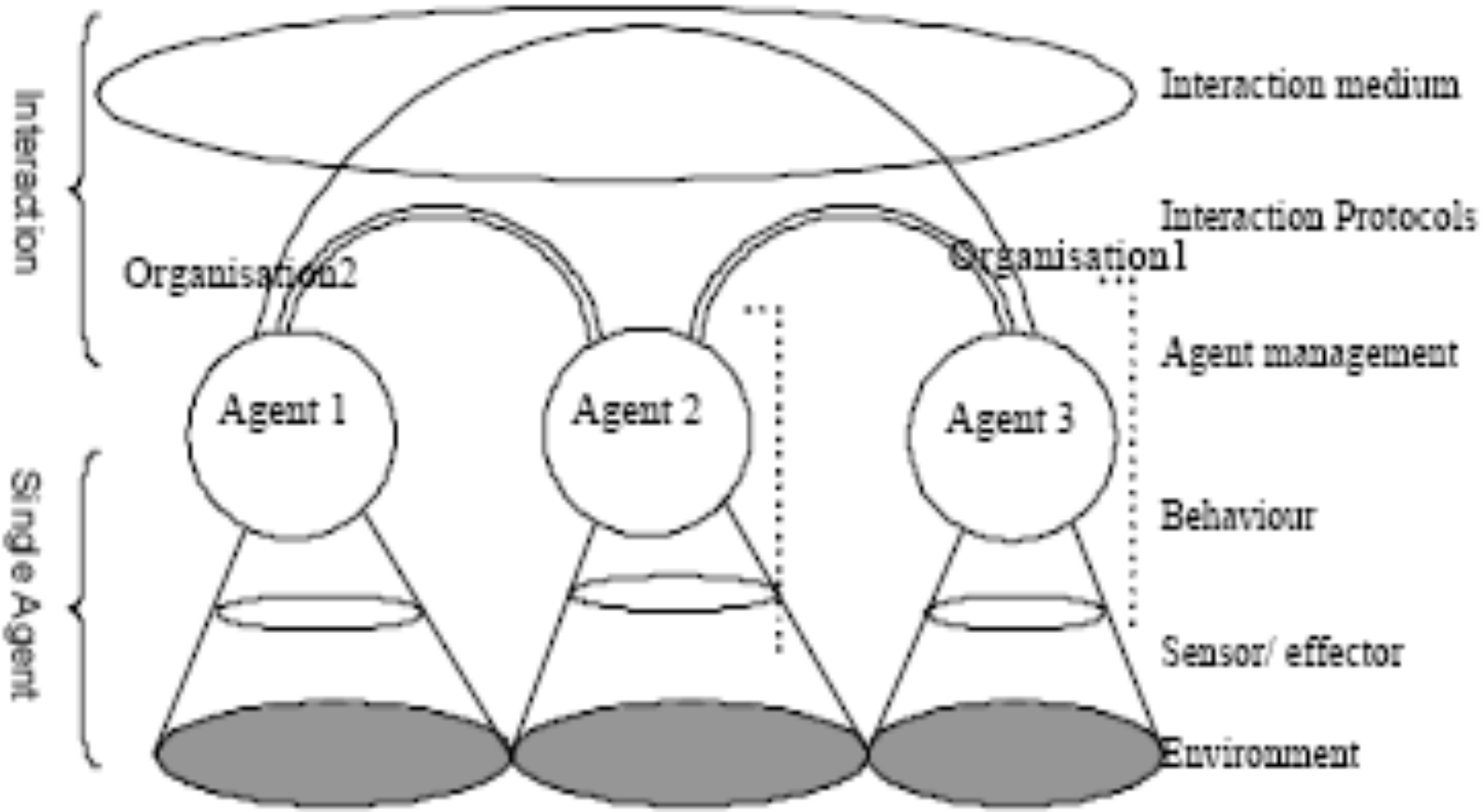
Single Agent vs. MAS

- Single agent
  - An application that presents some degree of agent's criteria.
- MAS
  - An application that consists of many agents which communicate each other to achieve its goal.

# Programming Agents: Focus on functionality



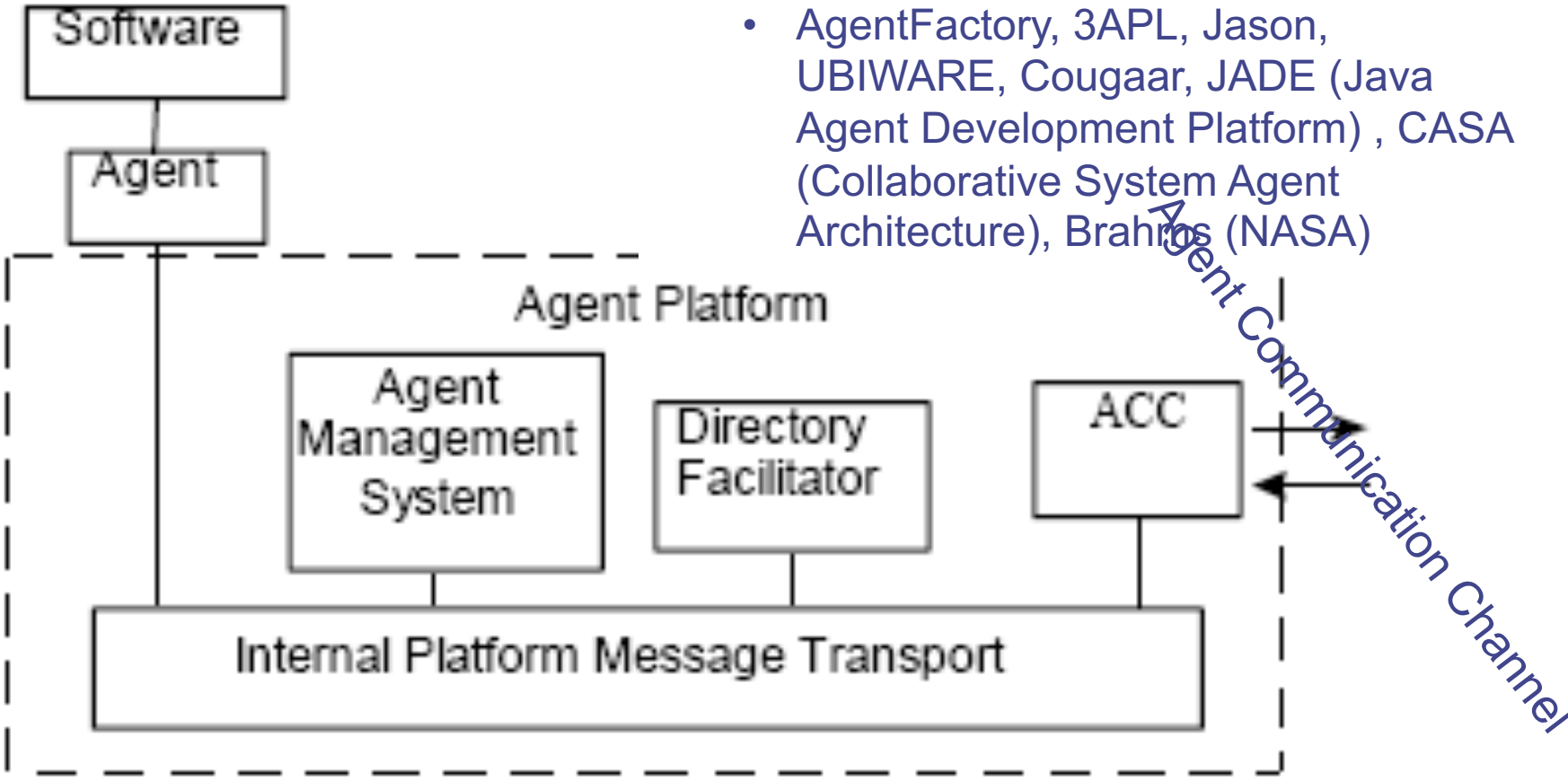
# MAS architecture



# JADE Framework

| Product /Criteria | JADE [JADE98] | JAFMAS [JAF97] | ZUES [ZUES00] | JATLITE [JAT97] | Agent Builder [BUIL00] | April [APRIL00] |
|---|---|---|---|---|---|---|
| AI features | 2 | 3 | 2 | 2 | 3 | 2 |
| Open architecture | 3 | 1 | 3 | 1 | 2 | 2 |
| Extension | 3 | 2 | 3 | 2 | 2 | 2 |
| ACL | FIPA | KQML | KQML/ FIPA | KQML | KQML | KQML |
| Support Distribution Location | 3 | 1 | 3 | 2 | 3 | 2 |
| Basic properties | Autonomy reactive | Autonom y reactive | Autonom y reactive | Autonom y reactive | BDI | Autonomy reactive |

- There exist many agent middleware platforms like
  - AgentFactory, 3APL, Jason, UBIWARE, Cougaar, JADE (Java Agent Development Platform) , CASA (Collaborative System Agent Architecture), Brahms (NASA)




