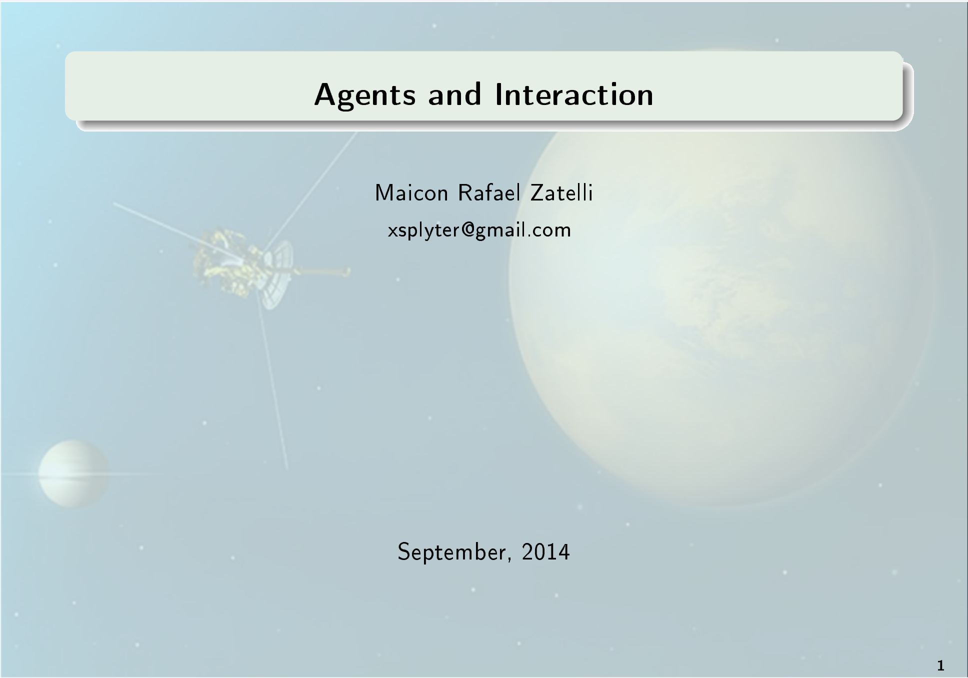


Agents and Interaction

Maicon Rafael Zatelli
xsplyter@gmail.com

September, 2014

Outline

Outline

Introduction

Communication in concurrent systems

Synchronization of multiple processes

- E.g.: solving the "lost update scenario":
- Two processes `p1` and `p2` access the shared variable `v`
- During modifying of `v` by `p1`, `p2` reads `v` and writes back the old value
- Update from `p1` is lost

Communication in OOP

Method invocation between different modules

- E.g.: object `o2` invokes method `m1` on object `o1` by executing the code `o1.m1(arg)`, where "`arg`" is the argument to communicate
- Which object makes the decision about the execution of `m1`?

Introduction

Communication in MAS

Autonomous agents encapsulate state, behaviour, and control

- Methods are executed according to the agent's self-interest
- However, agents can perform communicative actions, i.e. attempt to influence other agents
- Agent communication implies interaction, i.e. agents perform communication acts

Outline

Speech Acts

Most treatments of communication in (multi-) agent systems borrow their inspiration from speech act theory

Speech act theories are pragmatic theories of language, i.e., theories of language use: they attempt to account for how language is used by people every day to achieve their `goals` and `intentions`

The origin of speech act theories are usually traced to Austin's 1962 book, How to Do Things with Words

Speech Acts

Austin noticed that some utterances are rather like "physical actions" that appear to change the state of the world

E.g.:

- declaring war
- "I now pronounce you man and wife"

Austin identified a number of `performative verbs`, which correspond to various different types of speech acts

- Examples of performative verbs are `request`, `inform`, and `promise`

Speech Acts

Searle (1969) extended Austin's work and identified the following five key classes of possible types of speech acts:

`Representatives`: commits the speaker to the truth of an expression, e.g., "It is raining" (informing)

`Directives`: attempts to get the hearer to do something e.g., "please make the tea" (requesting)

`Commissives`: which commit the speaker to do something, e.g., "I promise to ..." (promising)

`Expressives`: whereby a speaker expresses a mental state, e.g., "thank you!" (thanking)

`Declaratives`: effect change of state, such as "declaring war" (declaring)

