Introduction to Jadex programming

Reza Saeedi

Reza.saeedi@alumni.um.ac.ir

Outline

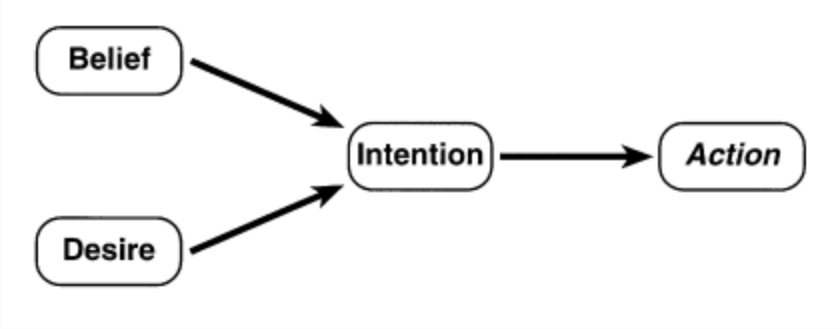
- Multi agent systems
- BDI Model
- Jadex
- Jadex concepts
  - Belief
  - Goal
  - Plan
- Jadex Programming
- Jadex Enviroment

Multi-agent system

- A multi-agent system is a system composed of multiple interacting intelligent agents within an environment.
- Multi-agent systems can be used to solve problems that are difficult or impossible for an individual agent or a monolithic system to solve
- Multi-agent systems consist of agents and their environment.
- The agents in a multi-agent system have several important characteristics:
  - Autonomy
  - Local views
  - Decentralization

BDI model

- A software model developed for programming intelligent agents.
- This model has three concepts:
  - Belief
  - Desire
  - intention



- actually uses these concepts to solve a particular problem in agent programming.

Jadex

- An agent-oriented reasoning engine for writing rational agents
- Based on BDI model
  - **Beliefs** and goals leading to the selection and stepwise execution of plans
  - **Goals:** conflict-free desires, modeled as events
  - **Plans:** executable representation of intentions
- Integrate agent theories with object-orientation and XML descriptions
- No new programming language is introduced
- For help using www.activecomponents.org