- http://jade.tilab.com/papers/PAAM.pdf

# Jade get started

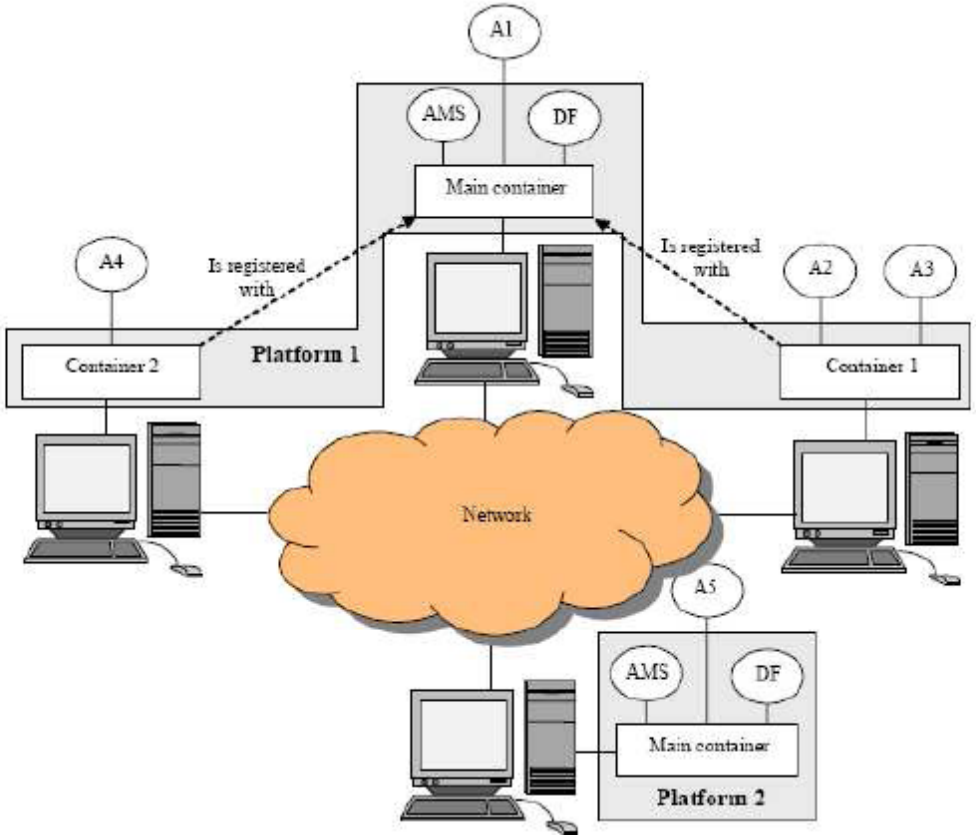
**JADE:**
is a framework to develop multi-agent systems in compliance with the ***FIPA*** specification

**FIPA :** Foundation For Intelligent Physical Agents
IEEE Computer Society standards organization that promotes agent-based technology and the interoperability of its standards with other technologies.
URL: http://www.fipa.org/

**System Requirements:**
Java Run Time Environment

**JADE Website:**
1. http://jade.tilab.com/

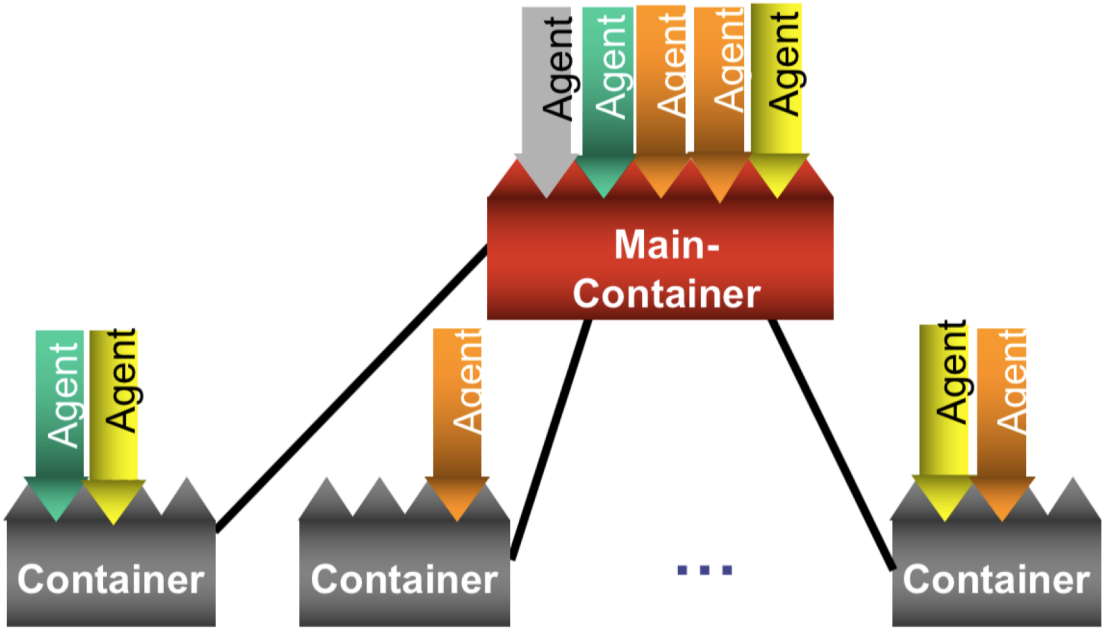
# Jade Runtime




- Each running instance of the JADE runtime environment is called a
  - ***Container*** *as it can contain several agents*
- The set of active containers is called a ***Platform***.

# Jade runtime environment

- Main-Container:
  - *must always be active in a platform and a*ll Other containers register with it as soon as they start.
- Container:
  - When starting secondary containers, they must know the address of the main container
- Agent




Normally, one container is run on host.
However, can also run several containers on the same host.

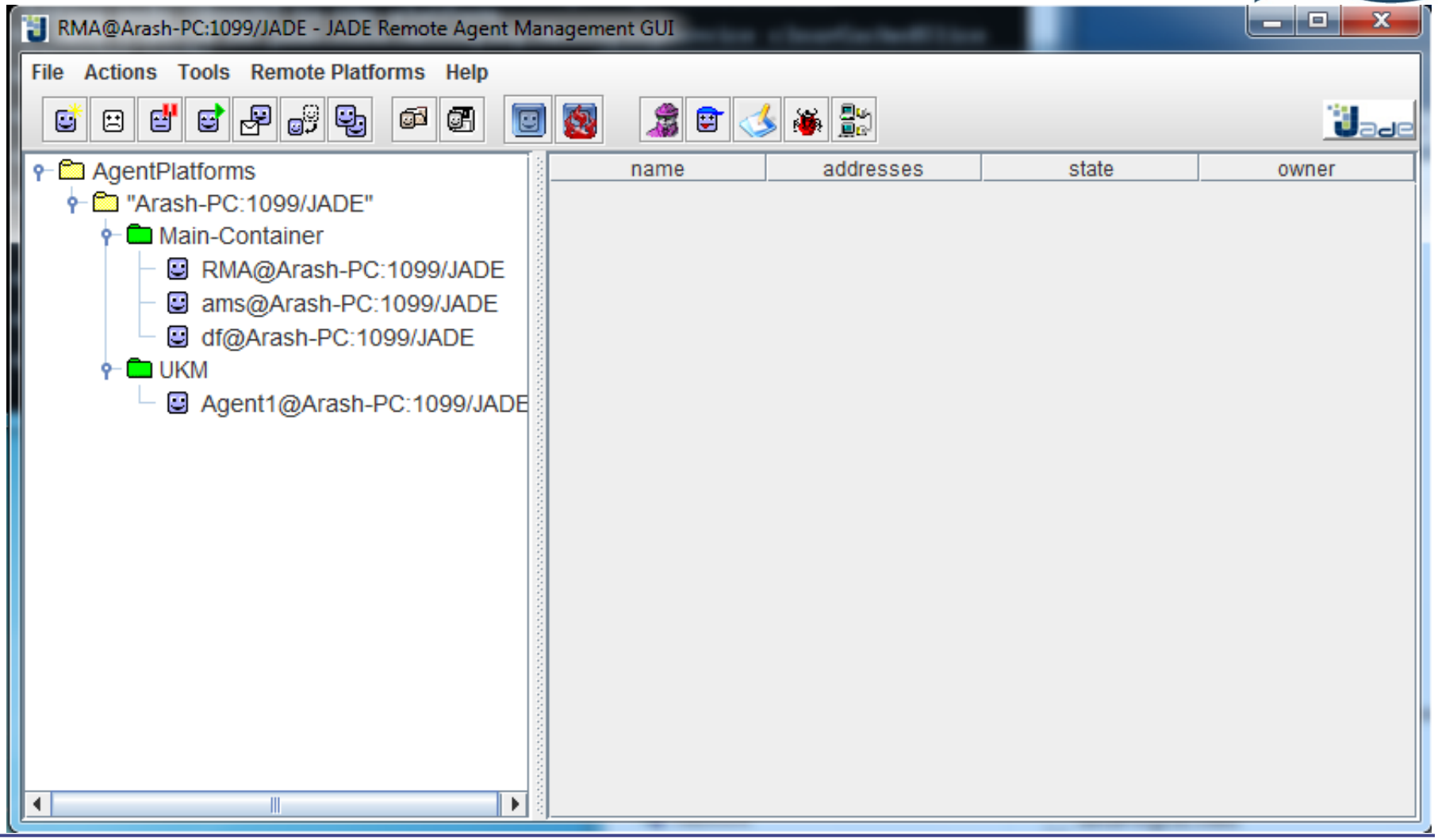
# Downloading and running

- **Downloading Site:**
- Download the JADE from the following site:
  - http://jade.tilab.com/
- **Installation:**
- Just unzip the file into some folder

- **Read "Tutorials and Guides" to get started**
  - https://jade.tilab.com/documentation/tutorials-guides/

- **Starting JADE with launching Remote Monitoring Agent (RMA) GUI:**
- Type following on the command line:
  - **::>> java jade.Boot –gui**

