Giovanni Caire

JADE Board Technical Leader

giovanni.caire@tilab.com

JADE Tutorial for beginners

organized by the JADE Board

The Hague, 12th Oct. 2004

Outline

- JADE basics
  - Creating Agents
  - Agent tasks
  - Agent communication
  - The Yellow Pages service
- JADE advanced
  - Managing content expressions
  - Interaction protocols
  - Working with the AMS
  - Mobility
  - Security
  - Others
- JADE on mobile phones
  - The LEAP add-on
  - Creating JADE-based MIDlet

The HalloWorld agent

- A type of agent is created by extending the `jade.core.Agent` class and redefining the `setup()` method.
- Each Agent instance is identified by an AID (`jade.core.AID`).
  - An AID is composed of a unique name plus some addresses
  - An agent can retrieve its AID through the `getAID()` method of the `Agent` class

```
import jade.core.Agent;

public class HalloWorldAgent extends Agent {

  protected void setup() {
    System.out.println("Hallo World! my name is "+getAID().getName());
  }
}
```

Local names, GUID and addresses

- Agent names are of the form <local-name>@<platform-name>
- The complete name of an agent must be globally unique.
- The default platform name is <main-host>:<main-port>/JADE
- The platform name can be set using the `-name` option
- Within a single JADE platform agents are referred through their names only.
- Given the name of an agent its AID can be created as
  - `AID id = new AID(localname, AID.ISLOCALNAME);`
  - `AID id = new AID(name, AID.ISGUID);`
- The addresses included in an AID are those of the platform MTPs and are ONLY used in communication between agents living on different FIPA platforms

Passing arguments to an agent

- It is possible to pass arguments to an agent
  - `java jade.Boot .... a:myPackage.MyAgent(arg1 arg2)`
  - The agent can retrieve its arguments through the `getArguments()` method of the `Agent` class

```
protected void setup() {
  System.out.println("Hallo World! my name is "+getAID().getName());
  Object[] args = getArguments();
  if (args != null) {
    System.out.println("My arguments are:");
    for (int i = 0; i < args.length; ++i) {
      System.out.println("- "+args[i]);
    }
  }
}
```

Agent termination

- An agent terminates when its `doDelete()` method is called.
- On termination the agent's `takeDown()` method is invoked (intended to include clean-up operations).

```
protected void setup() {
  System.out.println("Hallo World! my name is "+getAID().getName());
  Object[] args = getArguments();
  if (args != null) {
    System.out.println("My arguments are:");
    for (int i = 0; i < args.length; ++i) {
      System.out.println("- "+args[i]);
    }
  }
  doDelete();
}

protected void takeDown() {
  System.out.println("Bye...");
}
```

Outline

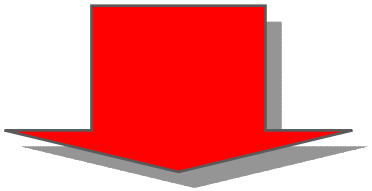
- JADE basics
  - Creating Agents
  - Agent tasks
  - Agent communication
  - The Yellow Pages service
- JADE advanced
  - Managing content expressions
  - Interaction protocols
  - Working with the AMS
  - Mobility
  - Security
  - Others
- JADE on mobile phones
  - The LEAP add-on
  - Creating JADE-based MIDlet

The Behaviour class

- The actual job that an agent does is typically carried out within "behaviours"
- Behaviours are created by extending the `jade.core.behaviours.Behaviour` class
- To make an agent execute a task it is sufficient to create an instance of the corresponding `Behaviour` subclass and call the `addBehaviour()` method of the `Agent` class.
- Each Behaviour subclass must implement
  - `public void action()`: what the behaviour actually does
  - `public boolean done()`: Whether the behaviour is finished

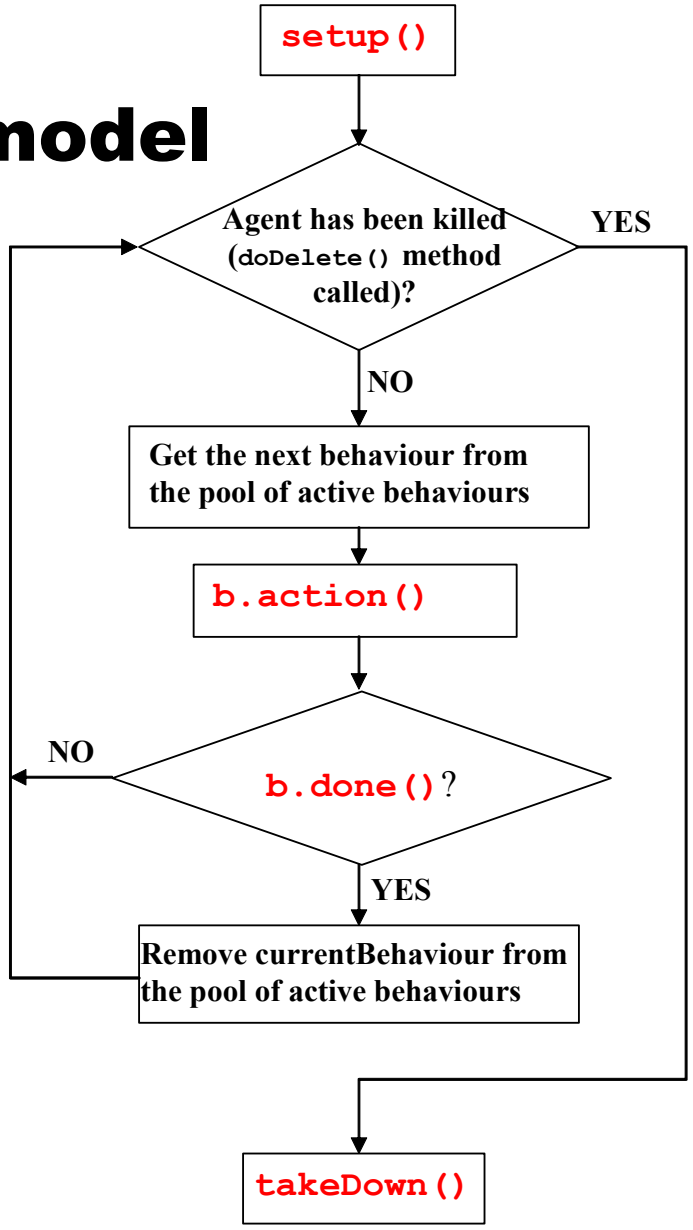
Behaviour scheduling and execution

- An agent can execute several behaviours in parallel, however, behaviour scheduling is not preemptive, but cooperative and everything occurs within a single Java Thread



Behaviour switch occurs only when the `action()` method of the currently scheduled behaviour returns.

