Multi-Agent Systems
Software Platforms/Frameworks

- JADE
- Repast
- SAJaS
- Jason
- Jadex
http://jade.tilab.com/
JADE

- An open-source framework for developing multi-agent systems
- FIPA-compliant (http://www.fipa.org/)

- Written in JAVA, consists of an API with several packages
JADE Architecture

Agent Management System

Directory Facilitator

Agents
JADE Main Features

• FIPA-compliant
  – Agent Platform
  – Agent Communication Language (ACL)
  – Interaction Protocols

• Distributed MAS
  – Agents execute within *containers* (JVMs)

• Infrastructure
  – Agent Management System (AMS)
  – Directory Facilitator (DF)
  – Message Transport System (MTS)
JADE Agents

• 1 Agent = 1 Thread

• Agent *mobility* and cloning
  – Agents can migrate throughout containers

• Agents have *behaviours*
  – Tasks, executed concurrently

• Behaviour scheduling
  – Not preemptive, but “cooperative”
  – Behaviours are executed in a “sequentially parallel” fashion
JADE Agent Execution

setup()

Agent has been killed (doDelete() method called)?

YES

NO

Get the next behaviour from the pool of active behaviours

b.action()

NO

b.done()?

YES

Remove current behaviour from the pool of active behaviours

takeDown()

- Initializations
- Addition of initial behaviours

Highlighted in red the methods that programmers have to implement

- Agent “life” (execution of behaviours)

- Clean-up operations
JADE Tools

- Remote Monitoring Agent (JADE’s “GUI”)
- Dummy Agent
- Sniffer Agent
- Directory Facilitator GUI
- Introspector Agent
http://repast.sourceforge.net/
http://repast.sourceforge.net/repast_3/
JADE vs. Repast

- **JADE** is meant for building agent systems, but has no simulation infrastructure
- **Repast** is meant for agent-based simulations and not for developing multi-agent systems

- **JADE** is not a simulation framework: it has no scheduler, nor a notion of a "clock"
- **Repast** is not for building MAS, and is not FIPA compliant

- The agents within **Repast** are not the same kind of agents within **JADE**: both systems are designed for two entirely different purposes
# JADE vs. Repast

<table>
<thead>
<tr>
<th>Feature</th>
<th>JADE</th>
<th>Repast</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distributed</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Simulation Tools</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Scalability</td>
<td>Limited</td>
<td>High</td>
</tr>
<tr>
<td>Open Source</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Agent Execution</td>
<td>Behaviours</td>
<td>Scheduler</td>
</tr>
<tr>
<td></td>
<td>Multi-thread</td>
<td>Single-thread</td>
</tr>
<tr>
<td></td>
<td>Event-driven</td>
<td>Tick-driven</td>
</tr>
<tr>
<td></td>
<td>Asynchronous</td>
<td>Synchronous</td>
</tr>
<tr>
<td>Interaction</td>
<td>FIPA ACL</td>
<td>Method calls</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Shared resources</td>
</tr>
<tr>
<td>Ontologies</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
Repast

- Agent-based modeling and simulation
  - An agent-based model (ABM) aims at simulating the actions and interactions of autonomous agents with a view to assessing their effects on the system as a whole

- Recursive Porous Agent Simulation Toolkit (REPASt)
  - Open-source framework for creating agent-based simulations

- Repast 3
  - Three flavors: RepastJ, RepastPy, Repast.Net

- Repast Simphony
Repast Features

- Data collection and output
- Displaying agent interaction
Repast Spaces

• Purposes:
  – Contain a collection of agents (spatial agents)
  – Define the spatial relationships of agents relative to each other

• Cellular
  – Boundaries: Grid, Torus
  – Shapes: rectangular, hexagonal
  – Neighborhood: Von Neumann, Moore, rings (hexagonal)

• Network

• GIS
Repast Simulations

• Discrete event simulator
  – Unit of time: *tick*
  – Events are scheduled for specific ticks

• Repast simulation
  – Collection of *agents*
  – *Model* that sets up and controls the execution of agent actions according to a schedule

• Schedule
  – Execution of agent behaviors (actions)
  – Model actions, e.g. display updates, data recording, snapshots, ...
Repast GUI

- Repast 3
- Repast Simphony
SAJaS
Simple API for JADE-based Simulations

SAJaS: why?

• **JADE:**
  – Multi-agent systems development
  – Not suited for multi-agent based simulation (MABS): scalability

• **Repast:**
  – Agent-based simulation
  – Lack support for agent programming and multi-agent features (communication, infrastructure, ...)

• **However:**
  – Need to *simulate while developing* a full-featured MAS, for testing purposes
### SAJaS Features

<table>
<thead>
<tr>
<th>Feature</th>
<th>JADE</th>
<th>Repast</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distributed</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Simulation Tools</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Scalability</td>
<td>Limited</td>
<td>High</td>
</tr>
<tr>
<td>Open Source</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Agent Execution</td>
<td>Behaviours</td>
<td>Scheduler</td>
</tr>
<tr>
<td></td>
<td>Multi-thread</td>
<td>Single-thread</td>
</tr>
<tr>
<td></td>
<td>Event-driven</td>
<td>Tick-driven</td>
</tr>
<tr>
<td></td>
<td>Asynchronous</td>
<td>Synchronous</td>
</tr>
<tr>
<td>Interaction</td>
<td>FIPA ACL</td>
<td>Method calls</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Shared resources</td>
</tr>
<tr>
<td>Ontologies</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

From the simulation scheduler point of view
Usage

- **JADE Developer**
  - Create JADE MAS
  - Convert to JS
  - SAJaS simulation
  - Run tests/simulations
  - Make enhancements