# Remote Monitoring Agent (RMA) Graphical Interface



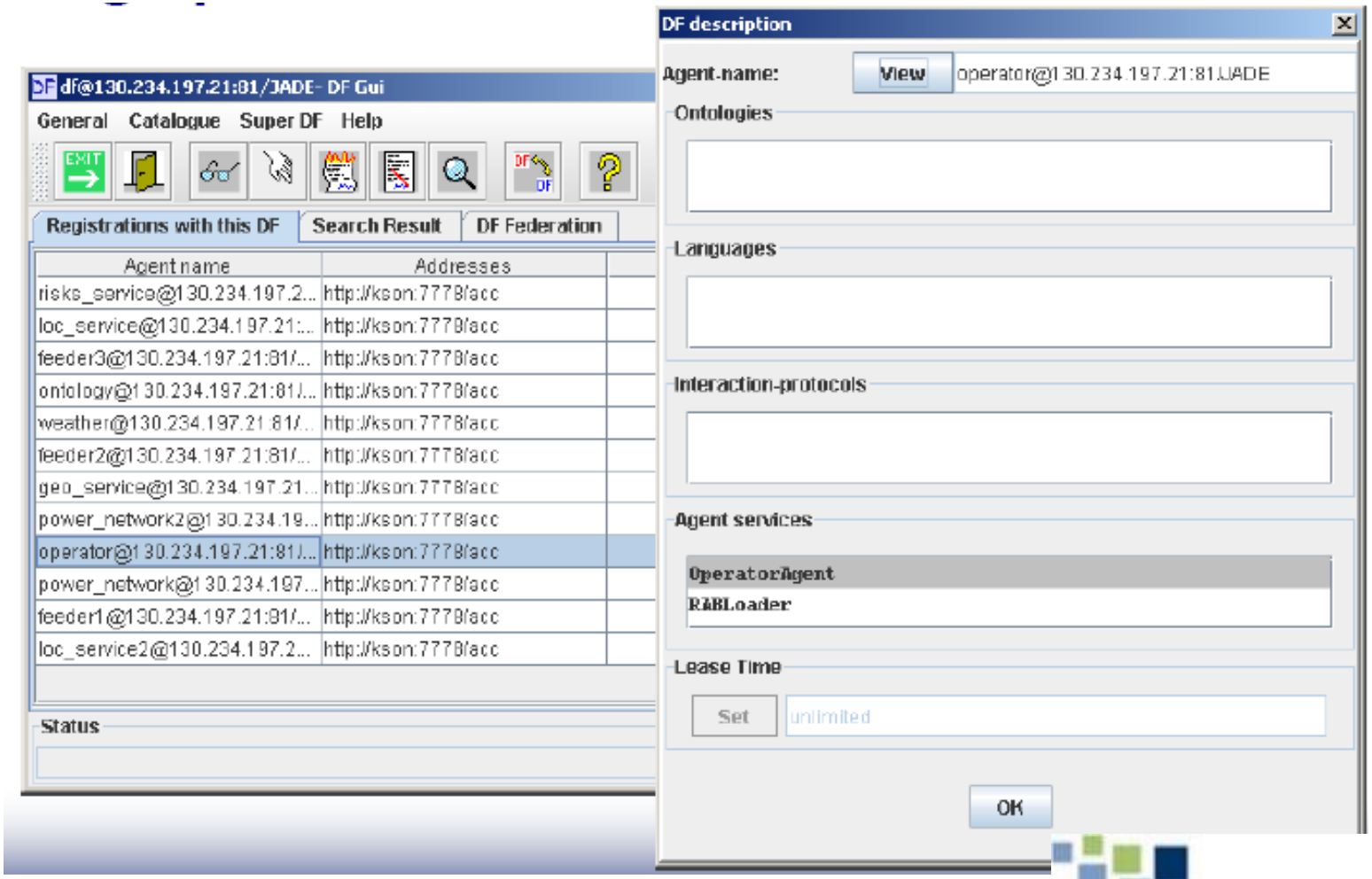
# Directory Facilitator (DF)

- *In order to find agents on a platform we use the Directory Facilitator (DF) agent which maintains a directory providing the following information:*

➢ A list of services which can be provided, and the agents who can provide them and what languages, interaction protocols and ontologies the services require.

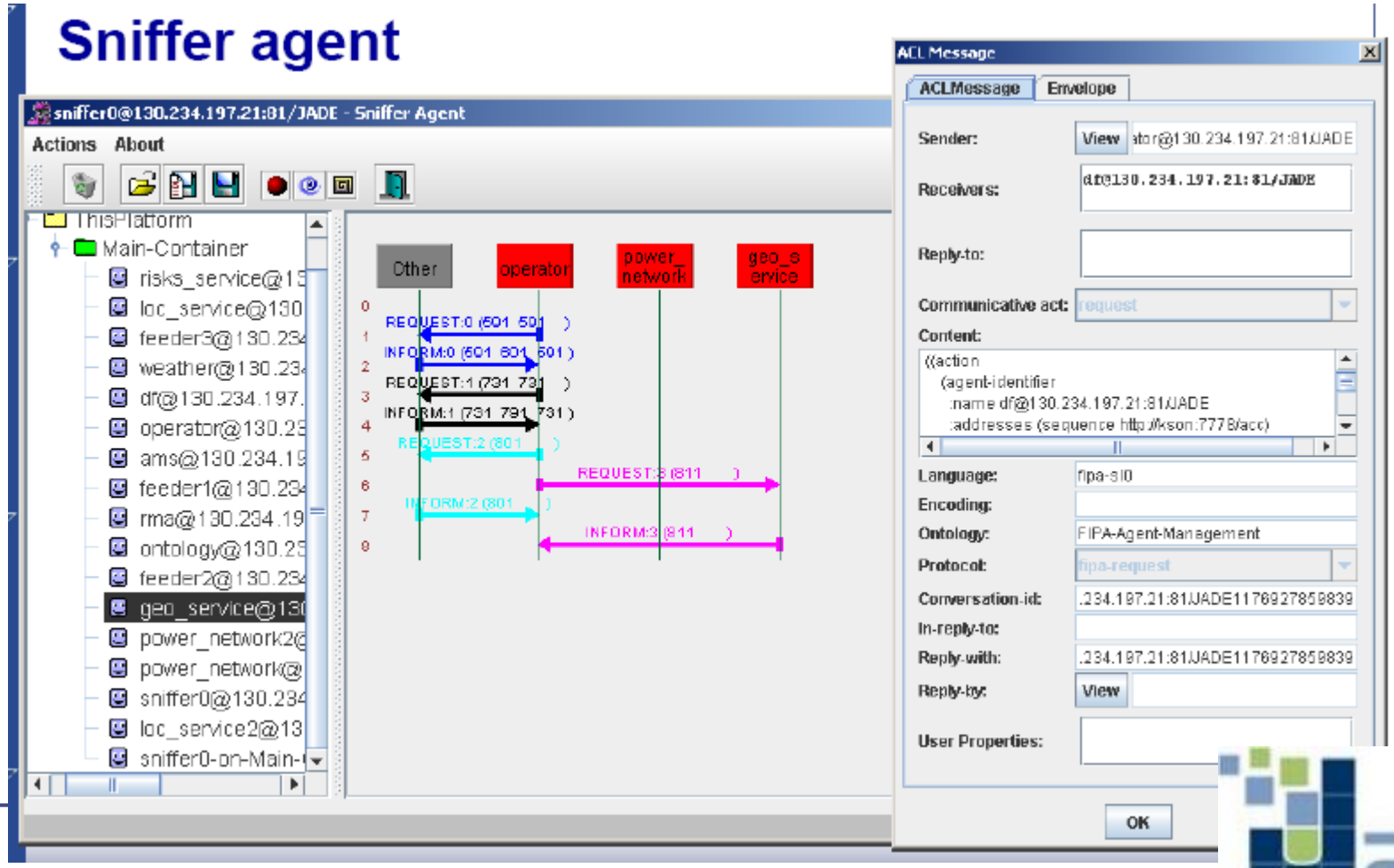
➢ In addition the DF can be used to maintain information about the languages, protocols and ontologies supported directly by agents (i.e. not related to particular services).

# Directory Facilitator (DF) Agent interface



# Sinffer agent graphical interface

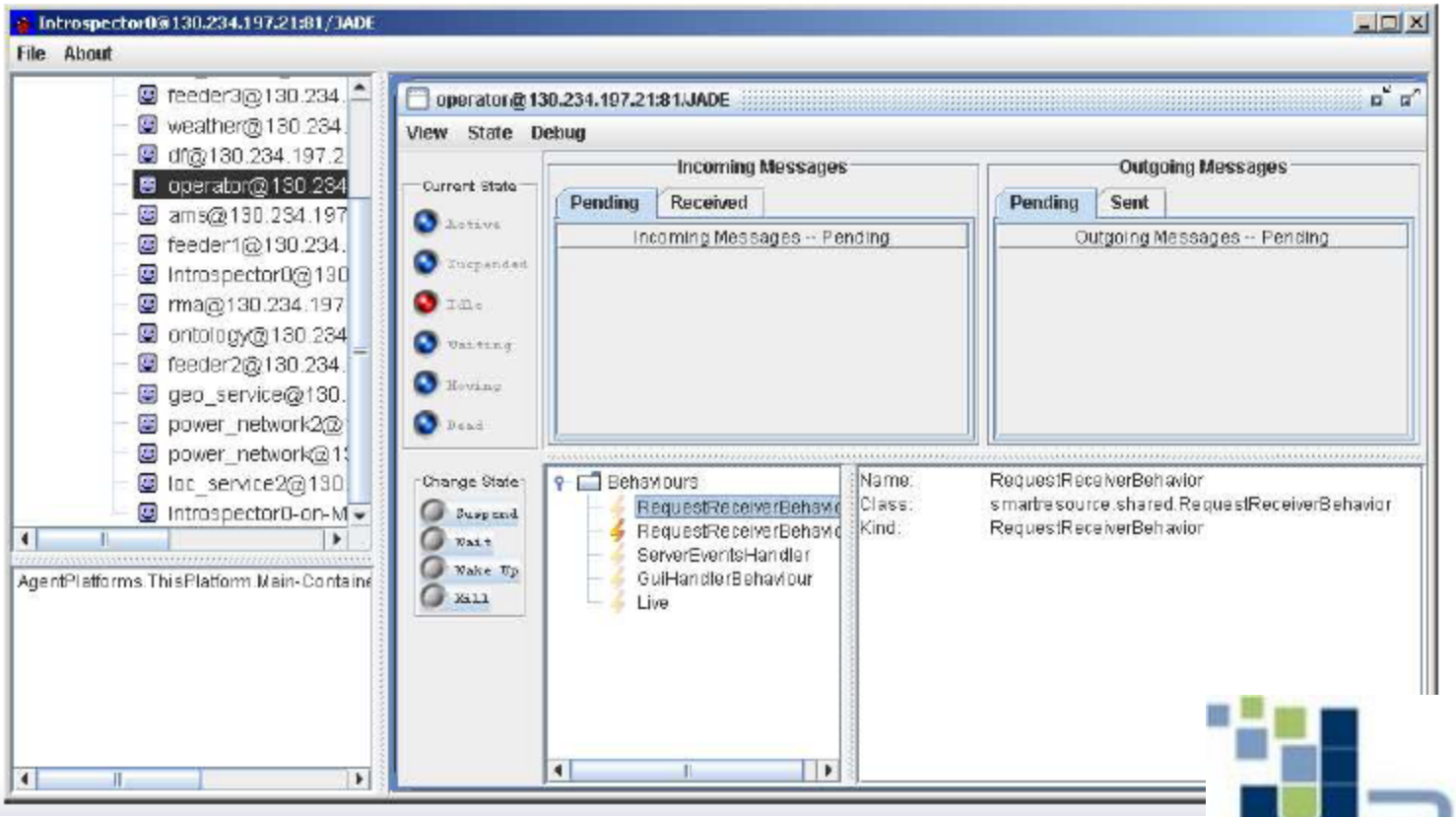
- This tool is extensively used for debugging, or simply documenting conversations between agents. Sniffer can track every message directed to or coming from an agent or a group of agents



