AOT LAB

Agent and **O**bject **T**echnology **Lab**
Dipartimento di Ingegneria dell'Informazione
Università degli Studi di Parma

Multi-Agent Systems

JADE

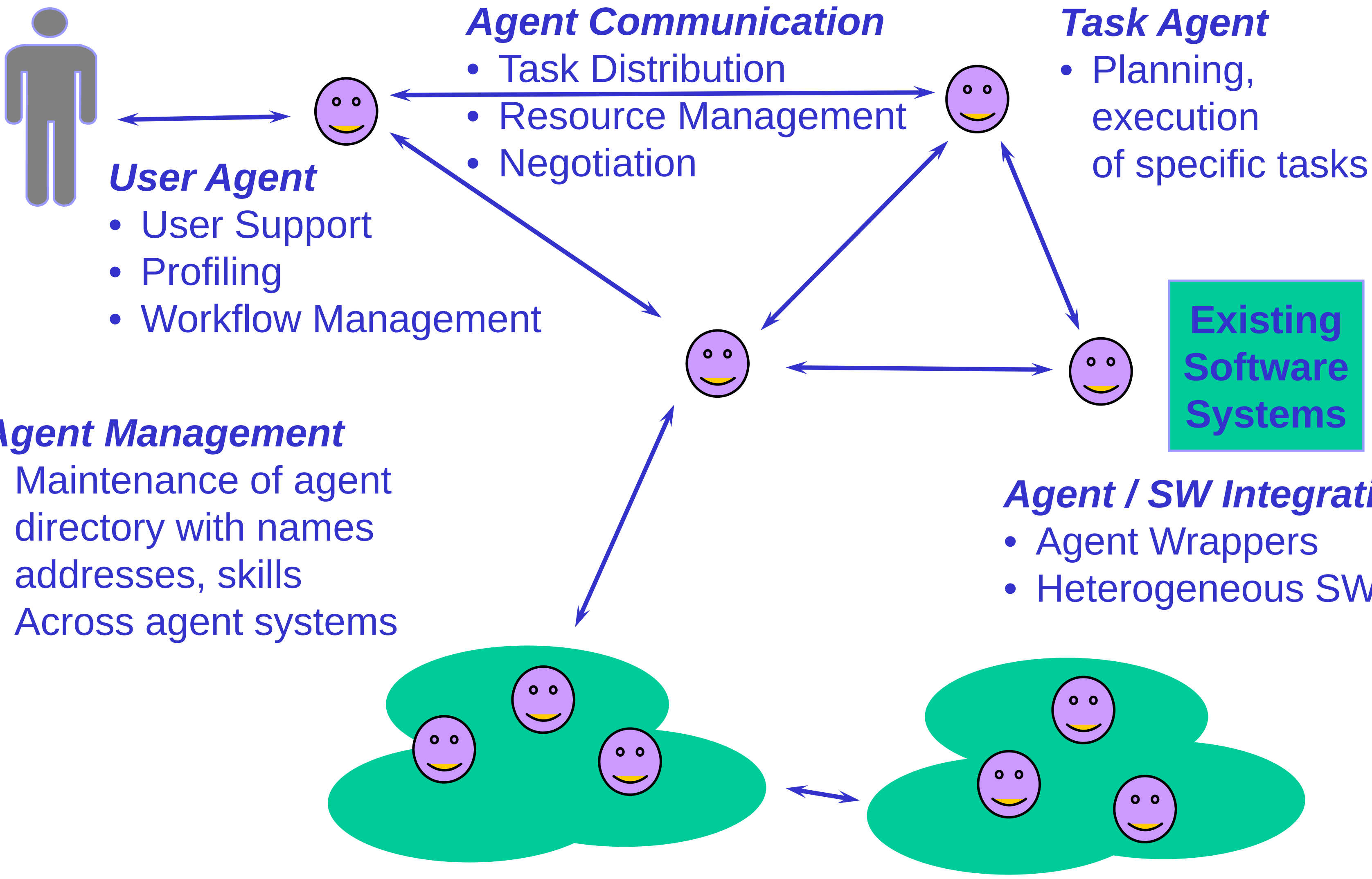
Prof. Agostino Poggi

What is FIPA?

- Foundation for Intelligent Physical Agents is a non-profit International standards body born in 1996
- More than 60 member companies
- Three set of specifications: FIPA97, FIPA98 and FIPA2000
- During 2005 becomes a standard committee of IEEE Computer Society

- FIPA mission is the promotion of technologies and interoperability specifications that facilitate the end-to-end interworking of intelligent agent systems in modern commercial and industrial settings
  - Only the external behavior of systems should be standardized

Areas for Standardization



Agent Communication
- Task Distribution
- Resource Management
- Negotiation

Task Agent
- Planning, execution of specific tasks

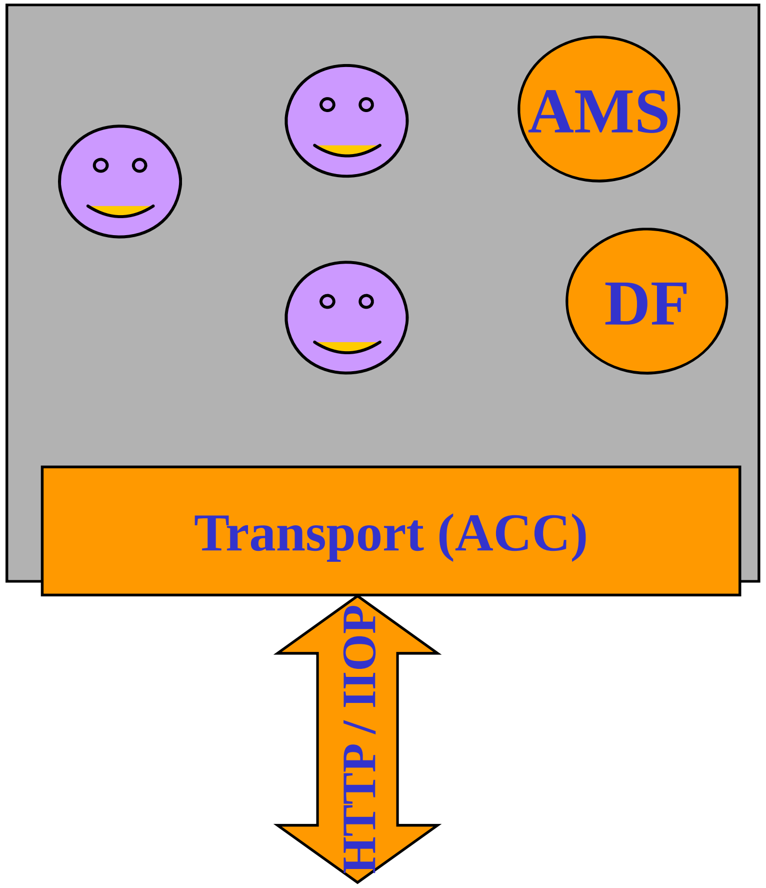
User Agent
- User Support
- Profiling
- Workflow Management

Agent Management
- Maintenance of agent directory with names addresses, skills
- Across agent systems

Agent / SW Integration
- Agent Wrappers
- Heterogeneous SW

The FIPA Agent Platform

- A software system providing a subset of FIPA specified services
- AMS
  - Authentication, Resources
  - White pages (naming)
- DF
  - Directory (yellow pages)
- ACC
  - Message transport



What is JADE?

- JADE (Java Agent DEvelopment framework) is a middleware that simplifies the development of Multi Agent applications providing the basic services and infrastructure

  - JADE project started in July 1998 as joint development of Telecom Italia Lab and Parma University
  - Fully implemented in Java
  - Runs on all JVM (J2SE, J2EE, J2ME)
  - Distributed in Open Source under the LGPL license and freely downloadable from http://jade.tilab.com (registration is required)
  - Compliant with the FIPA specifications
  - Last version (3.6.1) available from 04/11/2008

AOT LAB

JADE Hides FIPA from Programmers

- No need to implement the Agent Platform
  - AMS, DF executed at start-up
- No need to implement agent-management ontology and functionalities
  - An agent is registered with the Agent Platform within its constructor, it is given a name and an address
  - The DFService class provides a simplified interface to access the services of the DF (registration, searching, lease-renewal, …)
- No need to implement Message Transport and Parsing
  - Automatically (and possibly efficiently) done by the framework when sending/receiving messages
- Interaction Protocols must only be extended via handle methods

JADE Features

- Distributed Agent Platform
  - Seen as a whole from the outside world
  - Spanning multiple machines
- Transparent, multi-transport messaging
  - Event dispatching for local delivery
  - Java RMI for intra-platform deliver
  - FIPA 2000 MTP framework
- Two levels concurrency model
  - Inter-agent (pre-emptive, Java threads)
  - Intra-agent (co-operative, Behaviour classes)

JADE Features

- User defined content languages and ontologies
  - Message content is represented according to a meta-model
  - User defined classes can be used to model ontology elements (Actions, Concepts and Predicates)
- Supports intra-platform mobility and cloning
  - Agents can migrate between containers
  - Agents can clone across containers
  - JADE might also have provided inter-platform mobility, but … the support of a standard is a MUST or is just JADE2JADE mobility

JADE Features

- Agent based management and development tools
- Federation of DFs
  - Many FIPA-compliant DFs can be started at run time in order to implement multi-domain applications
- In-process interface
  - An external application can start a JADE runtime
  - Several containers can be created within the application JVM

