Toolchain - Introduction
Agent.Workbench + Eclipse + JADE

Agent Summer School - 15.07.2022

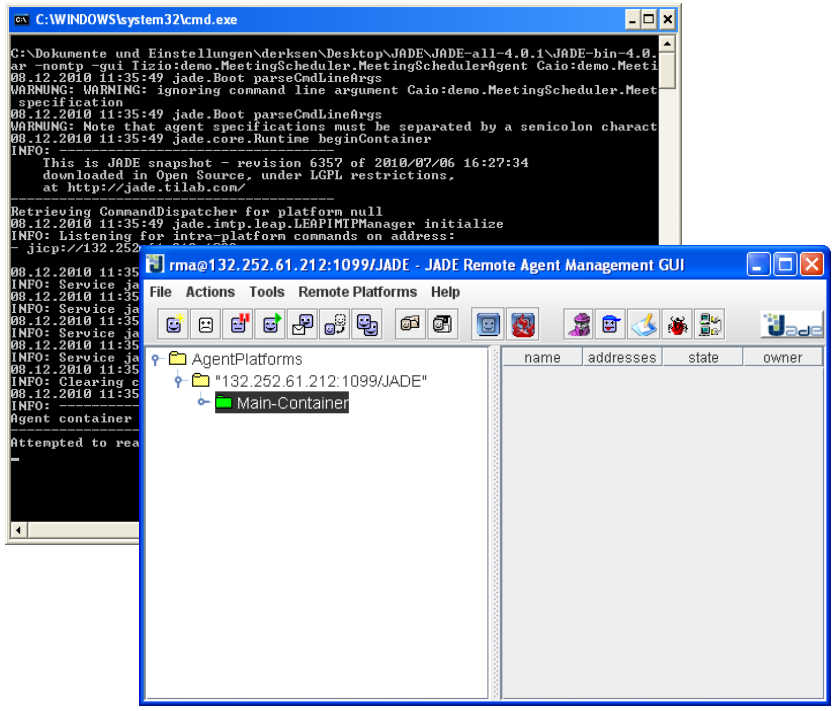
Christian Derksen ▪ SOFTEC ▪ University of Duisburg-Essen

Outline

- **Agent.Workbench (Runtime Environment + Library)**
  - Why did we develop Agent.Workbench? (AWB)? What it is?
  - What can you do with AWB? - Some example agent systems
- **Eclipse (IDE - Integrated Development Environment for Java)**
  - Some basics for the usage (Perspectives, Projects & Target Platform)
  - Create your first Plug-in Project
- **JADE (Library + Agent Runtime Environment)**
  - Platform structure, Agent Life Cycle
  - Base Classes, Behaviours
  - Messaging, Interaction Protocols

Agent.Workbench

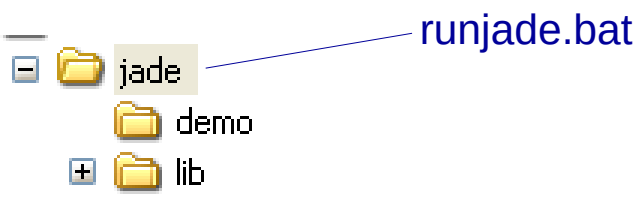
Why (Part 1): Starting JADE-Agents (the original approach)



Start using command line (an example):

java -classpath .;.\lib\jade.jar; .\lib\commons-codec\commons-codec-1.3.jar; jade.Boot -gui -services mas.service.SimulationService;jade.core.event.NotificationService;jade.core.mobility.AgentMobilityService;

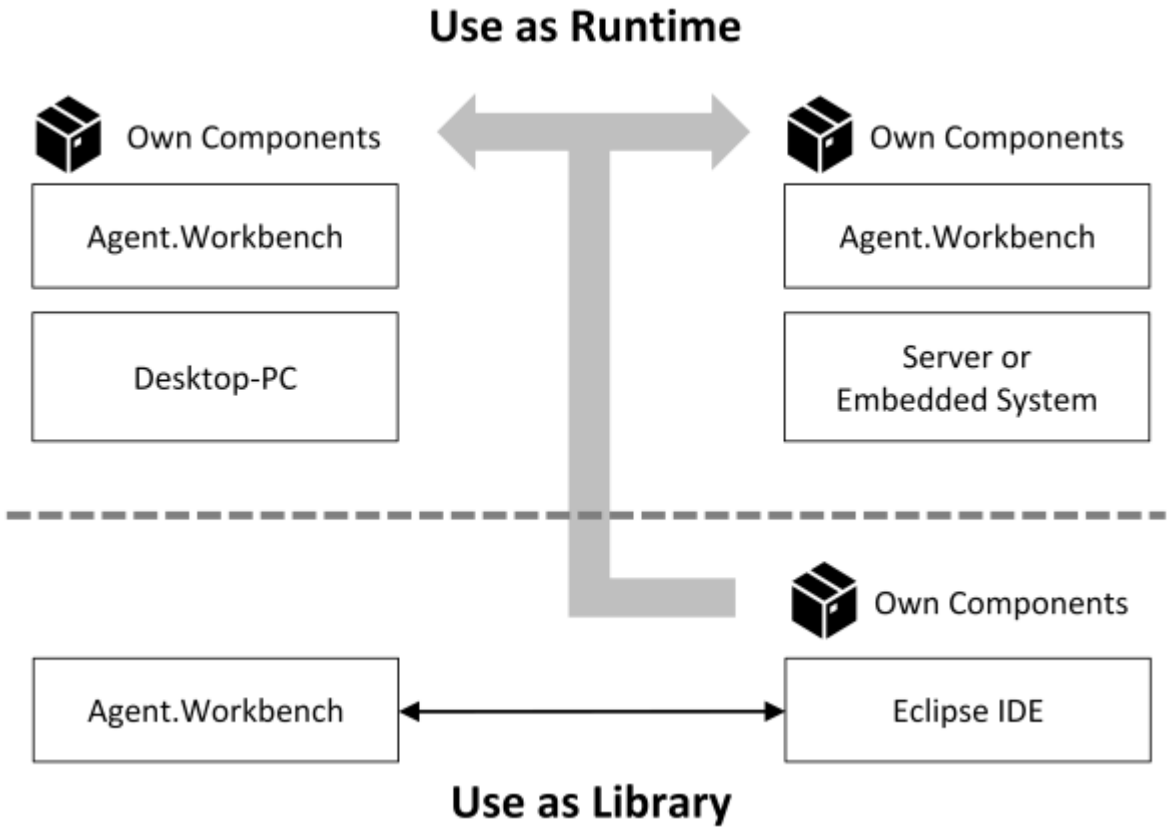
(„administratorsguide.pdf “, Page 4 ff.