# Introspector Agent graphical interface

➢ This agent is used to debug the behavior of a single agent. This tool is used to monitor the life cycle of an agent and its queues of sent and received messages.

## Introspector agent



# JADE Programming
# First JADE Agent: (HelloWorldAgent)

- **Programming JADE agents**
- Resources:
  - Programmer's tutorial: https://jade.tilab.com/doc/tutorials/JADEProgramming-Tutorial-for-beginners.pdf
  - Online JavaDocs: http://jade.tilab.com/doc/api/index.html

# JADE Programming
# First JADE Agent: (HelloWorldAgent)

- Creating a JADE agent requires writing a subclass of the class:
- *jade.core.Agent*
- Minimal agent program (HelloWorldAgent.java):
  - **import jade.core.Agent;**
  - **public class HelloWorldAgent extends Agent {**
  - **protected void setup() {**
  - **System.out.println("Hello World! My name is "+getLocalName()); } }**
- *To compile: javac HelloWorldAgent.java*
- *To start:*

*java jade.Boot –container John:HelloWorldAgent*

or

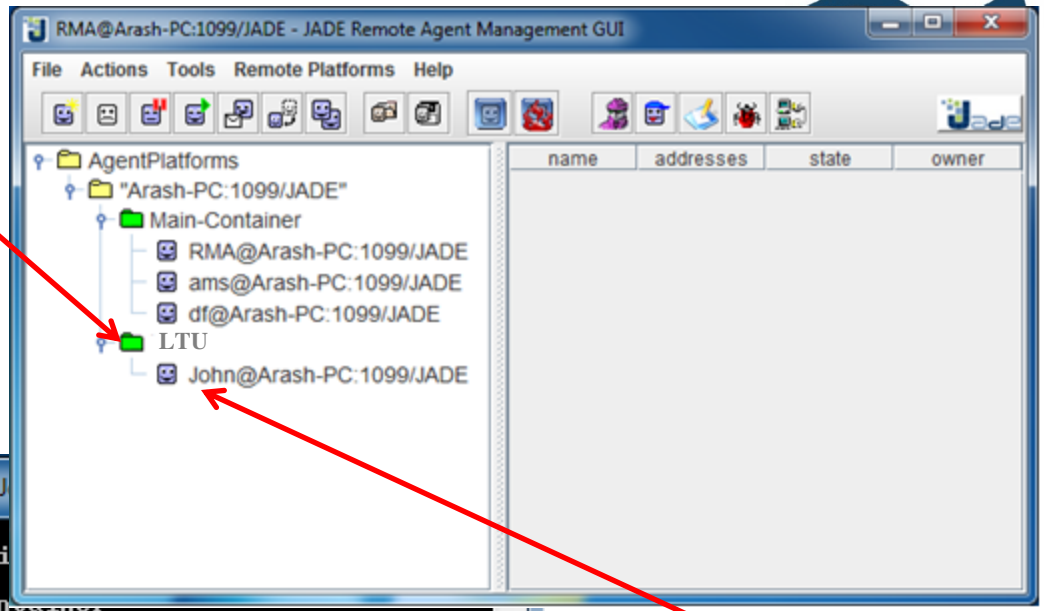
*java jade.Boot –container –container-name LTU Smith:HelloWorldAgent*

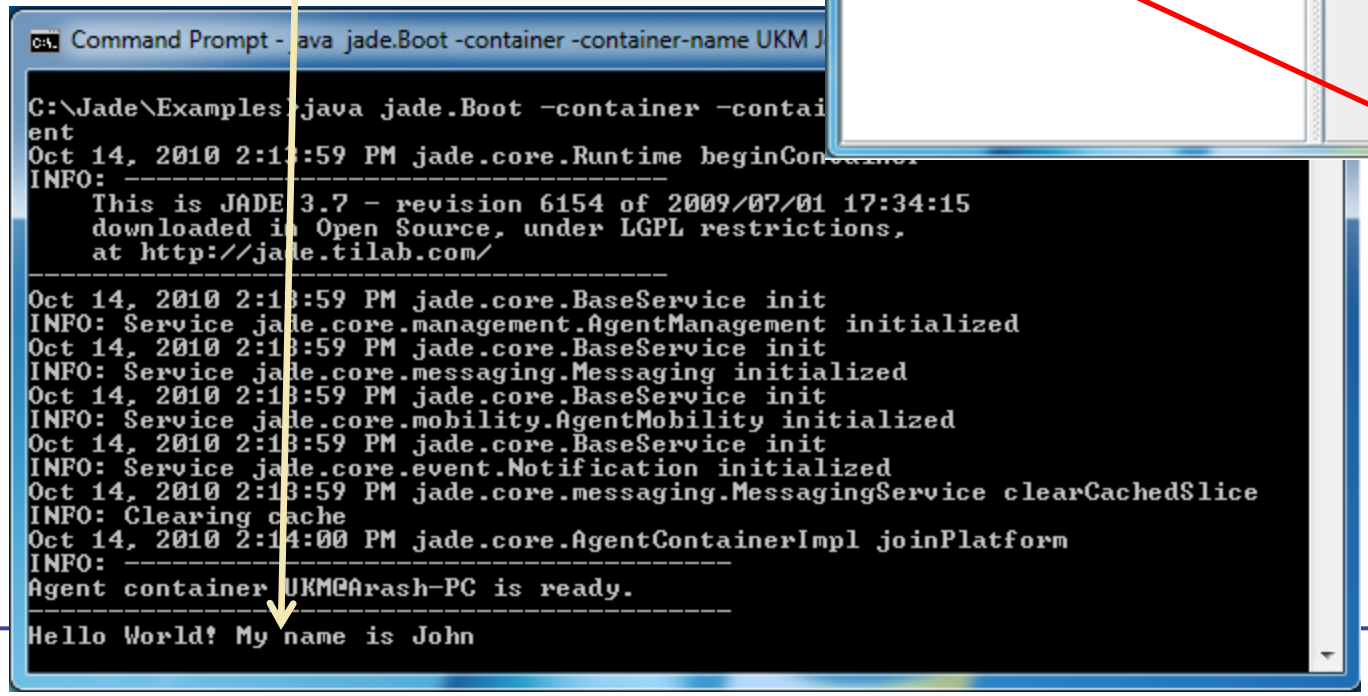
*(Creating an Agent in Main Container or automatic Naming for the Container or assign a name to the Container)*