JADE Basic Services

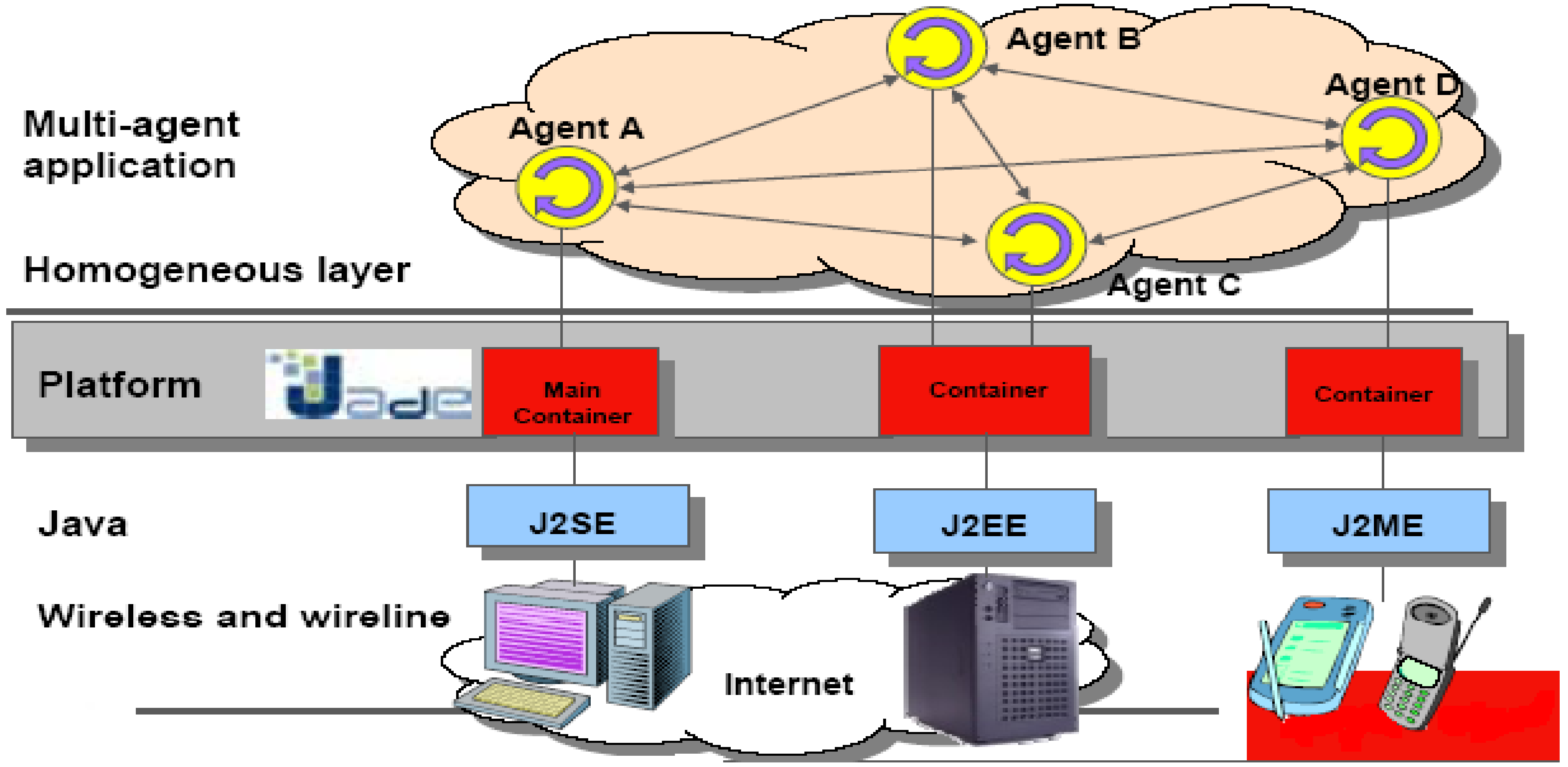
- Agent life-cycle and agent mobility
- White and yellow pages services
- Peer-2-peer message transport and multi-party communication
- Agent security
- Scheduling of multiple agent tasks
- Set of graphical tools to support monitoring, logging and debugging
- Extremely light-weight, ported to J2ME-CLDC-MIDP 1.0

JADE Platform Architecture

- A JADE-based application is composed of a collection of active components called agents
- Each agent has a unique name
- Each agent is a peer since he can communicate in a bidirectional way with all the other agents
- Each agent lives in a container and can migrate within the platform
- There is only one container that plays the role of MAIN, where AMS and DF live
- The Main Container can be replicated via replication service