Why (Part 2): Some things were missing

- **Development**
  - Beside the JADE library, what types of libraries are required (e.g. for database connections and other)?
  - How to have an extendible base for nearly any agent application?
- **Configuration**
  - How to easily configure the JADE agent platform?
  - How to visually configure agents?
- **Execution**
  - How to observe the agents and systems behaviour (especially on distributed systems)?
- **Shutdown**
  - How to proper shutdown an agent system (especially on distributed systems)?

What it is


Use as Runtime
Own Components
Agent.Workbench
Desktop-PC
Own Components
Agent.Workbench
Server or
Embedded System
Own Components
Agent.Workbench
Eclipse IDE
Use as Library

Structural Overview to Agent.Workbench

UNIVERSITÄT DUISBURG ESSEN

Offen im Denken

Handling Agent Projects with different setups

Providing a shared environment for agents (e.g., in simulations)

Connect to Databases

Use & Provide REST-Services

Core	Environment Model	Databases	REST
JADE	JUNG	Hibernate	Jetty
OSHI	OSM	MariaDB	Jersey
JFreeChart		PostgresSQL	Jackson
		MariaDB	Swagger

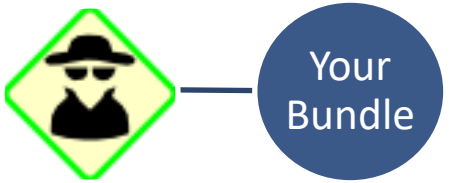
Customize the UI or extend AWB by writing:
- **OSGI-bundles with own agents**
- **an AWB - Plugin**
- **implement OSGI-Services**
- **connect to databases**
- **provide own REST-Services**
- **…**

Agent.Workbench
OSGI – Framework (Eclipse RCP)
Java
OS (Linux / Windows / Mac OS)

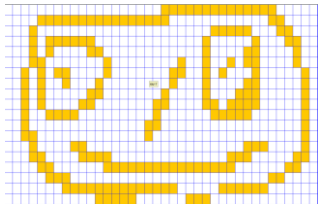
Have look on some example agent systems

- Ping-Pong: Observe the Messaging of two agents

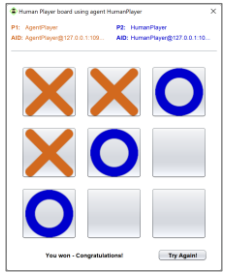
- Plugin-Example: Extend the AWB - UI



- Game of Life: A Cellular Automata Game



- Tic Tac Toe: The challenge of the day!



- Peak - Example: An electrical distribution grid with Energy-Agents

Some Links

- **Agent.Workbench on GitHub:**
  https://github.com/EnFlexIT/AgentWorkbench
- **Agent.Workbench on GitBook:**
  https://enflexit.gitbook.io/agent-workbench/
- **JADE: Java Agent DEvelopment Framework**
  https://jade.tilab.com

The Eclipse IDE for Enterprise Java and Web Developers

Have look to Eclipse

Some Links

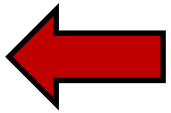
UNIVERSITÄT DUISBURG ESSEN

Offen im Denken

- **Define the Target Platform for development “… based on a AWB Installation”:**
  https://enflexit.gitbook.io/agent-workbench/development/basic-steps/define-a-target-platform/target-platform-based-on-awb-installation
- **Create an agent project:**
  https://enflexit.gitbook.io/agent-workbench/development/basic-steps/create-a-project-plugin
- **Starting AWB from Eclipse**
  https://enflexit.gitbook.io/agent-workbench/development/basic-steps/starting-awb-from-eclipse