The agent execution model



Highlighted in red the methods that programmers have to/can implement

- Initializations
- Addition of initial behaviours

- Agent "life" (execution of behaviours)

- Clean-up operations

Behaviour types

- "One shot" behaviours.
  - Complete immediately and their `action()` method is executed only once.
  - Their `done()` method simply returns `true`.
  - `jade.core.behaviours.OneShotBehaviour` class
- "Cyclic" behaviours.
  - Never complete and their `action()` method executes the same operation each time it is invoked
  - Their `done()` method simply returns `false`.
  - `jade.core.behaviours.CyclicBehaviour` class
- "Complex" behaviours.
  - Embed a state and execute in their `action()` method different operation depending on their state.
  - Complete when a given condition is met.

Scheduling operations at given points in time

- JADE provides two ready-made classes by means of which it is possible to easily implement behaviours that execute certain operations at given points in time
- `WakerBehaviour`
  - The `action()` and `done()` method are already implemented so that the `onWake()` method (to be implemented by subclasses) is executed after a given timeout
  - After that execution the behaviour completes.
- `TickerBehaviour`
  - The `action()` and `done()` method are already implemented so that the `onTick()` (to be implemented by subclasses) method is executed periodically with a given period
  - The behaviour runs forever unless its `stop()` method is executed.

More about behaviours

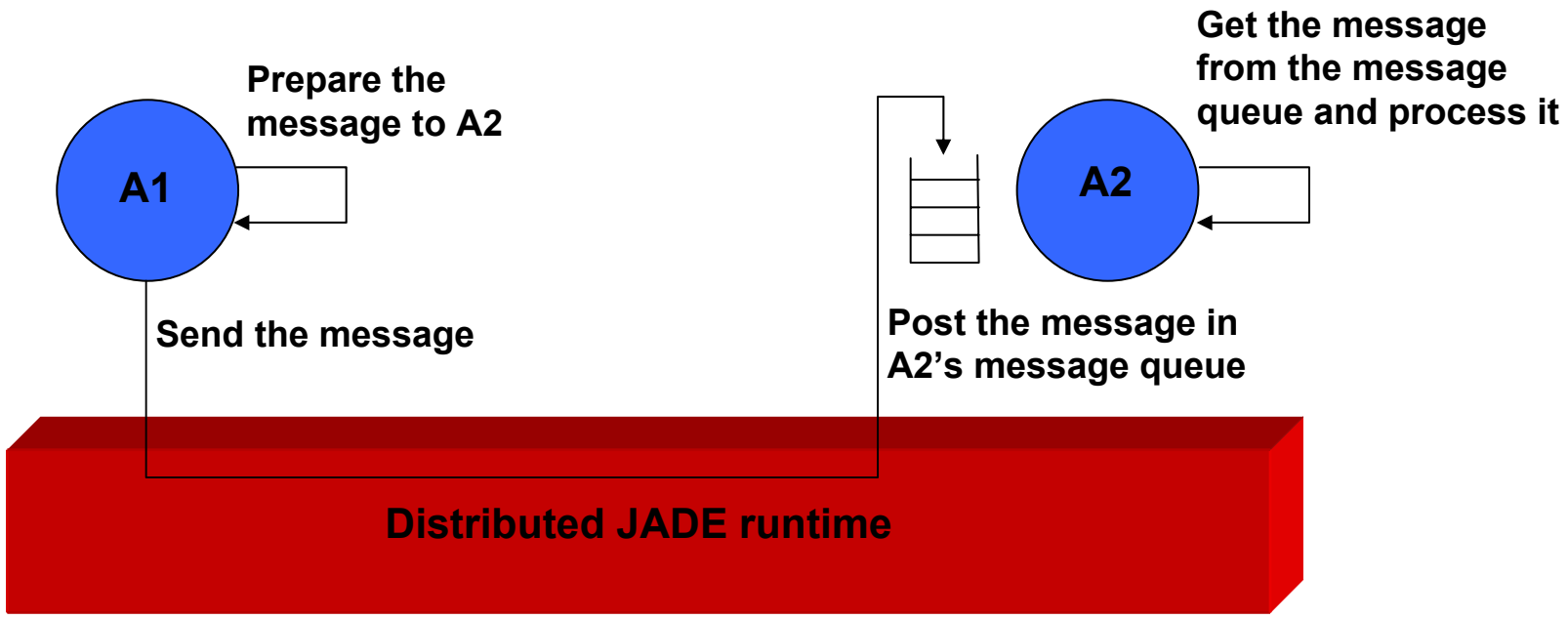
- The `onStart()` method of the `Behaviour` class is invoked only once before the first execution of the `action()` method. Suited for operations that must occur at the beginning of the behaviour
- The `onEnd()` method of the `Behaviour` class is invoked only once after the `done()` method returns `true`. Suited for operations that must occur at the end of the behaviour
- Each behaviour has a pointer to the agent executing it: the protected member variable `myAgent`
- The `removeBehaviour()` method of the `Agent` class can be used to remove a behaviour from the agent pool of behaviours. The `onEnd()` method is not called.
- When the pool of active behaviours of an agent is empty the agent enters the IDLE state and its thread goes to sleep

Outline

- JADE basics
  - Creating Agents
  - Agent tasks
  - Agent communication
  - The Yellow Pages service
- JADE advanced
  - Managing content expressions
  - Interaction protocols
  - Working with the AMS
  - Mobility
  - Security
  - Others
- JADE on mobile phones
  - The LEAP add-on
  - Creating JADE-based MIDlet

The communication model

- Based on asynchronous message passing
- Message format defined by the ACL language (FIPA)



The ACLMessage class

- Messages exchanged by agents are instances of the `jade.lang.acl.ACLMessage` class.
- Provide accessor methods to get and set all the fields defined by the ACL language
  - get/setPerformative();
  - get/setSender();
  - add/getAllReceiver();
  - get/setLanguage();
  - get/setOntology();
  - get/setContent();
  - ....