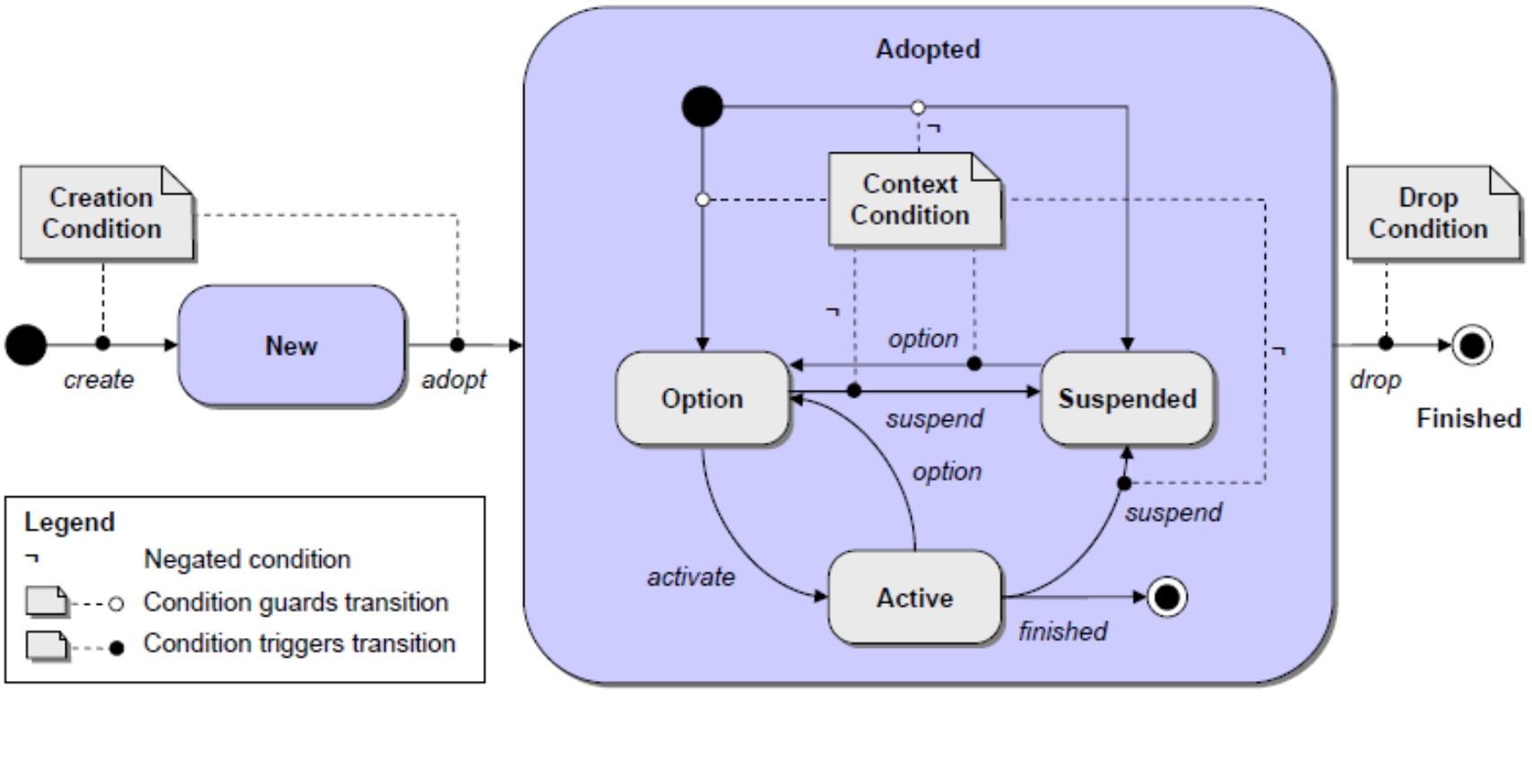
Jadex Beliefs

- Belief base contains the knowledge of an agent
  - Beliefs (single facts stored as Java objects)
  - Beliefsets (sets of facts as Java Objects)
  - Java objects
  - key / value pairs
- Advantages of storing information as facts
  - Central place for knowledge (accessible to all plans)
  - Allows queries over the agent‘s beliefs
  - Allows monitoring of beliefs and conditions (e.g. to trigger events / goals)

Jadex Goals

- Generic goal types
  - perform (some action)
  - achieve (a specified world state)
  - query (some information)
  - maintain (re-establish a specified world state whenever violated)
- Are strongly typed with
  - name, type, parameters
  - BDI flags enable non-default goal-processing(for example **retry=true**)
- Goal creation/deletion possibilities
  - initial goals for agents
  - goal creation/drop conditions for all goal kinds
  - top-level / subgoals from within plans