UNIVERSITÄT
DUISBURG
ESSEN
Offen im Denken
JADE
Java Agent DEvelopment Framework

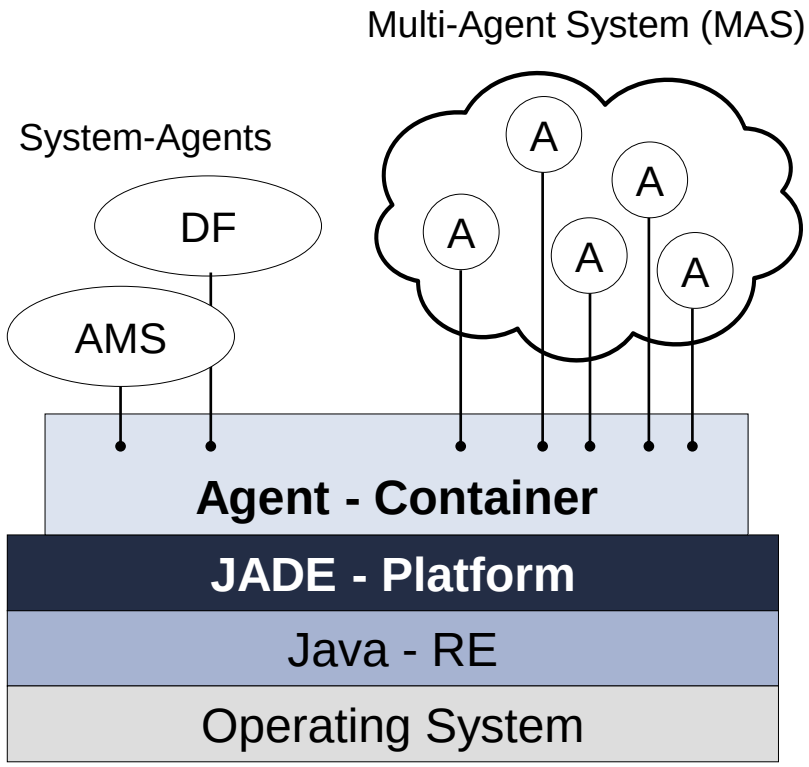
Why (Part 2): Some things were missing

- **Development**
  - Beside the JADE library, what types of libraries are required (e.g. for database connections and other)?
  - How to have an extendible base for nearly any agent application?
- **Configuration**
  - How to easily configure the JADE agent platform?
  - How to visually configure agents?
- **Execution** 
  - How to observe the agents and systems behaviour (especially on distributed systems)?
- **Shutdown**
  - How to proper shutdown an agent system (especially on distributed systems)?