Sending and receiving messages

- Sending a message is as simple as creating an `ACLMessage` object and calling the `send()` method of the `Agent` class

```
ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
msg.addReceiver(new AID("Peter", AID.ISLOCALNAME));
msg.setLanguage("English");
msg.setOntology("Weather-Forecast-Ontology");
msg.setContent("Today it's raining");
send(msg);
```

- Reading messages from the private message queue is accomplished through the `receive()` method of the `Agent` class.

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

Blocking a behaviour waiting for a message

- A behaviour that processes incoming messages does not know exactly when a message will arrive → It should poll the message queue by continuously calling `myAgent.receive()`.
- This of course would completely waste the CPU time.
- The `block()` method of the `Behaviour` class removes a behaviour from the agent pool and puts it in a blocked state (it's not a blocking call!!).
- Each time a message is received all blocked behaviours are inserted back in the agent pool and have a chance to read and process the message.

```
public void action() {
  ACLMessage msg = myAgent.receive();
  if (msg != null) {
    // Process the message
  }
  else {
    block();
  }
}
```

This is the strongly recommended pattern to receive messages within a behaviour

Selective reading from the message queue

- The `receive()` method returns the first message in the message queue and removes it.
- If there are two (or more) behaviours receiving messages, one may "steal" a message that the other one was interested in.
- To avoid this it is possible to read only messages with certain characteristics (e.g. whose sender is agent "Peter") specifying a `jade.lang.acl.MessageTemplate` parameter in the `receive()` method.

```
MessageTemplate tpl = MessageTemplate.MatchOntology("Test-Ontology");

public void action() {
  ACLMessage msg = myAgent.receive(tpl);
  if (msg != null) {
    // Process the message
  }
  else {
    block();
  }
}
```

Receiving messages in blocking mode

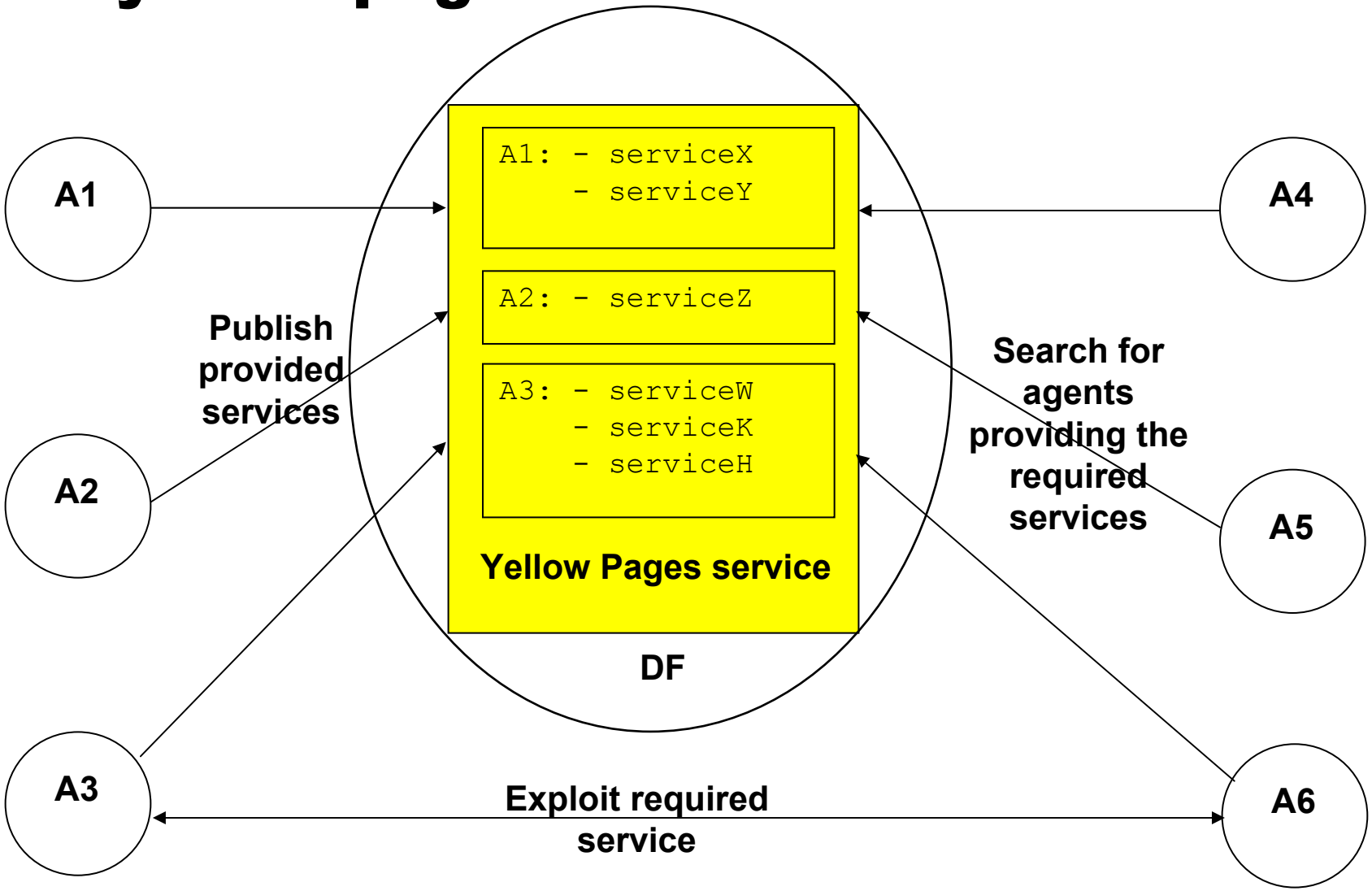
- The `Agent` class also provides the `blockingReceive()` method that returns only when there is a message in the message queue.
- There are overloaded versions that accept a `MessageTemplate` (the method returns only when there is a message matching the template) and or a timeout (if it expires the method returns null).
- Since it is a blocking call it is "dangerous" to use `blockingReceive()` within a behaviour. In fact no other behaviour can run until `blockingReceive()` returns.