Goal Lifecycle



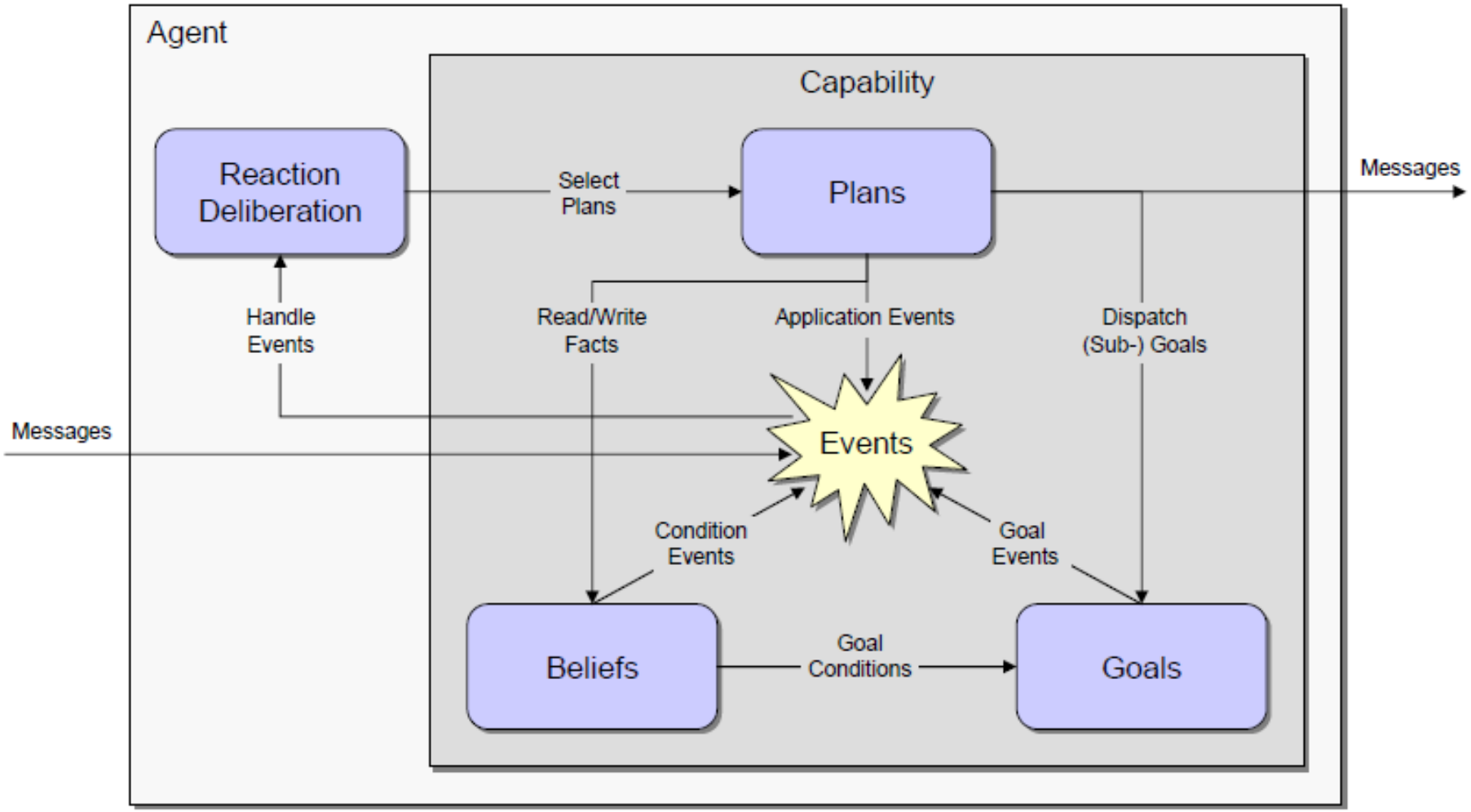
Plans

- Represent procedural knowledge
  - Means for goal achievement and reacting to events
  - Agent has library of pre-defined plans
- Realization of a plan
  - Plan head specified in ADF
  - Plan body coded in pure Java
- Assigning plans to goals/events
  - Plan head indicates ability to handle goals/events
  - Plan context / precondition further refines set of applicable plans

