

JADEX Framework

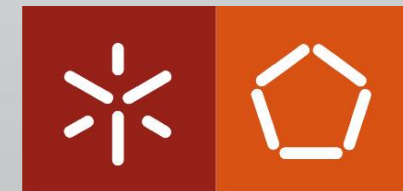
Integrated Master's in Informatics Engineering
Intelligent Agents
2017/2018

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/>

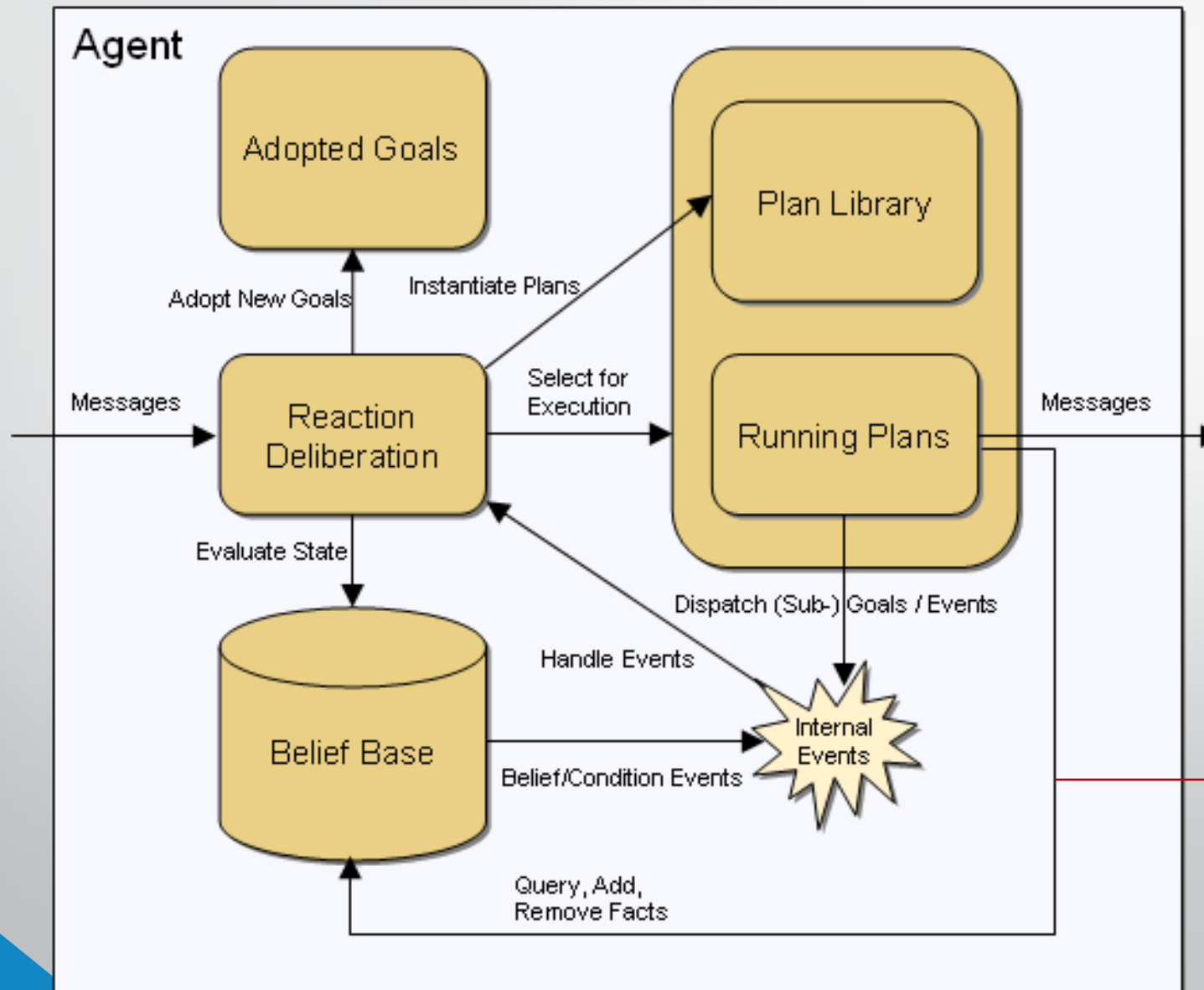
Jadex

- 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

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

```

01: select_expression ::= "SELECT" ("ALL" | "ANY" | "IOTA")?
02:   (
03:     (expression "FROM" ("$" identifier "IN" expression) ("," "$" identifier "IN" expression)* )
04:     | ("$" identifier "FROM" expression)
05:   )
06:   ("WHERE" expression)?
07:   ("ORDER" "BY" expression ("ASC" | "DESC")? )?