- Use `receive()` + `Behaviour.block()` to receive messages within behaviours.
- Use `blockingReceive()` to receive messages within the agent `setup()` and `takeDown()` methods.

Outline

- JADE basics
  - Creating Agents
  - Agent tasks
  - Agent communication
  - The Yellow Pages service
- JADE advanced
  - Managing content expressions
  - Interaction protocols
  - Working with the AMS
  - Mobility
  - Security
  - Others
- JADE on mobile phones
  - The LEAP add-on
  - Creating JADE-based MIDlet

The yellow pages service



Interacting with the DF Agent

- The DF is an agent and as such it communicates using ACL
- The ontology and language that the DF "understands" are specified by FIPA → It is possible to search/register to a DF agent of a remote platform.
- The `jade.domain.DFService` class provides static utility methods that facilitate the interactions with the DF
  - register();
  - modify();
  - deregister();
  - search();
- The JADE DF also supports a subscription mechanism

DFDescription format

- When an agent registers with the DF it must provide a description (implemented by the `jade.domain.FIPAAgentManagement.DFAgentDescription` class) basically composed of
  - The agent `AID`
  - A collection of service descriptions (implemented by the class `ServiceDescription`). This, on its turn, includes:
    - The service type (e.g. “Weather forecast”);
    - The service name (e.g. “Meteo-1”);
    - The languages, ontologies and interaction protocols that must be known to exploit the service
    - A collection of service-specific properties in the form key-value pair
- When an agent searches/subscribes to the DF it must specify another `DFAgentDescription` that is used as a template

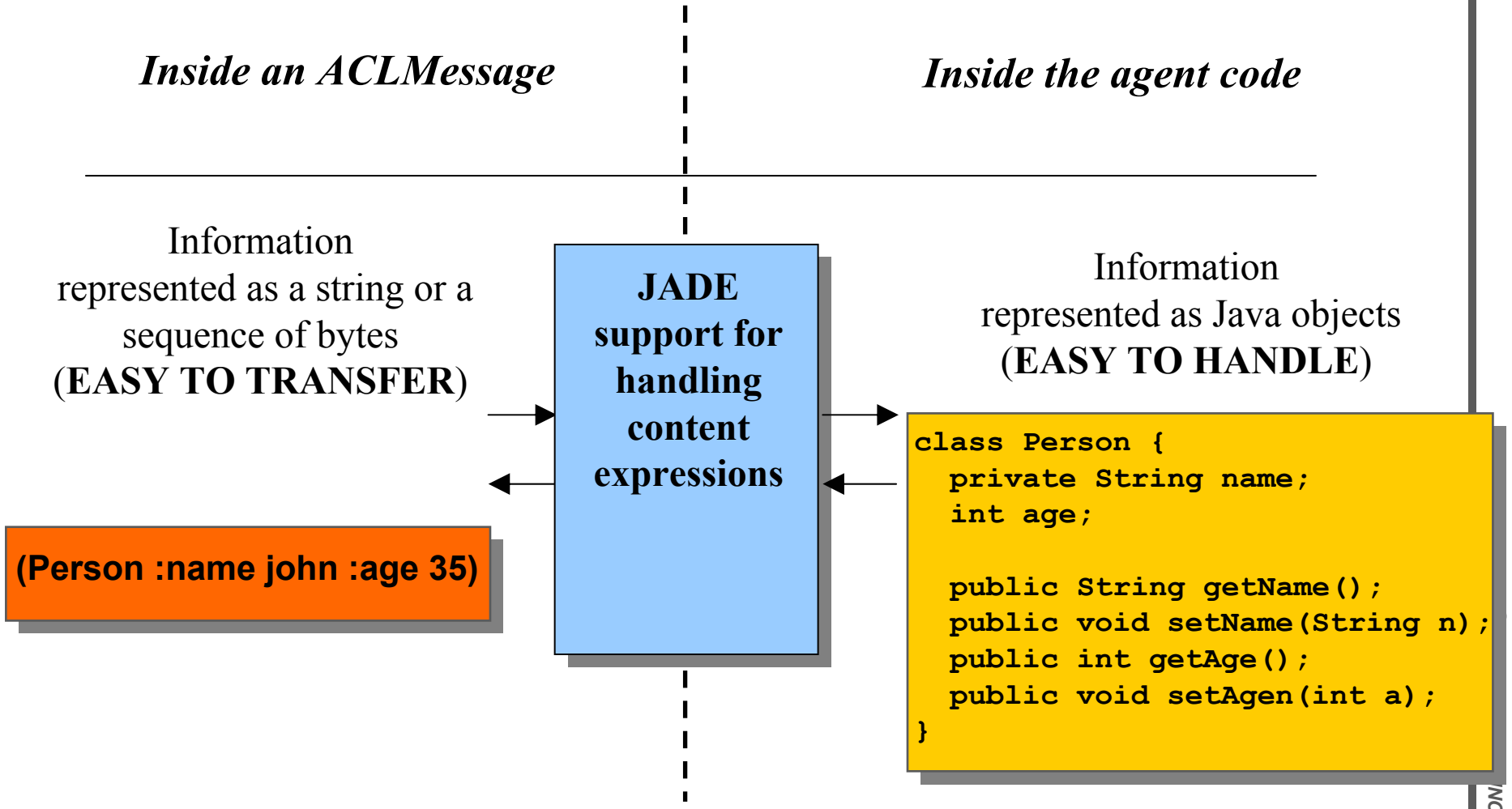
Outline

- JADE basics
  - Creating Agents
  - Agent tasks
  - Agent communication
  - The Yellow Pages service
- JADE advanced
  - Managing content expressions
  - Interaction protocols
  - Working with the AMS
  - Mobility
  - Security
  - Others
- JADE on mobile phones
  - The LEAP add-on
  - Creating JADE-based MIDlet