Outline

Synchronous and Asynchronous Communication

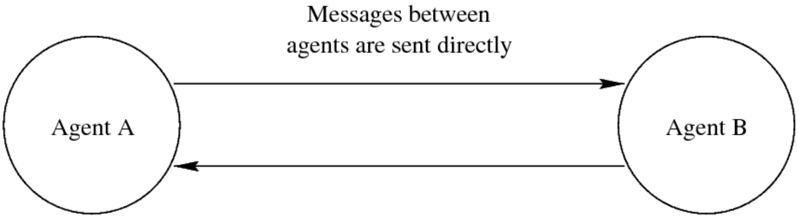
`Synchronous`: the agent `waits for` the reply of a message to proceed with the execution

- E.g.: the agent asks the price of a product in order to proceed with the payment

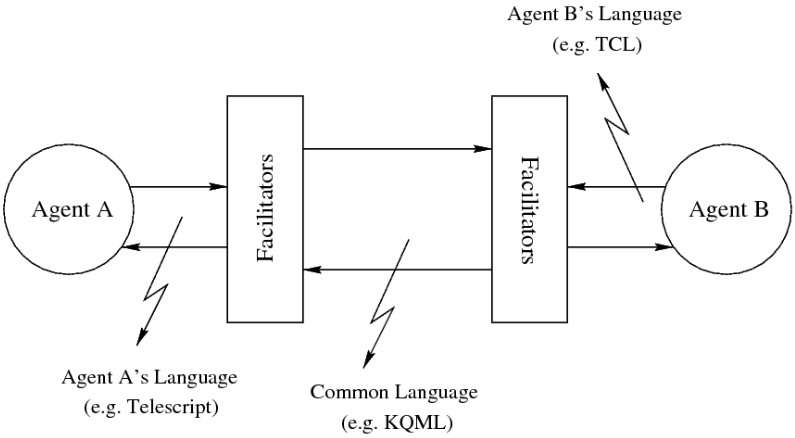
`Asynchronous`: the agent `does not wait for` the reply of a message to proceed with the execution

- E.g.: the agent asks the current time while it is “walking” in some place. It does not need to wait the reply to continue walking

Direct Interaction Among Agents

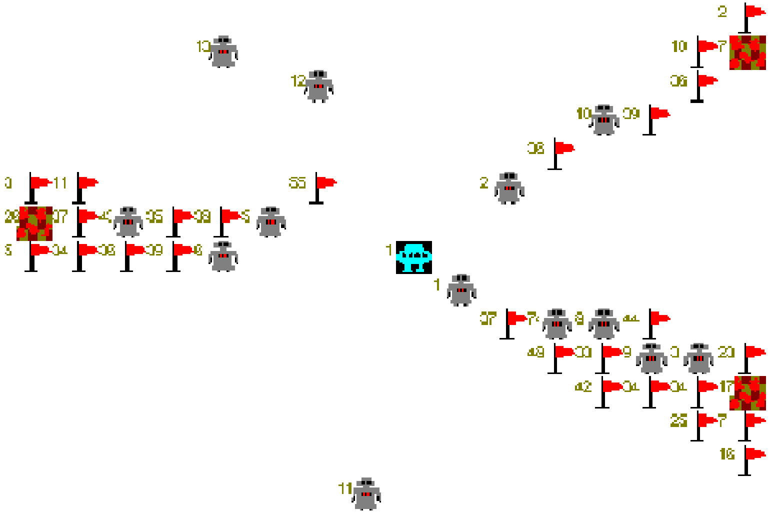


Indirect Interaction Among Agents



Indirect Interaction Among Agents

Stigmergy, overhearing, eavesdropping



Outline

Ontology

Ontologies ground the terminology used by the agents

E.g.:

An agent wants to buy a screw. But what means then "`size`"? Is it in `inch` or `centimeter`?

Ontology

An ontology formally represents knowledge as a hierarchy of concepts within an area of interest, using a shared `vocabulary` to denote the `types`, `properties` and `relationships` of those concepts.

E.g.:

$$\forall x \; (Block \; x) \;\Rightarrow\; (PhysicalObject \; x)$$

Outline

Outline

KQML

KQML (*Knowledge Query and Manipulation Language*) is an `outer` (envelope) language, that defines various acceptable `communicative verbs`, or `performatives`

E.g.:

- `ask-if` ("is it true that ... ")
- `perform` ("please perform the following action ...")
- `tell` ("it is true that ... ")
- `reply` ("the answer is ... ")

Example KQML 1

```
(ask-one
  :sender agent1
  :receiver agent2
  :content  (price ibm ?price)
  :language LPROLOG
  :ontology NYSE-TICKS
)
```

Example KQML 2

```
(reply
  :sender agent2
  :receiver agent1
  :content  (prive ibm 150)
  :language LPROLOG
  :ontology NYSE-TICKS
)
```

Outline

KIF

KIF (*Knowledge Interchange Format*) is a language for expressing message `content`

KIF allows agents to express:

- `properties` of things in a domain, e.g., “Michael is a vegetarian”
- `relationships` between things in a domain, e.g., “Michael and Janine are married”
- `general properties` of a domain, e.g., “All students are registered for at least one course” (quantification $\forall$)

Examples KIF

The temperature of m1 is 83 Celsius

```
(= (temperature m1) (scalar 83 Celsius))
```

An object is a bachelor if the object is a man and is not married

```
(defrelation bachelor (?x) := (and (man ?x)
                                   (not (married ?x))))
```

Any individual with the property of being a person also has the property of being a mammal:

```
(defrelation person (?x) :=> (mammal ?x))
```

Outline

FIPA-ACL

FIPA is the organization for developing standards in multiagent systems. It was officially accepted by the IEEE at its eleventh standards committee in 2005.

FIPA's goal in creating agent standards is to promote `interoperable agent applications` and agent systems Basic structure and concepts are very similar to KQML:

- `performative` (22 performatives[1] in FIPA)
- `housekeeping` (e.g., sender, receiver, etc.)
- `content` (the actual content of the message)

[1] http://www.fipa.org/specs/fipa00037/SC00037J.html

Example FIPA-ACL

```
(inform
  :sender agent1
  :receiver agent5
  :content (price good200 150)
  :language sl
  :ontology hpl-auction
)
```

FIPA-ACL Performatives

- accept-proposal
- agree
- cancel
- cfp
- confirm
- disconfirm
- failure
- inform
- inform-if
- inform-ref
- not-understood
- propagate
- propose
- proxy
- query-if
- query-ref
- refuse
- reject-proposal
- request
- request-when
- request-whenever
- subscribe

Outline

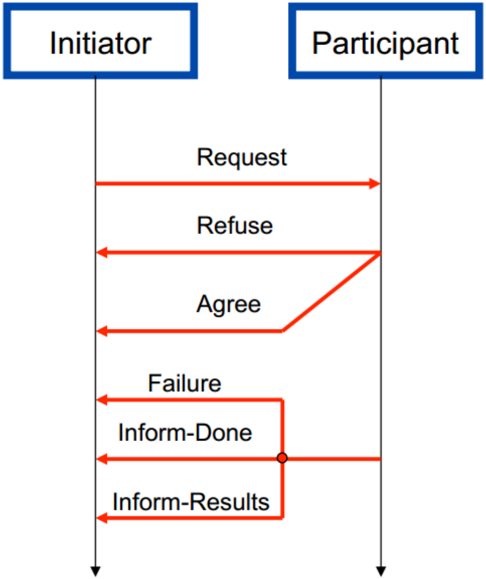
Interaction Protocol

Interaction protocols govern the exchange of a series of messages among agents - a conversation.

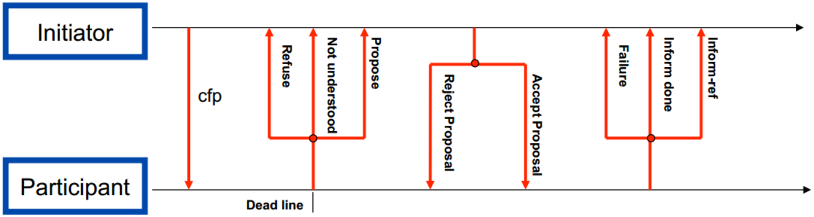
When agents have conflicting goals or are simply self-interested, the objective of protocols is to maximize the payoffs (utilities) of the agents.

When agents have similar goals or common problems, the objective of protocols is to maintain global globally coherent performance of the the agents without violating autonomy.

FIPA Request Protocol



FIPA Contract-Net Protocol



Outline

JADE

JADE is a framework for development of multi-agent systems

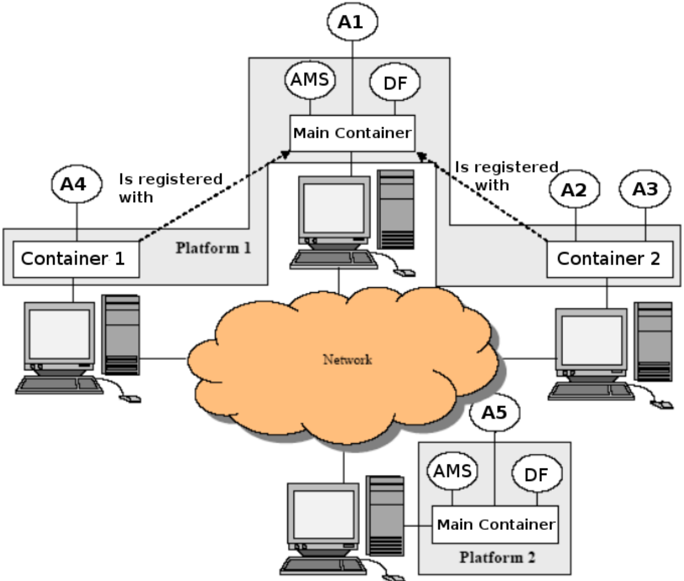
It implements the structure model of FIPA, with