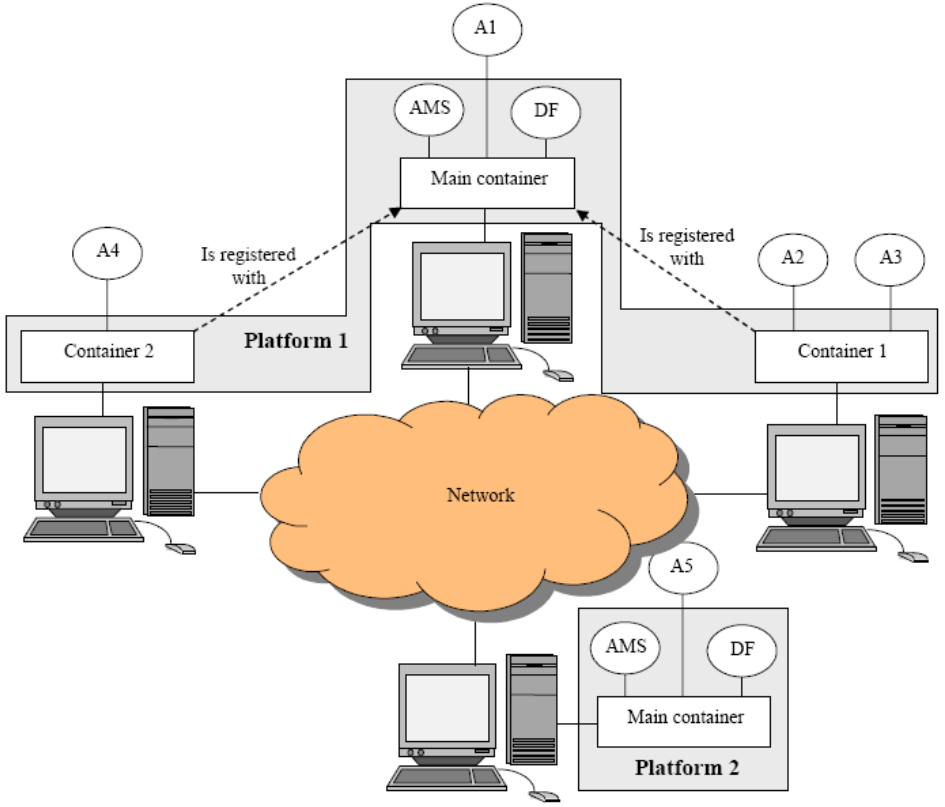
The JADE – Platform Execution

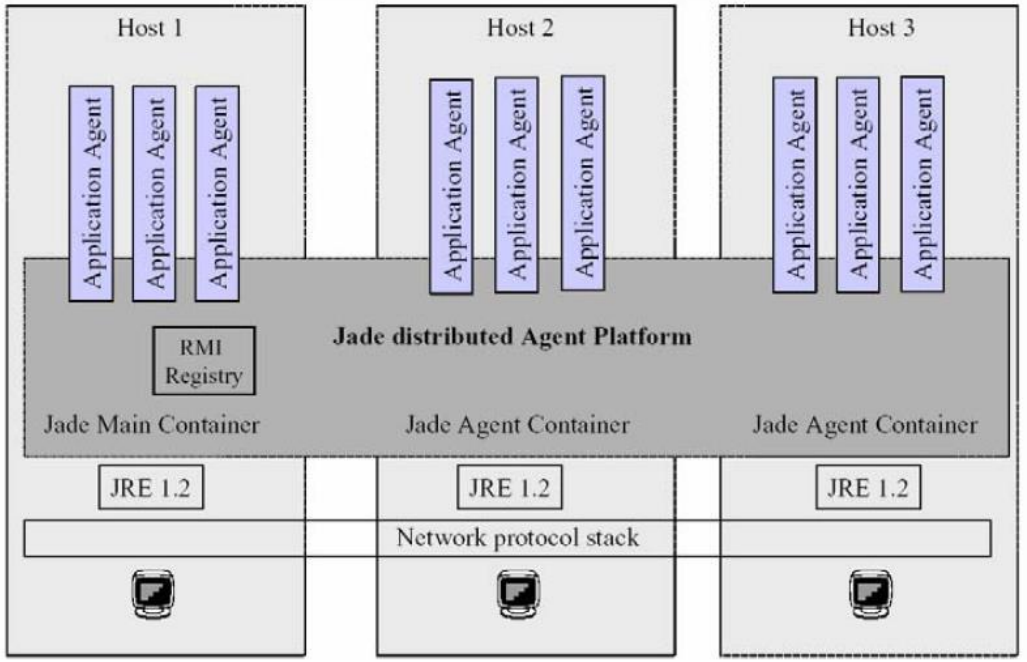
UNIVERSITÄT DUISBURG ESSEN
Offen im Denken

