JADEX Framework

Integrated Master's in Informatics Engineering

Intelligent Agents

2017/2018

Synthetic Intelligence Lab

Ricardo Ramos

Filipe Gonçalves

Paulo Novais

Useful Links

- https://www.activecomponents.org/
- https://download.actoron.com/docs/releases/latest/jadex-mkdocs/
- https://download.actoron.com/docs/releases/jadex-3.0.0-RC80/jadex-mkdocs/getting-started/getting-started/#project-setup-without-mavengradle
- https://download.actoron.com/docs/releases/latest/jadex-mkdocs/platform/platform/#jadex-platform
- https://download.actoron.com/docs/releases/jadex-3.0.0-RC80/jadex-mkdocs/getting-started/getting-started/#using-intellij-idea
- https://download.actoron.com/docs/releases/jadex-3.0.43/jadex-mkdocs/tutorials/bdiv3/01%20Introduction/

Jadex

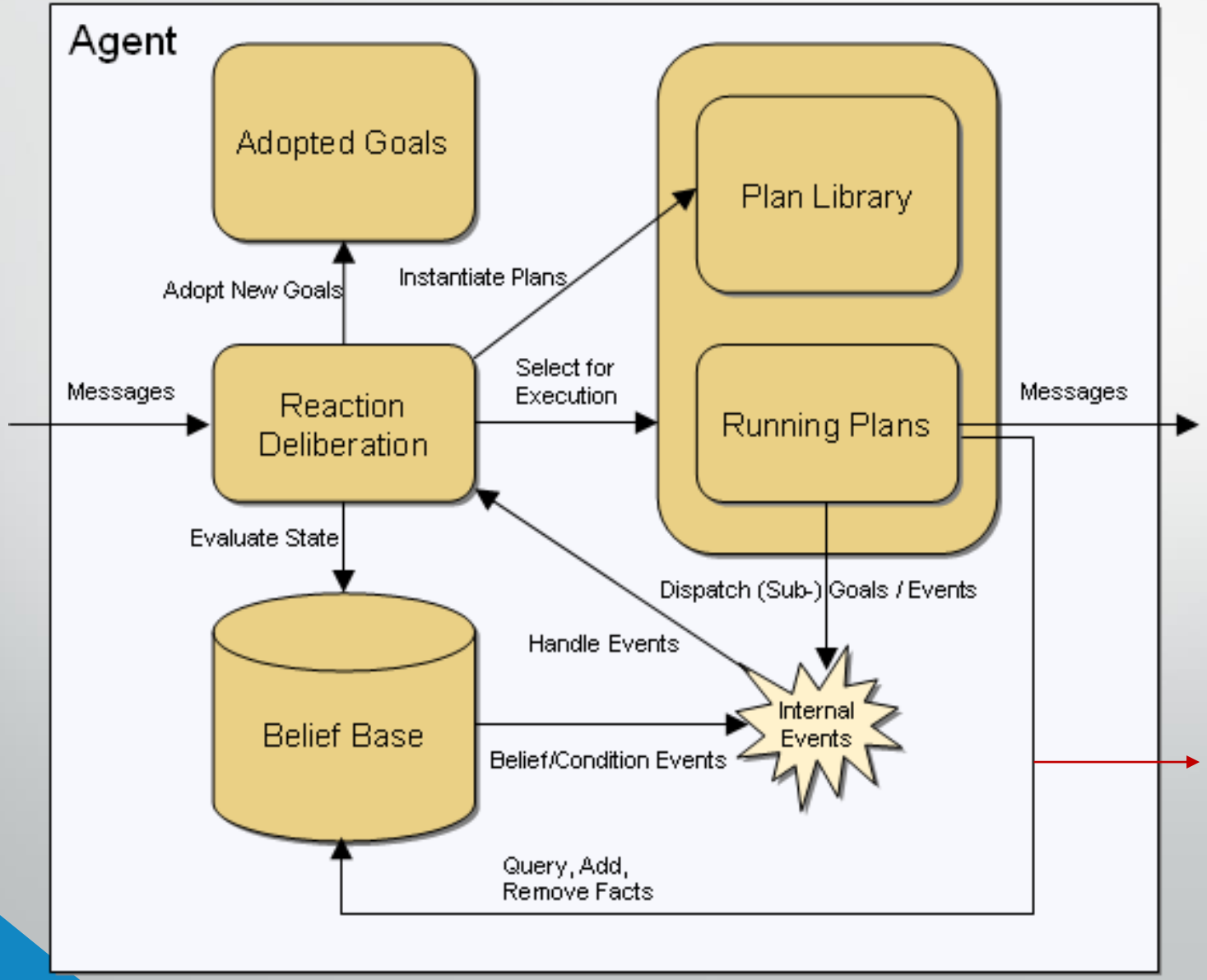
- Jadex = Jade + BDI Model (Belief Desire Intention)
- Jadex agents are goal-oriented (and not to behaviors such as the case of JADE agents)
- An architecture of agents is required to mount the many parts of a Distributed Intelligent Systems:
  - Internal agent
  - Company of agents
  - Artificial intelligence

Jadex

Goal-oriented Agents:

- Explicit classes:
  - **Belief, Goals, Plans**
- Agents have **Beliefs**, which presents knowledge that the agent has about himself and the environment
- **Goals** represent the agent's objectives and motivations, which will lead to the triggering of actions
- To achieve the **Goals**, the agents execute the **Plans**

Jadex



Plans can read and change agent beliefs

Jadex Agent Requirements

- **OQL-like query language**
  - Query language for facts on the **Beliefs Basis**
- **ADF - XML based Agent Definition File**
  - Specifies the agent's initial **Beliefs**, **Goals** and **Plans**
  - Jadex environment:
    - Reads file
    - Generate agent's mental model
    - Run agent according to the goals, selecting the plans
- **Plans – Java Classes with Actions**
  - Files that implement agent plans
  - Defines a set of predefined features/functionalities
    - E.g. Access the DF
  - File that can be plugged into agents (such as ADF)