- white & yellow-page services
- mechanisms for message exchanges
- agent security and mobility
- scheduling of multiple agent tasks

Oriented to tasks: the behaviour of the agents is decomposed in smaller pieces (`Behaviour`), which are added to the agent when necessary

Provides a set of graphical tools to support monitoring, logging, and debugging

JADE - Containers and Platforms



JADE - Containers and Platforms

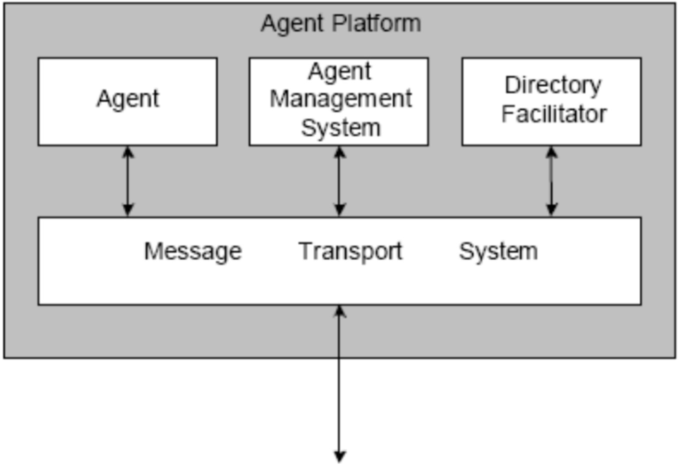
`Container`

- Instance of a JADE environment
- It's where the agents execute
- A `Main Container` is created as soon as JADE is started

`Platform`

- It's a set of active containers

JADE - The Platform



JADE - AMS and DF

`AMS (Agent Management System) - White Page`

- Agent that has the control of access and use of the platform
- There is only one AMS in each platform
- Keeps the list of identifiers of agents (AID) that are in the platform
- All agents must be registered in the AMS

`DF (Directory Facilitator) - Yellow Page`

- Provide the yellow pages service in the platform

JADE - Agent Class

It's the base class for agent definition

Each JADE agent is an instance of a class that extends the Agent class.

Provides all basic interactions with the platform (register, configuration, etc)

Provides a set of methods to implement the agent's behaviour

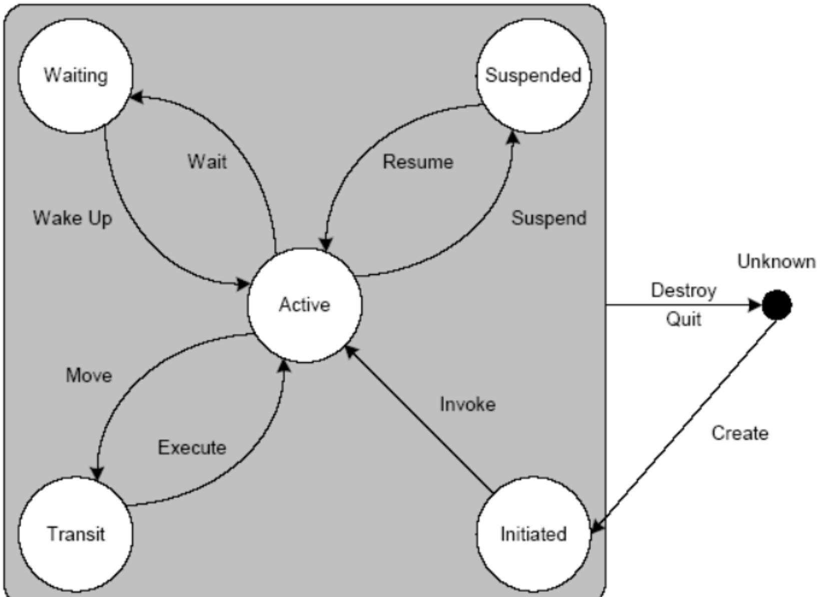
JADE - Agent Computational Model

An agent is multi-task, where the services are executed concurrently

Each service of an agent must be implemented as one or more behaviours

The Agent class provides a scheduler (not accessible to the developer), which automatically manages the scheduling of the behaviours

JADE - Life Cycle



JADE - Life Cycle

`Initiated` - The agent object is created, but still not registered in the AMS (i.e. it does not own an identifier and cannot communicate with other agents)

`Active` - The agent object is registered in the AMS, owns an identifier and can execute its services

`Suspended` - The agent object is stopped (i.e. its internal thread is suspended and the agent is not executing any service)

JADE - Life Cycle

`Waiting` - the agent object is blocked, waiting for some event (i.e. its internal thread is sleeping and will wake up when some condition becomes true)

`Transit` - an mobile agent is in this state when it is migrating to a new container. The system continues storing messages sent to this agent, which will be delivered to it when the migration is completed.