Platform and Container



Agent Distribution



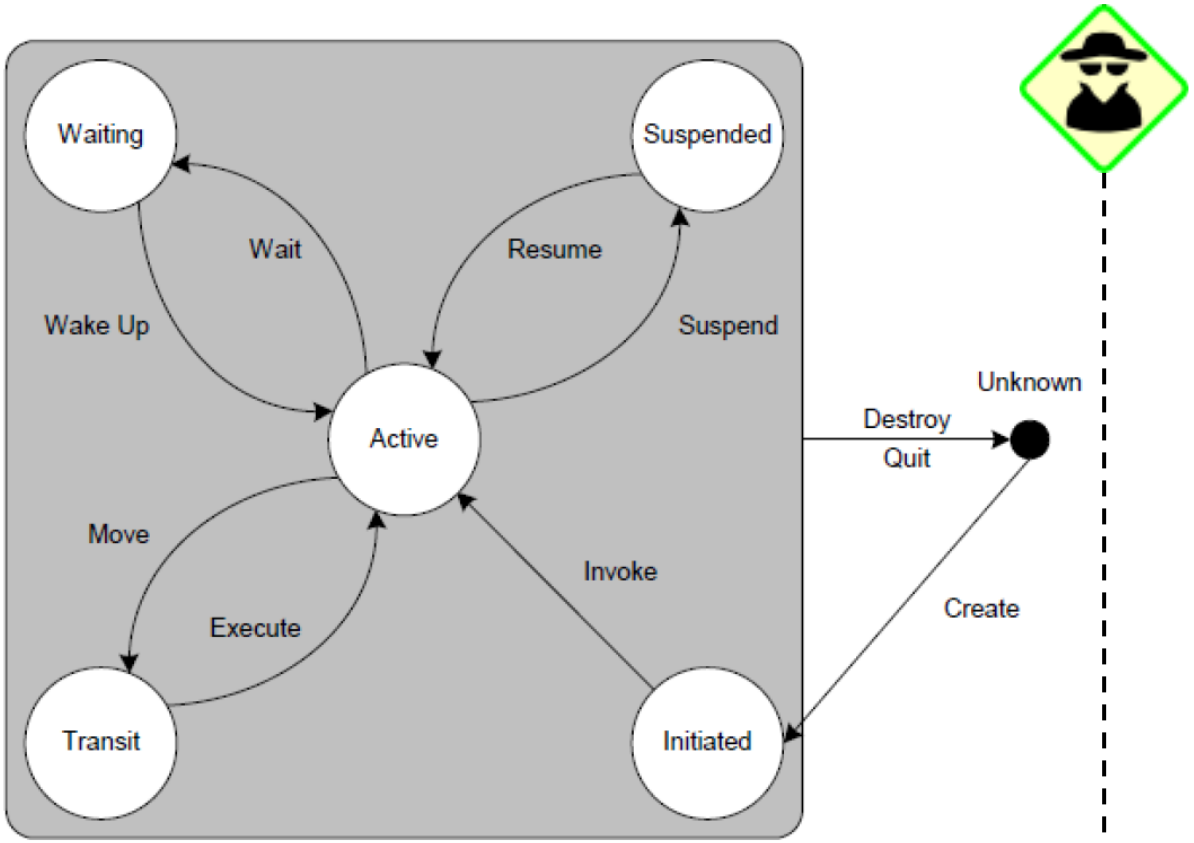


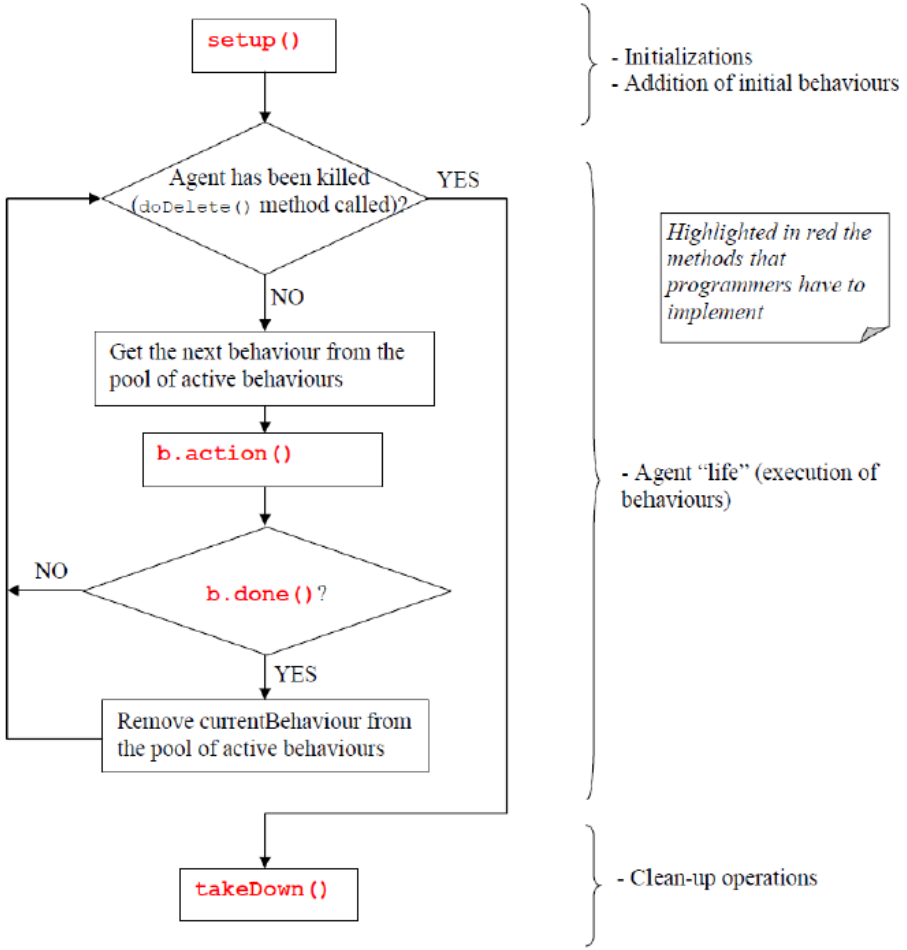
JADE - System-Agenten

UNIVERSITÄT DUISBURG ESSEN
Offen im Denken

- **RMA (Remote Management Agent):** a graphical user interface for the platform management. Basically the RMA can be used to start further JADE tooling agents (described in the following).
- **Introspector:** an agent that allows to monitor an agents state (e.g., its life cycle state) with respect to currently available or executed behaviours
- **Sniffer:** represents an agent that allows to monitor message sending between agents. This will be presented similar to an UML sequence diagram. Very useful to observe communication between multiple agents!
- **Dummy Agent:** is a Debugging- and Test-Tool for ACL-Messages. Then usage of this agent enables to create and send ACL-Messages to specific agents.
- **Directory Facilitator (DF):** the DF and its graphical representation enables to monitor the “Yellow Pages Service that can be started as an additional JADE-Service
- **LogManager**: Enables to configure the log level of the internal Logging-System