# HelloWorldAgent output

Container LTU
is Created

Agent John
Introducing Himself



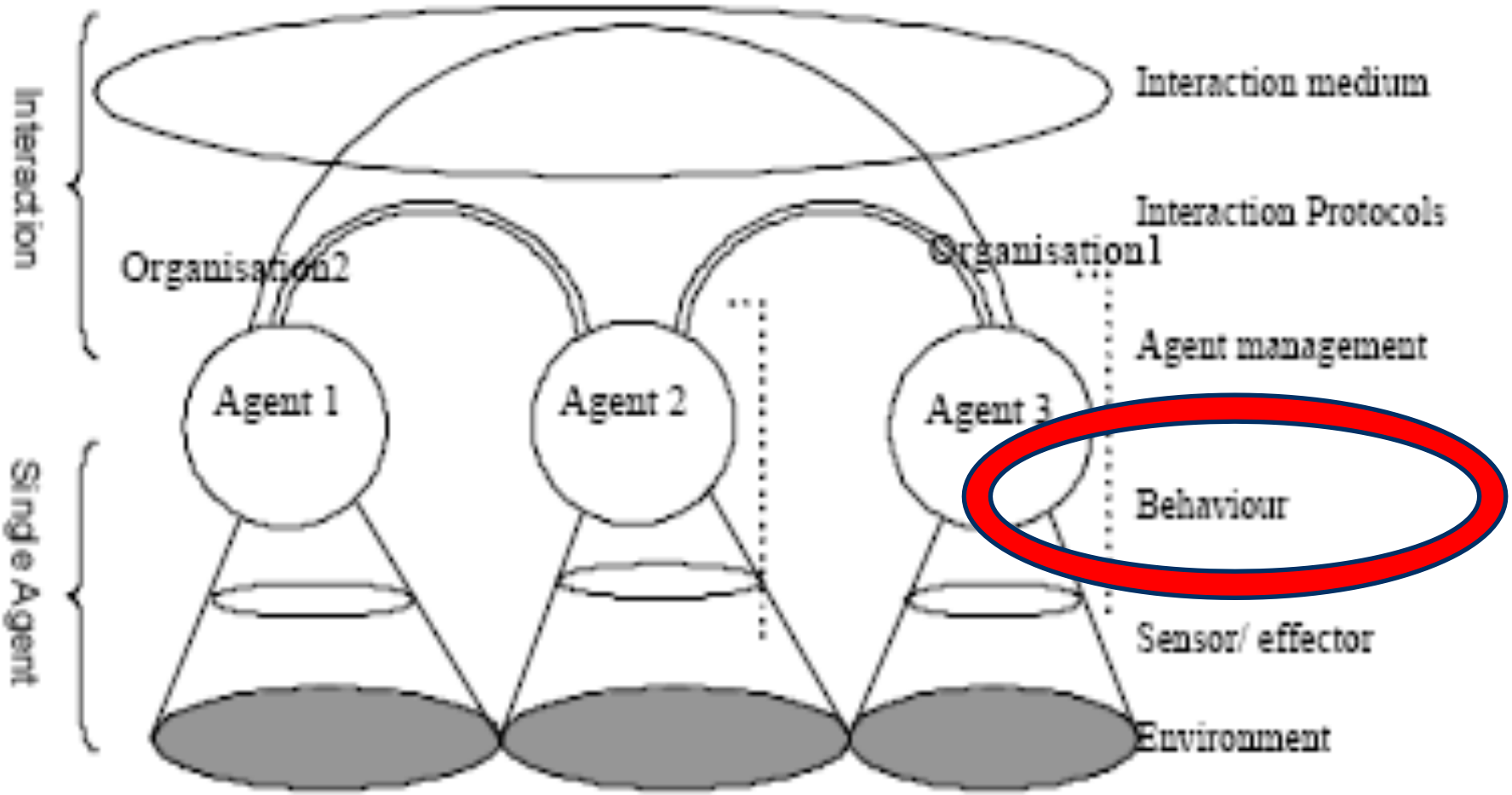

Agent John is
Created

# Accessing Agent's start parameters

```
protected void setup() {
Object[] args = getArguments();
if(args!=null && args.length>0){
for (int i=0; i<args.length; i++){
System.out.println(args[i].toString());
}
}
```

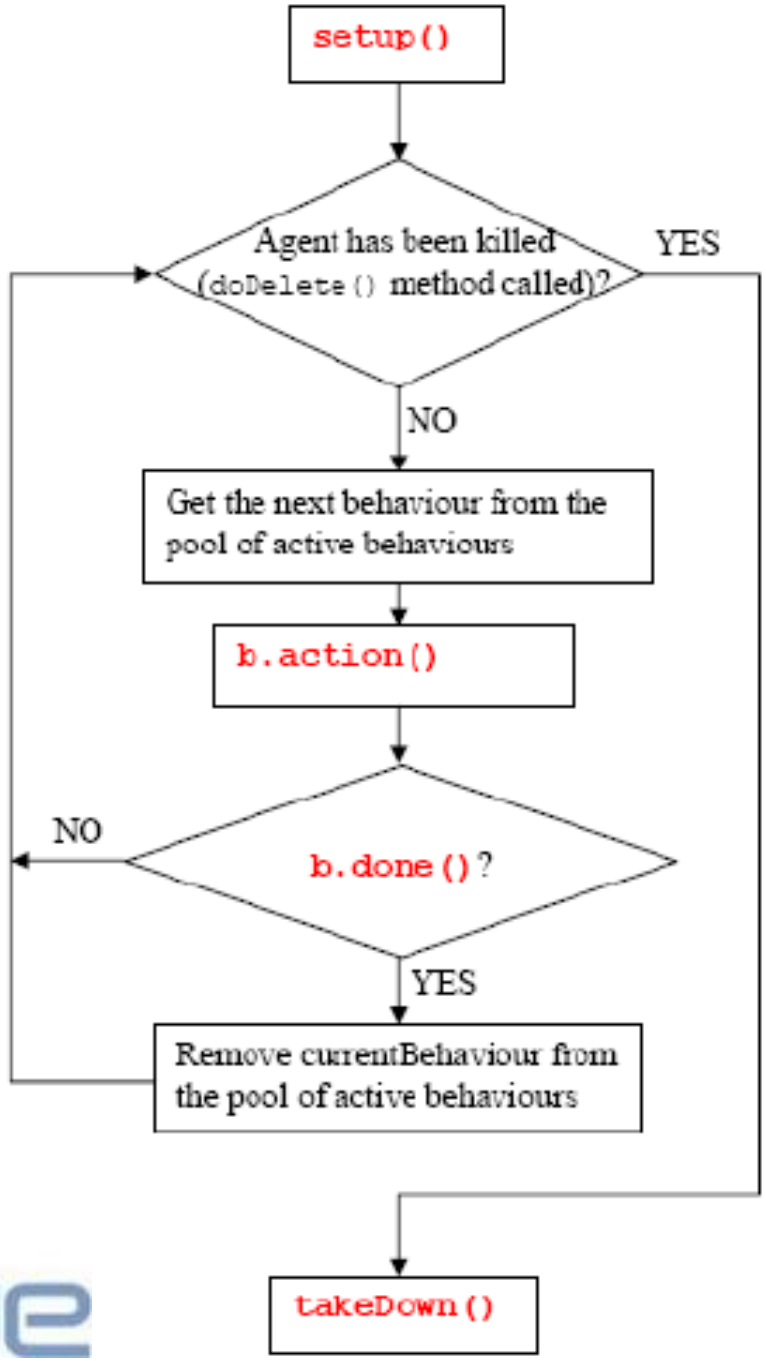
Parameters are passed from the command line starting JADE, commaseparated,
e.g. *john:HelloWorldAgent(parameter1,parameter2);*

# MAS architecture



# Behavior execution

- Each agent has its own thread (process in Java Virtual Machine).
- Agent's core uses simple queue for sequential execution of behaviors.
- This means that until *action() of the* current behavior returns, no other behavior can be executed.
- If behavior is supposed to have more iterations/steps, its *done() method has* to return *false. Then, the behavior will* be put in the end of the queue and its action will be called again when time comes.
- If behavior's *done() returns true, the* behavior is removed as finished.



# Standard Behavior classes

- **SimpleBehaviour – basic behavior class, the programmer has to**
  - implements both *action() and done()*
- **OneShotBehaviour – behavior with one iteration/step only, its *done()***
  - always returns *true*
- **CyclicBehaviour – never-ending behavior, its *done() always returns false***
- **TickerBehaviour - periodically executes a user-defined piece of code, has**
  - *action() already implemented, programmer writes onTick()*
- **WakerBehaviour - executed only once just after a given timeout is**
  - elapsed, has *action() already implemented, programmer writes onWake()*
- **ParallelBehaviour – composite behavior whose sub-behaviors are**
  - executed in parallel, finishes when all, any, or specified number of children finishes.
- **SequentialBehaviour - composite behavior whose sub-behaviors are**
  - executed sequentially.