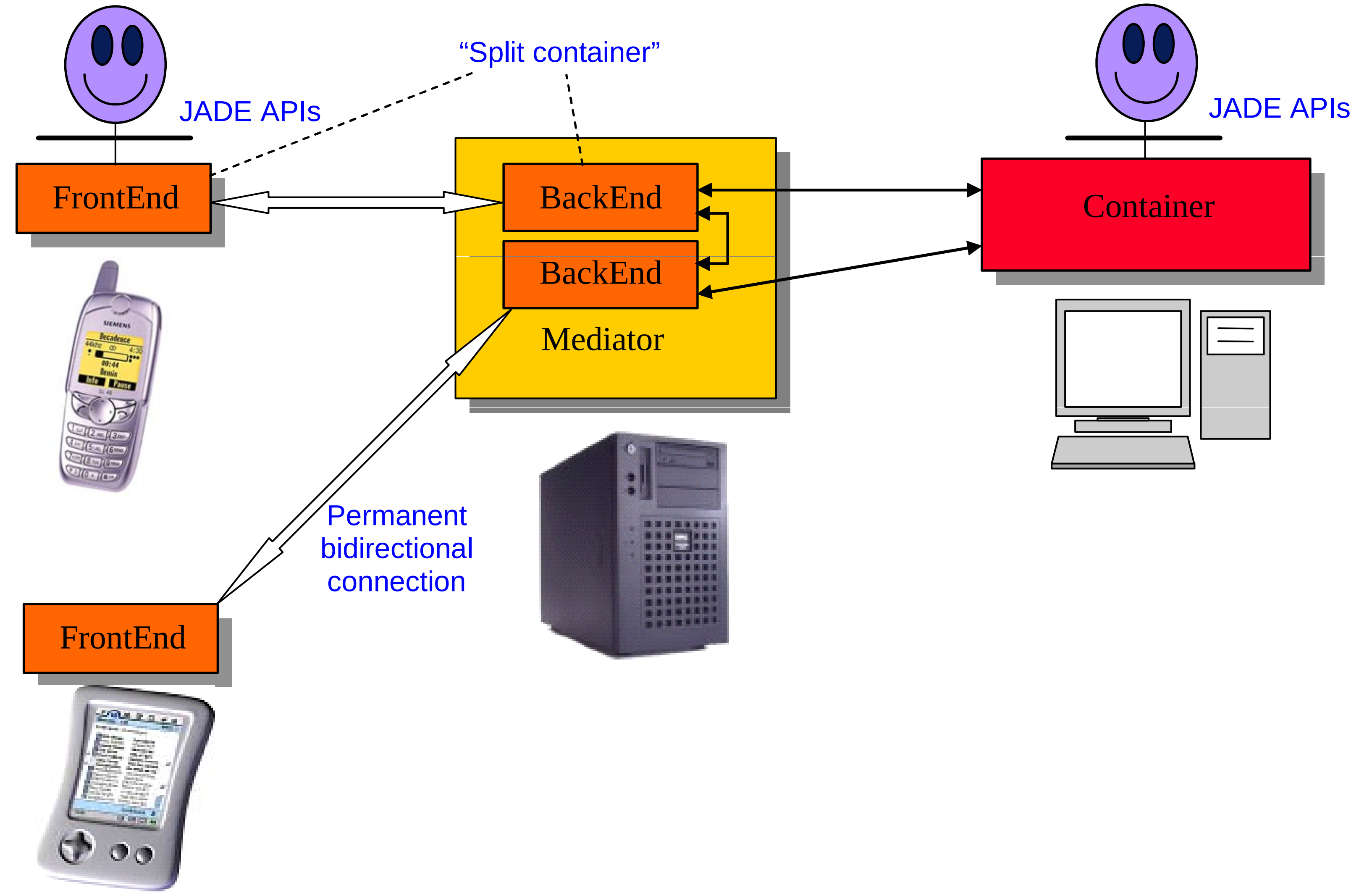
AOT
LAB

JADE Platform Architecture



AOT LAB

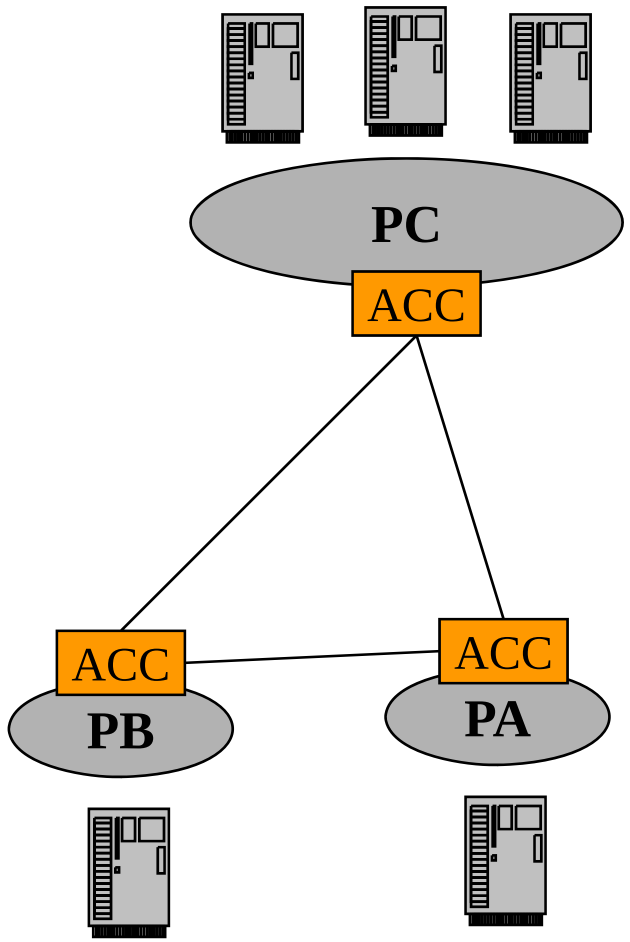
JADE Platform Architecture



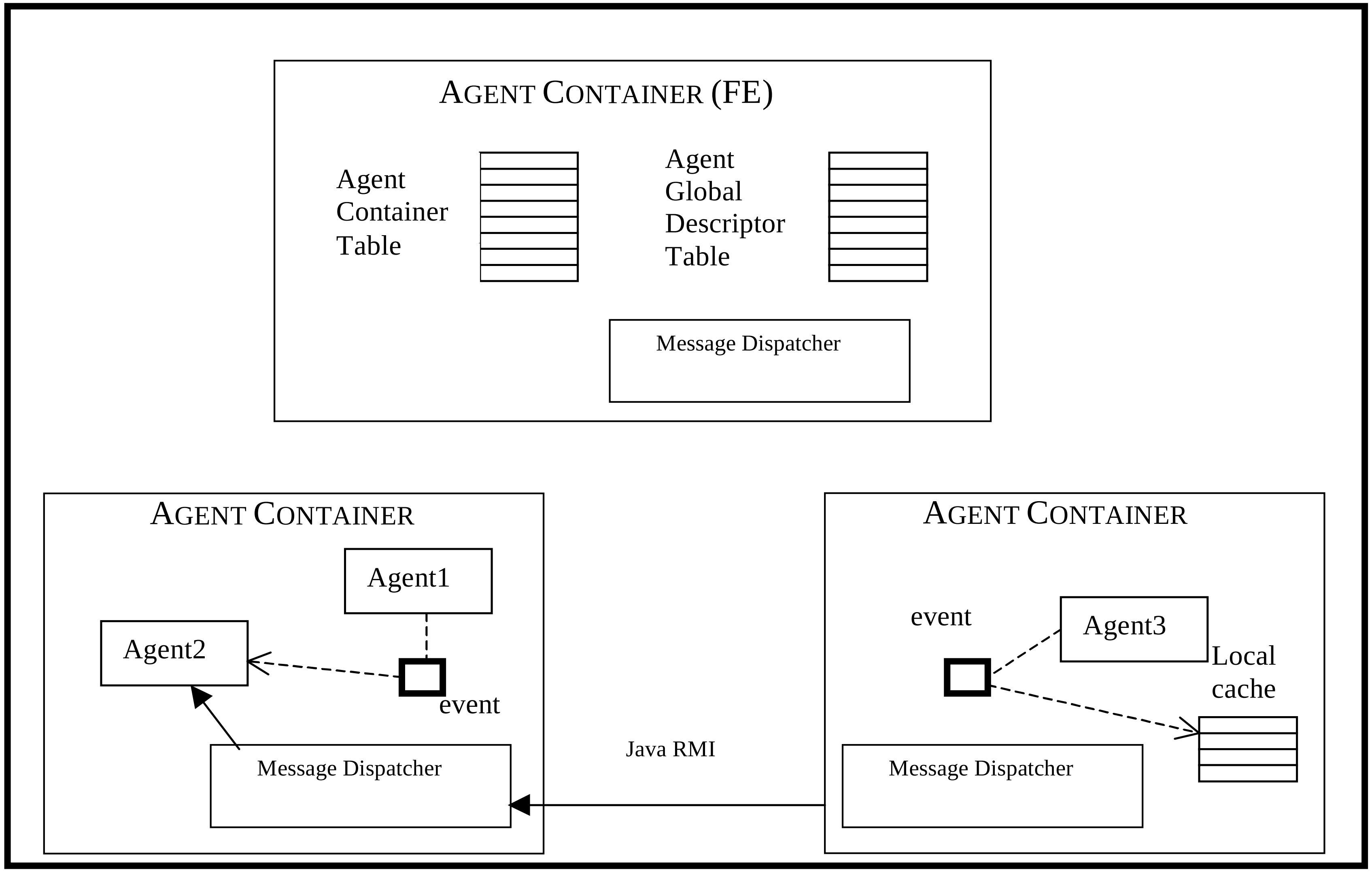
AOT LAB

Communication

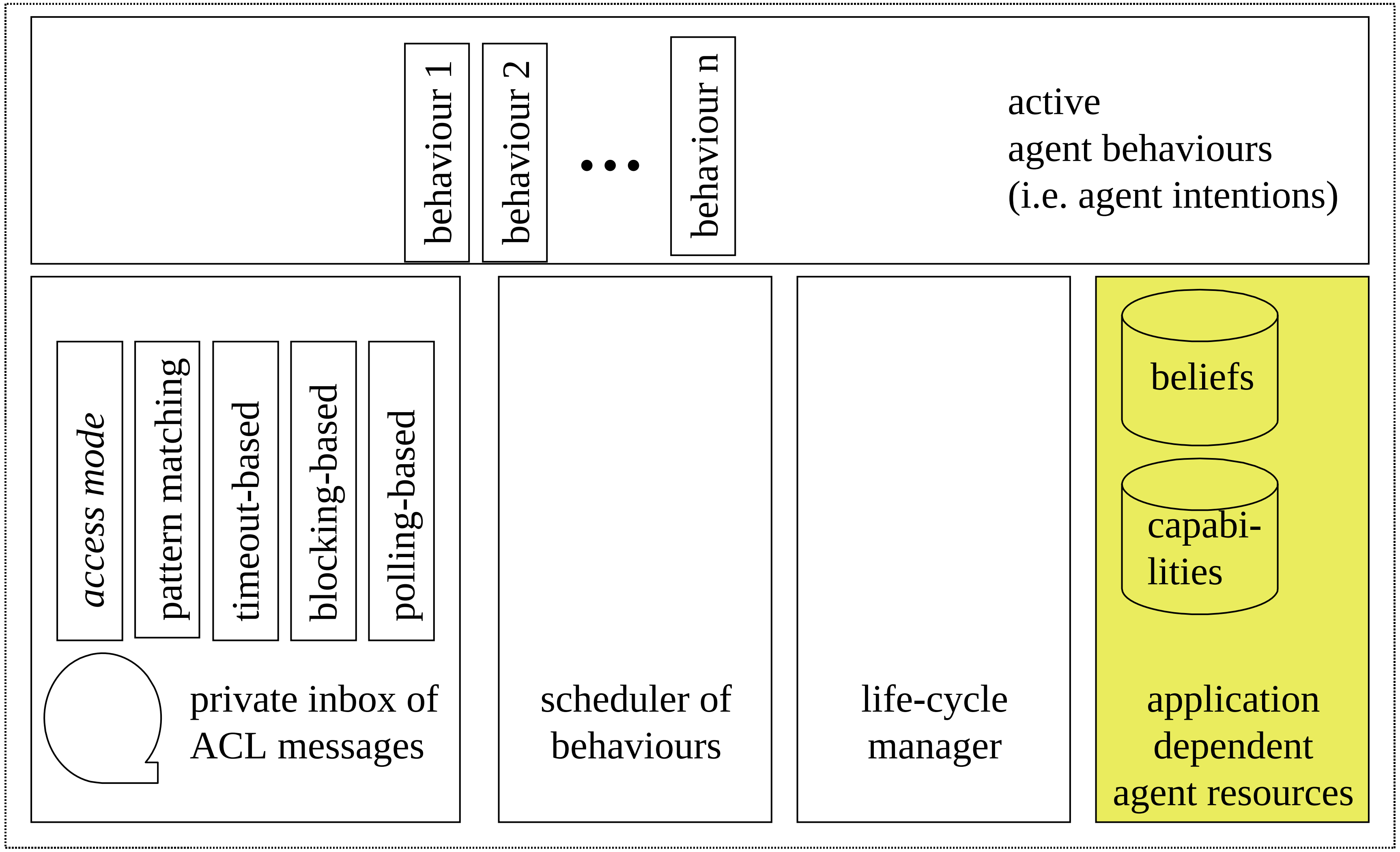
- Communication between agents can be
  - Intra-Platform
    - Non-standard technologies
  - Inter-Platforms
    - Uses the ACC and standard FIPA Message transport protocols
- Agent Environment on each platform
  - Different languages
  - Different APIs
  - Different support features
  - Different agent architecture
  - Common Base services
  - Common transports
  - Common communication languages



JADE Message Dispatching



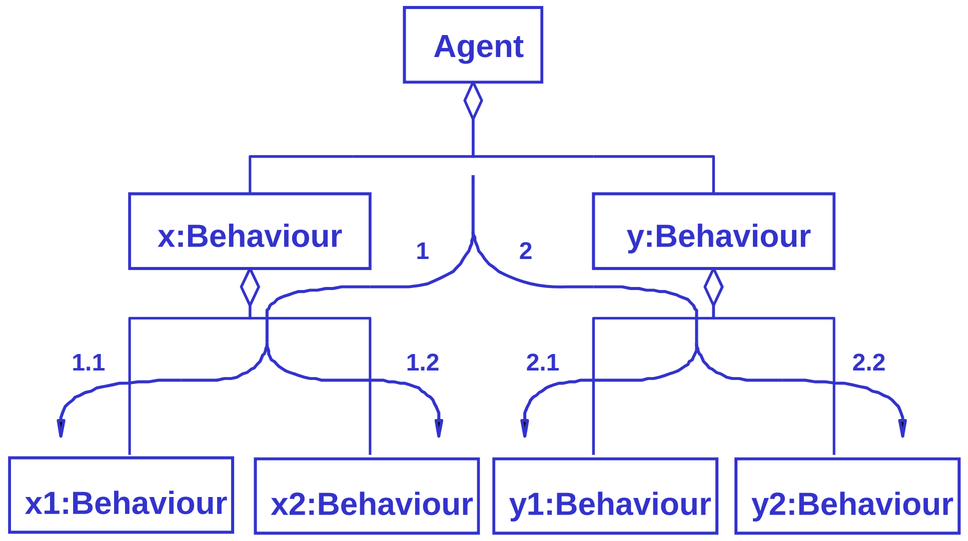
JADE Agent Architecture



JADE Concurrency Model

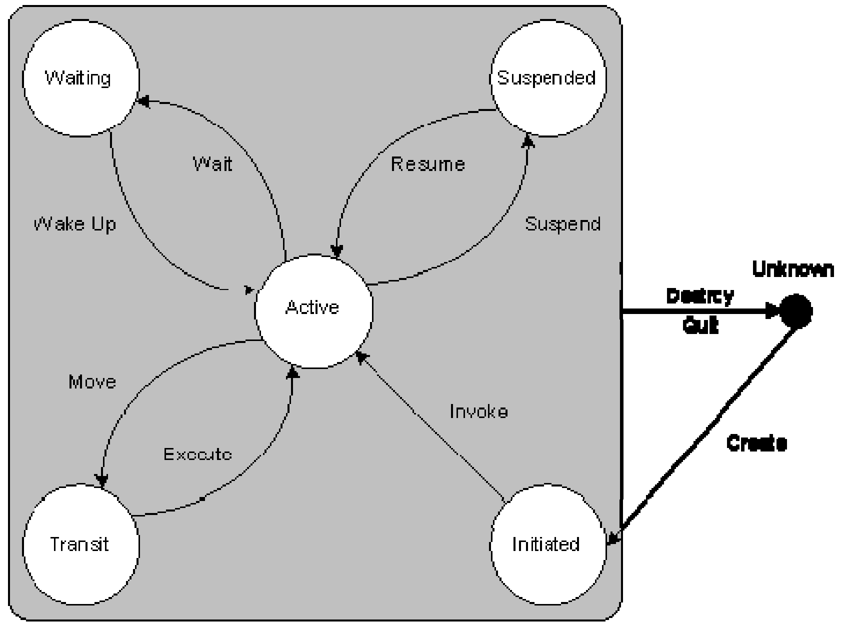
- Agent execution is based on a multithreaded inter-agent scheduling

