JADE

Java Agent DEvelopment Framework

A comprehensive guide to developing multi-agent systems

May 19, 2025

Contents

1 What is JADE? 3
1.1 Key Features of JADE . . . 3

2 What JADE Gives You 3
2.1 Agent Container & Lifecycle . . . 3
2.1.1 Agent Lifecycle in JADE . . . 3
2.2 Agent Communication . . . 5
2.2.1 Why Use ACL? . . . 5
2.2.2 ACL Message Structure . . . 5
2.2.3 Message Encoding . . . 6
2.2.4 Message Transport . . . 6
2.2.5 Message Delivery . . . 6
2.2.6 Code Example – Two Agents Communicating . . . 7
2.3 Directory Facilitator (DF) & AMS . . . 8
2.3.1 What is the DF (Directory Facilitator)? . . . 8
2.3.2 What is the AMS (Agent Management System)? . . . 10
2.4 Standard Behaviors . . . 11
2.4.1 JADE Standard Behaviors (Overview) . . . 12
2.5 Management GUIs . . . 14
2.5.1 What is the RMA? . . . 14
2.5.2 What Can You Do With RMA? . . . 15
2.5.3 How to Use RMA . . . 15
2.5.4 RMA Main Panels . . . 15

3 What's an Agent in JADE? 16
3.1 Agent Lifecycle . . . 16
3.2 Behaviours . . . 16
3.3 Messaging (ACLMessage) . . . 17
3.4 Service Registration & Discovery . . . 17

4 Spinning Up Containers & Agents 18
4.1 Main Container (Bootstrapping) . . . 18
4.2 Hosting Additional Containers (Remote) . . . 18

5 Mobility & Migration 18

6 Debugging & Monitoring 19

7 Common Utilities & Tips 19

8 Quick Reference: Core Classes 20

9 Documentation & Resources 20
9.1 Official Docs & Tutorials . . . 20
9.2 Books & Articles . . . 20

10 Build, Integrate & Deploy 20
10.1 Prerequisites . . . 20
10.2 Installing JADE . . . 21
10.2.1 Download Manually . . . 21

10.2.2 Using Maven 21
10.3 Project Setup 21
10.3.1 Directory Structure 21
10.3.2 IDE Configuration 21
10.4 Your First Agent 21
10.5 Running the Platform 22
10.5.1 Command-Line Bootstrap 22
10.5.2 Key Options 22

11 Advanced Topics & Patterns **22**
11.1 Design Patterns in MAS 22
11.2 Ontology & Content Languages 23
11.3 Agent Mobility & Cloning 23
11.4 Security & ACL Filtering 23
11.5 Performance Tuning 23

12 Common Pitfalls **23**

13 Community & Support **23**
13.1 JADE Mailing List 23
13.2 StackOverflow "jade-framework" Tag 23
13.3 GitHub Forks & Extensions 24

1 What is JADE?

JADE Info

JADE (Java Agent DEvelopment Framework) is a middleware platform used to develop multi-agent systems in Java. It simplifies the creation of distributed applications with intelligent agents that can communicate, move, and work together.

1.1 Key Features of JADE

JADE provides a comprehensive framework for developing multi-agent systems with the following features:

- Agent container and lifecycle management
- Standardized agent communication
- Directory services for agent discovery
- Graphical management tools
- Behavior-based agent programming model
- Support for mobility and cloning
- FIPA compliance

2 What JADE Gives You

2.1 Agent Container & Lifecycle

JADE runs a "container" (a JVM process) that hosts one or more agents. It handles starting, stopping, and monitoring them throughout their lifecycle.