Jadex - OQL

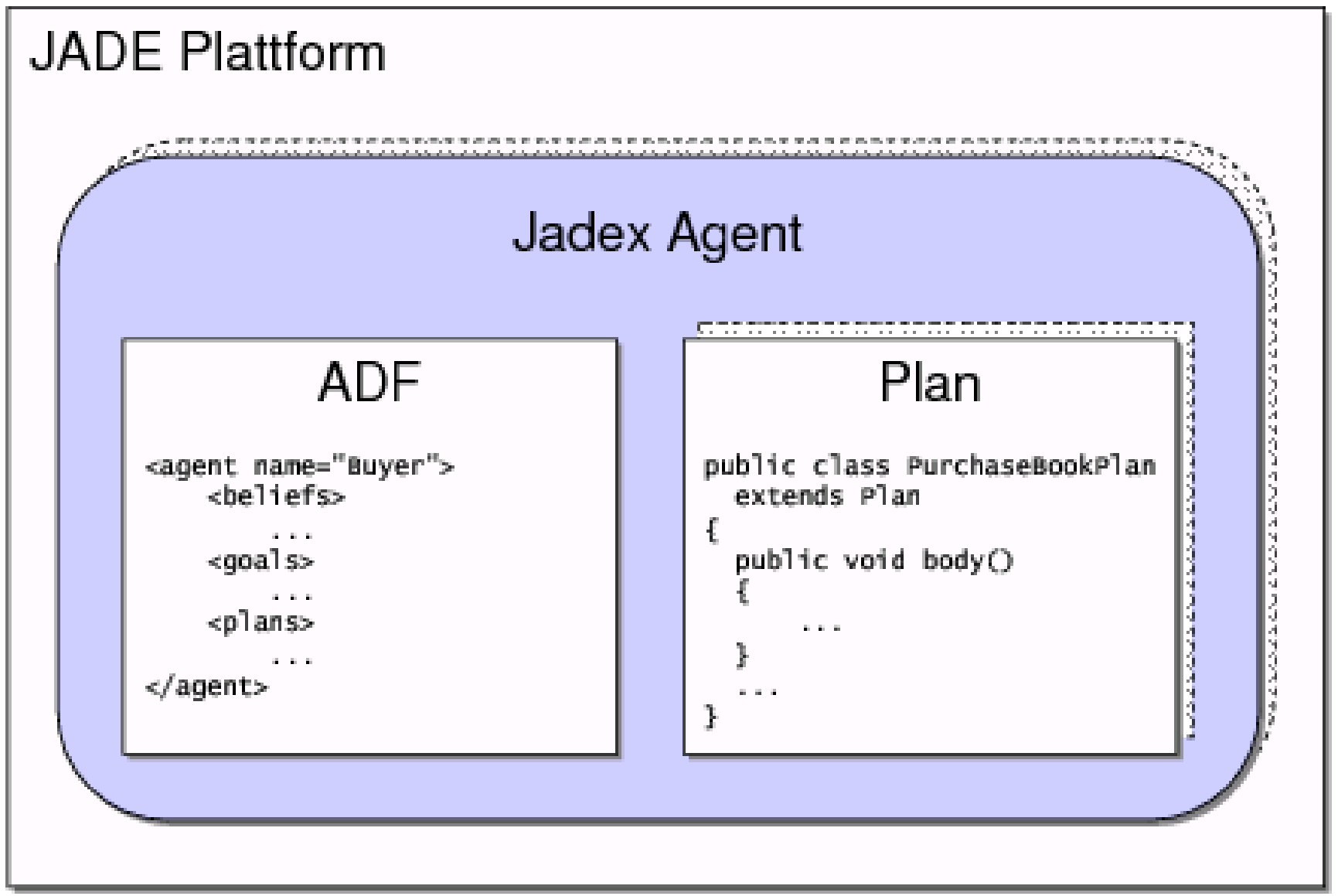
```
select_expression ::= "SELECT" ("ALL" | "ANY" | "IOTA")?
  (
    (expression "FROM" ("$" identifier "IN" expression) ("," "$" identifier "IN" expression)* )
    | ("$" identifier "FROM" expression)
  )
  ("WHERE" expression)?
  ("ORDER" "BY" expression ("ASC" | "DESC")? )?
```

Example: **SELECT** $block **FROM** $beliefbase.blocks **WHERE** $block.isClear()