`Removed` - The agent object is finished (i.e. its internal thread finished the execution and the agent is no longer registered in the AMS)

JADE - Creating an JADE Agent

JADE manages the creation of a new agent with the following steps:

- The constructor of the agent is executed
  - The agent receives an identifier from the MAS (e.g. `agente@localhost:1099/JADE`)
  - The agent becomes `Active`
- The method `setup()` is executed
  - This method is responsible for initializing the agent behaviours

JADE - Creating an JADE Agent

```
import jade.core.Agent;
public class HelloWorldAgent extends Agent {
  protected void setup() {
    // Mostra uma mensagem de Hello
    System.out.println("Hello World! My name is "
                          + getAID().getName());
  }
}
```

JADE - Destroying an JADE Agent

Even if the agent is not doing anything, the agent continues executing

To finish an agent, it must be executed the method `doDelete()`, which calls the method `takeDown()`

Thus, all references of the agent in the platform are removed

JADE - Destroying an JADE Agent

```
import jade.core.Agent;
public class HelloWorldAgent extends Agent {
  protected void setup() {
    // Mostra uma mensagem de Hello
    System.out.println("Hello World! My name is "
                       + getAID().getName());
    doDelete();
  }

  protected void takeDown() {
    // Imprimindo uma mensagem de saida
    System.out.println("Agent "
                       + getAID().getName()
                       + " finishing.");
  }
}
```

Outline

JADE - Behaviours

All tasks of the agents are executed by means of `behaviours`

A behaviour is an object of the `Behaviour` class

The agent adds a behaviour by means of the method `addBehaviour()`

Behaviours can be added at any moment

- In the method `setup()`
- Inside other behaviours

JADE - Behaviours

Method `action()`

- This method defines the operations that are executed when the behaviour is executing

Method `done()`

- This method specifies whether a behaviour was completed and be removed of the pool of behaviours that the agent is executing

JADE - Behaviours

The agent can execute several behaviours concurrently

The scheduling of behaviours is not preemptive

When a behaviour is scheduled for the execution, the method `action()` is called and executes until it returns

The developer must define when the execution must pass from one to another behaviour

JADE - Behaviours

```
public class Behaviour1 extends Behaviour {
  public void action() {
    while (true) {
      // behaviour's code
    }
  }

  public boolean done() {
    return true;
  }
}
```

JADE - Kinds of Behaviour

`OneShotBehaviour` - Executes just once and cannot be blocked

`CyclicBehaviour` - Executes forever

`SequentialBehaviour` - Executes a set of sub-behaviours sequentially and finishes when all sub-behaviours are executed

`ParallelBehaviour` - Executes a set of sub-behaviours concurrently and finishes when a condition about the sub-behaviours is true

JADE - Kinds of Behaviour

`TickerBehaviour` - Executes something periodically

`WakerBehaviour` - Waits for a certain amount of time (in ms) to actually execute

`FSMBehaviour` - Executes a set of sub-behaviours like a finite-state machine. The output of a sub-behaviour is used to calculate the transition to the next sub-behaviour. This behaviour finishes when the final sub-behaviour is executed

JADE - Example Behaviour

```
public class OneShotAgent extends Agent {
  protected void setup() {
    addBehaviour(new OneShotBehaviour() {
      public void action() {
        // Execute operation Z
      }
    } );
  }
}
```

The operation `Z` will be executed only once

JADE - Example Behaviour

```
public class CyclicAgent extends Agent {
  protected void setup() {
    addBehaviour(new CyclicBehaviour() {
      public void action() {
        // Execute operation W
      }
    } );
  }
}
```

The operation `W` will be executed forever in each cycle of the agent

JADE - Example Behaviour

```
public class WakerAgent extends Agent {
  protected void setup() {
    System.out.println("Added waker behaviour");
    addBehaviour(new WakerBehaviour(this, 10000) {
      protected void handleElapsedTimeout() {
        // Execute operation X
      }
    } );
  }
}
```

The operation `X` will be executed in 10s after the message "*Added waker behaviour*" be printed