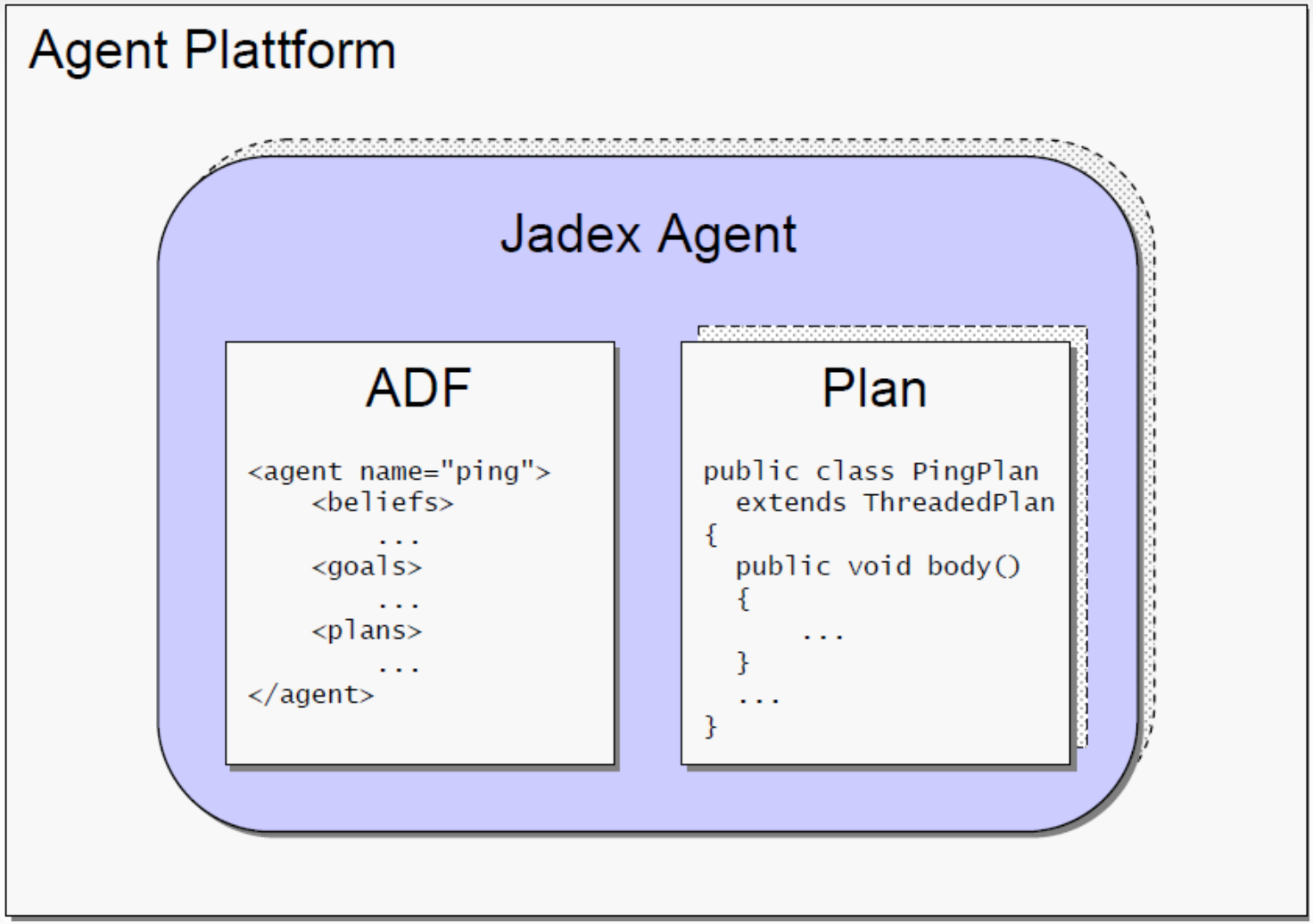
Jadex Event

- Three types of events
  - *Message event* denotes arrival/sending messages
  - *Goal event* denotes a new goal to be processed or the state of an existing goal is changed
  - *Internal event*
    - *Timeout* event denotes a timeout has occurred, e.g., waiting for arrival of messages/achieving goals/waitFor(duration) actions.
    - *Execute plan* event denotes plan to be executed without meta- level reasoning, e.g., plans with triggering condition
    - *Condition-triggered* event is generated when a state change occur that satisfies the trigger of a condition