Outline

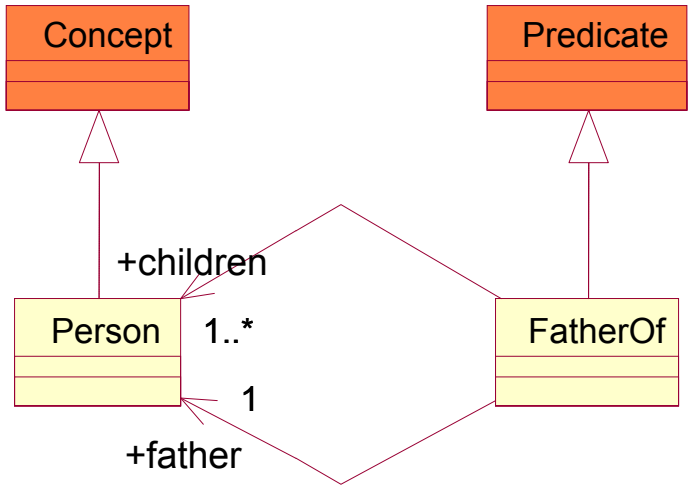
- JADE basics
  - Creating Agents
  - Agent tasks
  - Agent communication
  - The Yellow Pages service
- JADE advanced
  - Managing content expressions
  - Interaction protocols
  - Working with the AMS
  - Mobility
  - Security
  - Others
- JADE on mobile phones
  - The LEAP add-on
  - Creating JADE-based MIDlet

JADE support for handling content expressions



How it works

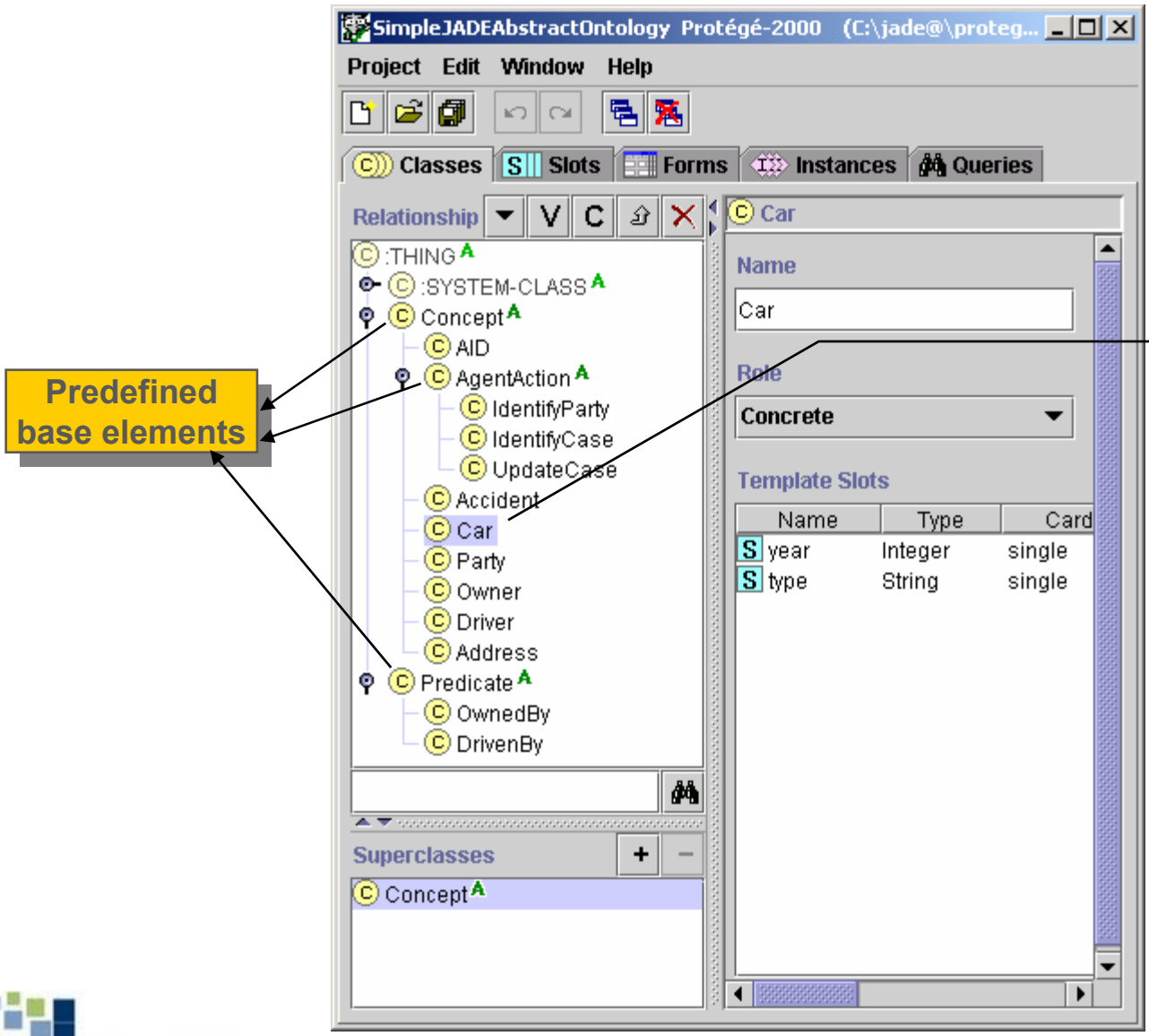
- Creating the Ontology (domain specific)
  - Defining the schemas ontology elements
  - Defining the corresponding Java classes
- Handling content expressions as Java objects
- Using the ContentManager to fill and parse message contents




```
Person john = new Person("John", 35);
Person bill = new Person("Bill", 67);
FatherOf f = new FatherOf();
f.setFather(bill);
f.addChildren(john);
```

(father-of (person :name Bill :age 67) (set (person :name John :age 35)))

Creating ontologies with Protege`



Beangenerator

```
class Car extends Concept {
   private int year;
   private String type;
   .....
}
```