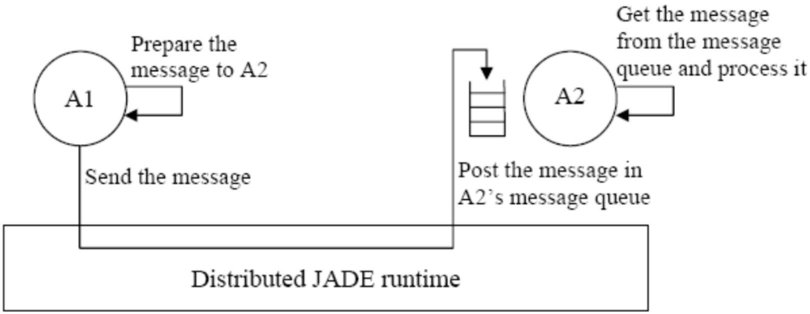
JADE - Example Behaviour

```
public class TickerAgent extends Agent {
  protected void setup() {
    addBehaviour(new TickerBehaviour(this, 10000) {
      protected void onTick() {
        // Execute operation Y
      }
    } );
  }
}
```

The operation `Y` will be executed each 10s

Outline

JADE - Communication


Prepare the message to A2
A1
Send the message
Post the message in A2's message queue
A2
Get the message from the message queue and process it
Distributed JADE runtime

JADE - Send Message

- Create the `ACLMessage` object
- Call the `send()` method

```
ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
msg.addReceiver(new AID("Ag2", AID.ISLOCALNAME));
msg.setLanguage("English");
msg.setOntology("Weather-forecast-ontology");
msg.setContent("Today it’s raining");
send(msg);
```

JADE - Receive Message

- Call the `receive()` method
- This method removes a message of the stack or returns null if the stack is empty

```
ACLMessage msg = receive();
if (msg != null) {
  // process the message
}
```

JADE - Receive Message

The scheduler of actions does not stop the execution if there is no messages for an agent

In order to stop the execution of an action to wait for a message, it must be used the `block()` method

```
ACLMessage msg = receive();
if (msg != null) {
  // process the message
} else {
  block();
}
```

JADE - Example Communication

```
public class ServerAgent extends Agent {
  protected void setup() {
    addBehaviour(new CyclicBehaviour() {
      public void action() {
        ACLMessage msg = myAgent.receive();
        if (msg != null) {
          System.out.println(msg.getSender().getName() + ": "
                                   + msg.getContent());
        } else block();
      }
    } );
  }
}
```

JADE - Example Communication

```
public class ClientAgent extends Agent {
  protected void setup() {
    addBehaviour(new OneShotBehaviour() {
      public void action() {
        ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
        msg.addReceiver(new AID("tom", AID.ISLOCALNAME));
        msg.setLanguage("English");
        msg.setOntology("Weather-forecast-ontology");
        msg.setContent("Today it's raining");
        send(msg);
      }
    } );
  }
}
```

JADE - Filtering Message

It is possible to select a kind of message using `MessageTemplate`

```
public void action() {
  MessageTemplate mt;
  mt = MessageTemplate.MatchPerformative(ACLMessage.CFP);
  ACLMessage msg = myAgent.receive(mt);
  if (msg != null) {
    // Message of kind CFP received and being processed
  } else block();
}
```

JADE - Filtering Message

MessageTemplate.MatchSender

MessageTemplate.MatchContent

MessageTemplate.MatchLanguage

MessageTemplate.MatchOntology

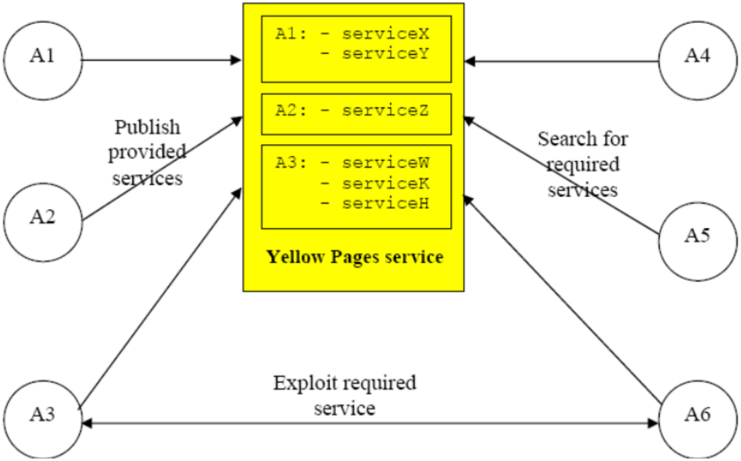
...