2.1.1 Agent Lifecycle in JADE

Agents in JADE follow a defined lifecycle managed by their containers:

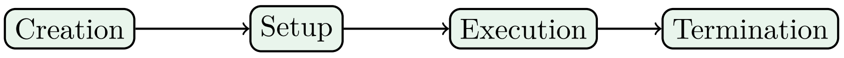

Figure 1: Agent Lifecycle in JADE

1. Creation Agents are created using the `createNewAgent` method.

Code Example

```java
AgentController agent = container.createNewAgent("MyAgent", "
mypackage.MyAgentClass", null);
agent.start();
```

The container:

- Allocates memory for the agent

- Initializes the agent object
- Places the agent in the "initiated" state

2. Setup The `setup()` method of the agent is called.

Code Example

```java
protected void setup() {
    // Agent initialization code
}
```

This is where you:

- Define the agent's behavior
- Register it with services like Directory Facilitator (DF)

3. Execution (Running) The agent enters the "active" state and:

- Performs tasks
- Listens for messages
- Reacts to events

This stage involves different behaviors, such as:

- SimpleBehaviour
- CyclicBehaviour
- OneShotBehaviour
- TickerBehaviour

4. Termination When the agent is done or explicitly told to stop:

Code Example

```java
protected void takeDown() {
    // Cleanup code before agent is terminated
}
```

This is where:

- Cleanup happens (e.g., deregister from DF)
- Resources are released
- The container removes the agent from memory

Monitoring by the Container JADE containers also:

- Monitor the health of agents
- Can restart agents that fail
- Can move or clone agents to other containers (mobility)
- Log lifecycle events (creation, start, stop, etc.)

Concept	**Description**
Container	JVM process that hosts agents
Main Container	Central hub with JADE services (AMS, DF)
Agent Lifecycle	Create → Setup → Run → Terminate
Agent Control	Container starts/stops/monitors agents
Distributed	Containers can be on different machines, coordinated by the main container

2.2 Agent Communication

Built-in ACL (Agent Communication Language) messaging provides encoding, transport, and delivery. Agent communication is one of the core concepts in JADE and multi-agent systems in general.

2.2.1 Why Use ACL?

ACL is designed to allow agents to:

- Share information
- Request actions
- Coordinate plans
- Negotiate, argue, etc.

JADE provides a built-in communication system that handles the encoding, transport, and delivery of these messages.

2.2.2 ACL Message Structure

An ACL message is an object of the class `jade.lang.acl.ACLMessage`. It contains:

Field	Description
performative	Type of communication act (e.g., INFORM, REQUEST, PROPOSE...)
sender	The agent sending the message
receivers	One or more agents to receive the message
content	The actual data or command being communicated
language	The language used to encode the content
ontology	The domain or context of the content
conversation-id	Used to track a conversation
reply-with, **in-reply-to**	Used to manage replies

2.2.3 Message Encoding

JADE uses Java objects to build messages, but messages are sent in text format using FIPA-ACL rules:

Code Example

```
(REQUEST
 :sender agent1@host
 :receiver agent2@host
 :content "Please send me the latest data"
 :language English
 :ontology WeatherOntology
)
```

2.2.4 Message Transport

JADE handles the transport layer internally:

- Messages are sent over TCP/IP or HTTP depending on the platform configuration
- Each container can be local or distributed across machines
- The Main container coordinates the routing of messages between agents in different containers

2.2.5 Message Delivery

JADE ensures reliable asynchronous delivery:

- The message is stored in the receiver agent's message queue
- The receiver must use behaviors (like CyclicBehaviour) to poll and process incoming messages

2.2.6 Code Example – Two Agents Communicating

SenderAgent.java

Code Example

```java
import jade.core.Agent;
import jade.core.behaviours.OneShotBehaviour;
import jade.lang.acl.ACLMessage;

public class SenderAgent extends Agent {
    protected void setup() {
        System.out.println("Sender ready.");

        addBehaviour(new OneShotBehaviour() {
            public void action() {
                ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
                msg.addReceiver(new jade.core.AID("receiver", AID.
                ISLOCALNAME));
                msg.setLanguage("English");
                msg.setContent("Hello from SenderAgent!");
                send(msg);
                System.out.println("Message sent.");
            }
        });
    }
}
```

ReceiverAgent.java

Code Example

```java
import jade.core.Agent;
import jade.core.behaviours.CyclicBehaviour;
import jade.lang.acl.ACLMessage;

public class ReceiverAgent extends Agent {
    protected void setup() {
        System.out.println("Receiver ready.");

        addBehaviour(new CyclicBehaviour() {
            public void action() {
                ACLMessage msg = receive();
                if (msg != null) {
                    System.out.println("Received message: " + msg.
                    getContent());
                } else {
                    block();  // Waits until a message arrives
                }
            }
        });
    }
}
```

Run with JADE Boot Command

Code Example

```
java -cp .:jade.jar jade.Boot -gui sender:SenderAgent receiver:ReceiverAgent
```

On Windows, use `;` instead of `:` for the classpath.