```

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

Figure 1.5. OQL syntax in EBNF and query example

Agent Platform

JADE Plattform

Jadex Agent

ADF

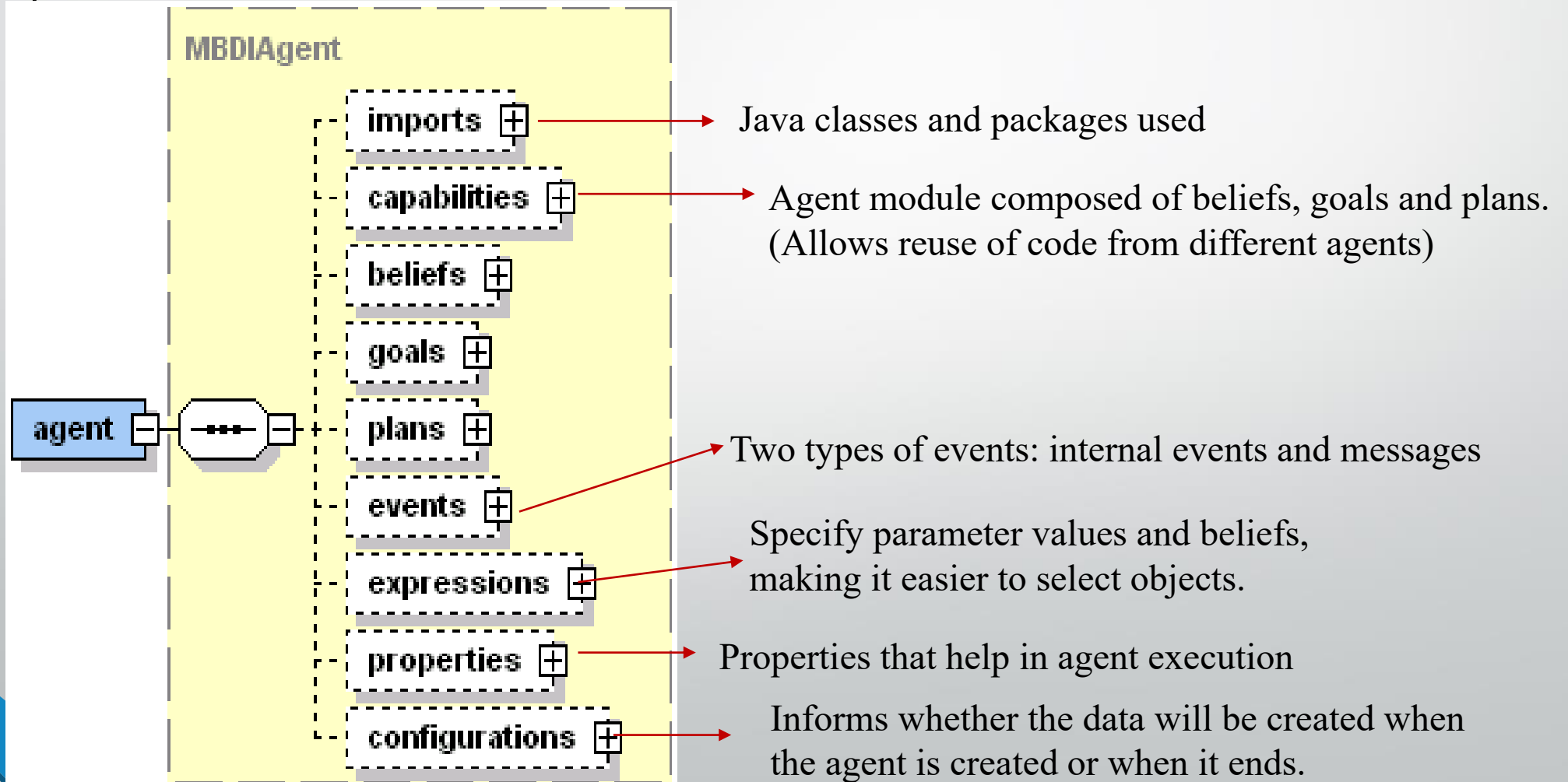
```
<agent name="Buyer">  
  <beliefs>  
    ...  
  <goals>  
    ...  
  <plans>  
    ...  
</agent>
```

Plan

```
public class PurchaseBookPlan  
  extends Plan  
{  
  public void body()  
  {  
    ...  
  }  
  ...  
}
```


ADF File

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



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)
 - Plans can be:
 - **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