- Behaviour abstraction
  - Composite for structure
  - Chain of Responsibility for scheduling
  - No context saving



AOT LAB

Agent Life Cycle



- **INITIATED**
  - Agent object is built
  - Not registered with AMS
  - No name, no address
- **ACTIVE**
  - Registered with AMS
  - Can access all JADE features
- **SUSPENDED**
  - No agent behaviour is executed
- **WAITING**
  - Agent object is blocked and its corresponding thread is sleeping on a Java monitor
- **DELETED**
  - Agent thread has terminated
  - No more registered with AMS
- **TRANSIT**
  - A mobile agent is migrating to a new location

AOT
LAB

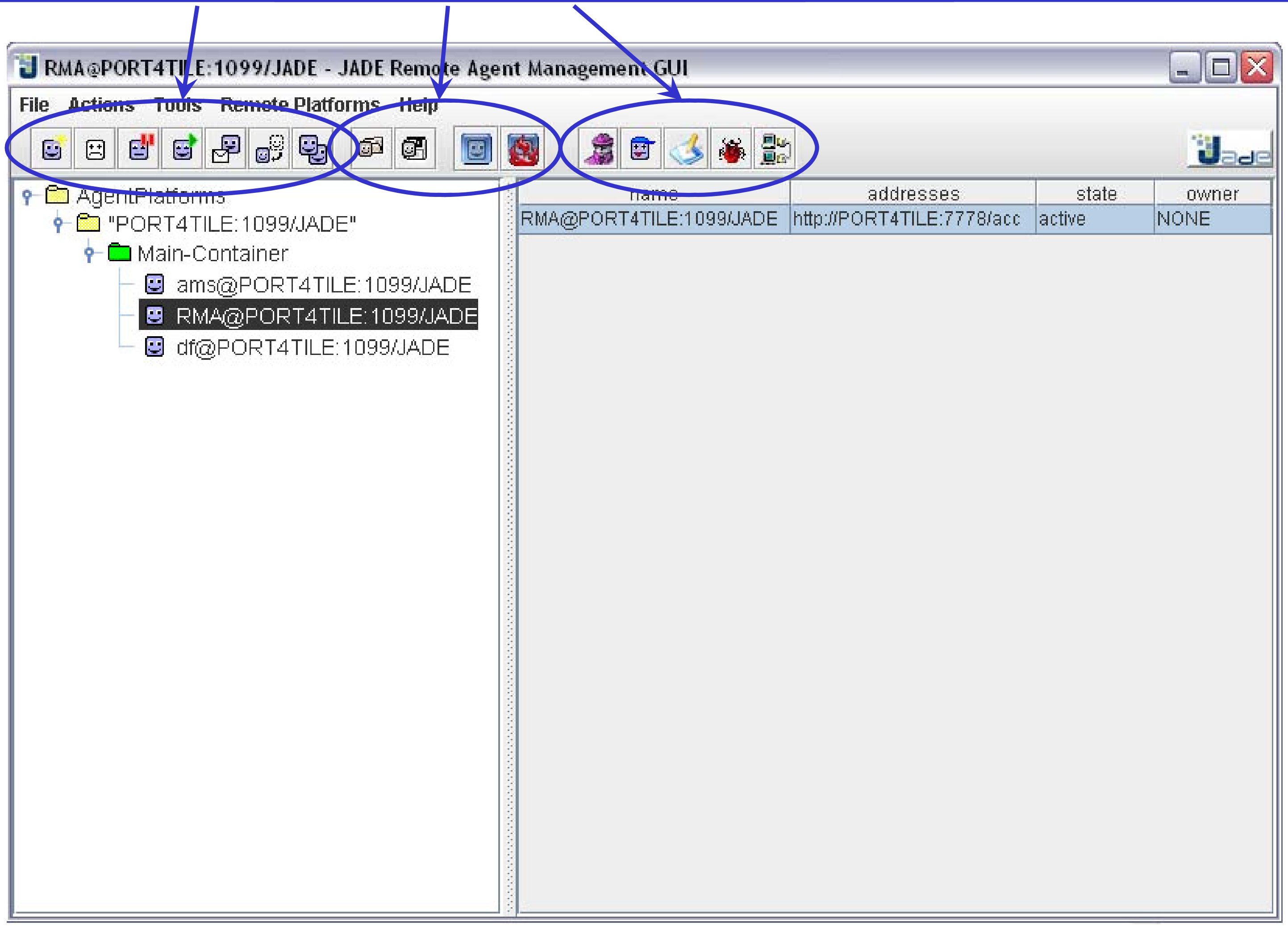
Graphical Tools

- RMA (Remote Monitoring Agent)
  - White pages GUI
  - Agent Life Cycle Handle
- DF (Directory Facilitator) GUI
  - Yellow Pages GUI
- Dummy Agent
  - End Point Debugger
- Sniffer Agent
  - Man-in-the-Middle
- Introspector Agent
  - Behaviour Level Debugger
- Log Manager Agent

AOT
LAB

Remote Management Agent (RMA)

St Start/Kill/Suspend/Res Load/Save/Freeze/Thaw Agent Agent ectorAgent/AddPlatform

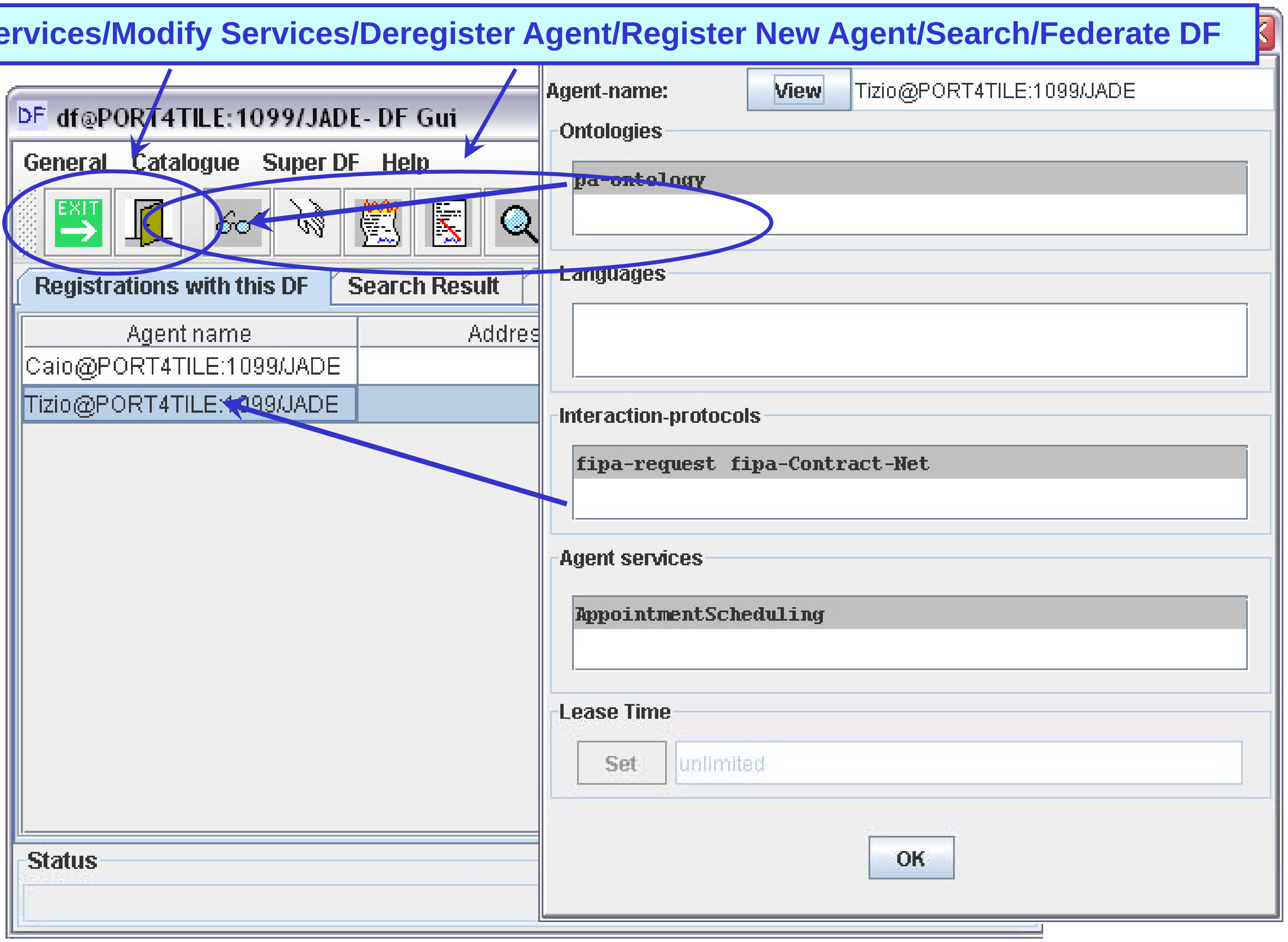


name	addresses	state	owner
RMA@PORT4TILE:1099/JADE	http://PORT4TILE:7778/acc	active	NONE

AOT
LAB

DF-GUI

View Services/Modify Services/Deregister Agent/Register New Agent/Search/Federate DF