Outline

JADE - Yellow Pages

Each agent must register its service in the DF

There is only one DF in each JADE platform



JADE - Yellow Pages - Registering

It must be created a `ServiceDescription` object and call the `register()` method of a DF

Usually made in the `setup()` method

```
protected void setup() {
  DFAgentDescription dfd = new DFAgentDescription();
  dfd.setName(getAID());

  ServiceDescription sd = new ServiceDescription();
  sd.setType("book-selling");
  sd.setName("JADE-book-trading");

  dfd.addServices(sd);

  try { DFService.register(this, dfd); }
  catch (FIPAException fe) { fe.printStackTrace(); }
}
```

JADE - Yellow Pages - Removing register

The method `deregister()` of the DF must be called

Usually made in the `takeDown()` method

```
protected void takeDown() {
  try { DFService.deregister(this); }
  catch (FIPAException fe) { fe.printStackTrace();
}
```

JADE - Yellow Pages - Searching

It must be created a `DFAgentDescription` object and call the `search()` method of a DF

```
DFAgentDescription template = new DFAgentDescription();
ServiceDescription sd = new ServiceDescription();
sd.setType("book-selling");
template.addServices(sd);

try {
  DFAgentDescription[] result =
    DFService.search(myAgent, template);

  AID sellerAgents[] = new AID[result.length];

  for (int i = 0; i < result.length; ++i)
    sellerAgents[i] = result[i].getName();
}
catch (FIPAException fe) { fe.printStackTrace(); }
```

Outline

JADE - Running

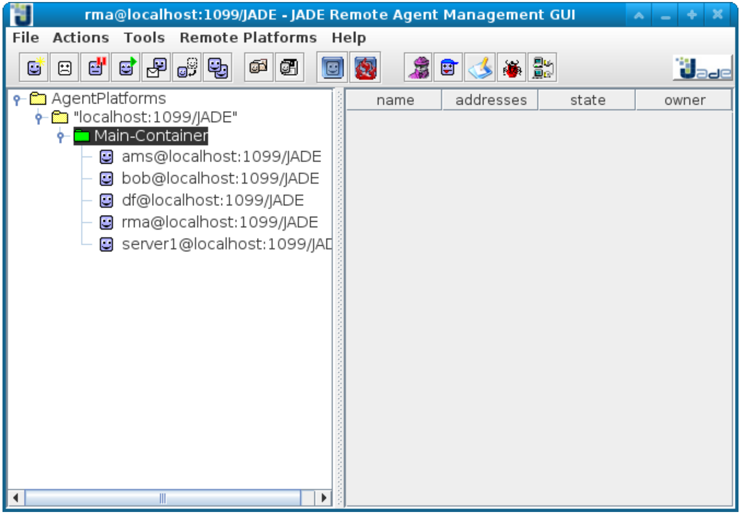
Running JADE

- E.g.: `java -cp lib/jade.jar jade.Boot -gui`

Running JADE agent

- E.g.: `java -cp lib/jade.jar:bin jade.Boot -agents simpleAgent1:agents.SimpleAgent`

JADE - GUI



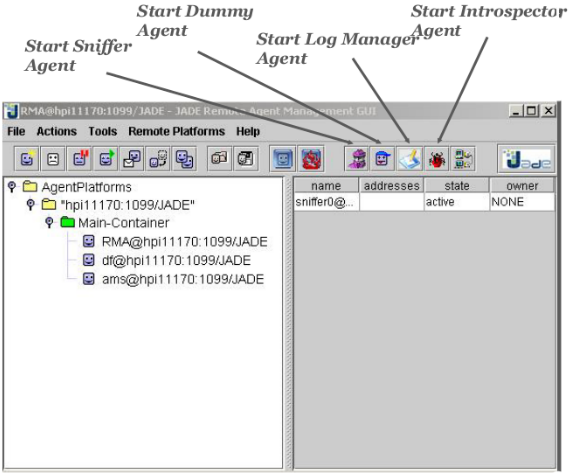
ams@localhost:1099/JADE
bob@localhost:1099/JADE
df@localhost:1099/JADE
rma@localhost:1099/JADE
server1@localhost:1099/JAD

JADE - GUI

Remote Management Agent (RMA)

Provided functionalities:

- monitor and control the platform and all its remote containers
- remote management of the life-cycle of agents (creating, suspending, resuming, killing, migrating, cloning)
- compose and send a custom message to an agent
- launch the other graphical tools
- monitor (just read operations) other FIPA-compliant platforms