Figure 1.5. OQL syntax in EBNF and query example

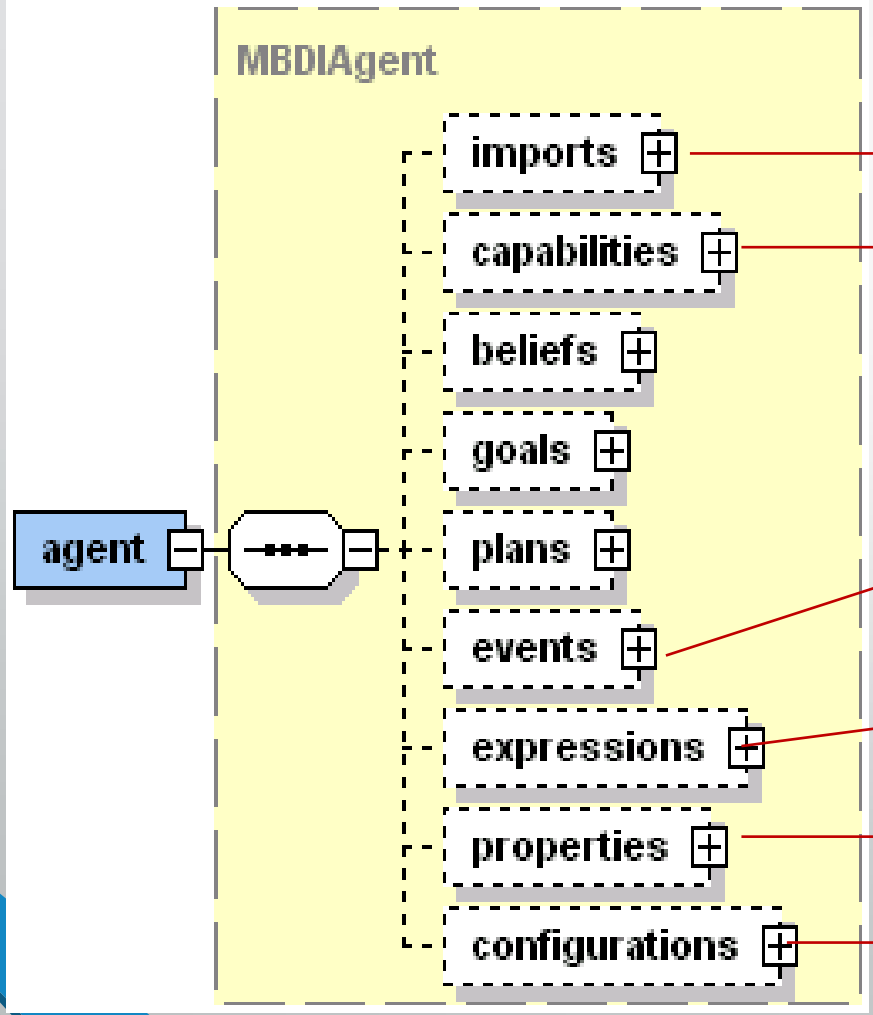
Create file types

Agent Platform



ADF File

- When the ADF file is loaded into Jadex, objects are created according to the specification



Java classes and packages used

Agent module composed of beliefs, goals and plans. (Allows reuse of code from different agents)

Two types of events: internal events and messages

Specify parameter values and beliefs, making it easier to select objects.

Properties that help in agent execution

Informs whether the data will be created when the agent is created or when it ends.

Plans

- Describe how the agent's actions are processed, according to the occurrence of **Events and Goals**.
- In Jadex the plans are divided into two parts:
  - **Head (ADF)**: specify when the plan is to be executed (XML File)
    - Put the reference to the plan class in the ADF file (including the import for the package where the class is)
  - **Body (Class)**: describe the plan implementation (Java Class File)
    - <u>Plans can be:</u>
      - **Active**: Always running
      - **Passive**: created at each event

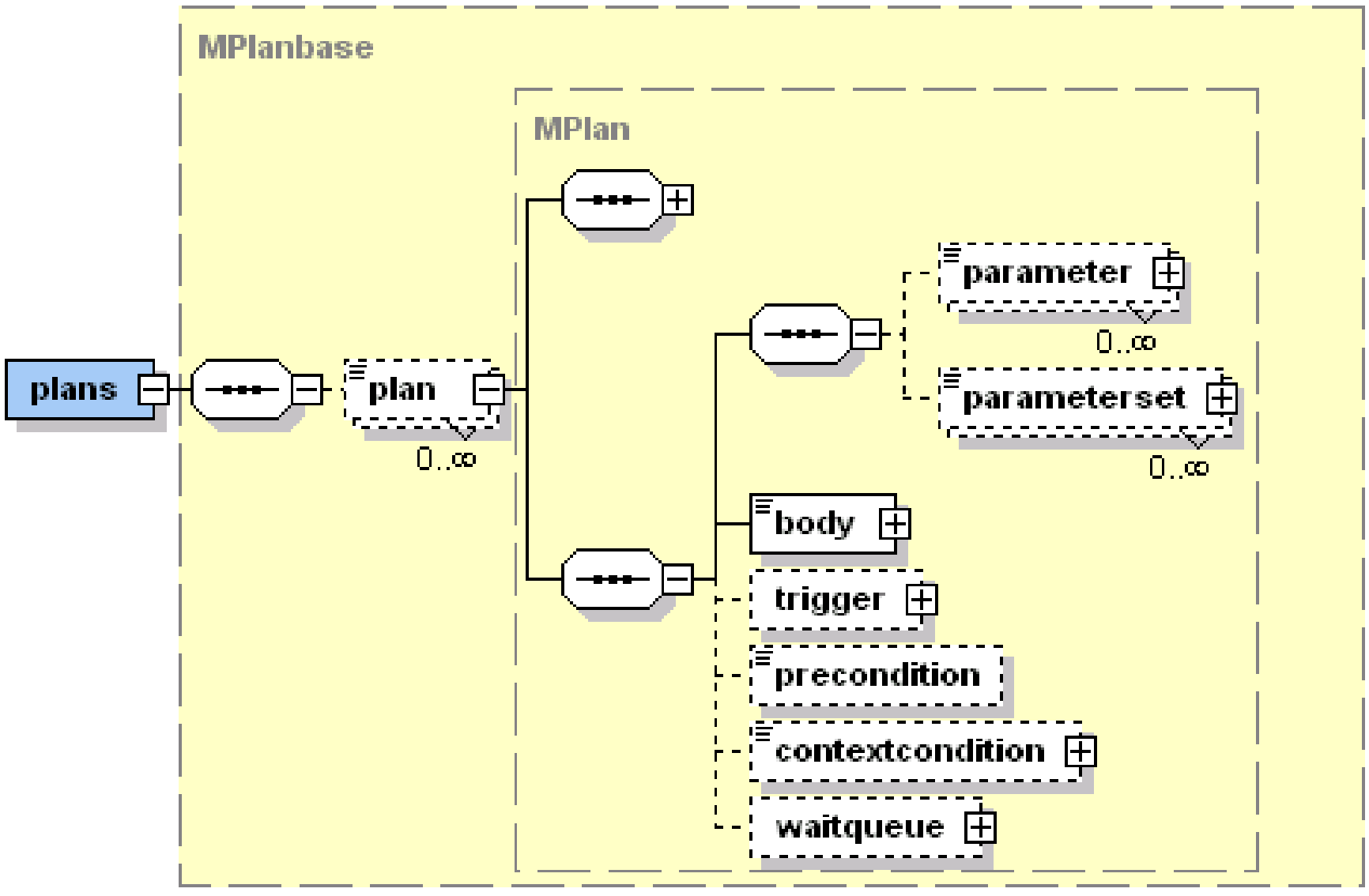
Plans Head Skeleton

```
<agent>
    ...
    <plan name="plan_name">
        ...
        <body class="JavaClass">
            <trigger>
                <goal ref="goal_name"/>
            </trigger>
        </body>
    </plan>
    ...
</agent>
```