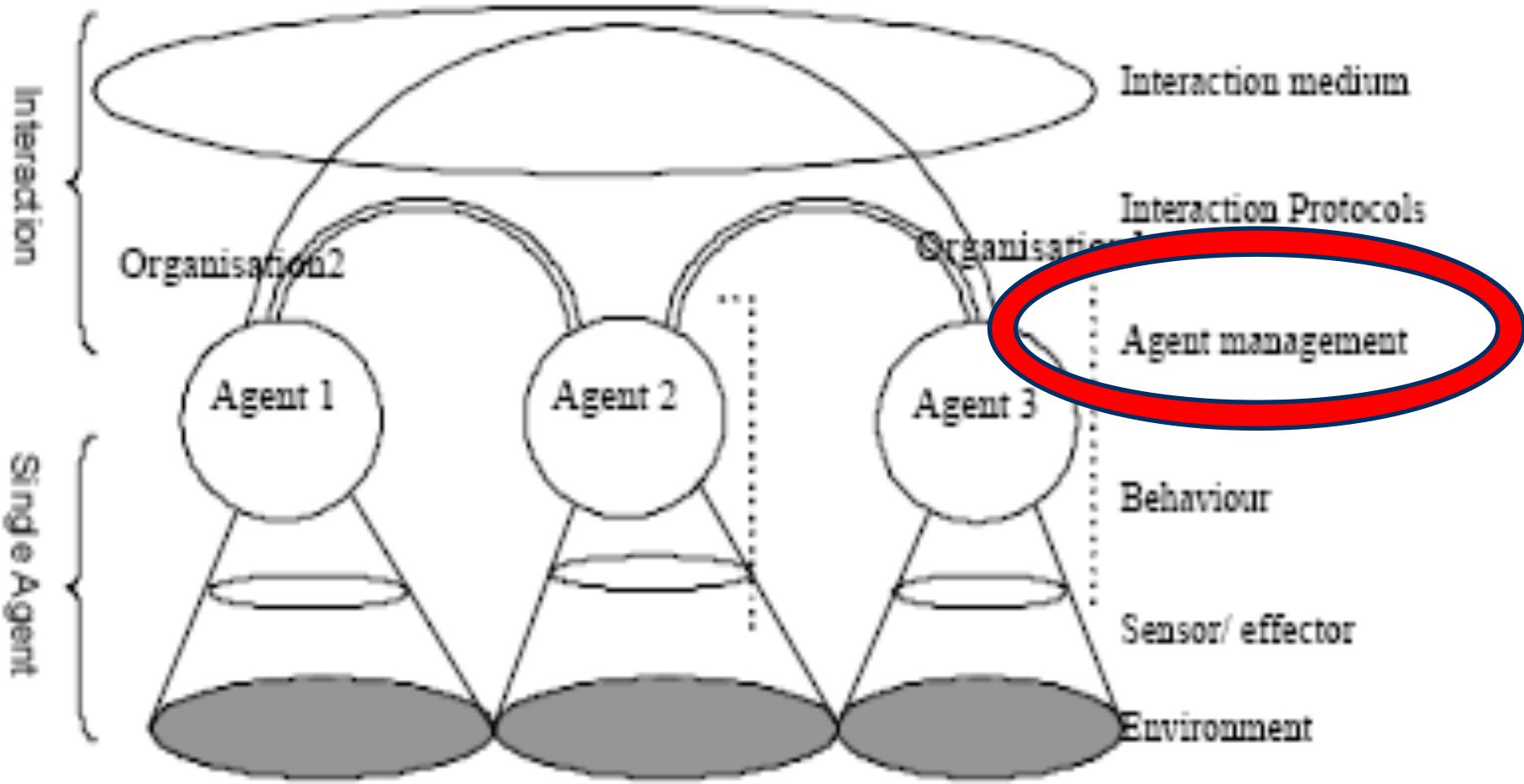
# JADE Agent with Behavior

- Instead of putting the actions to be performed inside *setup(), they may be* put inside an *agent behavior.*

➢ JADE provides class jade.core.behaviours.Behaviour and several subclasses of it, e.g. OneShotBehaviour, CyclicBehaviour.

```
import jade.core.Agent;
import jade.core.behaviours.OneShotBehaviour;
public class HelloWorldAgent extends Agent {
protected void setup() {
    addBehaviour(new OneShotBehaviour() {
        public void action(){
            System.out.println("Hello World! My name is "+getLocalName());
        }
        });
}
```

# MAS architecture



# Searching with DF

```
import jade.domain.DFService;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.FIPAAgentManagement.ServiceDescription;
import jade.domain.FIPAException;
…
DFAgentDescription template = new DFAgentDescription();
ServiceDescription sd = new ServiceDescription();
sd.setName(”Seller”);
sd.setType(”Book-Trading");
template.addServices(sd);
try {
DFAgentDescription[] result = DFService.search(myAgent, template);
for(int i=0; I < result.length; i++){
String name=result[i].getName().getLocalName();
…
}
}catch (FIPAException fe) {…}
```

# Mobility

- **Using JADE we can move an Agent from one Platform to another or from one Container to another in the same machine or platform.**

```
import jade.core.ContainerID;
import jade.core.Location;
…
Location loc = here();
if (!loc.getName().equals("Main-Container")){
doMove( new ContainerID("Main-Container", null) );
}
…
protected void beforeMove() {}
protected void afterMove() {
System.out.println( here().getAddress() );
System.out.println( here().getName() );
}
```

- The actual transition happens after the behavior that called *doMove()* ends. *beforeMove() is called before actual transition and can be* implemented to do some preparing actions. *afterMove() is called after the* transition is complete and can be implemented to perform some actions.

# Cloning

- **Using JADE we can Clone an Agent to do exactly the same tasks as the original Agent.**

```
doClone( new ContainerID("Main-Container", null), “copyOfJohn”);
…
protected void beforeClone() {
…
}
protected void afterClone() {
}
```

- The actual cloning happens after the behavior that called *doClone() ends. beforeClone() is called for mother agent before actual cloning and can be* implemented to do some preparing actions. *afterClone() is called for* daughter agent after the cloning is complete and can be implemented to perform some actions.

# Killing

- **Using JADE we can Clone an Agent to do exactly the same tasks as the original Agent.**

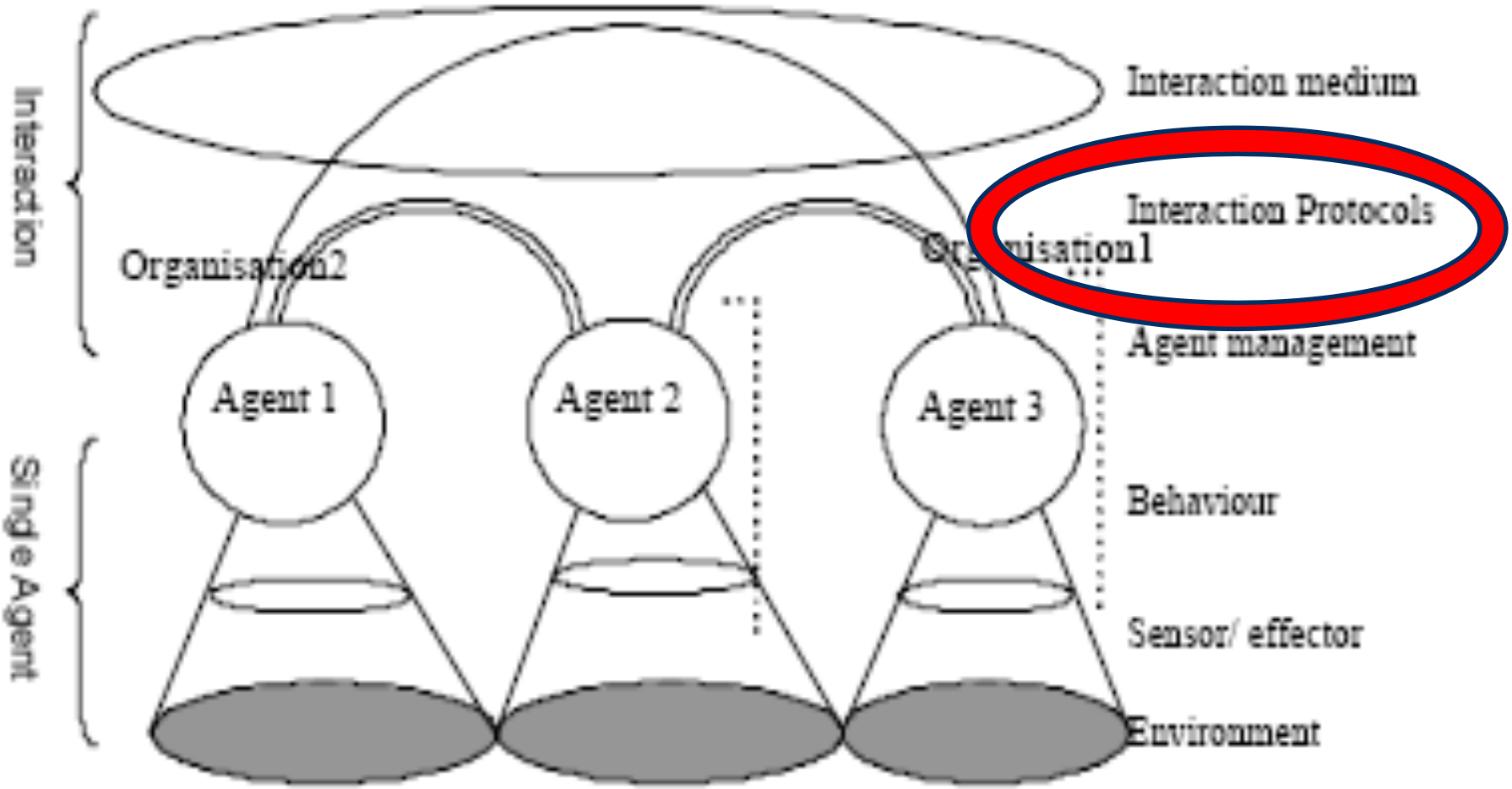
- ***Killer***

```
AgentContainer container = myAgent.getContainerController();
AgentController victim = container.getAgent("John");
victim.kill();
```