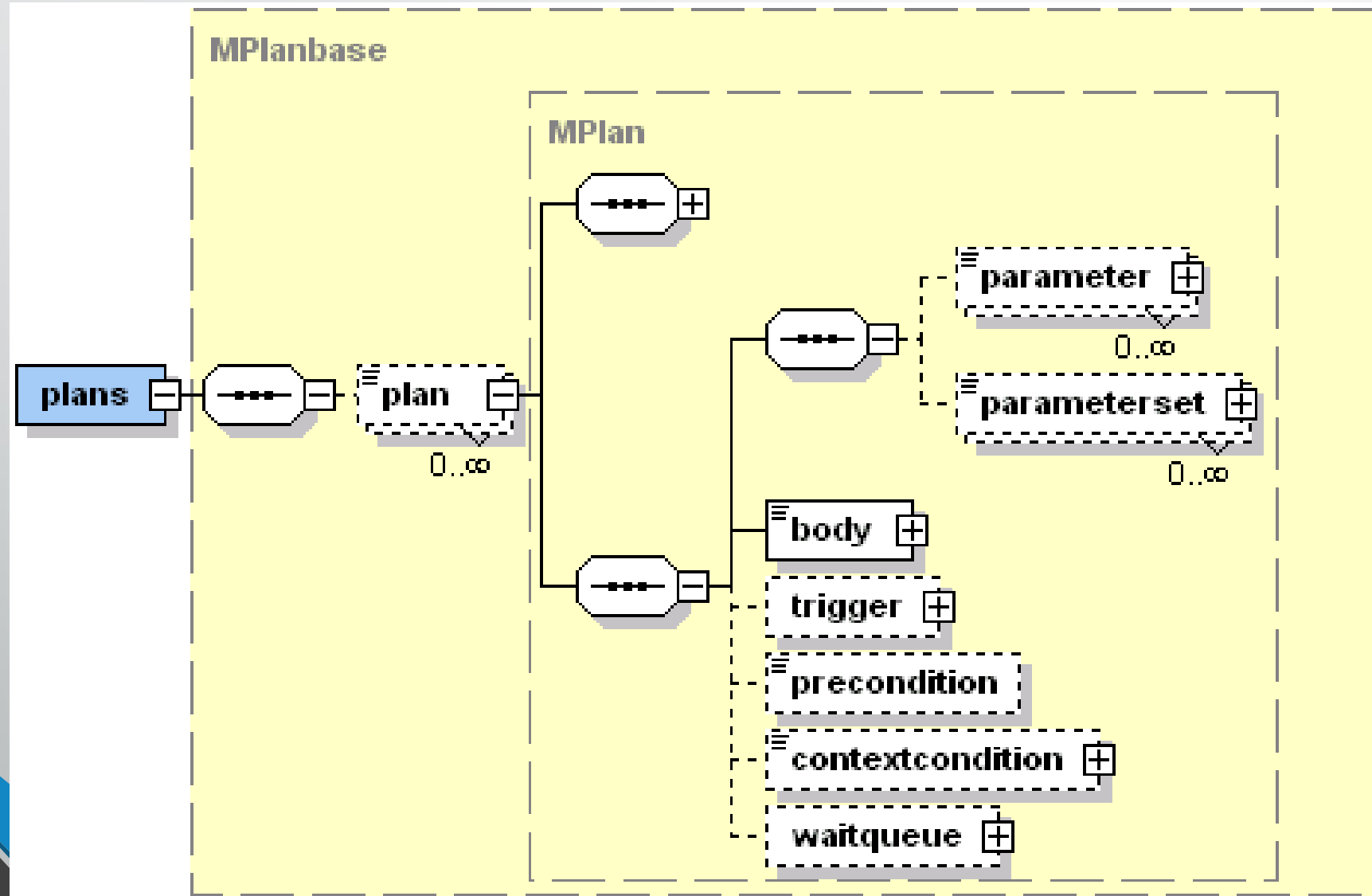
```
public class MyPlan extends Plan {  
    public void body() {  
        // Application code goes here.  
        ...  
    }  
}
```

```
    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



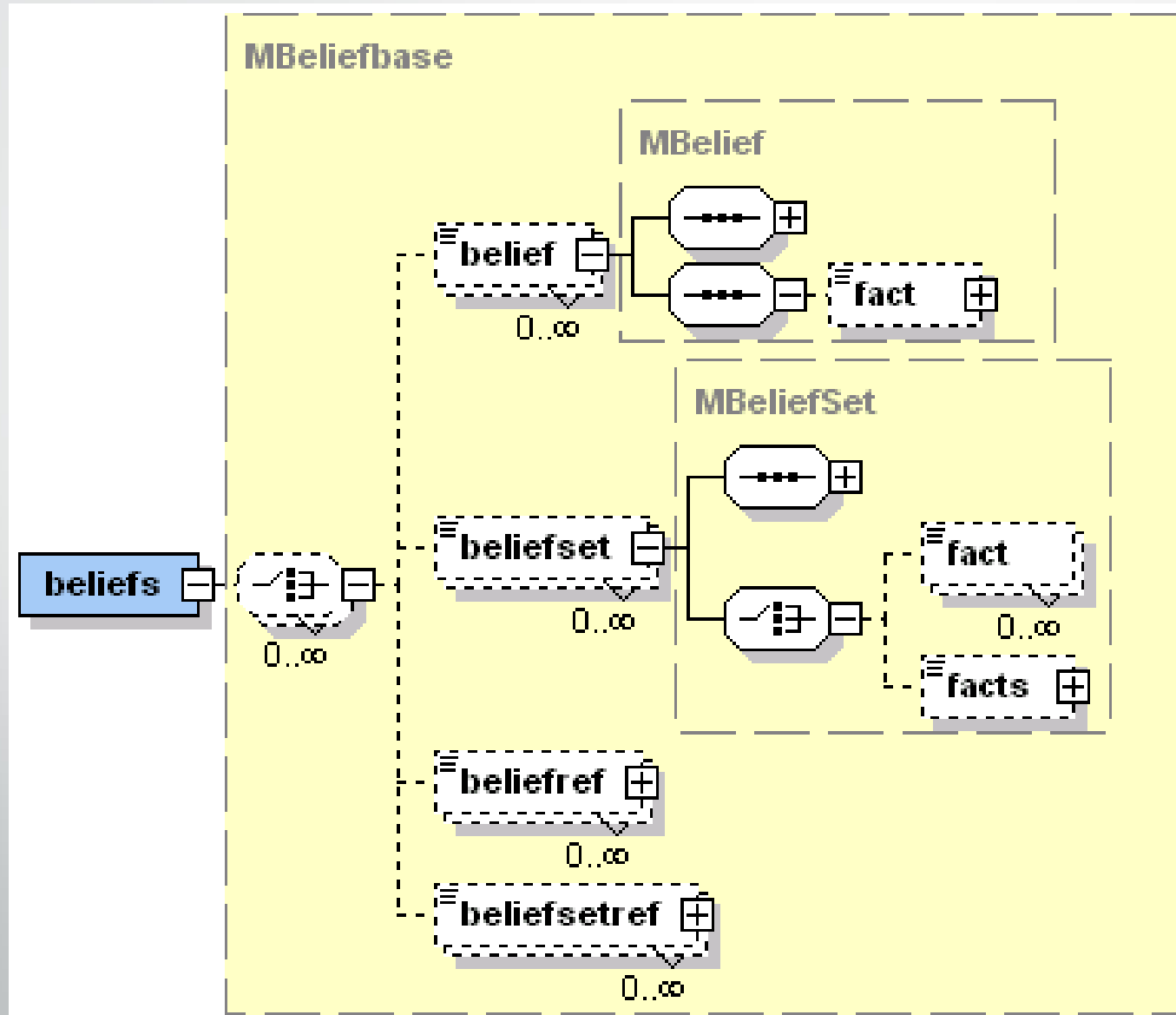
Beliefs

- 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



Goals

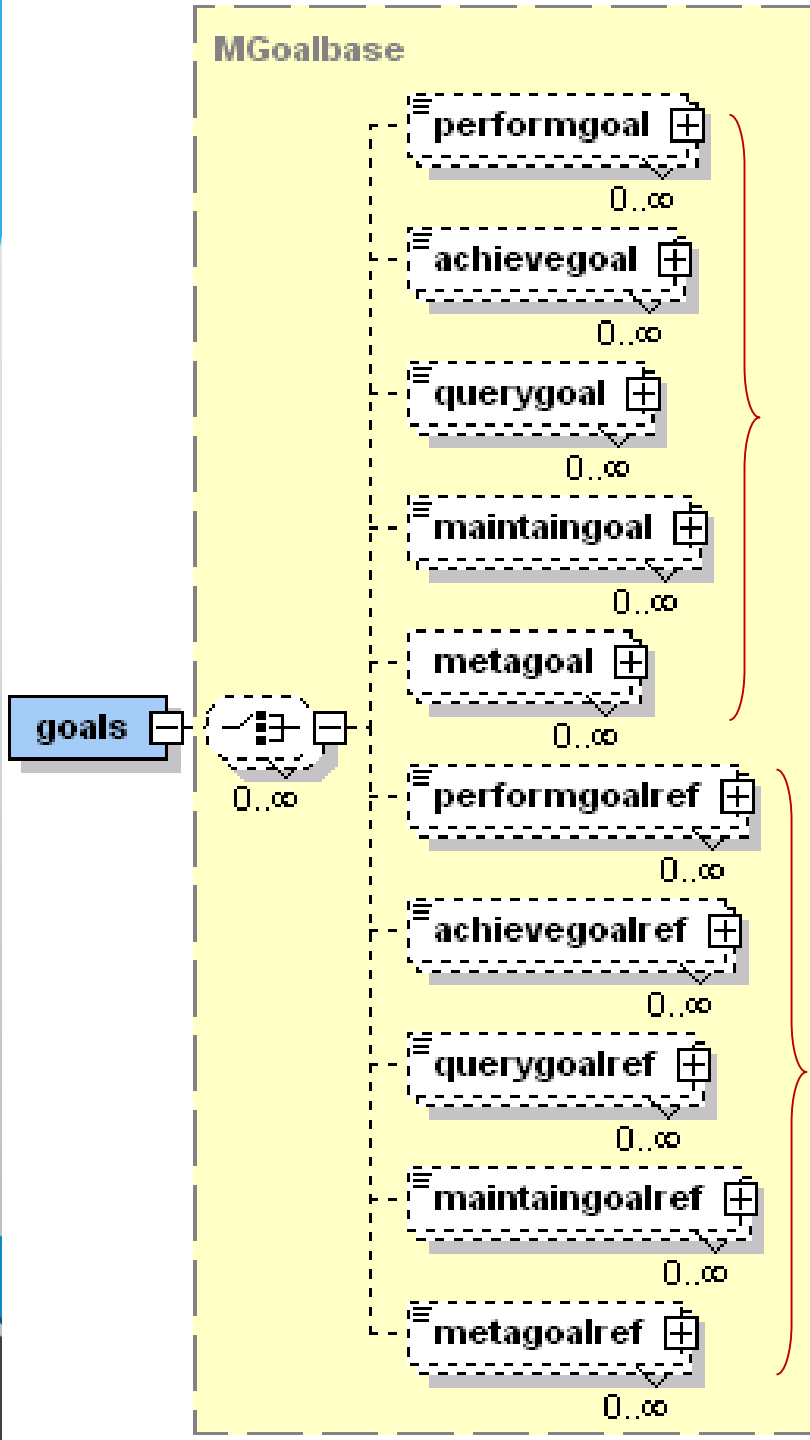
- **Perform:**