Summary

Step	Responsibility	Description
1. Encoding	`ACLMessage`	Create and fill out message fields in Java. JADE handles ACL forma
2. Transport	JADE platform	Routes messages through containers using TCP/HTTP.
3. Delivery	JADE agent runtime	Places messages into the receiver's message queue for processing.

Common Performatives

Performative	Meaning
`INFORM`	"Here's some information."
`REQUEST`	"Please do this."
`PROPOSE`	"Let's negotiate."
`AGREE`	"I will do it."
`REFUSE`	"I won't do it."
`FAILURE`	"It didn't work."
`CONFIRM`, `DISCONFIRM`	Confirm or deny info.

2.3 Directory Facilitator (DF) & AMS

A yellow-pages (DF) so agents can register/find services; the AMS keeps track of all agents and policies.

2.3.1 What is the DF (Directory Facilitator)?

The DF is like the "Yellow Pages" of JADE:

- Agents can register the services they provide
- Agents can search for services offered by other agents

DF Responsibilities:

- Agents register the services they offer
- Agents search the DF to discover other agents by service name/type
- Enables dynamic and decoupled communication: agents don't need to know each other in advance

Example: Agent Registers a Service with DF

Code Example

```java
import jade.core.Agent;
import jade.domain.DFService;
import jade.domain.FIPAAgentManagement.*;
import jade.lang.acl.ACLMessage;
import jade.domain.FIPAException;

public class RegisterAgent extends Agent {
    protected void setup() {
        System.out.println("Agent " + getLocalName() + " started.");

        // Prepare description of the service
        DFAgentDescription dfd = new DFAgentDescription();
        dfd.setName(getAID());  // Agent's ID

        // Define a service
        ServiceDescription sd = new ServiceDescription();
        sd.setType("weather-info");
        sd.setName("weather-service");

        dfd.addServices(sd);

        try {
            DFService.register(this, dfd);  // Register with DF
            System.out.println("Service registered with DF.");
        } catch (FIPAException e) {
            e.printStackTrace();
        }
    }

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

Another Agent Searches for That Service

Code Example

```java
DFAgentDescription template = new DFAgentDescription();
ServiceDescription sd = new ServiceDescription();
sd.setType("weather-info");
template.addServices(sd);

try {
    DFAgentDescription[] result = DFService.search(myAgent, template)
    ;
    if (result.length > 0) {
        AID provider = result[0].getName();
        System.out.println("Found provider: " + provider.getLocalName
        ());
    }
} catch (FIPAException fe) {
    fe.printStackTrace();
}
```

2.3.2 What is the AMS (Agent Management System)?

The AMS is the "White Pages" and governing authority in JADE:

- It tracks all agents, their lifecycles, and enforces rules and policies of the platform

AMS Responsibilities:

- Creates, deletes, and monitors agents
- Manages agent names (AIDs) and addresses
- Enforces security and permissions
- Can process management requests like create, kill, suspend, move, etc.

Example: Create or Kill an Agent via AMS (Advanced Use) You can programmatically request the AMS to create or kill agents using FIPA-ACL messages with specific performatives and ontologies. This is mostly for advanced or administrative agents.

DF vs AMS – Summary Table

Feature	**Directory Facilitator (DF)**	**Agent Management System (AMS)**
Purpose	Service discovery (yellow pages)	Agent management (white pages)
Agents Register	Services they offer	Agent name/address (automatically)
Agents Can Search	For services (by type/name)	Not typically searched manually
Used By	Most user agents	System/internal or privileged agents
Examples	"weather-info", "chatbot", etc.	Creating, killing, moving agents

Real-World Analogy

JADE Component	**Real-World Analogy**
DF	Yellow Pages business list
AMS	Government registry office

2.4 Standard Behaviors

Reusable building-blocks like periodic tasks, one-shot tasks, cyclic listeners, finite-state machines, etc.

In JADE, behaviors are the core way to define what an agent does. JADE provides a library of standard behavior classes—modular, reusable building blocks for writing agent logic.

These are essential to model agent activities such as:

- Performing a task once
- Listening for messages
- Executing tasks periodically
- Managing complex workflows (e.g., state machines)

2.4.1 JADE Standard Behaviors (Overview)

Behavior Type	Class	Description
One-shot	`OneShotBehaviour`	Executes once and then ends
Cyclic	`CyclicBehaviour`	Runs indefinitely (like a message listener)
Ticker	`TickerBehaviour`	Executes repeatedly with a fixed time delay
Waker	`WakerBehaviour`	Executes once after a delay
FSM (Finite-State)	`FSMBehaviour`	Complex workflows; transitions between states
Parallel	`ParallelBehaviour`	Run multiple behaviors concurrently
Sequential	`SequentialBehaviour`	Run behaviors one after another

1. OneShotBehaviour – Do something once

Code Example