Documentation

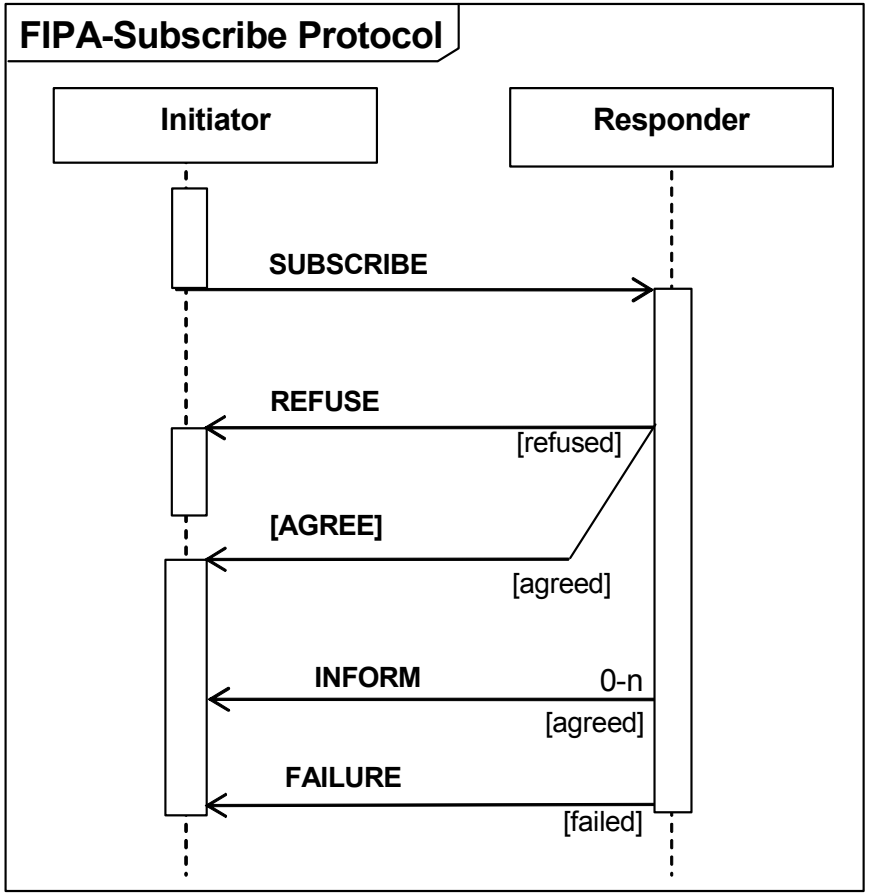
- A complete tutorial is available on the JADE site:
- http://jade.tilab.com/doc/CLOntoSupport.pdf
- API documentation (javadoc): `jade.content` package and subpackages
- Sample code: `examples.content` package in the examples included in the JADE distribution.

Outline

- JADE basics
  - Creating Agents
  - Agent tasks
  - Agent communication
  - The Yellow Pages service
- JADE advanced
  - Managing content expressions
  - Interaction protocols
  - Working with the AMS
  - Mobility
  - Security
  - Others
- JADE on mobile phones
  - The LEAP add-on
  - Creating JADE-based MIDlet

Interaction protocols

- Having a standard set of types of messages (INFORM, REQUEST, PROPOSE ....) allows specifying predefined sequences of messages exchanged by agents during a conversations.
- These are known as Interaction Protocols



Support for interaction protocols

- The `jade.proto` package contains behaviours for both the initiator and responder role in the most common interaction protocol:
  - FIPA-request (`AchieveREInitiator/Responder`)
  - FIPA-Contract-Net (`ContractNetInitiator/Responder`)
  - FIPA-Subscribe (`SubscriptionInitiator/Responder`)
- All these classes automatically handle
  - the flow of messages checking that it is compliant to the protocol
  - The timeouts (if any)
- They provide callback methods that should be redefined to take the necessary actions when e.g. a message is received or a timeout expires.

Documentation

- Chapter 3.5 in the Programmers guide included in the JADE distribution provides a detailed explanation of the interaction protocol support
- API documentation (javadoc): `jade.proto` package
- Sample code: `examples.protocols` package in the examples included in the JADE distribution.

Outline

- JADE basics
  - Creating Agents
  - Agent tasks
  - Agent communication
  - The Yellow Pages service
- JADE advanced
  - Managing content expressions
  - Interaction protocols
  - Working with the AMS
  - Mobility
  - Security
  - Others
- JADE on mobile phones
  - The LEAP add-on
  - Creating JADE-based MIDlet

Working with the AMS

- The AMS (Agent Management System) represents the authority in a JADE platform.
- All platform management actions (creating/killing agents, killing containers...) are under the control of the AMS.
- Other agents can request the AMS to perform these actions by using
  - The fipa-request interaction protocol
  - The SL language
  - The JADE-Management ontology and related actions
- The AID of the AMS can be retrieved through the `getAMS()` method of the `Agent` class

Outline

- JADE basics
  - Creating Agents
  - Agent tasks
  - Agent communication
  - The Yellow Pages service
- JADE advanced
  - Managing content expressions
  - Interaction protocols
  - Working with the AMS
  - Mobility
  - Security
  - Others
- JADE on mobile phones
  - The LEAP add-on
  - Creating JADE-based MIDlet

Mobility

- JADE supports “hard mobility” i.e. mobility of status and code.
  - Status: an agent can i) stop its execution on the local container ii) move to a remote container (likely on a different host) and iii) restart its execution there from the exact point where it was interrupted.
  - Code: If the code of the moving agent is not available on the destination container it is automatically retrieved on demand.
- In order to be able to move, an agent must be `Serializable`
- Mobility can be
  - self-initiated through the `doMove()` method of the `Agent` class
  - forced by the AMS (following a request from another agent)
- Besides mobility also agent cloning is available (method `doClone()`)

Documentation

- Chapter 3.7 in the Programmers guide included in the JADE distribution provides a detailed explanation of the mobility support
- API documentation (javadoc): `jade.core.Agent` class, `jade.core.Location` interface and `jade.domain.mobility` package
- Sample code: `examples.mobile` package in the examples included in the JADE distribution.