Agent name
Caio@PORT4TILE:1099/JADE
Tizio@PORT4TILE:1099/JADE

Agent-name: Tizio@PORT4TILE:1099/JADE

Ontologies
pa-ontology

Languages

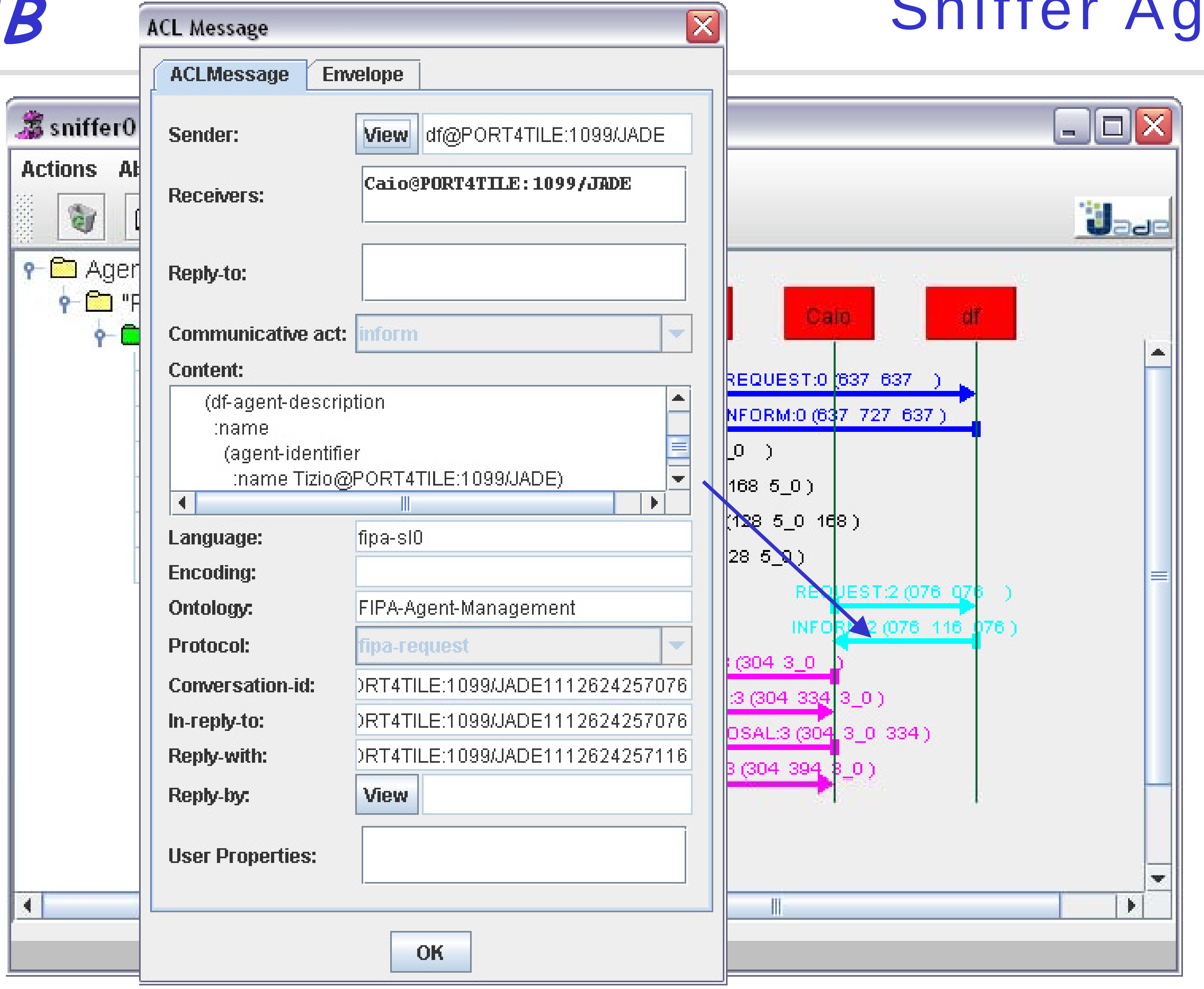
Interaction-protocols
fipa-request fipa-Contract-Net

Agent services
AppointmentScheduling

Lease Time
unlimited

AOT LAB

Sniffer Agent



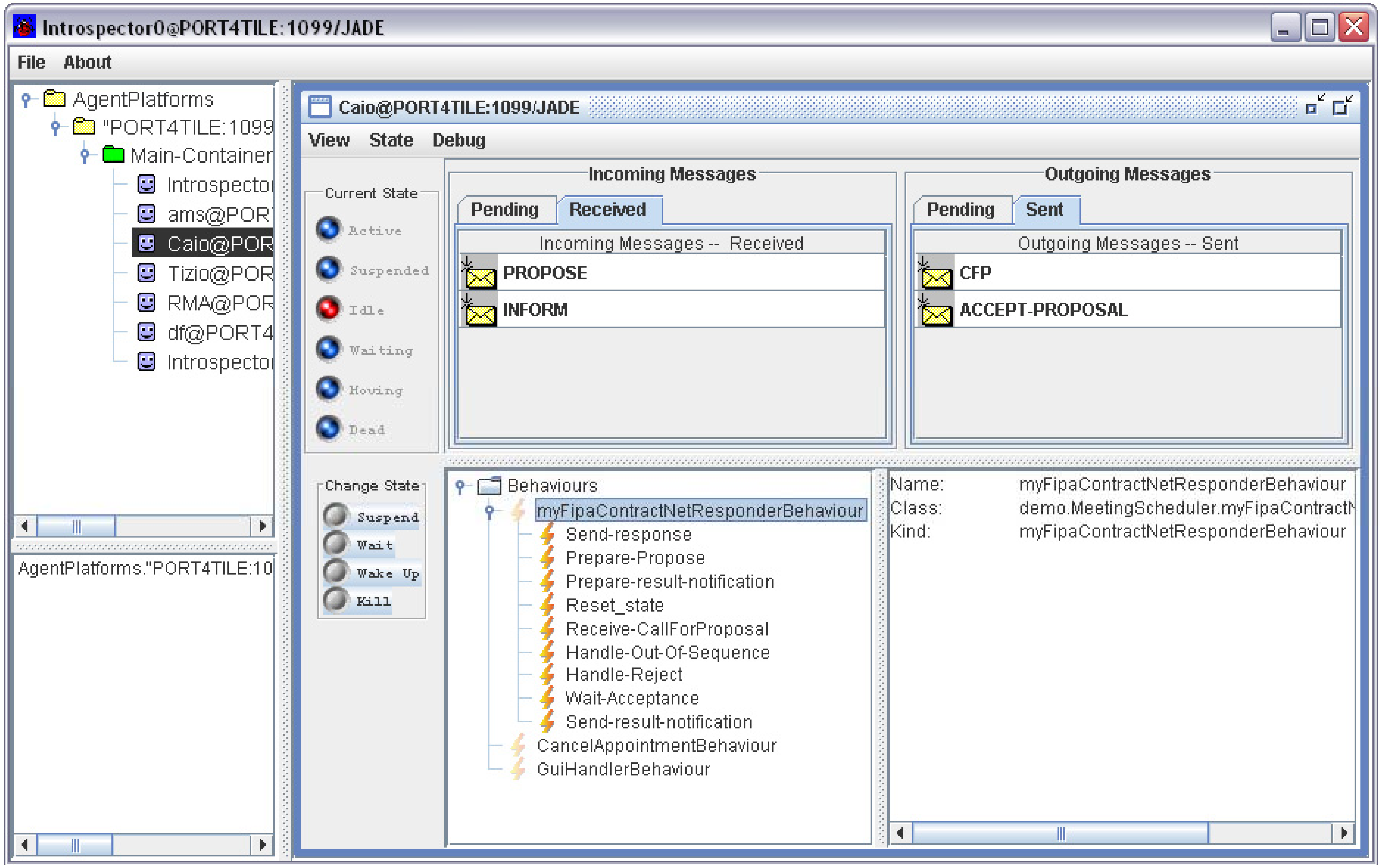
Sender: df@PORT4TILE:1099/JADE
Receivers: Caio@PORT4TILE:1099/JADE
Reply-to:
Communicative act: inform
Content:

```
(df-agent-description
  :name
   (agent-identifier
    :name Tizio@PORT4TILE:1099/JADE)
```

Language: fipa-sl0
Encoding:
Ontology: FIPA-Agent-Management
Protocol: fipa-request
Conversation-id:)RT4TILE:1099/JADE1112624257076
In-reply-to:)RT4TILE:1099/JADE1112624257076
Reply-with:)RT4TILE:1099/JADE1112624257116
Reply-by:
User Properties:

AOT LAB

Introspector Agent


Introspector0@PORT4TILE:1099/JADE

File About

AgentPlatforms
"PORT4TILE:1099
Main-Container
Introspecto
ams@POR
Caio@POR
Tizio@POR
RMA@POR
df@PORT4
Introspecto

AgentPlatforms."PORT4TILE:10

Caio@PORT4TILE:1099/JADE

View State Debug

Current State: Active, Suspended, Idle, Waiting, Moving, Dead

Change State: Suspend, Wait, Wake Up, Kill

Incoming Messages — Pending | Received

Incoming Messages -- Received
- PROPOSE
- INFORM

Outgoing Messages — Pending | Sent

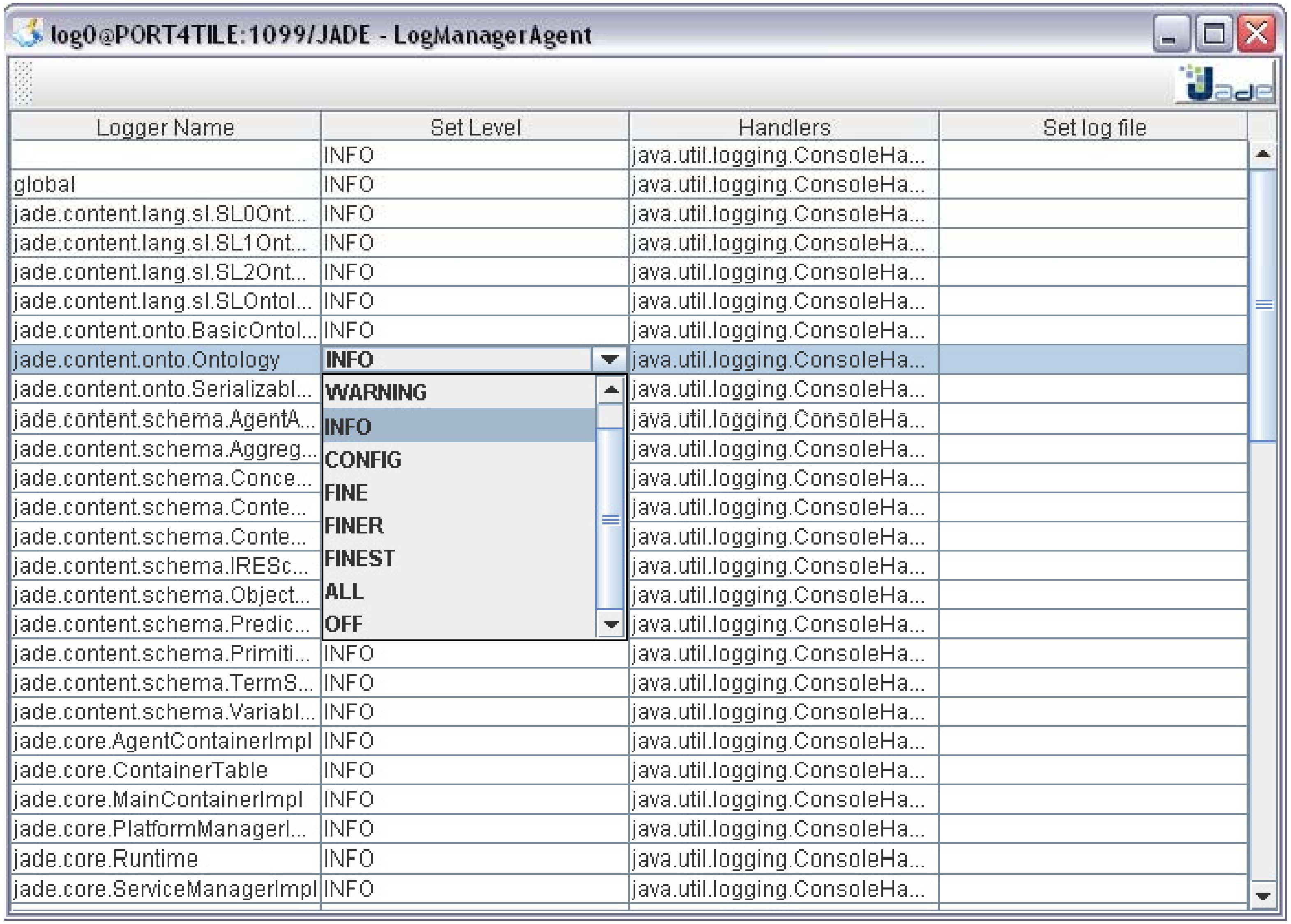
Outgoing Messages -- Sent
- CFP
- ACCEPT-PROPOSAL

