http://jadex-agents.informatik.uni-hamburg.de/

AIAD, September 2012
Jadex BDI

- Jadex BDI is an agent-oriented reasoning engine for writing rational agents with XML and the Java programming language.

- 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* *motivational* attitudes, and *intentions* *deliberative* attitudes of agents.

• Execution model for software agents: beliefs, goals, and plans
  – Agents have beliefs, which can be any kind of Java object and are stored in a belief base.
  – Goals represent the concrete motivations (e.g. states to be achieved) that influence an agent's behavior.
  – To achieve its goals the agent executes plans, which are procedural recipes coded in Java.
Jadex Agent Abstract Architecture
The Beliefbase

• The beliefbase stores believed facts and is an access point for the data contained in the agent.

• **Belief** = *identifier + belief value* (a Java object)

• Beliefs can be stored as expressions and evaluated dynamically on demand

• Using beliefs:
  – OQL-like query language
  – conditions that trigger plans or goals when some beliefs change (resembling a rulebased programming style)
Goals

• Goals are concrete, momentary desires of an agent
  – An agent will more or less directly engage into suitable actions, until it considers the goal as being reached, unreachable, or not desired any more

• The structure of currently adopted goals is stored in the goalbase of an agent.
• The agent has a number of top-level goals, which serve as entry points in the goalbase.
• Goals in turn may have subgoals, forming a hierarchy or tree of goals.
Types of Goals

• **Perform** goal: *something should be done* but may not necessarily lead to any specific result
  – waste-pickup robot wondering around

• **Achieve** goal: abstract *target state* to be reached, without specifying how to achieve it
  – player agent that needs certain resources

• **Query** goal: a *need for information*
  – cleaner robot needs to know where the next waste bin is located

• **Maintain** goal: *state that should be kept* once it is achieved
  – keep a reactor temperature below a certain level
Plans

• The concrete actions an agent may carry out to reach its goals are described in plans.

• Plan = head + body
  – the head contains the conditions under which the plan may be executed (activation triggers)
    • specified in the agent definition file (ADF)
  – the body of the plan is a procedural recipe describing the actions to take
    • specified in Java

• At runtime, plans are instantiated to handle events and to achieve goals
  – the plan’s body may also dispatch subgoals and wait for events to occur
Agent Definition File (ADF)

- XML file containing all relevant properties of an agent (e.g. the beliefs, goals and plans)
- The ADF is a kind of a class description for agents: agents are instantiated from the ADF
ADF

- **<imports>**
  - classes and packages that can be used by expressions throughout the ADF
- **<capabilities>**
  - modularize agent functionality, for reuse
  - an encapsulated agent module composed of beliefs, goals, and plans
  - a capability is basically the same as an agent, but without its own reasoning process
- **<beliefs>, <goals>, <plans>**
  - core part of the agent specification
- **<events>**
  - events known by the agent
- **<expressions>**
  - specify expressions and conditions, which can be used as predefined queries from plans
- **<properties>**
  - custom settings such as debugging and logging options
- **<configurations>**
  - predefined configurations, such as initial beliefs, goals, and plans, as well as end goals and plans
Defining Beliefs

<belief> and <beliefset>

Dynamically evaluated beliefs

...<beliefs>
    <belief name="my_location" class="Location">
        <fact>new Location("Hamburg")</fact>
    </belief>
    <beliefset name="my_friends" class="String">
        <fact>"Alex"</fact>
        <fact>"Blandi"</fact>
        <fact>"Charlie"</fact>
    </beliefset>
    <beliefset name="my_opponents" class="String">
        <facts>Database.getOpponents()</facts>
    </beliefset>
...
</beliefs>

...
Accessing Beliefs from Plans

- **IBeliefbase** getBeliefbase()
  - **IBelief** getBelief()
    - Object getFact()
    - setFact(Object)
  - **IBeliefSet** getBeliefSet()
    - Object getFact()
    - Object[] getFacts()
    - addFact(Object)
    - addFacts(Object[])
    - removeFact(Object)
    - removeFacts()

```java
public void body
{
    ...
    IBelief hungry = getBeliefbase().getBelief("hungry");
    hungry.setFact(new Boolean(true));
    ...
    Food[] food = (Food[])getBeliefbase().getBeliefSet("food").getFacts();
    ...
}
```
Defining Goals

- **Common:**
  - parameters: in, out, inout
  - `<unique/>
  - `<creationcondition`
  - `<contextcondition`
  - `<dropcondition`
  - `<deliberation`