Plans Body Skeleton

```
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.
        ...
        getException().printStackTrace();
    }

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

Plans


MPlanbase
MPlan
plans
plan
0..∞
parameter
0..∞
parameterset
0..∞
body
trigger
precondition
contextcondition
waitqueue

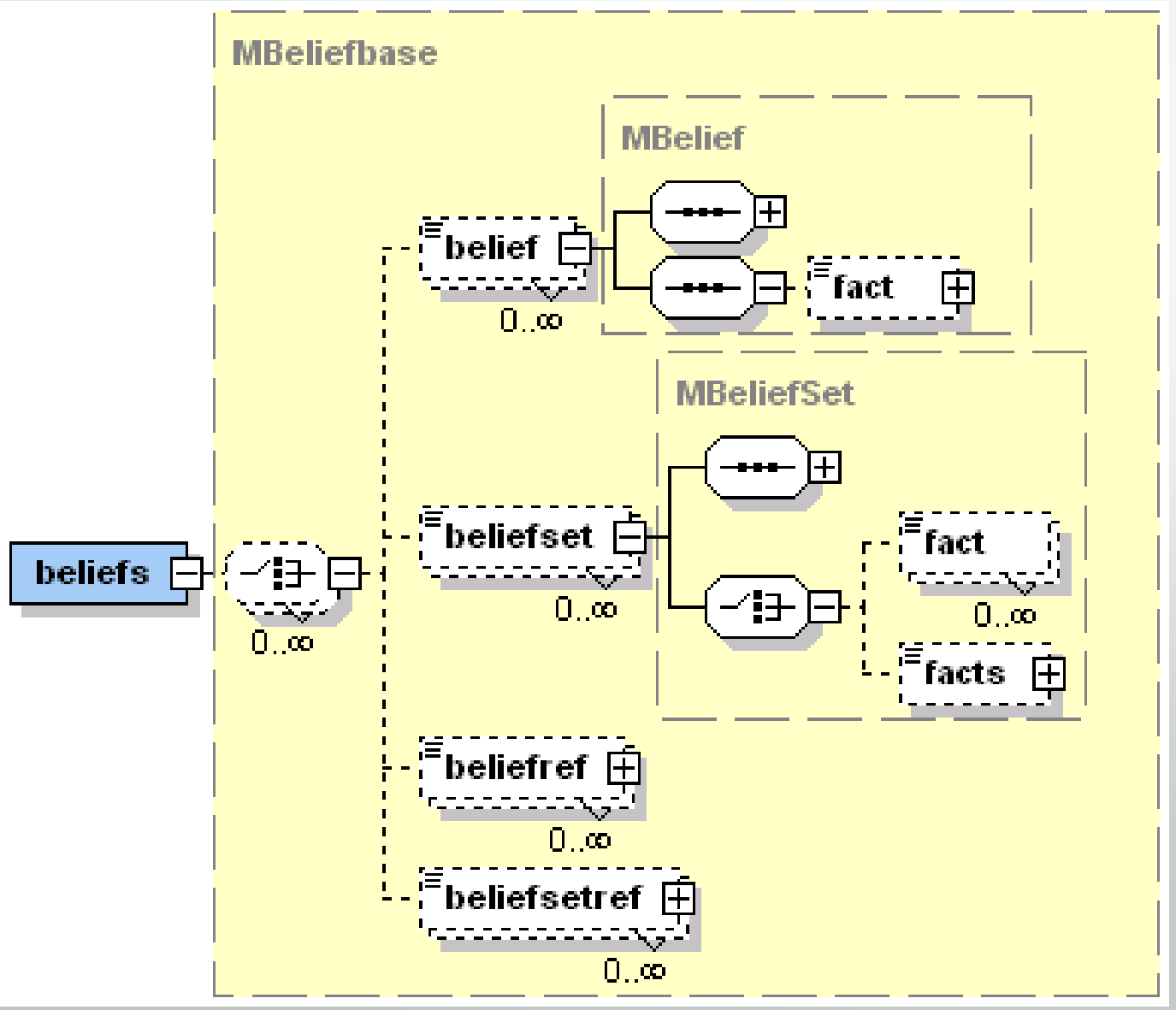
Beliefs

- Represents agents knowledge of the environment
- Must be declared in ADF file
- Can dynamically query, add and remove objects from **Belief Base**

Beliefs Skeleton