 - Something must be done but no specific result is expected.
 - E.g. Pick up the trash
- **Achieve:**
 - Describes a state you want to reach
 - Different execution alternatives (plans) can be used, not specifying the way to get there
 - E.g. Collected garbage
- **Query:**
 - It represents 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>	<!--Goal condition to become active-->
...	>
<goals>	<createcondition></createcondition>
<!-- Belief declared -->	</achievegoal>
<performgoal name="goal_name"	<querygoal name="goal_name"
retry="true">	retry="true">
<!-- All goal types may have drop	...
conditions-->	</querygoal>
<dropcondition></dropcondition>	<maintaingoal name="goal_name">
</performgoal>	...
<achievegoal name="goal_name"	</maintaingoal>
retry="false">	...
<parameter	</goals>
name="parameter_name" class="parameter_class">	...
<value>value_of_parameter</value>	</agent>
</parameter>	

Goals

- **Perform:**
 - Something must be done but no specific result is expected.
 - E.g. Pick up the trash
- **Achieve:**
 - 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

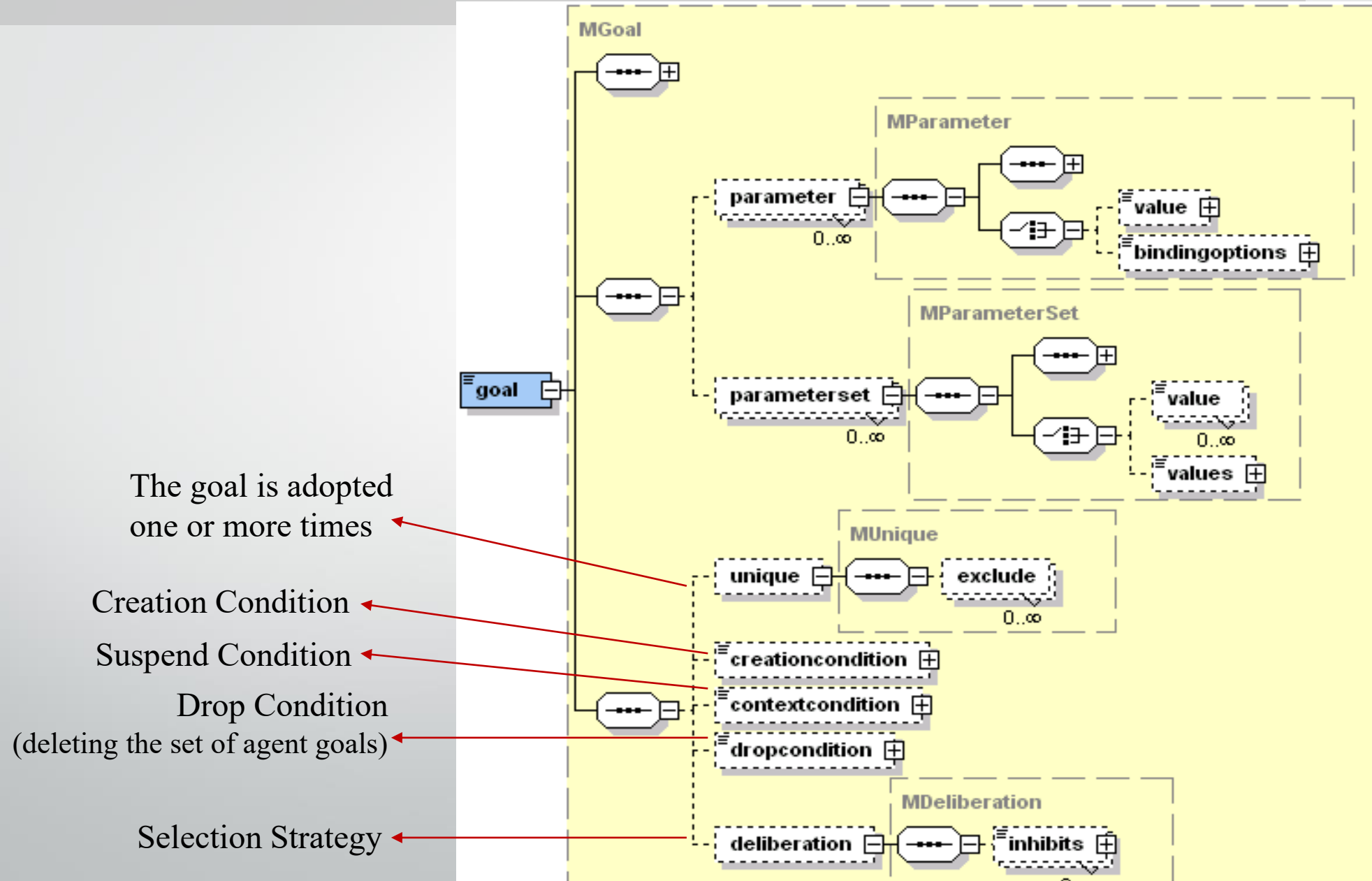


Goals

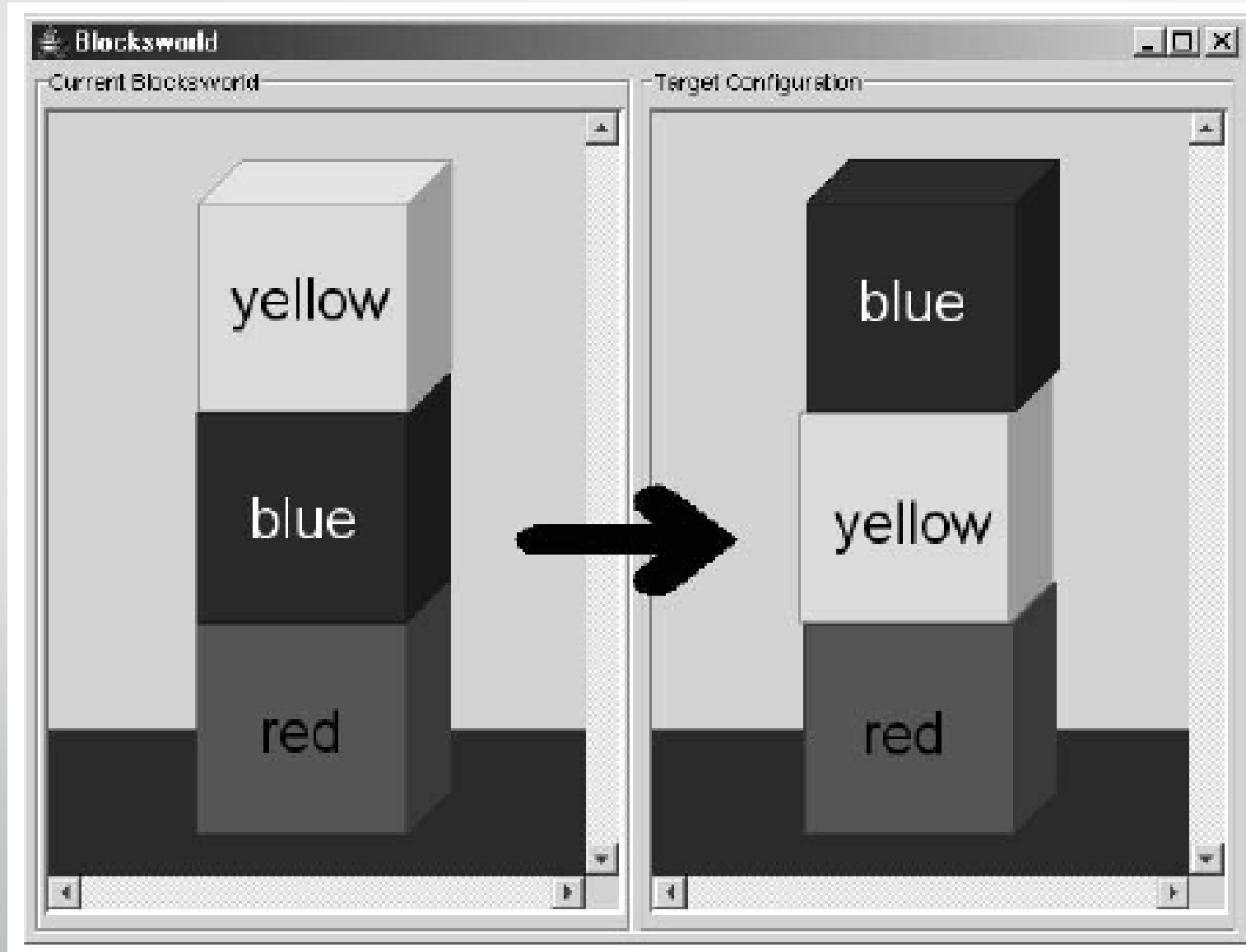
Goals Types

References to others
defined goals
in capacities

Goals



Example



Agent Model