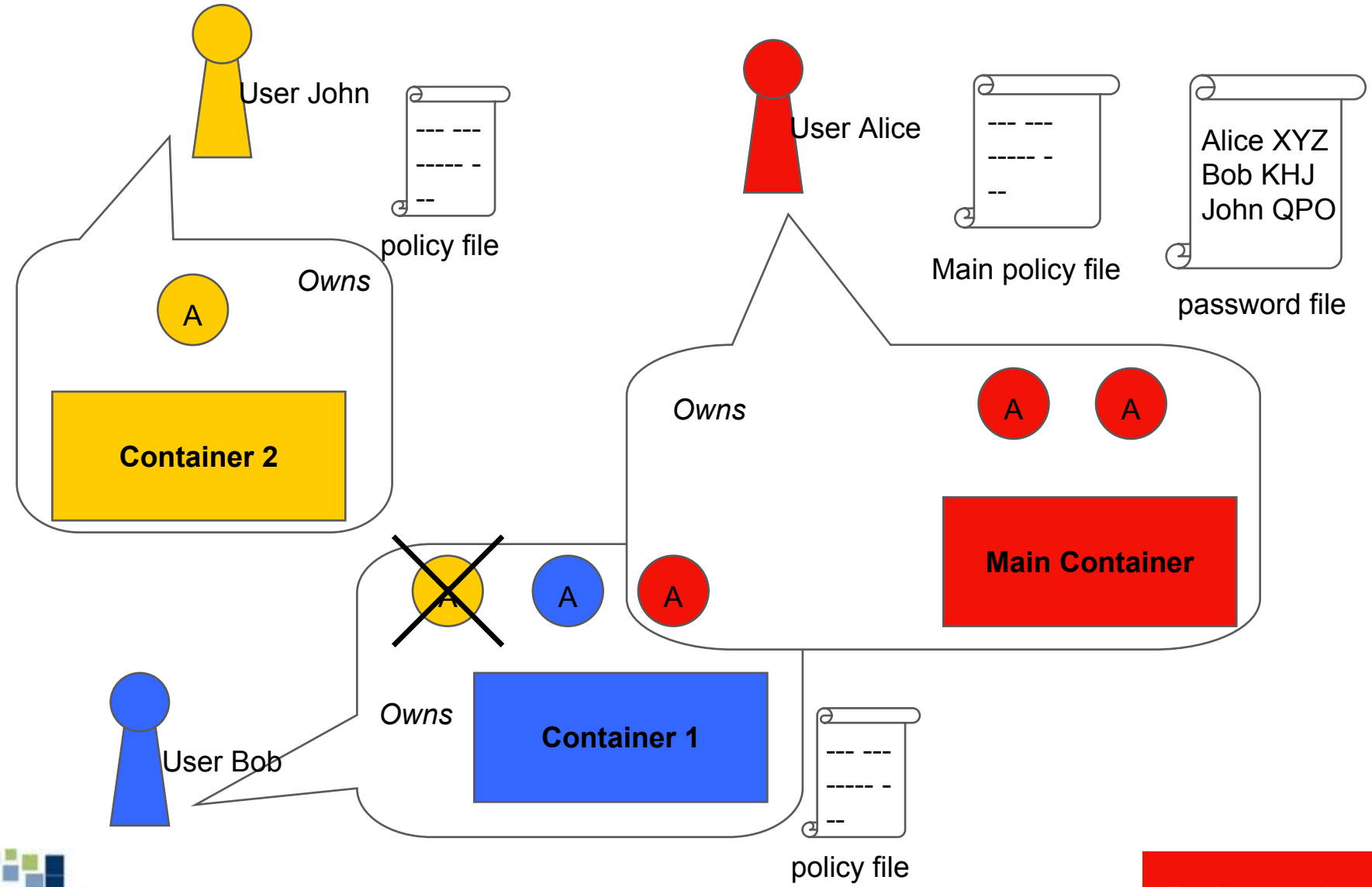
Outline

- JADE basics
  - Creating Agents
  - Agent tasks
  - Agent communication
  - The Yellow Pages service
- JADE advanced
  - Managing content expressions
  - Interaction protocols
  - Working with the AMS
  - Mobility
  - Security
  - Others
- JADE on mobile phones
  - The LEAP add-on
  - Creating JADE-based MIDlet

Security

- Distributed as an add-on
- Examples of potential threats:
  - A malicious agent can request the AMS to kill another agent
  - A malicious agent can request the AMS to shutdown the platform
  - A malicious entity can sniff or tamper sensible information included in an ACLMessage
- Prevents the above threats by providing support for:
  - Authentication and authorization
  - End-to-end message integrity and confidentiality
- Based on JAAS
- Further evolutions are expected → Comments and contributions are Welcome

JADE as a multi-user environment



Policy file

- grant principal jade.security.Name "alice" {
  permission jade.security.PlatformPermission "", "create";
  permission jade.security.ContainerPermission "", "create";
  permission jade.security.AMSPermission "agent-class=*", "register, derister,modify";
  permission jade.security.AgentPermission "agent-class=*", "create, kill";
  permission jade.security.MessagePermission "agent-owner:alice", "send-to";
};

Signing and encrypting messages

- All security related API are embedded into the `SecurityHelper`
- Can be retrieved through the `getHelper()` method of the `Agent` class

```
// Create the message
ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
......

// Retrieve the SecurityHelper
SecurityHelper myHelper = (SecurityHelper)
getHelper("jade.core.security.Security");

// The message must be signed
mySecurityHelper.setUseSignature(msg);

// Send the message
send(msg);
```

Documentation

- The Security add-on comes with a complete guide and some code examples.