```
<agent>
    ...
    <beliefs>
        <!-- Belief declared -->
        <belief name="juice" class="boolean">
            <fact>false</fact>
        </belief>
        <!-- Inherited from capability -->
        <beliefref name="myself">
            <concrete ref="movement.myself" />
        </beliefref>
    ...
    </beliefs>
    ...
</agent>
```

Beliefs


MBeliefbase
MBelief
belief
0..∞
fact
MBeliefSet
beliefs
0..∞
beliefset
0..∞
fact
0..∞
facts
beliefref
0..∞
beliefsetref
0..∞

Goals

- **Perform:**
  - Something must be done but no specific result is expected.
  - E.g. Pick up the trash
- **Archieve:**
  - Describes a state you want to reach
  - Different execution alternatives (plans) can be used, not specifying the way to get there
  - E.g. Collected garbage
- **Query**:
  - It represents the need to obtain / query information
  - E.g. Want to know where the trash is
- **Maintain:**
  - Keep one or more conditions always satisfied.
  - E.g. Keep the environment free of garbage

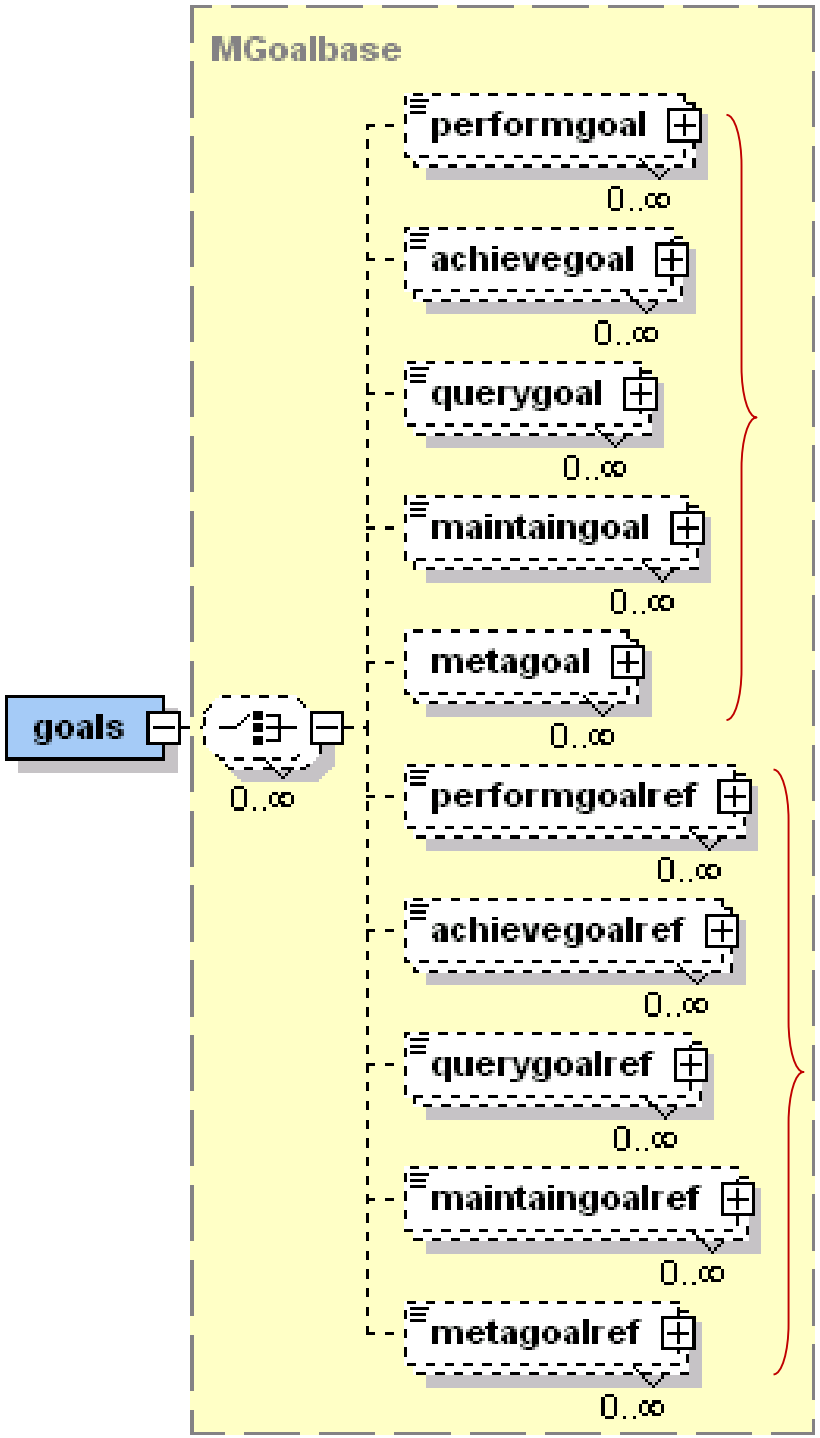
Goals Skeleton

```
<agent>
    ...
    <goals>
        <!-- Belief declared -->
        <performgoal name="goal_name" retry="true">
            <!-- All goal types may have drop conditions-->
            <dropcondition></dropcondition>
        </performgoal>
        <archievegoal name="goal_name" retry="false">
            <parameter name="parameter_name" class="parameter_class">