- **JADE MAS**
  - Convert to SJ
  - Create simulation with SAJaS
  - Run tests/simulations
  - Make enhancements

- **Repast Developer**
  - Convert to SJ
  - Create simulation with SAJaS
  - Run tests/simulations
  - Make enhancements
SAJaS Overview
Benchmark Scenario

- **Service Consumer/Provider**
  - $5 \times n$ providers; $2 \times n$ consumers (AllProv, ProvSel)
  - DF, behaviours, FIPA-protocols, ACL, ontologies

Main Advantages

• MABS programmer
  – has a rich set of multi-agent programming features offered by JADE
  – may explore simulation-related features offered by the simulation infrastructure (e.g. Repast)

• Same implementation can be used both for simulation and deployment purposes
  – checkout MASSim2Dev

• Simulation performance gains in certain scenarios: high communication-to-computation ratio
Jason: interpreter for AgentSpeak

• Developing cognitive agents with the BDI (Beliefs-Desires-Intentions) architecture
  – Beliefs are information the agent has about the world
  – Desires are all the possible states of affairs that the agent might like to accomplish
  – Intentions are the states of affairs that the agent has decided to work towards

• Practical reasoning (directed towards actions): deliberation and means-ends reasoning
  – Beliefs, goals, plans
  – AgentSpeak: a programming language for BDI agents guided by practical reasoning – based on logic programming
Jason/AgentSpeak Main Concepts

• **Beliefs**
  – Ako Prolog facts, which can be annotated
    ```prolog
    publisher(wiley).
    colour(box1,blue)[source(bob)].
    ```
  – Strong negation (no closed world assumption)
    ```prolog
    ~colour(box1,white)[source(john)].
    ```

• **Goals**
  – Achievement goals (!)
    ```prolog
    !write(book)
    ```
  – Test goals (?)
    ```prolog
    ?publisher(P)
    ```
Jason/AgentSpeak Main Concepts

**Plans**

\[
\text{trig\_event} : \text{context} \leftarrow \text{body}.
\]

- **Triggering events**: changes in beliefs and in goals
- **Context**: applicability of the plan
- **Body**: a course of action for addressing the triggering event
  - Actions
    - Environmental actions
    - Belief changes (mental notes)
    - Internal actions (\(\cdot\)) do not change the (physical) environment
      - Standard
      - User-defined
  - (Sub)goals

<table>
<thead>
<tr>
<th>Notation</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>+l</td>
<td>Belief addition</td>
</tr>
<tr>
<td>-l</td>
<td>Belief deletion</td>
</tr>
<tr>
<td>+l!l</td>
<td>Achievement-goal addition</td>
</tr>
<tr>
<td>-l!l</td>
<td>Achievement-goal deletion</td>
</tr>
<tr>
<td>+l?l</td>
<td>Test-goal addition</td>
</tr>
<tr>
<td>-l?l</td>
<td>Test-goal deletion</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>l</td>
<td>The agent believes (l) is true</td>
</tr>
<tr>
<td>\neg l</td>
<td>The agent believes (l) is false</td>
</tr>
<tr>
<td>not l</td>
<td>The agent does not believe (l) is true</td>
</tr>
<tr>
<td>not \neg l</td>
<td>The agent does not believe (l) is false</td>
</tr>
</tbody>
</table>


Jason Reasoning Cycle
MAS and Environment

- MAS configuration file:

\[
\text{MAS <name> } \{
\begin{align*}
\text{infrastructure: } & [\text{Centralised } | \text{ Jade}] \\
\text{environment: } & <\text{environment\_simulation\_class}> \\
\text{executionControl: } & <\text{execution\_control\_class}> \\
\text{agents: } & <\text{ag1}>; \ldots;
\end{align*}
\}
\]

- Simulated environment implementation:
  - Extend class \text{jason.environment.Environment}
Jason Tools

- IDE: jEdit plugin
- MAS Console
- Mind Inspector

- Infrastructure: Jade
  - Jade Management Agent
  - Jade Sniffer
http://www.activecomponents.org/
http://paginas.fe.up.pt/~eol/AIAD/jadex/doku.php
Jadex BDI

- Jadex BDI is an agent-oriented reasoning engine for writing rational agents

- Agents represent active components with individual reasoning capabilities

- Agents can exhibit reactive behavior (responding to external events) as well as pro-active behavior (motivated by the agents own goals)
Beliefs, Goals and Plans

• BDI
  – Beliefs capture informational attitudes, desires capture motivational attitudes, and intentions capture deliberative attitudes of agents

• Execution model for software agents: beliefs, goals, and plans
  – Agents have beliefs (Java objects stored in a belief base)
  – Goals represent motivations (e.g. states to be achieved) that influence an agent’s behavior.
  – To achieve its goals the agent executes plans
Jadex Agent Abstract Architecture
Jadex Active Components

• Extension of the Service Component Architecture (SCA)
  – Components as active entities – agents – acting as service providers and consumers

• Communication is done using service invocations
Asynchronous Programming

- Asynchronous programming is the basis of the Jadex active components programming model
  - Synchronous call
  - Asynchronous call
Agent applications consist not only of agents but also of an environment where the agents are situated in
Jadex Control Center

The mars world application descriptor

Can be used to launch the mars world example. It consists of three different kinds of agents:
- **Sentry agents** are responsible for examining ore locations.
- **Production agents** produce ore at available ore location.
- **Carry agents** are able to carry ore to the base.

Objective is to carry as much ore as possible to the home base in a predefined mission time.