Other advanced features

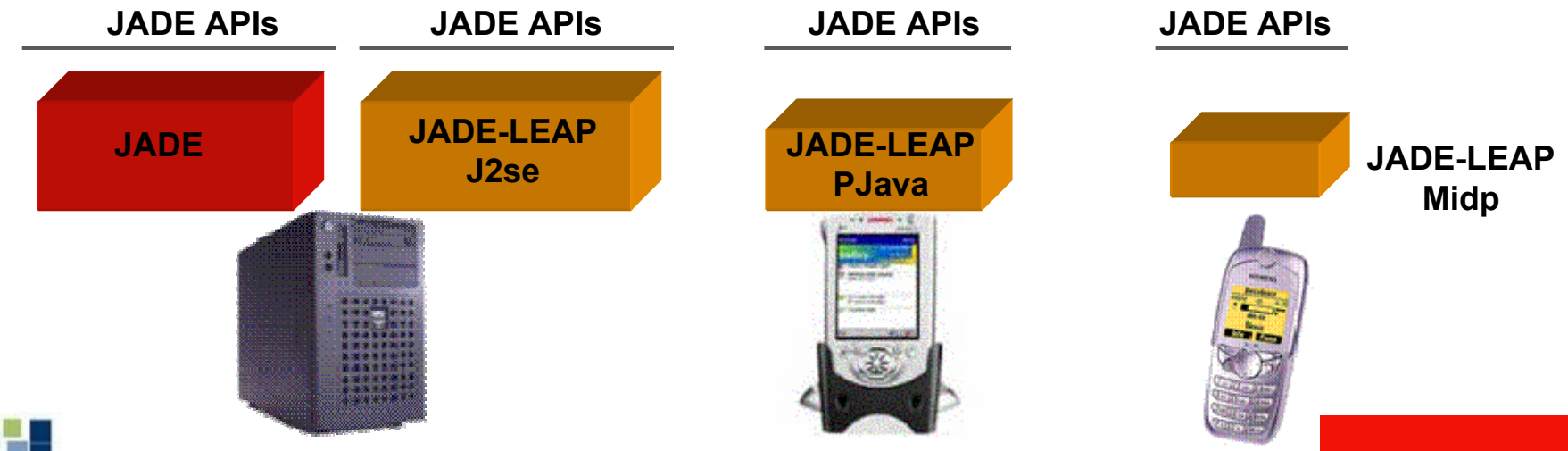
- **In-process interface**
  - Allow using JADE (i.e. creating a container and starting agents on it) from an external Java program.
  - `jade.core.Runtime` class and `jade.wrapper` package
  - Documentation: chapter 3.8 of the JADE programmers guide.
- **Threaded behaviours**
  - Allow executing a normal JADE behaviour in a dedicated thread
  - `jade.core.behaviours.ThreadedBehaviourFactory` class
  - Documentation: chapter 3.4.13 of the JADE programmers guide
- **Persistence**
  - Allow saving and reloading agent state on relational DB
  - Based on Hibernate (http://www.hibernate.org)
  - Distributed as an add-on
  - Still in experimental version → Comments and contributions welcome

Outline

- JADE basics
  - Creating Agents
  - Agent tasks
  - Agent communication
  - The Yellow Pages service
- JADE advanced
  - Managing content expressions
  - Interaction protocols
  - Working with the AMS
  - Mobility
  - Security
  - Others
- JADE on mobile phones
  - The LEAP add-on
  - Creating JADE-based MIDlet

The LEAP add-on

- Allows running JADE agents on PDA and mobile phones (MIDP, PersonalJava)
- Developed in the LEAP IST European project (Motorola, BT, Broadcom, Siemens, Univ. Parma, ADAC, TILAB)
- Replaces some parts of JADE → JADE-LEAP (J2se, PJava, MIDP)
- Different internal implementation
- Same APIs for agents



Building JADE-LEAP

- JADE-LEAP can be downloaded in binary form (for J2se, PJava and MIDP) directly from the download area of the JADE site.
- Alternatively one can download the JADE sources, then the LEAP add-on and finally build JADE-LEAP using an ANT build file included in the LEAP add-on (ANT 1.6 is required).
- The LEAP add-on also includes the LEAP user guide that provides all the details about using JADE-LEAP.
- Using JADE-LEAP for J2se is identical to using JADE (same tools, configuration parameters...) except for:
  - Agents on the command line must be separated by ';' instead of blank
  - Agent arguments must be separated by ',' instead of a blank (both on the command line and in the RMA)

Creating JADE-based MIDlet

- All MIDP applications are called MIDlets and their main class extends `javax.microedition.midlet.MIDlet`.
- JADE-LEAP for MIDP is itself a MIDlet
  - The main class is `jade.MicroBoot`
- A MIDlet must be packaged as a single JAR file
  - A JADE-based MIDlet JAR file must include both JADE-LEAP classes and application specific classes and resources (e.g. images)
  - The typical build process includes the following steps:
    - Compile the application sources with JADE-LEAP in the classpath
    - Mix application .class files and JADE-LEAP .class files
    - Pre-verify the whole
    - Jar the whole

Configuration parameters in MIDP

- JADE accepts a number of configuration parameters either on the command line or in a property file
- Both ways of specifying parameters are not available in MIDP
- In MIDP configuration parameters are specified in the JAD/manifest of the MIDlet.
- Example
  - java ... jade.Boot .... –host ibm8695 –port 2222 ....
  - Manifest

```
....
MIDlet-LEAP-host: ibm8695
MIDlet-LEAP-port: 2222
....
```

- Only a subset of the JADE configuration parameters are available/meaningful in a MIDP manifest/JAD (e.g. –container)

Minimization

- JADE-LEAP for MIDP is ~450 Kbytes: VERY BIG!
- In general however its really rare that an application exploits all features provided by JADE. It would be highly desirable to remove all classes related to unused features from the application JAR file
- This can be achieved by means of the minimize target of the ANT build file included in the LEAP add-on
  - `ant minimize -DJAR=<jar-file> -DDLC=<dlc-file> -DMANIFEST=<manifest>`
  - Where the dlc-file includes the list of dynamically loaded classes
- After the minimization process an average JADE-based application becomes ~150Kbytes
- Using a good obfuscator it is possible to sensibly reduce the code size further

Logging

- When running on a cell phone typically printouts produced by `System.out.println()` have no effect at all.
- The `jade.util.Logger` class provides a uniform way over J2SE, PJava and MIDP to produce printouts.
  - In J2SE it uses JAVA Logging and therefore allow exploiting all its flexibility
  - In PJava it simply uses `System.out.println()`
  - In MIDP it writes the output in a proper RecordStore where it can be later viewed by means of the `jade.util.leap.OutputViewer` MIDlet

Thanks for your attention!!!!

- Questions?