    <value>value_of_parameter</value>
            </parameter>
            <!--Goal condition to become active-->
            <createcondition></createcondition>
        </archievegoal>
        <querygoal name="goal_name" retry="true">
            ...
        </querygoal>
        <maintaingoal name="goal_name">
            ...
        </maintaingoal>
        ...
    </goals>
    ...
</agent>
```

Goals

- **Perform:**
  - Something must be done but no specific result is expected.
  - E.g. Pick up the trash
- **Archieve:**
  - Describes a state you want to reach
  - Different execution alternatives (plans) can be used, not specifying the way to get there
  - E.g. Collected garbage
- **Query**:
  - It represents the need to obtain / query information
  - E.g. Want to know where the trash is
- **Maintain:**
  - Keep one or more conditions always satisfied.
  - E.g. Keep the environment free of garbage

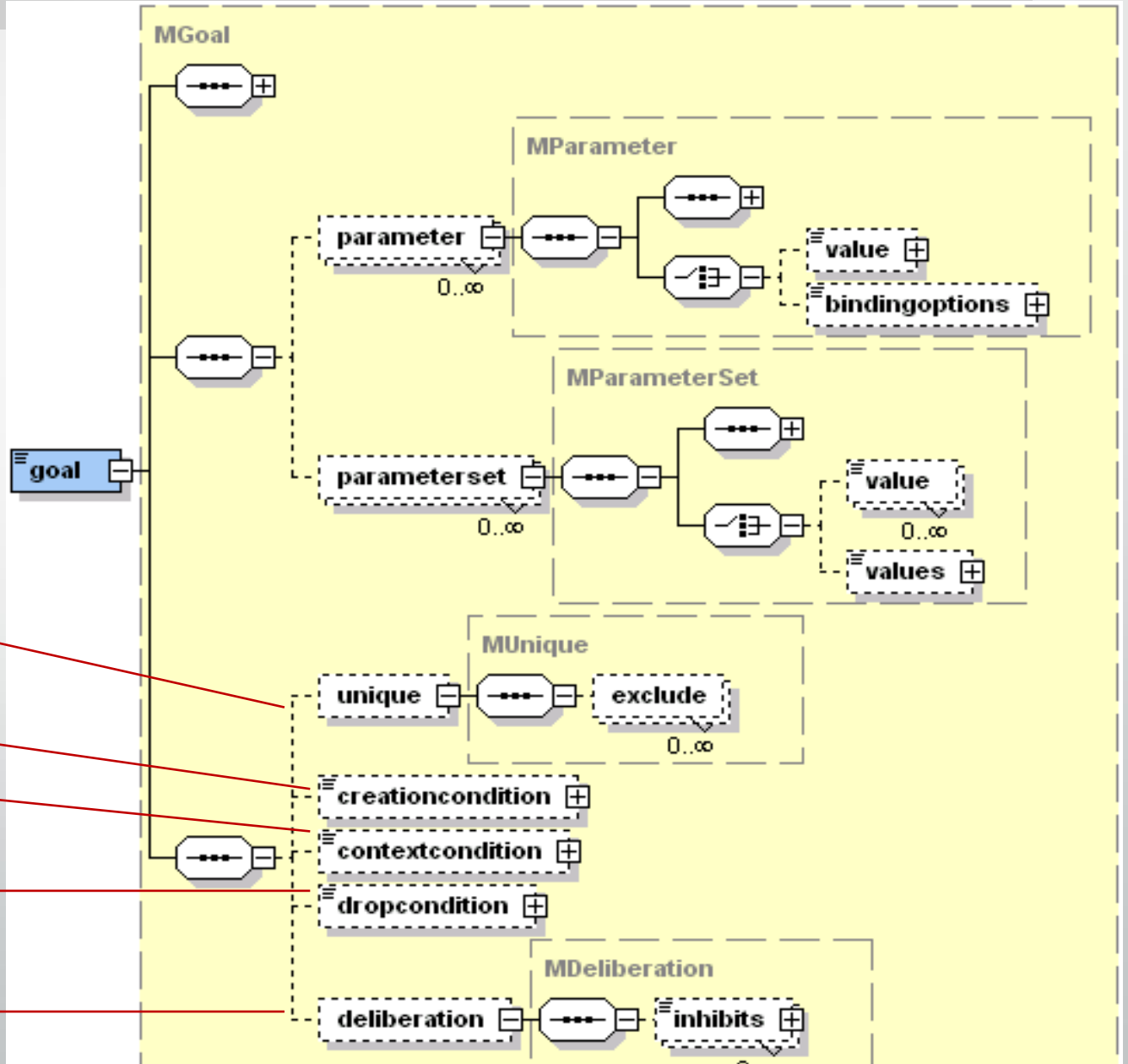
Goals



Goals Types

References to others defined goals in capacities

Goals



The goal is adopted one or more times

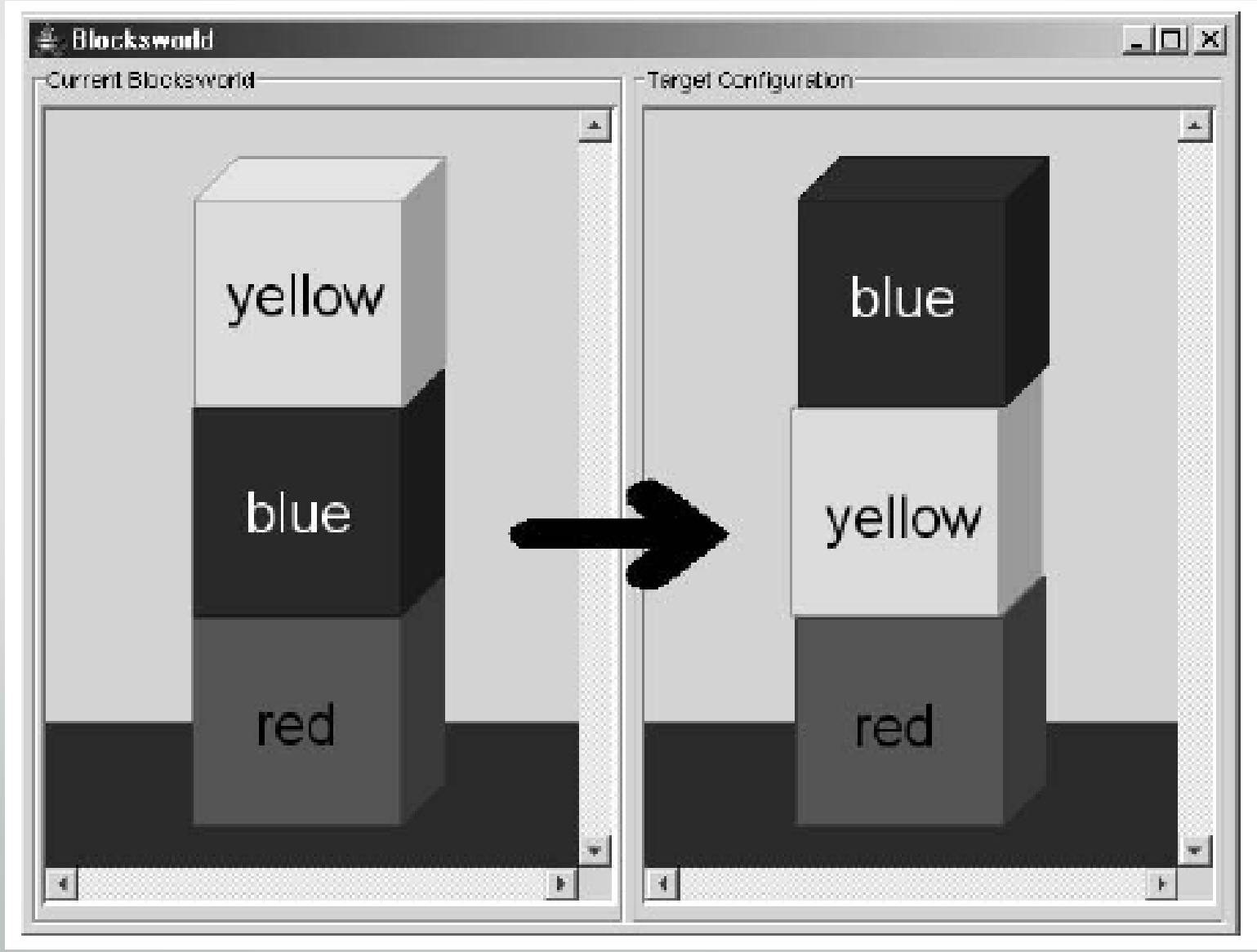
Creation Condition

Suspend Condition

Drop Condition (deleting the set of agent goals)

Selection Strategy

Example


Blocksworld
Current Blocksworld
yellow
blue
red
Target Configuration
blue
yellow
red

Example

Agent Model