```java
addBehaviour(new OneShotBehaviour() {
    public void action() {
        System.out.println("One-shot task executed.");
    }
});
```

Use this for initialization, sending a message, or a one-time computation.

2. CyclicBehaviour – Loop forever

Code Example

```java
addBehaviour(new CyclicBehaviour() {
    public void action() {
        ACLMessage msg = receive();
        if (msg != null) {
            System.out.println("Received: " + msg.getContent());
        } else {
            block(); // Wait for next message
        }
    }
});
```

Best for message handling, monitoring, or event listening.

3. TickerBehaviour – Repeat every X ms

Code Example

```java
addBehaviour(new TickerBehaviour(this, 5000) {  // 5000 ms = 5 sec
    protected void onTick() {
        System.out.println("Periodic check or heartbeat.");
    }
});
```

Use for polling, health checks, scheduled updates.

4. WakerBehaviour – Wake up after a delay

Code Example

```java
addBehaviour(new WakerBehaviour(this, 10000) {  // Wait 10 sec
    protected void onWake() {
        System.out.println("Time's up! Doing delayed task.");
    }
});
```

Great for timeout triggers, delayed start, etc.

5. FSMBehaviour – Finite State Machine Ideal for modeling complex tasks like negotiations, protocols, or mission steps.

Code Example

```java
FSMBehaviour fsm = new FSMBehaviour(this);

// Define states
fsm.registerFirstState(new OneShotBehaviour() {
    public void action() {
        System.out.println("State A");
    }
}, "A");

fsm.registerState(new OneShotBehaviour() {
    public void action() {
        System.out.println("State B");
    }
}, "B");

// Define transitions
fsm.registerLastState(new OneShotBehaviour() {
    public void action() {
        System.out.println("Final state C");
    }
}, "C");

fsm.registerTransition("A", "B", 1);
fsm.registerTransition("B", "C", 2);

addBehaviour(fsm);
```

6. ParallelBehaviour – Run things in parallel

Code Example

```java
ParallelBehaviour parallel = new ParallelBehaviour(this,
ParallelBehaviour.WHEN_ALL);

parallel.addSubBehaviour(new OneShotBehaviour() {
    public void action() {
        System.out.println("Task 1");
    }
});

parallel.addSubBehaviour(new OneShotBehaviour() {
    public void action() {
        System.out.println("Task 2");
    }
});

addBehaviour(parallel);
```

Options:

- WHEN_ALL – ends when all sub-behaviors finish
- WHEN_ANY – ends when one sub-behavior finishes

7. SequentialBehaviour – Step by step

Code Example