An Agents Life Cycle



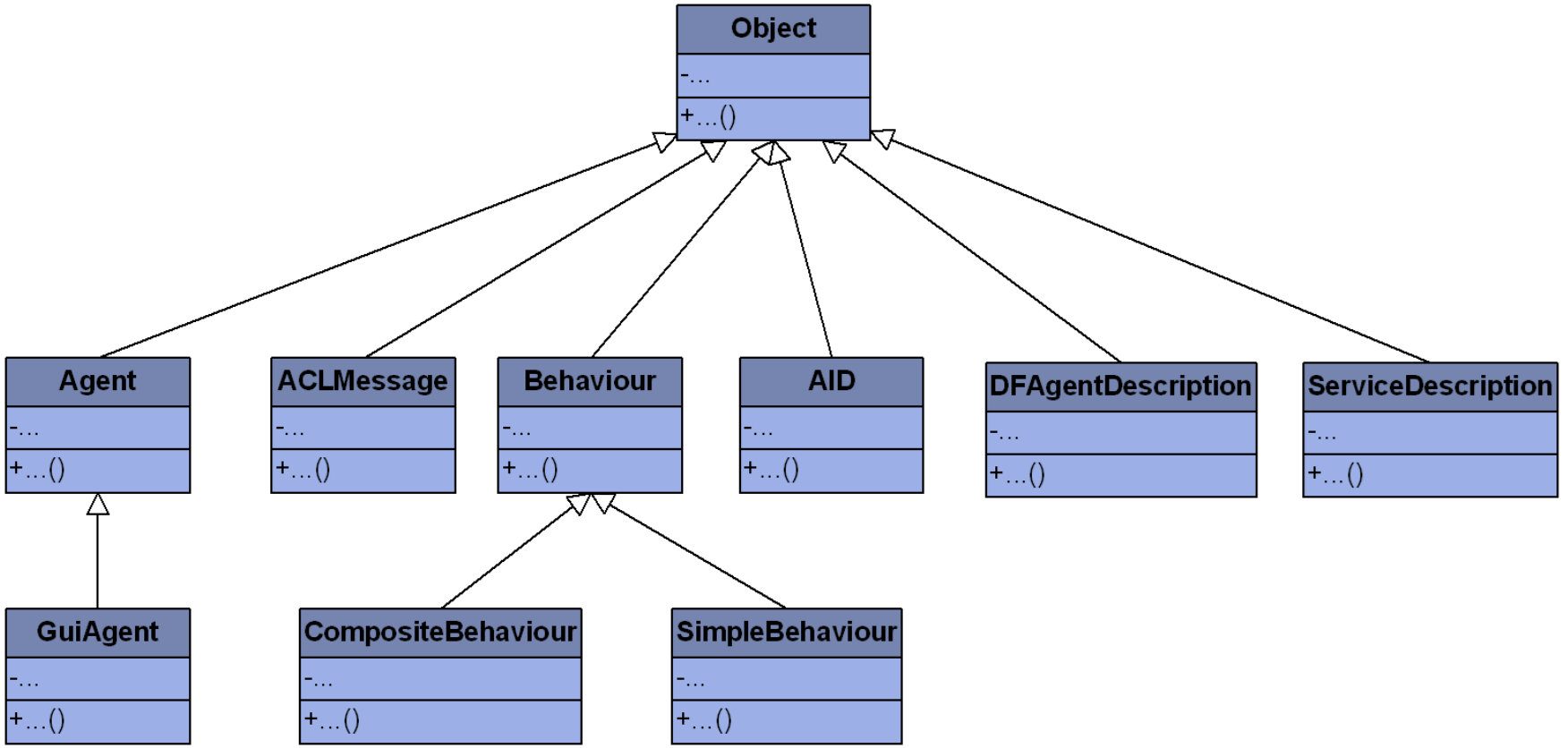


The beginning of an agent development

Creating a JADE agent is as simple as defining a class extending the `jade.core.Agent` class and implementing the `setup()` method as shown in the code below.

```
import jade.core.Agent;

public class BookBuyerAgent extends Agent {
  protected void setup() {
    // Printout a welcome message
    System.out.println("Hallo! Buyer-agent "+getAID().getName()+" is ready.");
  }
}
```

Some important JADE classes



A simple Agent example


```
import jade.core.Agent; import jade.core.behaviours.*;

public class Simple0 extends Agent {
    protected void setup() {
        addBehaviour( new B1( this ) );
    }
}

private class B1 extends SimpleBehaviour {
    private boolean finished = false;

    public B1(Agent agent) {
        super(agent);
    }

    public void action() {
        System.out.println( "Hello World! My name is "+ myAgent.getLocalName());
    }
    public boolean done() {
        return finished;
    }

} //End class B1
```

Behaviours and their execution

- One agent is a single Java-Thread in which the added behaviours are executed sequentially
- By adding a behaviour to an agent, the behaviour is placed in the agents internal behaviour queue.
- The agents internal scheduler takes care of the execution of that behaviours

- Side Note: It is possible to add / execute another Java-Thread for an agent by using so-called ***ThreadedBehaviour***. Consequently, an agent can also act as a manager of several parallel control processes

Behaviour Overview I

Primitive Behaviours

- **SimpleBehaviour:** an basic class that you can extend in various ways and which often turns out to be the best solution when other promising Behaviours are found to have some hidden quirks
- **CyclicBehaviour:** This behaviour stays active as long as its agent is alive and will be called repeatedly after every event. Quite useful to handle message reception.
  - **TickerBehaviour**: a cyclic behaviour which periodically executes some user-defined piece of code
- **OneShotBehaviour:** This behaviour executes ONCE and dies....
  Not really that useful since the *one* shot may be triggered at the wrong time.
  - **WakerBehaviour**: which executes some user code once at a specificized time
  - **ReceiverBehaviour**: which triggers when a given type of message is received (or a timeout expires).