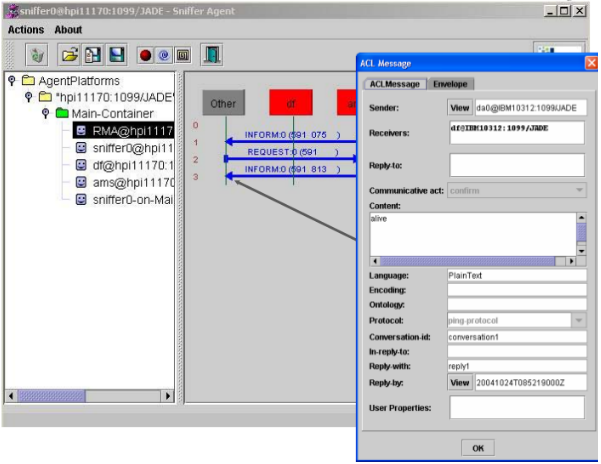
name	addresses	state	owner
sniffer0@...		active	NONE

JADE - GUI

Sniffer Agent

Functionalities:

- display the flow of interactions between selected agents
- display the content of each exchanged message
- save/load the flow on/from a file



INFORM:0 (591 075)
REQUEST:0 (591)
INFORM:0 (591 813)

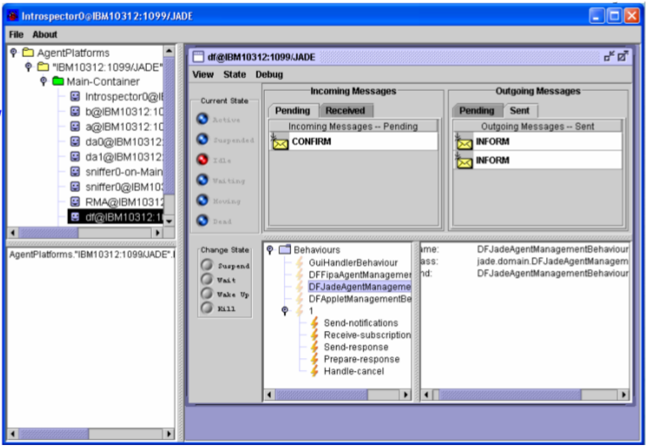
Sender:	da0@IBM10312:1099/JADE
Receivers:	df@IBM10312:1099/JADE
Reply-to:	
Communicative act:	confirm
Content:	alive
Language:	PlainText
Encoding:	
Ontology:	
Protocol:	ping-protocol
Conversation-id:	conversation1
In-reply-to:	
Reply-with:	reply1
Reply-by:	20041024T085219000Z
User Properties:	

JADE - GUI

Introspector Agent

Functionalities:

- **monitoring agent internal state**
  - **received/sent/ pending msg**
  - **scheduled behaviours (active, blocked) and sub-behaviours**
  - **agent state**
- **debugging execution**
  - **step-by-step**
  - **slowly**
  - **break points**



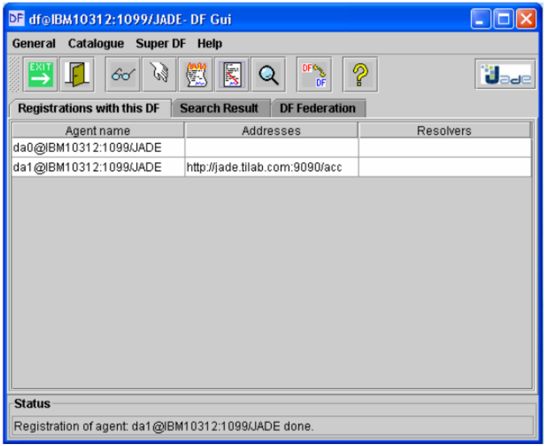
JADE - GUI

DFGUI

GUI of the yellow-page service,

it allows to:

- browse, register, deregister, modify, search agent descriptions
- federate with other DFs
- execute federated searchs



Agent name	Addresses	Resolvers
da0@IBM10312:1099/JADE		
da1@IBM10312:1099/JADE	http://jade.tilab.com:9090/acc	

Registration of agent: da1@IBM10312:1099/JADE done.

More Information

http://jade.tilab.com/

http://jade.tilab.com/documentation/tutorials-guides/

http://jade.tilab.com/doc/tutorials/JADEProgramming-Tutorial-for-beginners.pdf

http://www.fipa.org/

Review

1. Introduction
   - Speech Acts
   - Kinds of Interaction
   - Ontology
2. Communication Languages
   - KQML
   - KIF
   - FIPA-ACL
3. Interaction Protocol
4. JADE
   - Behaviours
   - Communication
   - Yellow Pages
   - Running Agents

EOF

Sources

These slides were strongly based on the slides provided in:

- Jade - Java Agent Development Framework (PUC-Rio/LES)
- Introduction to Multi-Agent Programming - Agent Communication (Alexander Kleiner, Bernhard Nebel)
- Java Agent DEvelopment Framework (JADE) (Elena Nardini)
- JADE Tutorial for beginners (Fabio Bellifemine)