```

01: <agent name="Blocksworld" package="jadex.examples.blocksworld">
02:   <imports>
03:     <import>java.awt.Color</import>
04:   </imports>
05:
06:   <beliefs>
07:     <belief name="table" class="Table">
08:       <fact>new Table()</fact>
09:     </belief>
10:     <beliefset name="blocks" class="Block">
11:       <fact>new Block(new Color(240, 16, 16), $beliefbase.table)</fact>
12:       <fact>new Block(new Color(16, 16, 240), $beliefbase.table.allBlocks[0])</fact>
13:       <fact>new Block(new Color(240, 240, 16), $beliefbase.table.allBlocks[1])</fact>
14:     ...
15:   </beliefset>
16: </beliefs>
17:
18:   <goals>
19:     <achievegoal name="clear">
20:       <parameter name="block" class="Block" />
21:       <targetcondition>$goal.block.isClear()</targetcondition>
22:     </achievegoal>
23:     <achievegoal name="stack">
24:       <parameter name="block" class="Block" />
25:       <parameter name="target" class="Block" />
26:       <targetcondition>$goal.block.lower==$goal.target</targetcondition>
27:     </achievegoal>
28:     <achievegoal name="configure">
29:       <parameter name="configuration" class="Table" />
30:       <targetcondition>
31:         $beliefbase.table.configurationEquals($goal.configuration)
32:       </targetcondition>
33:     </achievegoal>
34:   </goals>
35:
36:   <plans>
37:     <plan name="stack">
38:       <body>new StackBlocksPlan($event.goal.block, $event.goal.target)</body>
39:       <trigger><goal ref="stack"/></trigger>
40:     </plan>
41:     <plan name="configure">
42:       <body>new ConfigureBlocksPlan($event.goal.configuration)</body>
43:       <trigger><goal ref="configure"/></trigger>
44:     </plan>
45:     <plan name="clear">
46:       <bindings>
47:         <binding name="upper">
48:           select $upper from $beliefbase.blocks where $upper.getLower()==$event.goal.block
49:         </binding>
50:       </bindings>
51:       <body>new StackBlocksPlan($upper, $beliefbase.table)</body>
52:       <trigger><goal ref="clear"/></trigger>
53:     </plan>
54:   </plans>
55: </agent>

```

Example

StackBlocks Plan