```java
SequentialBehaviour seq = new SequentialBehaviour();

seq.addSubBehaviour(new OneShotBehaviour() {
    public void action() {
        System.out.println("Step 1");
    }
});

seq.addSubBehaviour(new OneShotBehaviour() {
    public void action() {
        System.out.println("Step 2");
    }
});

addBehaviour(seq);
```

Great for workflows like: "ask", "wait", then "respond".

2.5 Management GUIs

RMA (Remote Monitoring Agent) lets you inspect running containers, agents, message flows, behaviors.

2.5.1 What is the RMA?

RMA = Remote Monitoring Agent
It is a GUI-based agent automatically launched with the main container, used to:

- Monitor running agents and containers
- View agent behaviors and their status
- Track messages being sent and received (ACL messages)
- Register/unregister agents to/from the DF (Directory Facilitator)
- Send management commands to agents (kill, move, suspend, etc.)

2.5.2 What Can You Do With RMA?

Feature	Description
View active **agents**	List of all agents running on the platform
Inspect **containers**	See how many agents run in each container
Monitor **behaviors**	View active and completed behaviors of each agent
Inspect **messages**	View incoming/outgoing ACL messages
Access **DF registry**	See which agents are offering which services
Use **AMS functions**	Create, move, or kill agents; change properties
Manual **agent interaction**	Send messages directly to other agents from the UI

2.5.3 How to Use RMA

When you run your JADE platform like this:

Code Example

```
java -cp jade.jar jade.Boot -gui
```

- The Main Container is started
- The RMA agent (with GUI) is launched automatically

2.5.4 RMA Main Panels

Agent Management Tab

- Shows all currently running agents
- Allows you to kill, suspend, resume, or move agents
- You can also create new agents

Container Panel

- Displays which agents are running on which containers
- Shows container names, platform addresses

Sniffer Tool

- Used to monitor ACL messages sent between agents in real time
- Helpful for debugging communication protocols
- You must add agents to the Sniffer to monitor their messages

DF Management

- Browse and manage service registrations in the Directory Facilitator
- Helps track which agents provide which services

3 What's an Agent in JADE?

In JADE, an Agent is just a Java class that:

- extends `jade.core.Agent`
- overrides key lifecycle methods
- uses behaviors to modularize its logic

Agents live in a container, can discover each other, and talk via ACL messages.

3.1 Agent Lifecycle

Code Example

```java
public class MyAgent extends Agent {
  @Override
  protected void setup() {
    // called once when agent starts
  }

  @Override
  protected void takeDown() {
    // called once before agent dies
  }
}
```

3.2 Behaviours

Behaviours encapsulate tasks. You attach them to agents:

Behaviour Type	Use Case	Key Methods
OneShotBehaviour	Run once, then done	action()
CyclicBehaviour	Loop forever processing messages/events	action(), done() always false
TickerBehaviour	Run periodically (e.g., every N ms)	onTick()
FSMBehaviour	Finite-state machine coordination	registerState(), registerTransition()
ParallelBehaviour	Run child behaviours in parallel	addSubBehaviour()

Attach a behaviour:

Code Example

```java
addBehaviour(new OneShotBehaviour() {
  public void action() {
    System.out.println("Hello from behaviour!");
  }
});
```

3.3 Messaging (ACLMessage)

Code Example

```java
// Sending
ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
msg.addReceiver(new AID("receiverAgent", AID.ISLOCALNAME));
msg.setContent("Do the thing");
send(msg);

// Receiving (non-blocking)
ACLMessage m = receive();
if (m != null) {
  String content = m.getContent();
  // ...
}

// Receiving (blocking)
ACLMessage m2 = blockingReceive(5000); // wait up to 5s
```

3.4 Service Registration & Discovery

Code Example

```java
// Register a "service" in the DF
DFAgentDescription dfd = new DFAgentDescription();
dfd.setName(getAID());
ServiceDescription sd = new ServiceDescription();
sd.setType("weather-info");
sd.setName("local-weather-service");
dfd.addServices(sd);
DFService.register(this, dfd);

// Find agents offering "weather-info"
DFAgentDescription template = new DFAgentDescription();
ServiceDescription tsd = new ServiceDescription();
tsd.setType("weather-info");
template.addServices(tsd);

DFAgentDescription[] results = DFService.search(this, template);
for (DFAgentDescription dfad : results) {
  AID provider = dfad.getName();
  // ...
}
```

4 Spinning Up Containers & Agents

4.1 Main Container (Bootstrapping)

Code Example