Behaviours
- myFipaContractNetResponderBehaviour
  - Send-response
  - Prepare-Propose
  - Prepare-result-notification
  - Reset_state
  - Receive-CallForProposal
  - Handle-Out-Of-Sequence
  - Handle-Reject
  - Wait-Acceptance
  - Send-result-notification
- CancelAppointmentBehaviour
- GuiHandlerBehaviour

Name:	myFipaContractNetResponderBehaviour
Class:	demo.MeetingScheduler.myFipaContractN
Kind:	myFipaContractNetResponderBehaviour

AOT
LAB

Log Manager Agent



Logger Name	Set Level	Handlers	Set log file
	INFO	java.util.logging.ConsoleHa...	
global	INFO	java.util.logging.ConsoleHa...	
jade.content.lang.sl.SL0Ont...	INFO	java.util.logging.ConsoleHa...	
jade.content.lang.sl.SL1Ont...	INFO	java.util.logging.ConsoleHa...	
jade.content.lang.sl.SL2Ont...	INFO	java.util.logging.ConsoleHa...	
jade.content.lang.sl.SLOntol...	INFO	java.util.logging.ConsoleHa...	
jade.content.onto.BasicOntol...	INFO	java.util.logging.ConsoleHa...	
jade.content.onto.Ontology	INFO	java.util.logging.ConsoleHa...	
jade.content.onto.Serializabl...	WARNING	java.util.logging.ConsoleHa...	
jade.content.schema.AgentA...	INFO	java.util.logging.ConsoleHa...	
jade.content.schema.Aggreg...	CONFIG	java.util.logging.ConsoleHa...	
jade.content.schema.Conce...	FINE	java.util.logging.ConsoleHa...	
jade.content.schema.Conte...	FINER	java.util.logging.ConsoleHa...	
jade.content.schema.Conte...	FINEST	java.util.logging.ConsoleHa...	
jade.content.schema.IRESc...	ALL	java.util.logging.ConsoleHa...	
jade.content.schema.Object...	OFF	java.util.logging.ConsoleHa...	
jade.content.schema.Predic...		java.util.logging.ConsoleHa...	
jade.content.schema.Primiti...	INFO	java.util.logging.ConsoleHa...	
jade.content.schema.TermS...	INFO	java.util.logging.ConsoleHa...	
jade.content.schema.Variabl...	INFO	java.util.logging.ConsoleHa...	
jade.core.AgentContainerImpl	INFO	java.util.logging.ConsoleHa...	
jade.core.ContainerTable	INFO	java.util.logging.ConsoleHa...	
jade.core.MainContainerImpl	INFO	java.util.logging.ConsoleHa...	
jade.core.PlatformManagerI...	INFO	java.util.logging.ConsoleHa...	
jade.core.Runtime	INFO	java.util.logging.ConsoleHa...	
jade.core.ServiceManagerImpl	INFO	java.util.logging.ConsoleHa...	

AOT
LAB

Agent Class

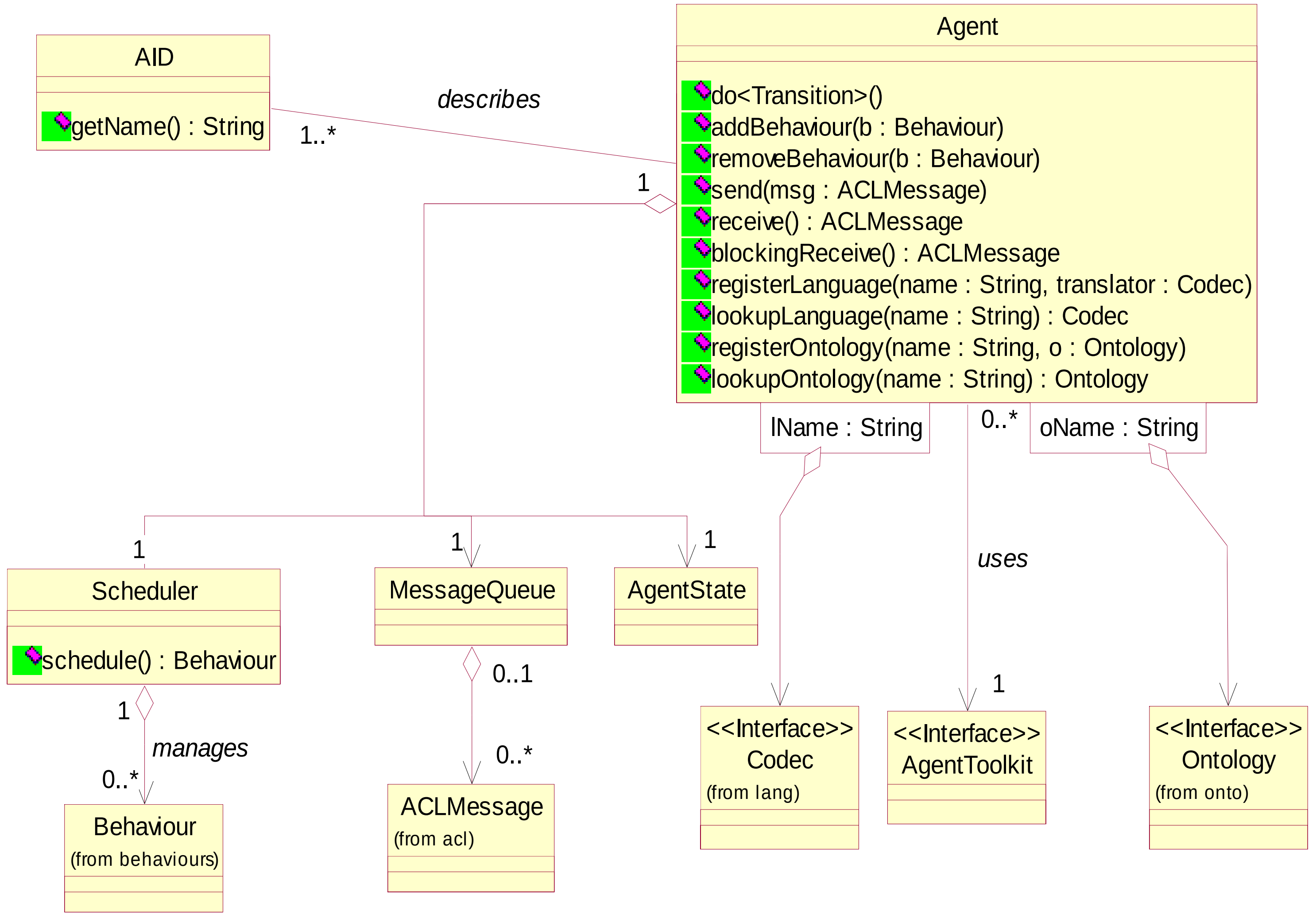
- The Agent class represents a common base class for user defined agents
- A JADE agent is simply an instance of a user defined Java class that extends the base Agent class
- The Agent class provides:
  - Features to accomplish basic interactions with the agent platform (registration, configuration, remote management, …)
  - A basic set of methods that can be called to implement the custom behaviour of the agent (e.g. send/receive messages, use standard interaction protocols, register with several domains, …)

Agent Class

- The computational model of an agent is multitask, where tasks (or behaviours) are executed concurrently
  - Each functionality/service provided by an agent should be implemented as one or more behaviours
  - A scheduler, internal to the base Agent class and hidden to the programmer, automatically manages the scheduling of behaviours in a queue of ready tasks

AOT LAB

Agent Class



Starting Agent Execution

- Birth of a new agent
  - Agent constructor is executed
  - Agent identifier (AID) is assigned
  - Agent is registered with the AMS
  - Agent is put in the AP_ACTIVE state
  - Agent’s setup() method is executed
- The setup() method is therefore the point where any application-defined agent activity starts
  - Initialize the agent
  - Add tasks using the method addBehaviour()

Terminating Agent Execution

- Any behaviour can call the Agent doDelete() method in order to stop agent execution
- The Agent takeDown() method is executed when the agent is going to be destroyed
  - The agent is still registered with the AMS and can therefore send messages to other agents
  - The takeDown() method can be overridden to implement any necessary cleanup, such as de-registering with DF agents
- Just after the takeDown() method is completed, the agent will be de-registered from AMS and its thread destroyed

AOT LAB

Example: Hello World

```
import jade.core.Agent;

public class HelloAgent extends Agent {
    protected void setup() {
            System.out.println("Hello World!!!");
            System.out.println("My name is "+ getLocalName());
    }
}
```