```
<agent name="Blocksworld" package="jadex.examples.blocksworld">
  <imports>
    <import>java.awt.Color</import>
  </imports>

  <beliefs>
    <belief name="table" class="Table">
      <fact>new Table()</fact>
    </belief>
    <beliefset name="blocks" class="Block">
      <fact>new Block(new Color(240, 16, 16), $beliefbase.table)</fact>
      <fact>new Block(new Color(16, 16, 240), $beliefbase.table.allBlocks[0])</fact>
      <fact>new Block(new Color(240, 240, 16), $beliefbase.table.allBlocks[1])</fact>
      ...
    </beliefset>
  </beliefs>

  <goals>
    <achievegoal name="clear">
      <parameter name="block" class="Block" />
      <targetcondition>$goal.block.isClear()</targetcondition>
    </achievegoal>
    <achievegoal name="stack">
      <parameter name="block" class="Block" />
      <parameter name="target" class="Block" />
      <targetcondition>$goal.block.lower==$goal.target</targetcondition>
    </achievegoal>
    <achievegoal name="configure">
      <parameter name="configuration" class="Table" />
      <targetcondition>
        $beliefbase.table.configurationEquals($goal.configuration)
      </targetcondition>
    </achievegoal>
  </goals>

  <plans>
    <plan name="stack">
      <body>new StackBlocksPlan($event.goal.block, $event.goal.target)</body>
      <trigger><goal ref="stack"/></trigger>
    </plan>
    <plan name="configure">
      <body>new ConfigureBlocksPlan($event.goal.configuration)</body>
      <trigger><goal ref="configure"/></trigger>
    </plan>
    <plan name="clear">
      <bindings>
        <binding name="upper">
          select $upper from $beliefbase.blocks where $upper.getLower()==$event.goal.block
        </binding>
      </bindings>
      <body>new StackBlocksPlan($upper, $beliefbase.table)</body>
      <trigger><goal ref="clear"/></trigger>
    </plan>
  </plans>
</agent>
```

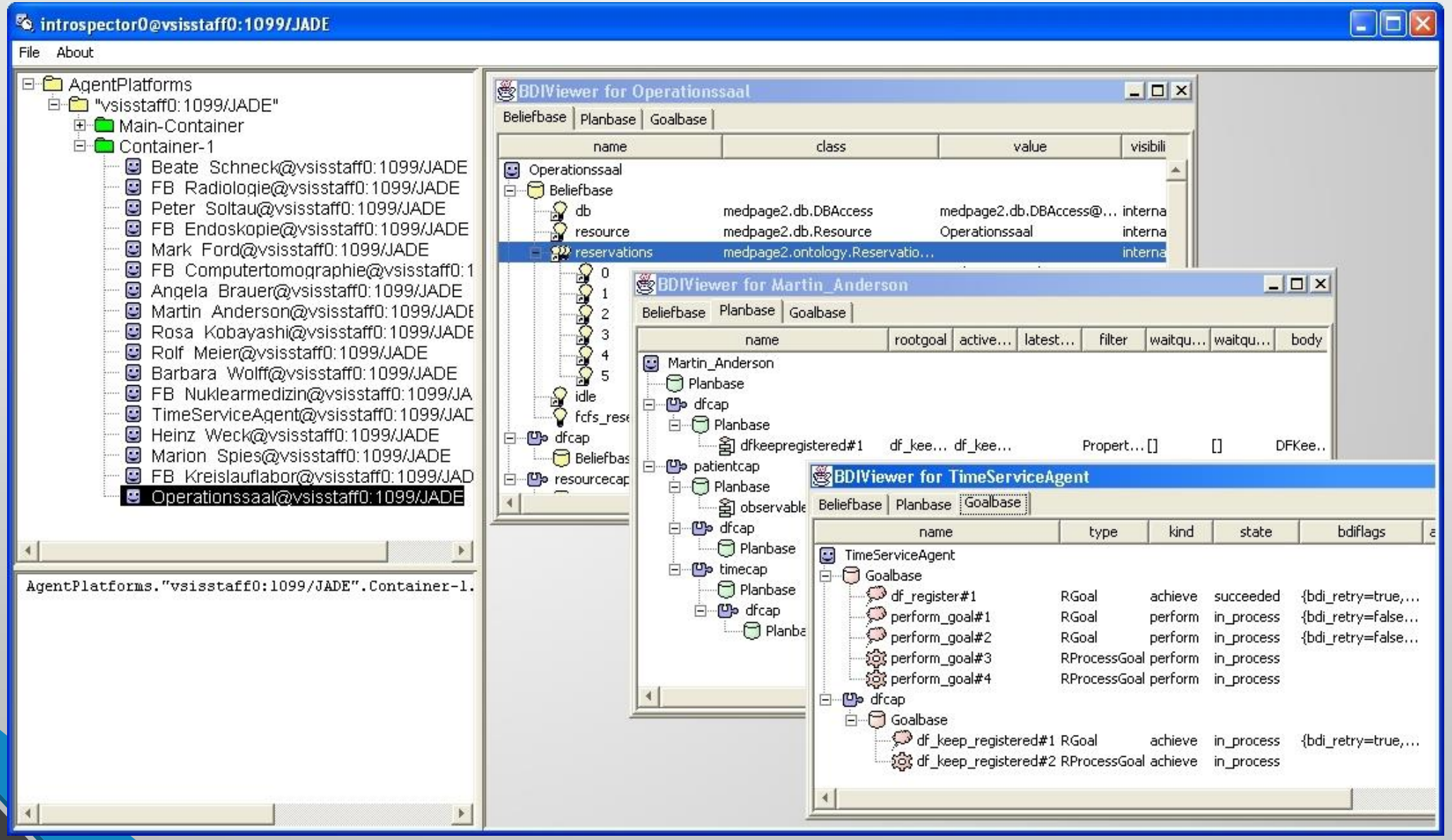
StackBlocks Plan