```
# On the command line, in your jade/lib folder:
java -cp jade.jar:jadeTools.jar jade.Boot \
     -gui              \  # start the RMA GUI
     -name Main        \  # name your main container
     -agents agent1:com.mycompany.MyAgent;agent2:com.mycompany.
     OtherAgent
```

4.2 Hosting Additional Containers (Remote)

Code Example

```
java -cp jade.jar:jadeTools.jar jade.Boot \
     -container \
     -host 192.168.1.2  \  # point to main-container address
     -agents remoteAgent:com.mycompany.RemoteAgent
```

You can also launch programmatically:

Code Example

```
Profile p = new ProfileImpl();
p.setParameter(Profile.MAIN_HOST, "localhost");
AgentContainer container = Runtime.instance().createAgentContainer(p)
;
AgentController ag = container.createNewAgent("agentName", MyAgent.
class.getName(), null);
ag.start();
```

Option	**Usage**
`-gui`	Enable RMA monitoring GUI
`-name <String>`	Name the container
`-port <Int>`	Set incoming port (default 1099)
`-agents list`	Comma-separated `name:full.class.Name`
`-container`	Launch a non-main (satellite) container

5 Mobility & Migration

Feature	**Snippet**
Make agent mobile	Extend `jade.core.Agent` + implement `beforeMove()` / `afterMove()`
Migrate	`doMove(new ContainerID("Main", "localhost"));`
Clone agent	`doClone(new ContainerID("Main","localhost"), "cloneName");`

6 Debugging & Monitoring

RMA GUI: See containers, agents, message traffic.

Sniffer Agent:

Code Example

```
java -cp jade.jar:jadeTools.jar jade.Boot \
     -agents sniffer:jade.tools.sniffer.Sniffer
```

Tracks ACL exchanges between two or more agents.

AMS Messages: Enable: `p.setParameter(Profile.GUI, "true");`

7 Common Utilities & Tips

Utility	**How to Use**
Logging	`ACLMessage m = receive(); logMyAgent.log(Level.INFO, m.toString());`
JSON Content	`msg.setContentObject(mySerializableObject);` (implements `Serializable`)
Timeout Patterns	Use `ReceiveTimeoutBehaviour` for built-in timeouts
ACL Templates	`MessageTemplate.MatchPerformative(ACLMessage.REQUEST)` to filter in `receive(te`

8 Quick Reference: Core Classes

Class	Role
`jade.core.Agent`	Base agent class
`jade.core.behaviours.*`	All behaviour types
`jade.lang.acl.ACLMessage`	Message container
`jade.core.AID`	Agent Identifier
`jade.domain.DFService`	Yellow Pages (Directory Facilitator)
`jade.domain.FIPAAgentManagement.*`	AMS and DF description structures
`jade.wrapper.AgentContainer`	Programmatic container control
`jade.wrapper.AgentController`	Start/stop agents programmatically

9 Documentation & Resources

9.1 Official Docs & Tutorials

JADE User's Guide The canonical manual covering every class, behavior, container option, mobility feature, content language, etc.

- `http://jade.tilab.com/doc/JADEUserGuide.pdf`

FIPA Specifications Since JADE implements FIPA, skim these to understand the messaging standards:

- `https://jade.tilab.com/documentation/tutorials-guides/`

JADE Examples Bundle In your jade/examples folder you'll find dozens of ready-to-run demos—from simple "ping-pong" agents to full contract-net negotiations.

9.2 Books & Articles

"Developing Multi-Agent Systems with JADE" by Bellifemine, Caire & Greenwood The gold-standard deep dive, packed with patterns, tips, and real-world case studies.

"JADE Essentials" by Fabio Luigi Bellifemine A shorter, hands-on guide—perfect for quickly getting up and running.

Research Papers & Use Cases Google Scholar for "JADE multi-agent" will turn up dozens of industrial and academic projects to inspire your architecture.

10 Build, Integrate & Deploy

10.1 Prerequisites

- Java JDK 8 or higher installed and JAVA_HOME configured.
- Maven (optional) or manual jade.jar download.
- Basic familiarity with Java programming.

10.2 Installing JADE

10.2.1 Download Manually

- Visit JADE official site.
- Download the latest stable jade-bin-x.y.z.zip and extract.
- Locate lib/jade.jar and optional examples in jadeExamples.

10.2.2 Using Maven

Add to your pom.xml:

Code Example