AOT LAB

Behaviour Class

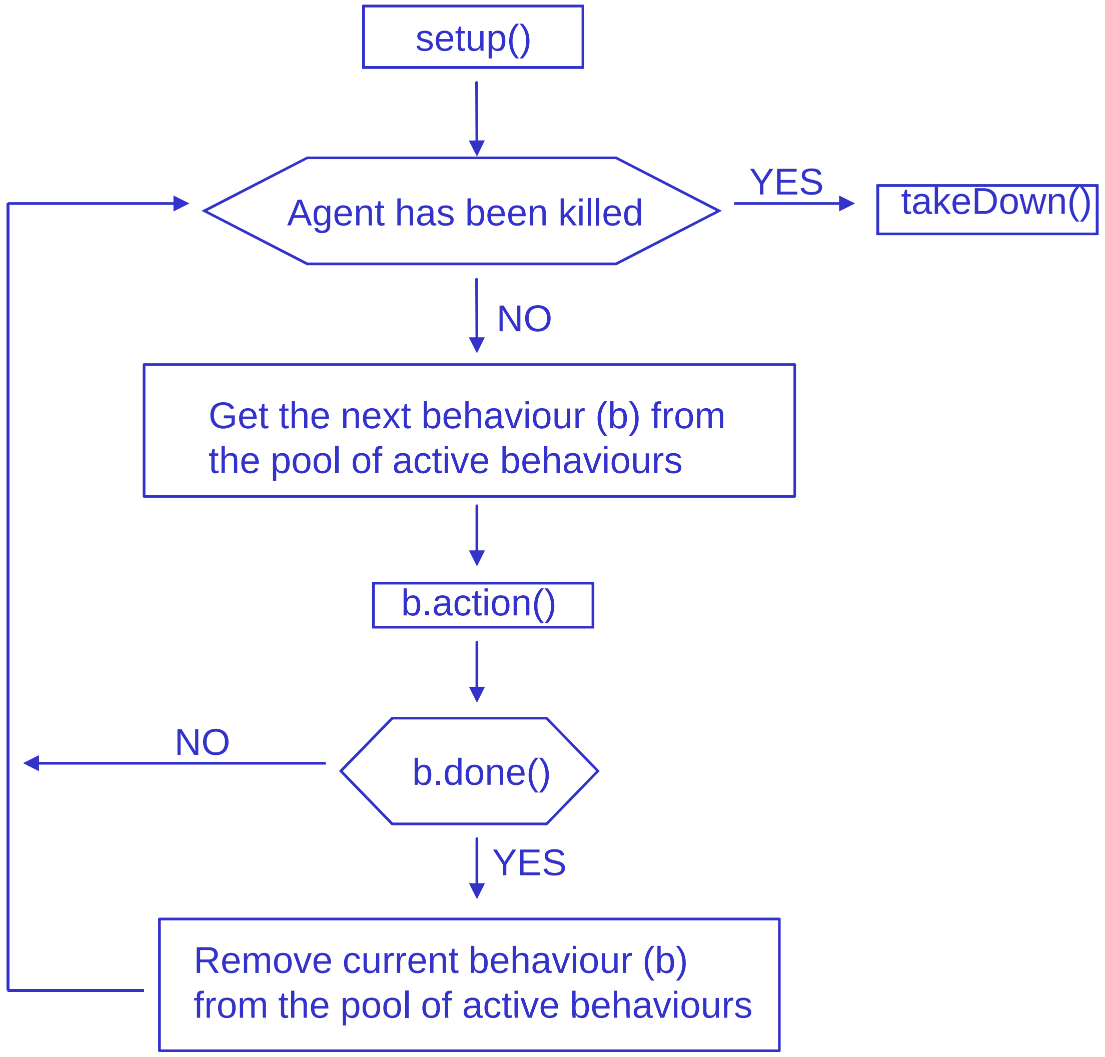
- The actual job/functionality/service that an agent does is typically carried out within “behaviours”
- Behaviours extend the Behaviour class
- To make an agent execute a task it is sufficient
  - To create an instance of the corresponding Behaviour subclass
  - Call the addBehaviour() method of the Agent class

Behaviour Class

- Each Behaviour subclass must implement
  - The action() method implementing what the behaviour actually does
  - The done() method checking whether the behaviour is finished
- Moreover some other methods can be implemented
  - The onStart() method that is invoked only once before the first execution of the action() method
  - The onEnd() method that is invoked only once after the done() method returns true
- Each behaviour has a pointer to the agent executing it
  - The removeBehaviour() method of the Agent class can be used to remove a behaviour from the agent pool of behaviours
  - In this case, the onEnd() method is not called

AOT LAB

Agent Execution Model


setup()
Agent has been killed
YES
takeDown()
NO
Get the next behaviour (b) from the pool of active behaviours
b.action()
NO
b.done()
YES
Remove current behaviour (b) from the pool of active behaviours

Task Scheduler

- The scheduler, implemented by the base Agent class and hidden to the programmer, carries out a round-robin non-preemptive scheduling
  - In detail, the agent scheduler executes action() method of each behaviour in the ready behaviours queue
  - Since no stack contest is saved, every time the action() method is run from the beginning
  - When action() method returns, the done() method is called in order to check if the behaviour has completed its task
  - If so, the behaviour object is removed from the queue

AOT
LAB

Blocked Tasks

- Behaviours work just like co-operative threads
- The method block() puts the behaviour in a queue of blocked behaviours as soon as the action() method returns
  - All blocked behaviours are rescheduled as soon as a new message arrives
  - It is possible also to set a timeout parameter
  - Programmer must block again a behaviour if it was not interested in the arrived message
- Because of the non preemptive multitasking model chosen for agent behaviours, agent programmers must avoid to use endless loops and even to perform long operations within action() methods

AOT LAB

Example: agent with behaviour

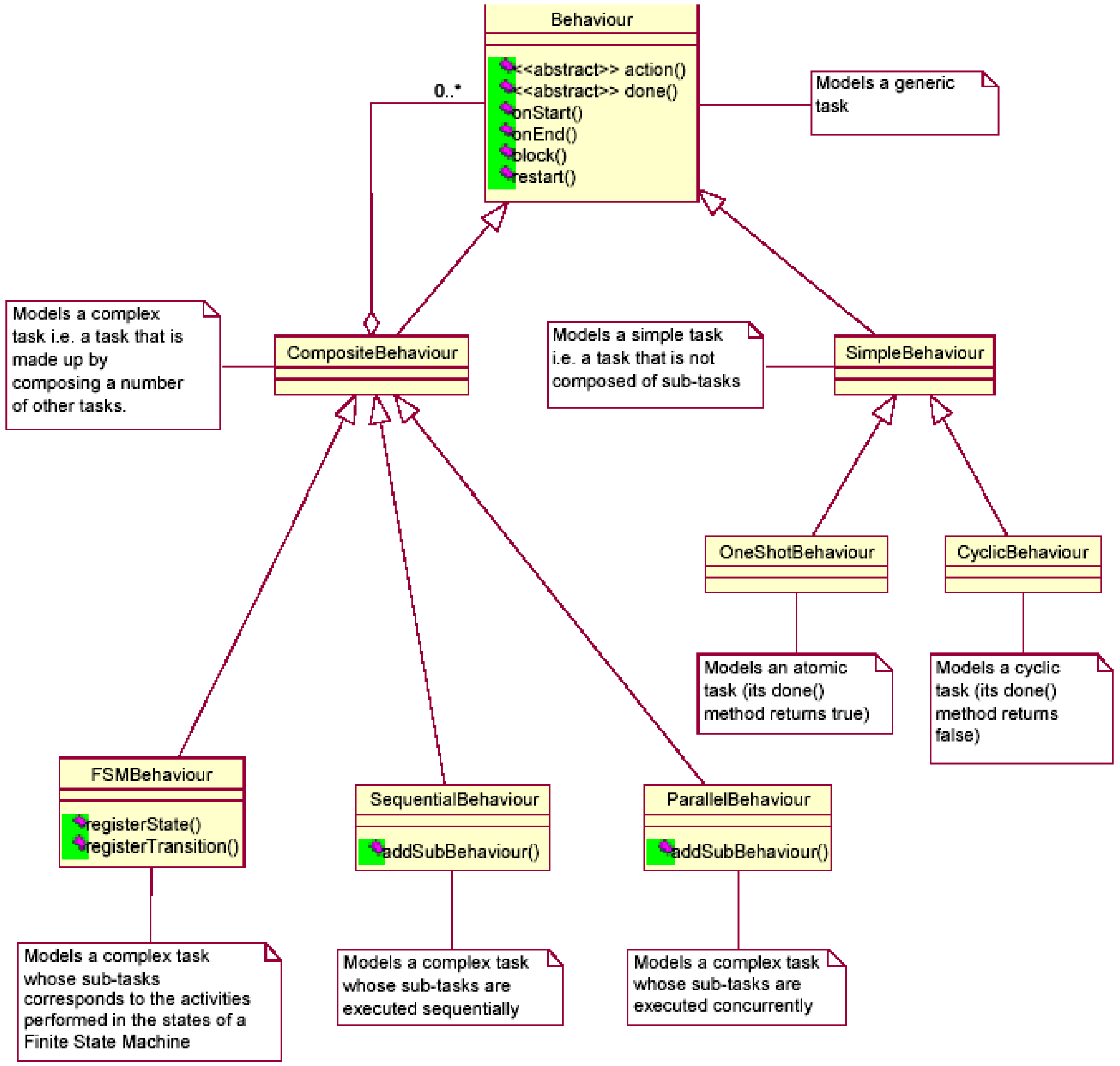
```
import jade.core.Agent;
import jade.core.behaviours.*;
public class ExampleAgent extends Agent {
        protected void setup() {
            addBehaviour( new HelloBehaviour(this));
        }
}
class HelloBehaviour extends SimpleBehaviour {
        private boolean finished = false;
        public HelloBehaviour(Agent a) {
            super(a);
        }
        public void action() {
            System.out.println( "Hello World! My name is " + myAgent.getLocalName());
        }
        public boolean done() {
            return finished;
        }
}
```

Behaviour Types

- Simple Behaviours
  - One shot behaviours
    - Complete immediately, the action() method is executed only once
  - Cyclic behaviours
    - Never complete, the action() method is executed forever
- Composite Behaviours
  - “FSM” behaviours
    - Each child is a state of the FSM
  - “Sequential” behaviours
    - Each child is executed in a sequential order
    - Terminates when the last child has ended
  - “Parallel” behaviours
    - Each child is executed concurrently
    - Terminates when a particular condition on its sub-behaviours is met