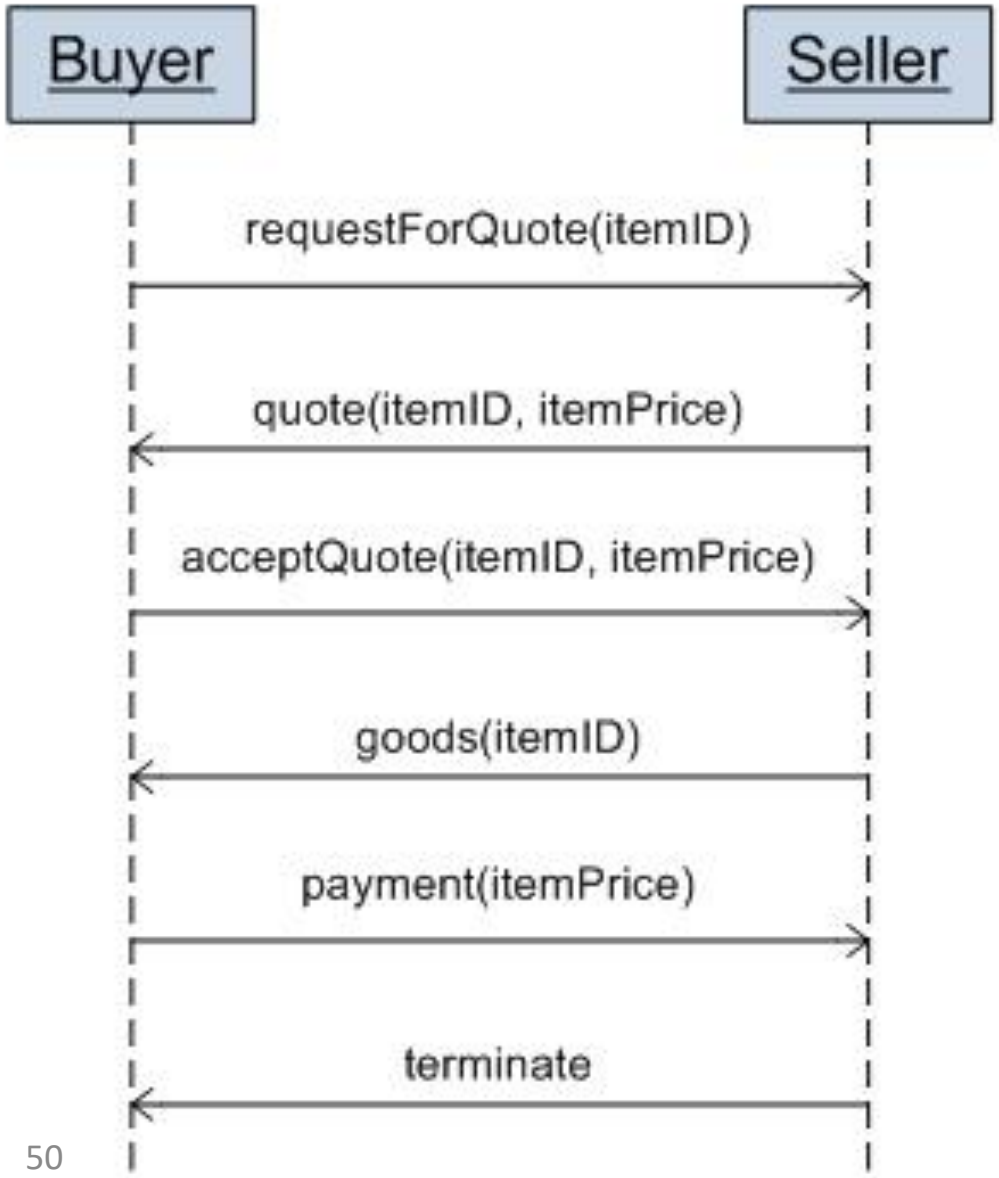
- ***Kamikaze (destroy the container and all the agents in it, including oneself)***

```
AgentContainer container = getContainerController();
container.kill();
```

# MAS architecture



# Agent Communication



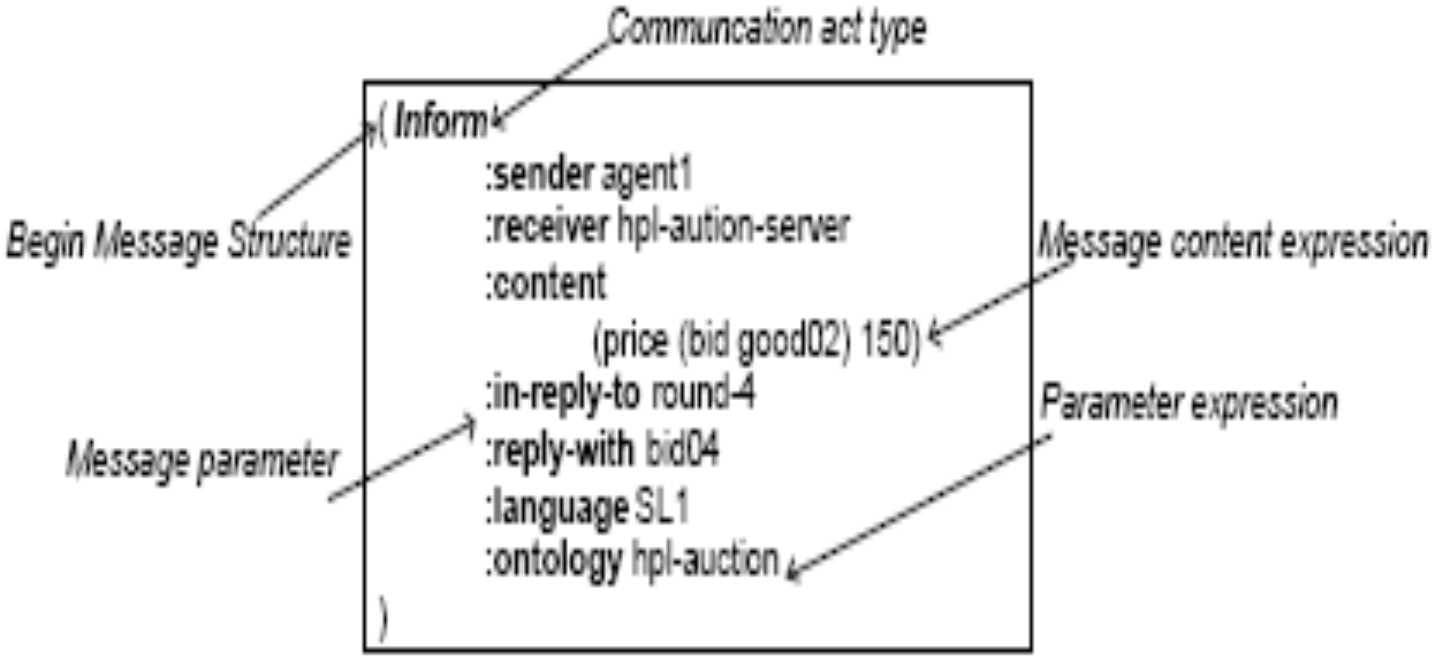

- **Agent communication Language**
- **A declarative language:** provides Logical Communication
- **FIPA**( Foundation for Intelligent Agents) – one of the standards describing a possible format and protocol
- **ACLs** demonstrate three key elements:
  - Common agent communication language and its protocol,
  - Common format for **content** of communication

# Message Syntax

Envelope/content

- Envelope can be read by all agents in the society
- Content is usually understood only by specialized agents

# Example FIPA: Performative




- Common ACL protocol is known as *performatives such as* send, accept, reject, etc. that mimic human of communication (refer FIPA specification)

# Common format content of communication

- The common format content of communication and shared ontology exhibits the openness of agent interaction.
- The common format content of communication describes language for interpreting the information in the content field of the message, called ‘ontolingua', whereas *ontology identifies the ontology to interpret the information in the content field of the message.*
- The content field provides the exact content of information to send or receive.

# Ontology - FYI

- *Ontologies are defined as specification schemes for describing concepts and their relationships* in a domain of discourse
- Ontolingy is often mentioned in the literature as a system that provides a vocabulary for the definition of reusable, portable and shareable ontologies.
- Ontolingy definitions are described using syntax and semantics similar to those of the Knowledge Interchange Format  also known as KIF, which is a format to standardised knowledge representation schemes based on first-order logic.

# Sending an ACL message

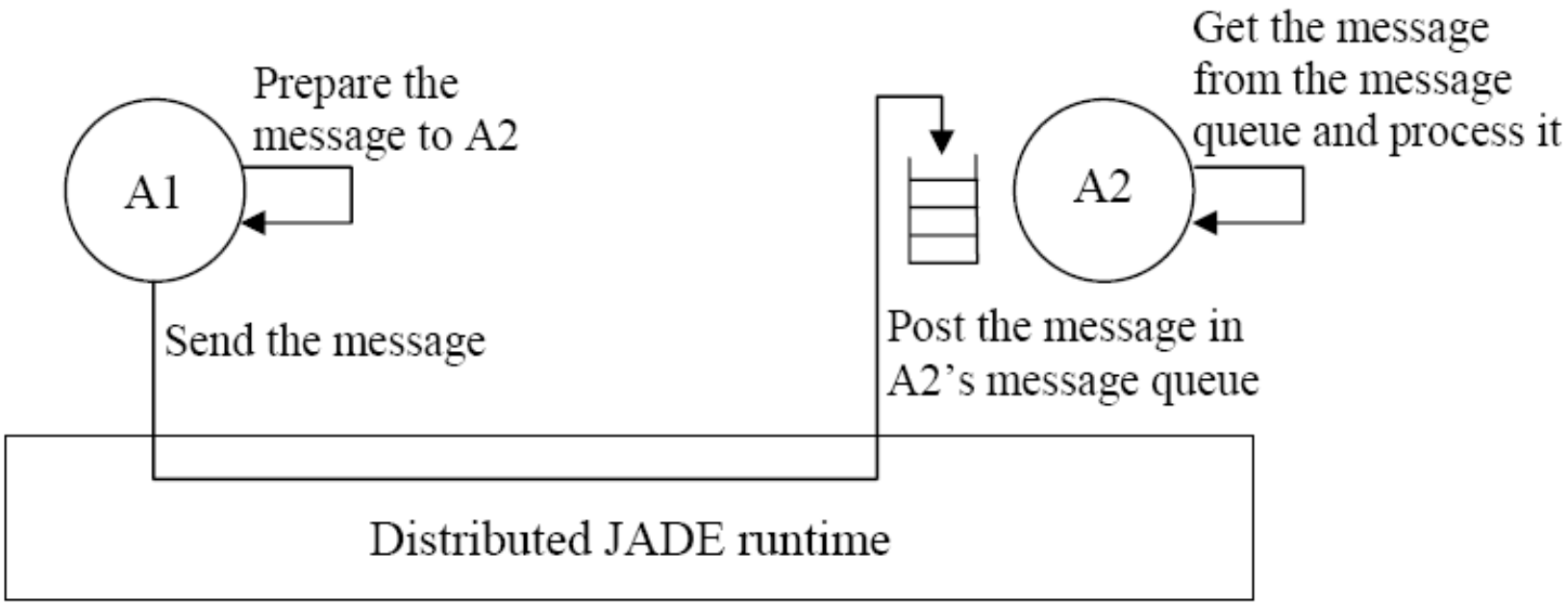
- Class *AID – enables working with Agent Identifiers according to FIPA* standards (includes localname, platform name, ACC addresses). For messaging inside an agent platform, local name is enough.