```
<dependency>
  <groupId>com.tilab</groupId>
  <artifactId>jade</artifactId>
  <version>4.5.0</version>
</dependency>
```

10.3 Project Setup

10.3.1 Directory Structure

Code Example

```
my-jade-project/
          lib/                     optional: place jade.jar here
          src/
                main/java/
                    agents/
                        HelloAgent.java
                        ...
          pom.xml                  if using Maven
```

10.3.2 IDE Configuration

- In Eclipse/IntelliJ: add jade.jar to project classpath or import Maven.
- Enable annotation processing if using SLCodec annotations.

10.4 Your First Agent

Create a simple HelloAgent:

Code Example

```
package agents;

import jade.core.Agent;

public class HelloAgent extends Agent {
  @Override
  protected void setup() {
    System.out.println("Hello! Agent " + getLocalName() + " is ready.
    ");
    // One-shot behaviour
    addBehaviour(new jade.core.behaviours.OneShotBehaviour(this) {
      @Override
      public void action() {
        System.out.println("Executing one-shot behaviour");
      }
    });
  }
}
```

10.5 Running the Platform

10.5.1 Command-Line Bootstrap

Code Example

```
# From project directory, ensure jade.jar on classpath
java -cp lib/jade.jar:bin jade.Boot -gui
```

- `-gui` launches RMA (Agent Management GUI).
- Use `-agents` to auto-launch agents:

Code Example

```
java -cp lib/jade.jar:bin jade.Boot -gui -agents "a1:agents.
HelloAgent;a2:agents.HelloAgent"
```

10.5.2 Key Options

- `-container host:port`: connect to remote main container.
- `-local-port`: change listening port.

11 Advanced Topics & Patterns

11.1 Design Patterns in MAS

- **Contract Net** (CFP / PROPOSE / ACCEPT_PROPOSAL)
- **Broker** (middleman service discovery + load balancing)
- **Observer** (publish-subscribe via DF notifications)
- **Auction** (bid collection with FIPA-Contract-Net protocol)

11.2 Ontology & Content Languages

- JADE supports SL (Semantic Language) out of the box; you can plug in XML, RDF or even JSON content languages.
- Write your own Ontology classes to map Java objects onto structured messages.

11.3 Agent Mobility & Cloning

- Understand the pitfalls of serializing resources (open sockets, DB connections) when moving agents.
- Use PlatformID and ContainerID wisely to migrate across JVMs.

11.4 Security & ACL Filtering

- Plug in SSL/TLS transport for ACL messages.
- Use MessageTemplate to filter unwanted performatives or malformed content.

11.5 Performance Tuning

- Don't jam heavy CPU or blocking I/O into CyclicBehaviour; delegate to separate threads or use BehaviourPool.
- Monitor memory with the RMA or JMX; large numbers of agents can exhaust the GC.

12 Common Pitfalls

JADE Info

- **Blocking in Behaviours:** Calling long tasks inside action() without yielding will stall the agent scheduler.
- **Thread Safety:** Behaviours run in a single thread per agent, but any shared static data must be synchronized.
- **Mis-used Performatives:** Using INFORM when you meant REQUEST breaks FIPA protocols—other agents will ignore your message.
- **DF "Clutter":** Always deregister() in takeDown() or your DF will fill up with ghost entries.

13 Community & Support

13.1 JADE Mailing List

jade-user@lists.sourceforge.net (archive available on SourceForge)

13.2 StackOverflow "jade-framework" Tag

Browse solved questions or ask your own—lots of experts monitor it.

13.3 GitHub Forks & Extensions

Search "JADE" on GitHub to find integrations with Spring, Camel, MQTT, ROS, etc.

JADE Info

Keep these at your fingertips as you architect real multi-agent systems. With the official guides, the example code, the design patterns, and the community behind you, you'll have everything—or know exactly where to look—to build robust, scalable JADE applications. Enjoy!