```
package jadex.examples.blocksworld;
import jadex.runtime.*;

/** Plan to stack one block on top of another target block. */
public class StackBlocksPlan extends Plan {
  protected Block block;
  protected Block target;

  public StackBlocksPaperPlan(Block block, Block target) {
    this.block = block;
    this.target = target;
  }

  public void body() {
    IGoal clear = createGoal("clear");
    clear.getParameter("block").setValue(block);
    dispatchSubgoalAndWait(clear);

    clear = createGoal("clear");
    clear.getParameter("block").setValue(target);
    dispatchSubgoalAndWait(clear);

    block.stackOn(target);
  }
}
```

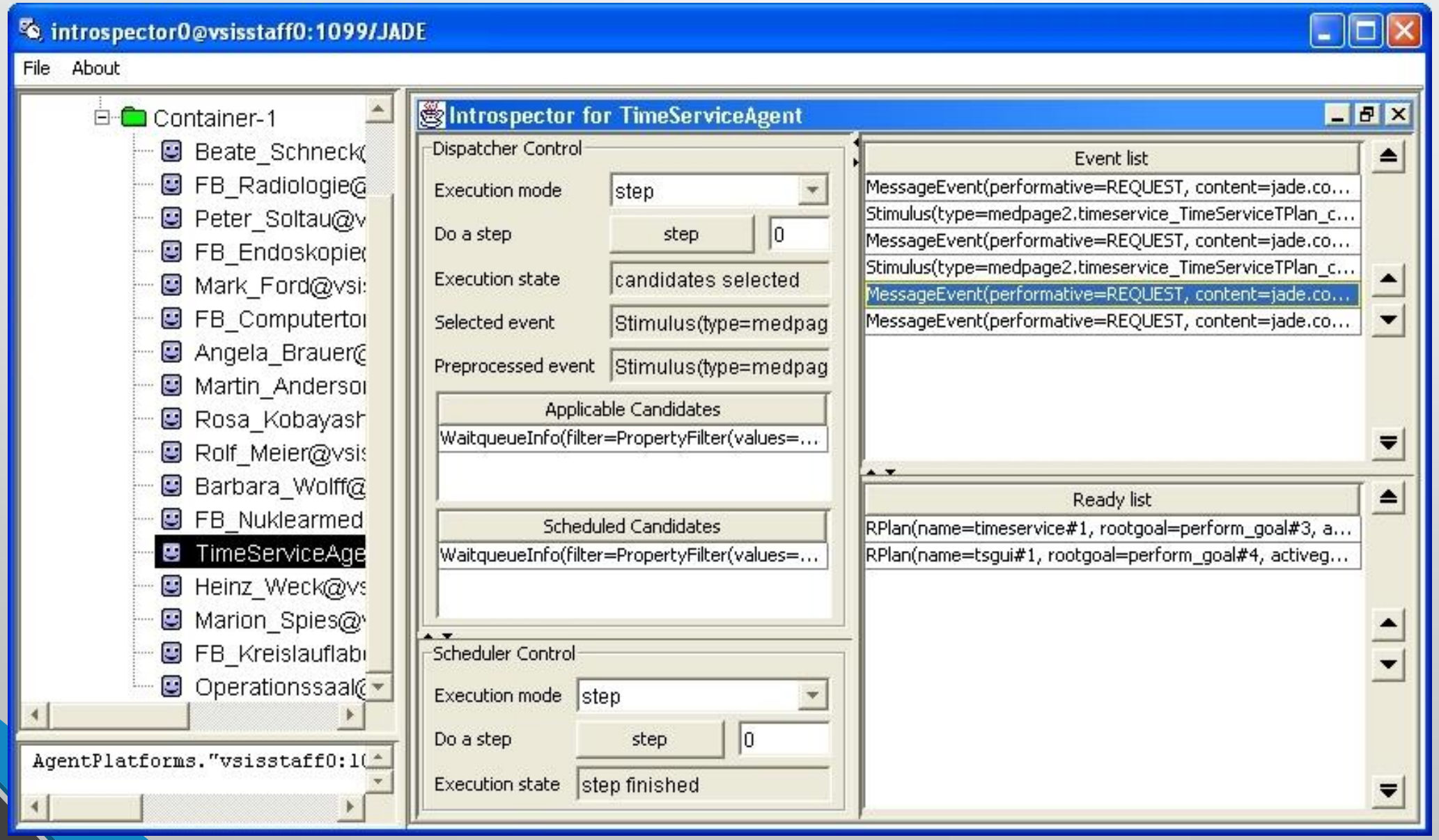
Jadex Tools

- Jadex presents all Jade tools plus:
  - **BDI Viewer Tool** makes it possible to see the internal state of the agents, i.e. their goals, plans and beliefs
  - **Jadex Introspector Tool** that allows monitoring the behavior of the agent and also modifies the execution of the agents

BDI Viewer Tool



Introspector Tool



JADEX Framework

Integrated Master's in Informatics Engineering

Intelligent Agents

2017/2018

Synthetic Intelligence Lab

Ricardo Ramos

Filipe Gonçalves

Paulo Novais