- **Achieve goal:** `<targetcondition>`

- **Maintain goal:** `<maintaincondition>`, `<targetcondition>`

```xml
<performgoal name="performlookforwaste" retry="true" exclude="never">
  <contextcondition language="jcl">
    $beliefbase.daytime
  </contextcondition>
</performgoal>

<achievegoal name="moveto">
  <parameter name="location" class="Location"/>
  <targetcondition>
    $beliefbase.my_location.isNear($goal.location)
  </targetcondition>
</achievegoal>

<maintaingoal name="maintainbatteryloaded">
  <deliberation>
    <inhibits ref="performlookforwaste" inhibit="when_in_process"/>
    <inhibits ref="achievecleanup" inhibit="when_in_process"/>
    <inhibits ref="performpatrol" inhibit="when_in_process"/>
  </deliberation>
  <maintaincondition language="jcl">
    $beliefbase.my_chargestate > 0.2
  </maintaincondition>
</maintaingoal>

<querygoal name="query_wastebin" exclude="never">
  <parameter name="result" class="Wastebin" evaluationmode="push" direction="out">
    <value variable="$wastebin">
      Wastebin $wastebin &amp;&amp; !$wastebin.isFull()
      &amp;&amp; !((Wastebin $wastebin2 &amp;&amp; !$wastebin2.isFull())
      &amp;&amp; $beliefbase.my_location.getDistance($wastebin2.getLocation())
      &gt; $beliefbase.my_location.getDistance($wastebin.getLocation()))
    </value>
  </parameter>
</querygoal>
```
Creating and Dispatching Goals

- Top-level goals
  - may be part of the agent’s initial state
  - may be later adopted at runtime
- Subgoals
  - can only be dispatched by a running plan

- `createGoal()`
- `dispatchSubgoal()`
- `dispatchSubgoalAndWait()`
- `dispatchTopLevelGoal()`
- `drop()`
Defining Plans

• The plan head defines the circumstances under which the plan body is instantiated and executed
• The plan body is declared using `<body>`
• `<trigger>`
  – goals, internal- or message events for which the plan is applicable: `<goal>`, `<goalfinished>`, `<internalevent>`, `<messageevent>`
  – `<factchanged>`, `<factadded>`, `<factremoved>`, ...

• Parameters: in, out, inout
• Plan applicability
  – `<precondition>`, `<contextcondition>`

```xml
<plans>
  <plan name="repair">
    <body impl="RepairPlan"/>
    <trigger>
      <condition>beliefbase.out_of_order</condition>
    </trigger>
    <contextcondition>beliefbase.repairable</contextcondition>
  </plan>
</plans>
<agent ...
  ...
  <plans>
    <plan name="ping">
      <body impl="PingPlan"/>
      <trigger>
        <messageevent ref="query_ping"/>
      </trigger>
    </plan>
  </plans>
  ...
  <events>
    <messageevent name="query_ping" type="fipa">
      ...
    </messageevent>
  </events>
  ...
</agent>
```
Plan Body in Java

- Standard plans inherit from `jadex.bdi.runtime.Plan` body()

- Atomic blocks

```java
public void body() {
    // Send request.
    ...

    // Wait for agree/refuse.
    IMessageEvent e1 = waitForMessageEvent(...);
    boolean agreed = ...;
    ...

    // Wait for inform/failure.
    if (agreed) {
        IMessageEvent e2 = waitForReply(...);
        boolean informed = ...;
        ...
        if (informed) {
            ...
        } else {
            ...
        }
    } else {
        ...
    }
}
```
Plan Success or Failure

```java
public class MyPlan extends Plan {
    public void body()
    {
        // Application code goes here.
        ...
    }

    public void passed()
    {
        // Clean-up code for plan success.
        ...
    }

    public void failed()
    {
        // Clean-up code for plan failure.
        ...
        System.out.println("Exception thrown.");
    }

    public void aborted()
    {
        // Clean-up code for an aborted plan.
        ...
        System.out.println("Goal achieved? "+isAbortedOnSuccess());
    }
}
```
Events

• Events are usually handled by plans
• Events are single points in time and therefore only support "in" parameters
• **Internal events:** IInternalEvent
  – occurrence inside the agent

```xml
<events>
  <internalevent name="gui_update">
    <parameter name="content" class="String"/>
  </internalevent>
</events>
```

• **Message events:** IMessageEvent
  – all message types an agent wants to send/receive are specified in the ADF
  – only incoming messages are handled by the event dispatching mechanism

```java
public void body()
{
  String update_info;
  ...
  // "gui_update" internal event type must be defined in the ADF
  IInternalEvent event = createInternalEvent("gui_update");
  // Setting the content parameter to the update info
  event.getParameter("content").setValue(update_info);
  dispatchInternalEvent(event);
  ...
}
Receiving/Sending Messages

```xml
<imports>
  <import>jadex.base.fipa.SFipa</import>
</imports>
...
<events>
  <!-- A query-ref message with content "ping" -->
  <messageevent name="query_ping" type="fipa" direction="receive">
    <parameter name="performative" class="String" direction="fixed">
      <value>SFipa.QUERY_REF</value>
    </parameter>
    <parameter name="content" class="String" direction="fixed">
      <value>"ping"</value>
    </parameter>
  </messageevent>

  <!-- An inform message where content contains the word "hello" -->
  <messageevent name="inform_hello" type="fipa" direction="receive">
    <parameter name="performative" class="String" direction="fixed">
      <value>SFipa.INFORM</value>
    </parameter>
    <match>((String)$content).indexOf("hello") != -1</match>
  </messageevent>
</events>

public void body()
{
    IMessageEvent me = createMessageEvent("query_ref");
    me.getParameterSet(SFipa.RECEIVERS).addValue(cid);
    //me.getParameter(SFipa.CONTENT).setValue("ping 2");
    sendMessage(me);
}
```
Starting the Jadex Platform

- java -jar jadex-platform-standalone-launch-2.1.jar