```

01: package jadex.examples.blocksworld;
02: import jadex.runtime.*;
03:
04: /** Plan to stack one block on top of another target block. */
05: public class StackBlocksPlan extends Plan {
06:   protected Block block;
07:   protected Block target;
08:
09:   public StackBlocksPaperPlan(Block block, Block target) {
10:     this.block = block;
11:     this.target = target;
12:   }
13:
14:   public void body() {
15:     IGoal clear = createGoal("clear");
16:     clear.getParameter("block").setValue(block);
17:     dispatchSubgoalAndWait(clear);
18:
19:     clear = createGoal("clear");
20:     clear.getParameter("block").setValue(target);
21:     dispatchSubgoalAndWait(clear);
22:
23:     block.stackOn(target);
24:   }
25: }

```

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

introspector0@vsiisstaff0:1099/JADE

File About

AgentPlatforms

- "vsiisstaff0:1099/JADE"
 - Main-Container
 - Container-1
 - Beate_Schneck@vsiisstaff0:1099/JADE
 - FB_Radiologie@vsiisstaff0:1099/JADE
 - Peter_Soltau@vsiisstaff0:1099/JADE
 - FB_Endoskopie@vsiisstaff0:1099/JADE
 - Mark_Ford@vsiisstaff0:1099/JADE
 - FB_Computertomographie@vsiisstaff0:1099/JADE
 - Angela_Brauer@vsiisstaff0:1099/JADE
 - Martin_Anderson@vsiisstaff0:1099/JADE
 - Rosa_Kobayashi@vsiisstaff0:1099/JADE
 - Rolf_Meier@vsiisstaff0:1099/JADE
 - Barbara_Wolff@vsiisstaff0:1099/JADE
 - FB_Nuklearmedizin@vsiisstaff0:1099/JADE
 - TimeServiceAgent@vsiisstaff0:1099/JADE
 - Heinz_Weck@vsiisstaff0:1099/JADE
 - Marion_Spies@vsiisstaff0:1099/JADE
 - FB_Kreislauflabor@vsiisstaff0:1099/JADE
 - Operationssaal@vsiisstaff0:1099/JADE**

AgentPlatforms."vsiisstaff0:1099/JADE".Container-1.

BDIVIEWER for Operationssaal

Beliefbase Planbase Goalbase

name	class	value	visibili
Operationssaal			
db	medpage2.db.DBAccess	medpage2.db.DBAccess@...	interna
resource	medpage2.db.Resource	Operationssaal	interna
reservations	medpage2.ontology.Reservatio...		interna

BDIVIEWER for Martin_Anderson

Beliefbase Planbase Goalbase

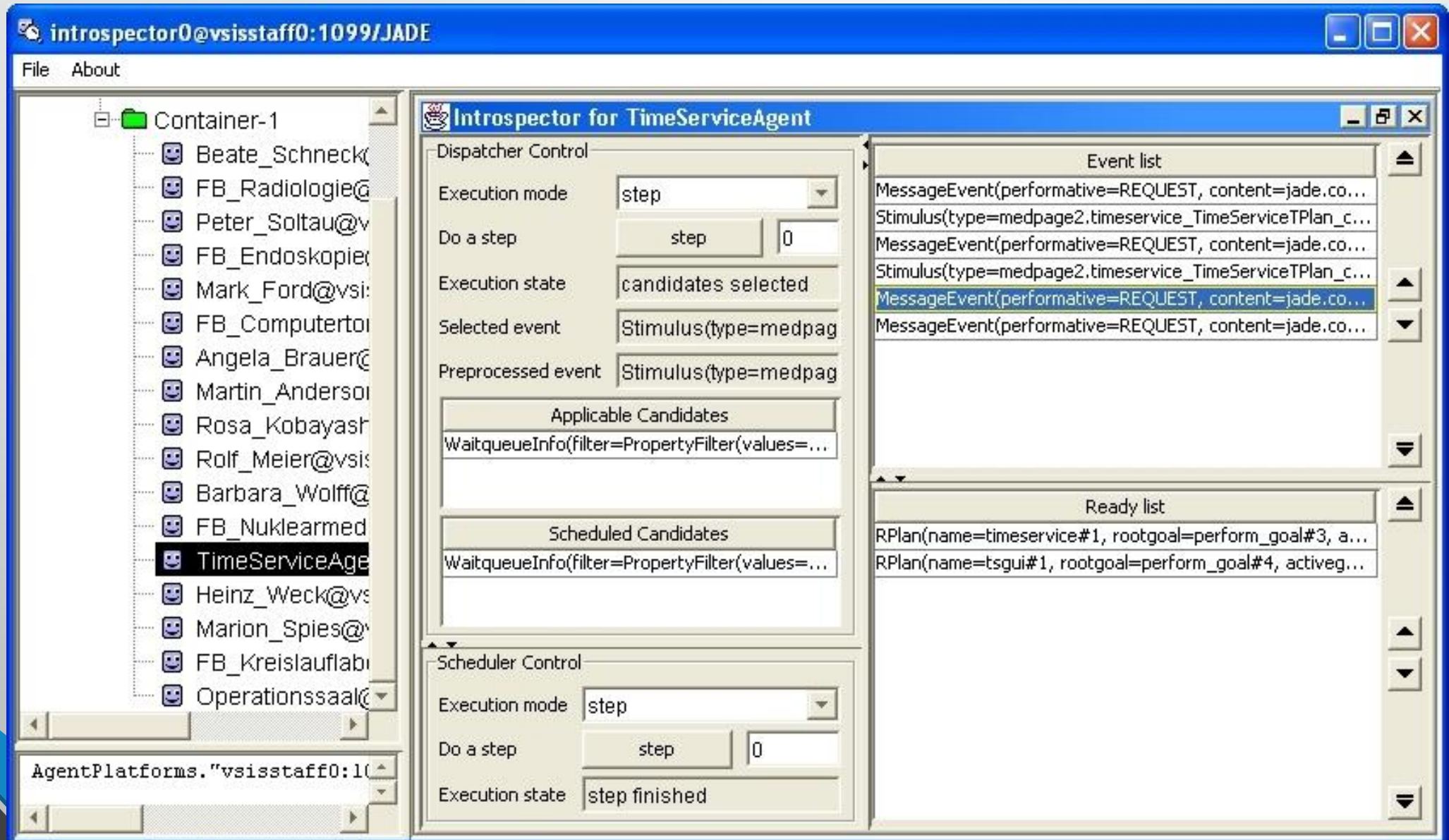
name	rootgoal	active...	latest...	filter	waitqu...	waitqu...	body
Martin_Anderson							
Planbase							
dfcap							
Planbase							
dfkeepregistered#1	df_kee...	df_kee...		Propert...			DFKee...
patientcap							
Planbase							
observable							
dfcap							
Planbase							
timecap							
Planbase							
dfcap							

BDIVIEWER for TimeServiceAgent

Beliefbase Planbase Goalbase

name	type	kind	state	bdi flags
TimeServiceAgent				
Goalbase				
df_register#1	RGoal	achieve	succeeded	{bdi_retry=true,...
perform_goal#1	RGoal	perform	in_process	{bdi_retry=false,...
perform_goal#2	RGoal	perform	in_process	{bdi_retry=false,...
perform_goal#3	RProcessGoal	perform	in_process	
perform_goal#4	RProcessGoal	perform	in_process	
dfcap				
Goalbase				
df_keep_registered#1	RGoal	achieve	in_process	{bdi_retry=true,...
df_keep_registered#2	RProcessGoal	achieve	in_process	

Introspector Tool



The screenshot displays the Introspector Tool interface, which is used for monitoring and controlling the execution of agents in a JADE environment. The main window is titled "introspector0@vsisstaff0:1099/JADE".

Left Panel (Agent List): A tree view showing the hierarchy of agents. Under "Container-1", several agents are listed, including "TimeServiceAgent", which is currently selected.

Right Panel (Introspector for TimeServiceAgent): This panel provides detailed control and monitoring for the selected agent.

- Dispatcher Control:**
 - Execution mode:
 - Do a step:
 - Execution state:
 - Selected event:
 - Preprocessed event:
- Applicable Candidates:**
 -
- Scheduled Candidates:**
 -
- Scheduler Control:**
 - Execution mode:
 - Do a step:
 - Execution state:
- Event list:** A list of events being processed, including:
 - MessageEvent(performative=REQUEST, content=jade.co...
 - Stimulus(type=medpage2.timeservice_TimeServiceTPlan_c...
 - MessageEvent(performative=REQUEST, content=jade.co...
 - Stimulus(type=medpage2.timeservice_TimeServiceTPlan_c...
 - MessageEvent(performative=REQUEST, content=jade.co...
 - MessageEvent(performative=REQUEST, content=jade.co...
- Ready list:** A list of ready-to-execute plans, including:
 - RPlan(name=timeservice#1, rootgoal=perform_goal#3, a...
 - RPlan(name=tsgui#1, rootgoal=perform_goal#4, activeg...

The bottom status bar shows "AgentPlatforms."vsisstaff0:1099/JADE".

JADEX Framework

Integrated Master's in Informatics Engineering
Intelligent Agents
2017/2018

Synthetic Intelligence Lab

Ricardo Ramos

Filipe Gonçalves

Paulo Novais