AOT LAB

Behaviour Class Hierarchy



Behaviour
<<abstract>> action()
<<abstract>> done()
onStart()
onEnd()
block()
restart()
0..*
Models a generic task
CompositeBehaviour
Models a complex task i.e. a task that is made up by composing a number of other tasks.
SimpleBehaviour
Models a simple task i.e. a task that is not composed of sub-tasks
OneShotBehaviour
Models an atomic task (its done() method returns true)
CyclicBehaviour
Models a cyclic task (its done() method returns false)
FSMBehaviour
registerState()
registerTransition()
Models a complex task whose sub-tasks corresponds to the activities performed in the states of a Finite State Machine
SequentialBehaviour
addSubBehaviour()
Models a complex task whose sub-tasks are executed sequentially
ParallelBehaviour
addSubBehaviour()
Models a complex task whose sub-tasks are executed concurrently

AOT LAB

Sending and Receiving Messages

- Message exchange is managed through the ACLMessage class

- An agent willing to send a message should:
  - Create a new ACLMessage object
  - Fill its attributes with appropriate values
  - Call its send() method

- An agent willing to receive a message should
  - Call its receive() or blockingReceive() methods

- Sending and receiving can be scheduled as independent agent activities by adding ReceiverBehaviour or SenderBehaviour to the agent queue of tasks

AOT
LAB

Example: SenderAgent

```
import jade.core.AID;
import jade.core.Agent;
import jade.lang.acl.ACLMessage;

public class SenderAgent extends Agent {
  protected void setup() {
    System.out.println("Hello!My name is “ + this.getLocalName());
    sendMessage();
  }

  private void sendMessage() {
    ACLMessage aclMessage = new ACLMessage(ACLMessage.REQUEST);
    aclMessage.addReceiver(new AID(“ag1“, AID.ISLOCALNAME ));
    aclMessage.setContent("Hello! How are you?");
    this.send(aclMessage);
     }
}
```

AOT LAB

Replying to a Message

- According to FIPA specifications, a reply message must be formed taking into account a set of well-formed rules, such as:
  - Setting the appropriate value for the attribute in-reply-to
  - Using the same conversation-id
  - …
- The method createReply() of the ACLMessage class returns a new ACLMessage object that is a valid reply to the current one
  - Then the programmer only needs to set the application specific communicative act and message content

Example: ResponderAgent

Write some behaviour in the **setup()** method in order to processing the incoming messages

```
import jade.core.Agent;

public class ResponderAgent extends Agent {

  // Agent Initialisation
  … … … … …

  protected void setup() {
    System.out.println("Hello. My name is "+this.getLocalName());
    addBehaviour(new ResponderBehaviour(this));
  }
}
```

AOT
LAB

Example: ResponderAgent

```
public class ResponderBehaviour extends SimpleBehaviour {

private static final MessageTemplate mt =
MessageTemplate.MatchPerformative(ACLMessage.REQUEST);

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

  public void action() {
    while (true) {
      ACLMessage aclMessage = myAgent.receive(mt);
      if (aclMessage!=null) {
        System.out.println(myAgent.getLocalName()+": I received a message.\n"+aclMessage);
      } else {
        this.block(); }
    }
  }

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

Message Templates

- Any agent should be provided with the capability of carrying on many simultaneous conversations
- Because the queue of incoming messages is shared by all the agent behaviours, Jade implements an access mode to that queue based on pattern matching, rather than FIFO only
  - The MessageTemplate class allows to build patterns to match ACL messages against
    - The programmer can create one pattern for each attribute of the ACLMessage
  - The receive() and blockingReceive() methods can be augmented with the pattern-matching capability, by passing a MessageTemplate parameter
  - Elementary patterns can be combined with AND, OR and NOT operators, in order to build more complex matching rules

AOT LAB

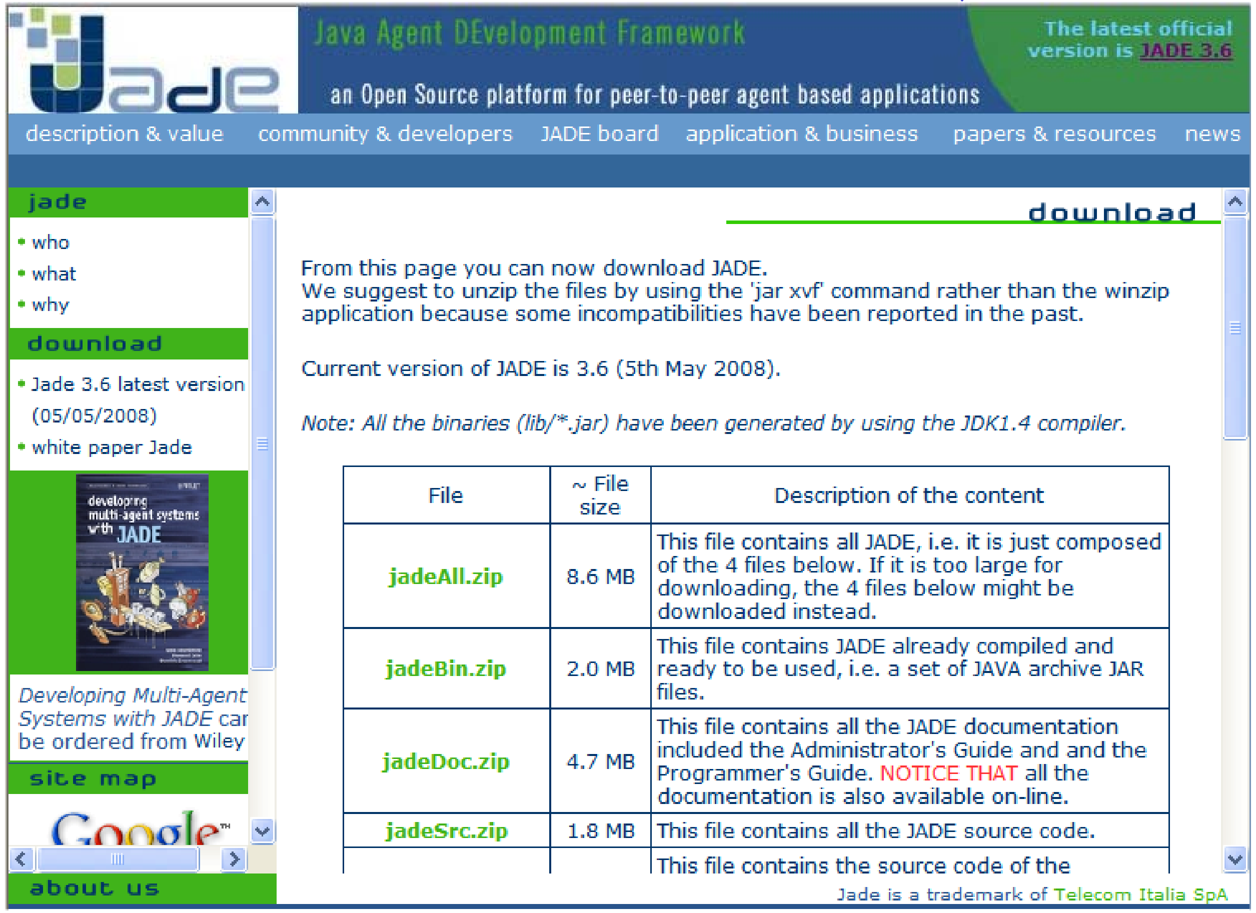
Interaction Protocols

- JADE provides ready-made behaviour classes for both roles in conversations following most FIPA interaction protocols
  - They offer a set of callback methods to handle the states of the protocols with an homogeneous API
  - All initiator behaviours terminate and are removed from the queue of the agent tasks as soon as they reach any final state of the interaction protocol
  - All responder behaviours are cyclic and are rescheduled as soon as they reach any final state of the interaction protocol

AOT LAB

Downloading JADE

http://jade.tilab.com


Java Agent DEvelopment Framework
an Open Source platform for peer-to-peer agent based applications
The latest official version is JADE 3.6
description & value
community & developers
JADE board
application & business
papers & resources
news
jade
who
what
why
download
Jade 3.6 latest version (05/05/2008)
white paper Jade
Developing Multi-Agent Systems with JADE can be ordered from Wiley
site map
about us
Jade is a trademark of Telecom Italia SpA

download

From this page you can now download JADE.
We suggest to unzip the files by using the 'jar xvf' command rather than the winzip application because some incompatibilities have been reported in the past.

Current version of JADE is 3.6 (5th May 2008).

Note: All the binaries (lib/.jar) have been generated by using the JDK1.4 compiler.*

File	~ File size	Description of the content
jadeAll.zip	8.6 MB	This file contains all JADE, i.e. it is just composed of the 4 files below. If it is too large for downloading, the 4 files below might be downloaded instead.
jadeBin.zip	2.0 MB	This file contains JADE already compiled and ready to be used, i.e. a set of JAVA archive JAR files.
jadeDoc.zip	4.7 MB	This file contains all the JADE documentation included the Administrator's Guide and and the Programmer's Guide. NOTICE THAT all the documentation is also available on-line.
jadeSrc.zip	1.8 MB	This file contains all the JADE source code.
		This file contains the source code of the

AOT LAB

Launching a JADE Platform

- A Jade platform/container can be launched through the following command

```
java jade.Boot [options] [agent specifiers]
```

- For example:

```
java jade.Boot –gui –nomtp

java jade.Boot –gui –port 1100 –host aot.ce.unipr.it
 –container example1: it.unipr.aotlab.demo.demoAgent
```

Options

- help: prints on standard output the help information
- container: creates a new container and joins it to an existing platform
- host <hostname> : specifies the host where the main container to register is running (default: “localhost”)
- port: specifies the port number where the main container to register is running (default: “1099”)
- gui: launches the RMA GUI of JADE
- mtp/nomtp: specifies a list of external Message Transport Protocols to be activated
- conf <filename>: creates/loads a configuration file
- <keyword> <value>: specifies other properties

Agent Specifiers

- A sequence of strings separated by a space specifying the agents to launch

  `<agentName>:<agentClass>(<agentArgs>)`

- <agentName> is the name of the Agent
- <agentClass> is the name of the Java class implementing the Agent. The name of the class must be fully qualified.
- <agentArgs> is a list of arguments dependent of the Agent implementation
  - They can be retrieved by using the **`getArguments()`** method of the Agent class (it is usually done in the setup() method