- Class *ACLMessage – enables working with FIPA's ACL Messages. Can* set and get the whole set of ACL standard fields. Also provides *createReply() method.*

```
import jade.lang.acl.ACLMessage;
import jade.core.AID;
…
ACLMessage message = new ACLMessage(ACLMessage.INFORM);
message.addReceiver (new AID ("mary", AID.ISLOCALNAME) );
message.setContent ("Hi Mary!");
myAgent.send (message);
```

# Incoming message queue

- Jade platform posts incoming messages to the agent's queue.
- It is responsibility of one of the agent's behaviors to get messages from the queue.



# Example. Ping Agent

PingAgent is a very simple example of an agent that receives a “ping” message and then reply it with “pong”.

In order to send a message to “PingAgent”, we have created another agent called “SendPing2” Agent.

Upon running SendPing2 agent you can just enter the word “ping” in the command line and then it will be sent to PingAgent which in turn with return “pong” back to SendPing2 agent and it will displays it on the command line.

A typical commandline command to run these agents. It will also run an instance of Sniffer agent to trace the message sent between these two agents.

:>> java jade.Boot -gui “***Ping***:PingAgent; ***PingSender***:SendPing2; ***sniffer***:jade.tools.sniffer.Sniffer(***P****)”

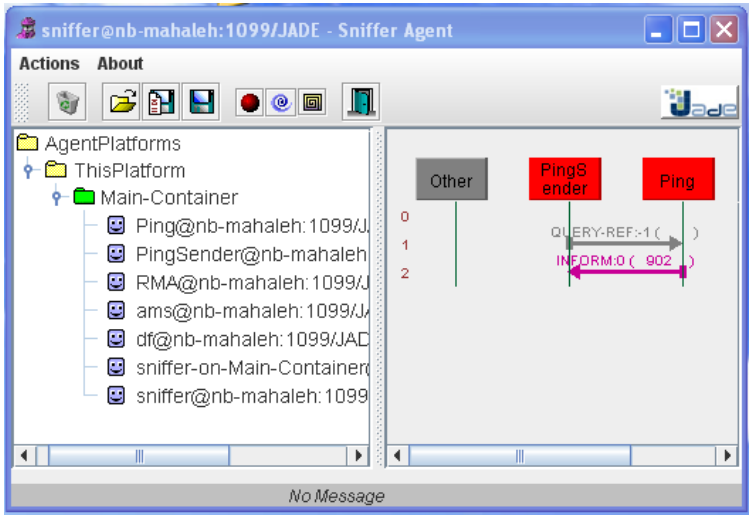
# Example (Ping Agent) Cont.. (PingAgent.java Code Review)

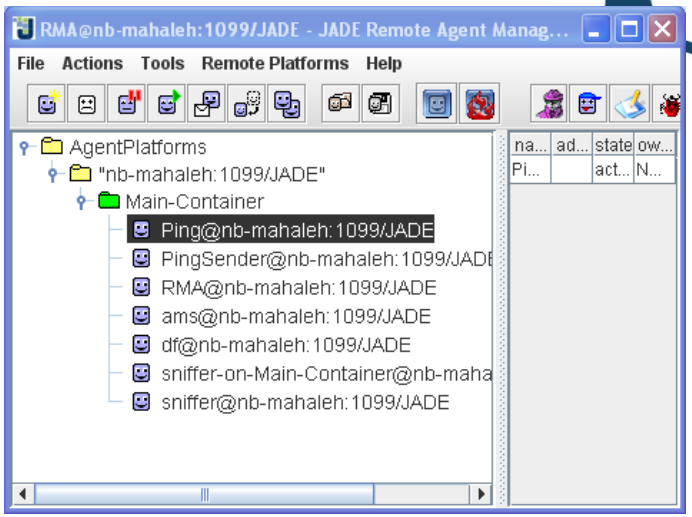
```
public WaitPingAndReplyBehaviour(Agent a) {
    super(a);
  }

  public void action() {

    ACLMessage  msg = blockingReceive();


   if(msg != null){
     if(msg.getPerformative() == ACLMessage.NOT_UNDERSTOOD)
     {

        log("Received the following message: "+ msg.toString());
        log("No reply message sent.");
     }
     else{
        log("Received the following message: "+ msg.toString());
        ACLMessage reply = msg.createReply();

          if((msg.getPerformative()== ACLMessage.QUERY_REF)||(msg.getPerformative()== ACLMessage.QUERY_IF))
        {
         String content = msg.getContent();
              if ((content != null) && (content.indexOf("ping") != -1))
         {{
           reply.setPerformative(ACLMessage.INFORM);
              reply.setContent("(pong)");
         }
         eelse
         {
           reply.setPerformative(ACLMessage.NOT_UNDERSTOOD);
           reply.setContent("( UnexpectedContent (expected ping))");
         }
        }
        else
        {
           reply.setPerformative(ACLMessage.NOT_UNDERSTOOD);
           reply.setContent("( (Unexpected-act "+ACLMessage.getPerformative(msg.getPerformative())+") ( expected (query-ref :content ping)))");
        }
        log("Replied with the following message: "+ reply.toString());

        send(reply);
     }

    }else{
      //System.out.println("No message received");
    }
  }
} //Endinner  class WaitPingAndReplyBehaviour
```

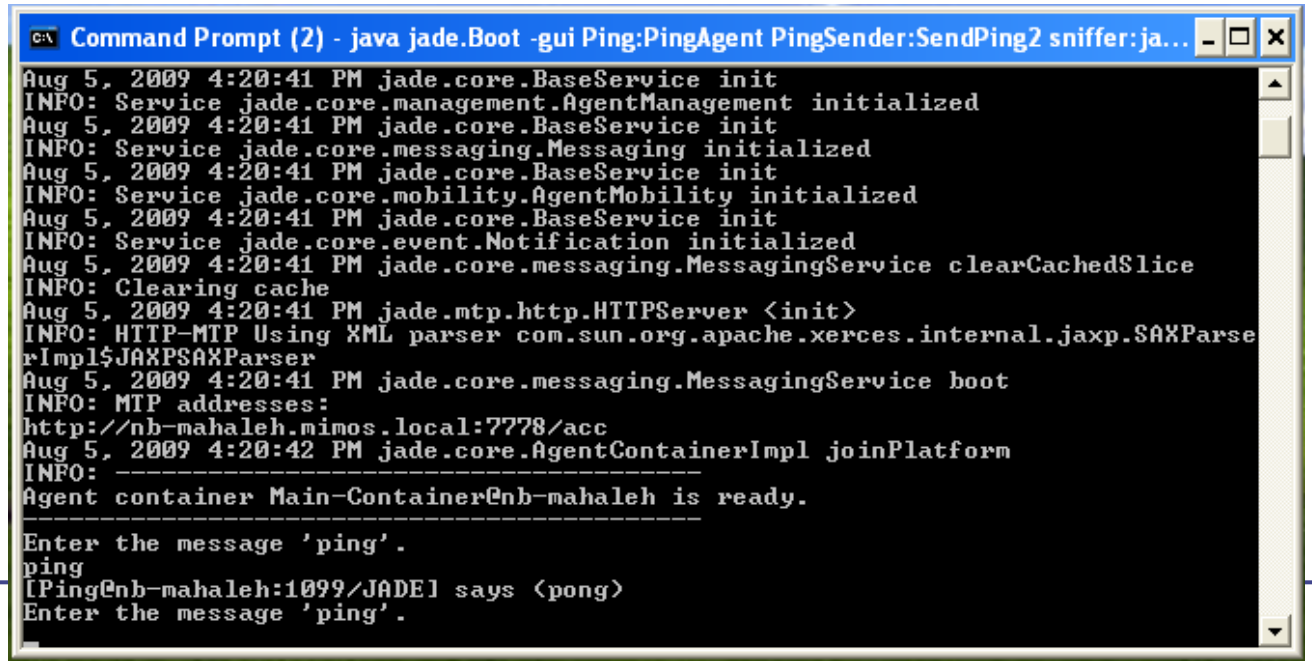
# Example (Ping Agent) Cont.. (SendPing2.java Code Review)

```
addBehaviour(new SimpleBehaviour() {

        private boolean finished = false;

        public void action() {
           System.out.println("Enter the message 'ping'.");
           String line = null;
           try {
              BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
              line = br.readLine();
           } catch (IOException ioe) {
              ioe.printStackTrace();
           }
           ACLMessage msg = new ACLMessage(ACLMessage.QUERY_REF);
           msg.setContent(line);
           msg.setSender(getAID());
           AID pingAgent = new AID("ping", false);
           msg.addReceiver(pingAgent);
           send(msg);

           msg = blockingReceive();
           if(msg != null) {
                if(msg.getPerformative() == ACLMessage.INFORM ) {
                System.out.println("[" + msg.getSender().getName()+ "] says " + msg.getContent());
              } else if(msg.getPerformative() == ACLMessage.NOT_UNDERSTOOD) {
                System.out.println("[" + msg.getSender().getName()+ "] says " + msg.getContent());
                 System.out.println("Not understood ?? This is the end!!");
                 finished = true;
               } else {
                 System.out.println("A mysterious message");
               }
           }
        } // end action()
```

# Example (Ping Agent) Cont..



# Receiving ACL message

```
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
…
MessageTemplate mt = MessageTemplate.MatchAll();
mt=MessageTemplate.and (mt,MessageTemplate.MatchPerformative
(ACLMessage.REQUEST) );
ACLMessage msg = myAgent.receive(mt);
if (msg != null){
…
}
else block();
```

- Class *MessageTemplate – enables matching performative, content,* encoding, ontology, conversation id, in-reply-to, sender and other fields using OR, AND, NOT logical operations.

# Question & Answer

# Thank You