Behaviour Overview II

Primitive Behaviours

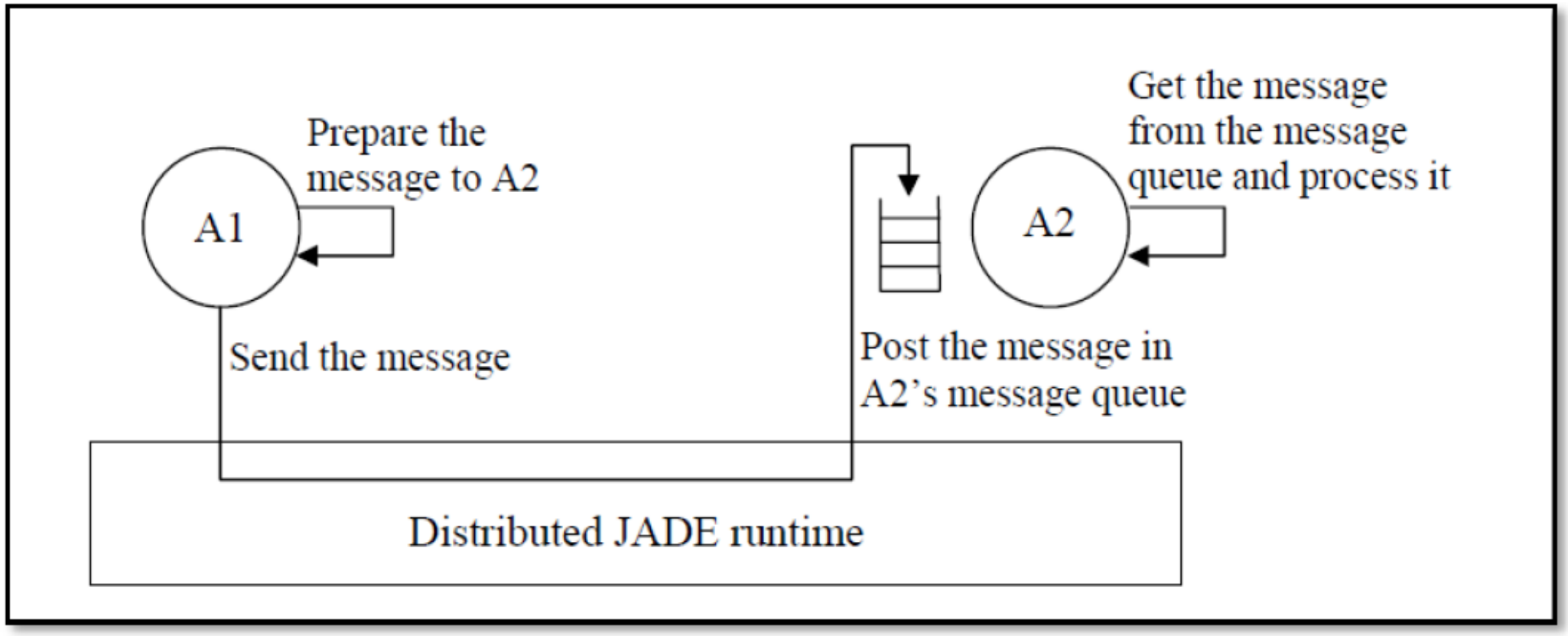
- **Note:** *TickerBehaviour*, *WakerBehaviour* and *ReceiverBehaviour* are conceptually subclasses of Cyclic and OneShot classes, but they are implemented as extensions of *SimpleBehaviour* and *Behaviour*

Composite Behaviours

- **ParallelBehaviour:** controls a set of children behaviours that execute in parallel. The important thing is the termination condition: we can specify that the group terminates when ALL children are done, N children are done or ANY child is done.

- **SequentialBehaviour:** this behaviour executes its children behaviours one after the other and terminates when the last child has ended.

Agent Communication



Send and receive Messages

Send a simple Message

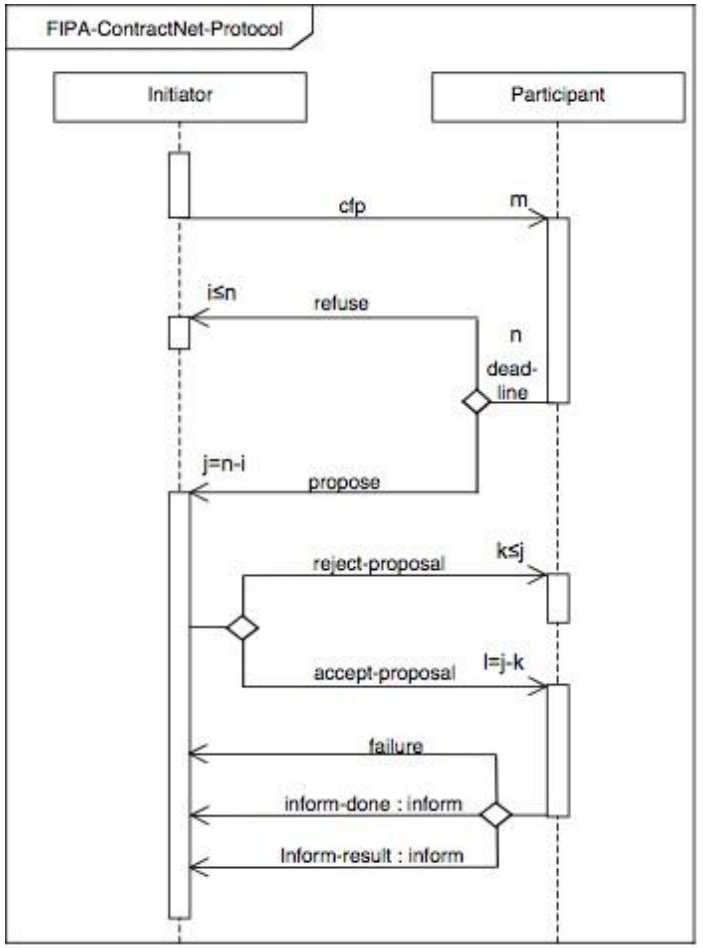
```
// --- Set receiver ------------------------------
AID receiver = new AID("pong", AID.ISLOCALNAME);

// --- Create simple message ---------------------
ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
msg.setSender(myAgent.getAID());
msg.addReceiver(receiver);
msg.setContent("ping");
myAgent.send(msg);
```

see AWB - Ping Pong example on Github!

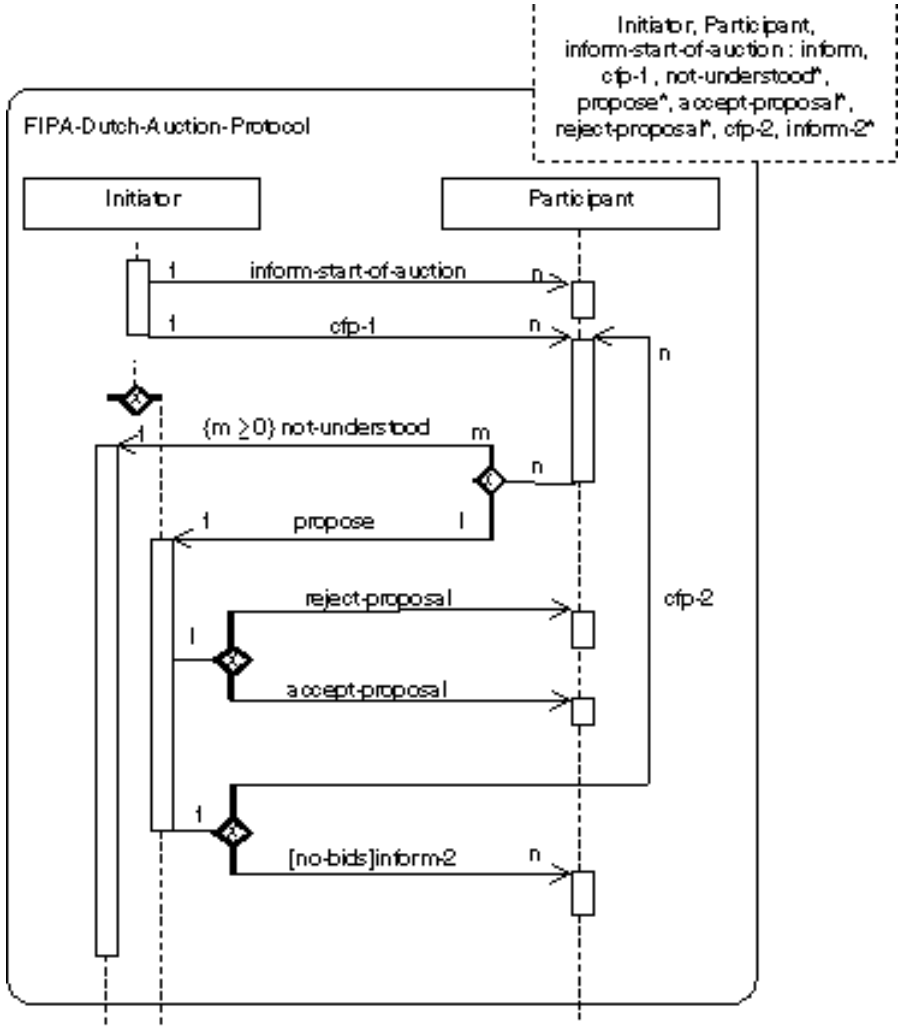
Receive a Message

```
class ReceiveBehaviour extends CyclicBehaviour {

  public void action() {

    ACLMessage msg = myAgent.receive();
    if (msg!=null) {
      // --- work on the delivered message -------
-
      => ...

    } else {
      // --- wait for the next incoming message --
-
      block();
    }
  }
}
```

Interaction Protocols available with JADE

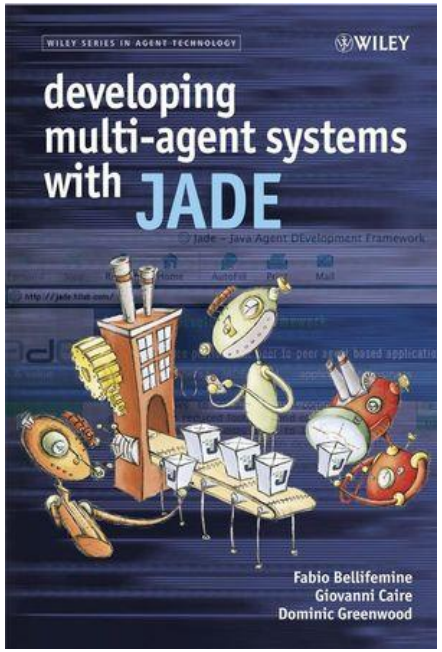


For further example protocols, see (old) FIPA - Site http://www.fipa.org



Some Links

- **JADE – Book**



- **JADE - Site:**
  https://jade.tilab.com/
- **FIPA - Site Create an agent project:**
  http://www.fipa.org
- **JADE - Repository on GitHub (by EnFlex.IT):**
  https://github.com/EnFlexIT/JADE

Thanks ! - Questions?

Dipl.-Ing. Christian Derksen
SOFTEC | paluno
Universität Duisburg-Essen
Universitätsstraße 9
45141 Essen

Web: https://www.softec.wiwi.uni-due.de
Mail: christian.derksen@uni-due.de
Tel.: + 49 201 183 4586
Fax: + 49 201 183 4460