Jadex Abstract Agent Architecture



Components of a Jadex Agent



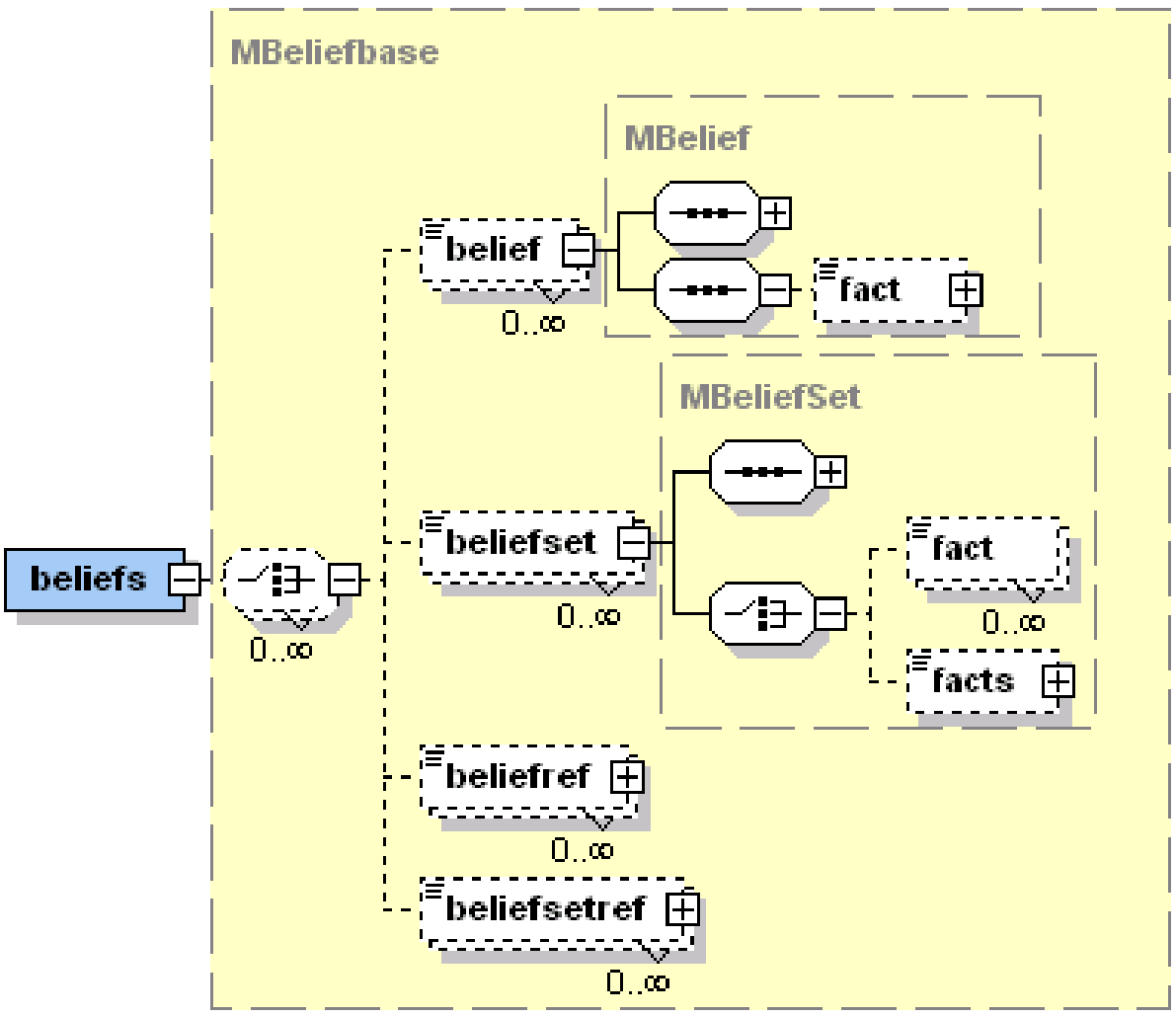
Jadex ADF

```
<agent
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation= "http://jadex.sourceforge.net/jadex
                     http://jadex.sourceforge.net/jadex-0.94.xsd"
name="Ping"
package="jadex.examples.ping"
properties="jadex.config.runtime" >
    <imports> ... <\imports>
    <capabilities> ... <\ capabilities >
    <beliefs> ... <\beliefs>
    <goals> ... <\goals>
    <plans> ... <\plans>
    <events> ... <\events>
    <expressions> ... <\ expressions>
    <properties> ... <\ properties >
    <initialistate> ... <\ initialistate >
</agent>
```

Jadex ADF (Cont.)

```
<imports>
    <import>java.util.HashMap</import>
    <import>java.awt.*</import>
    <import>movement.*</import>
</imports>
<capabilities>
    <capability name="movecap" file="Move"/>
</capabilities>
<beliefs>
    <belief name="data">
        <fact>new HashMap()</fact>
    </belief>
</beliefs>
```

Beliefs


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

Jadex ADF (Beliefs)

```
<agent ...>
    <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>
</agent>
```

Interface IBelief from Plans

- **getFact**() - Get the fact of a belief
- **setFact**(Object fact) - Set a fact of a belief
- **isAccessible**() - Is this belief accessable

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

Goal Creation

- Initial goals are created and adopted as top-level goals when the agent is born.

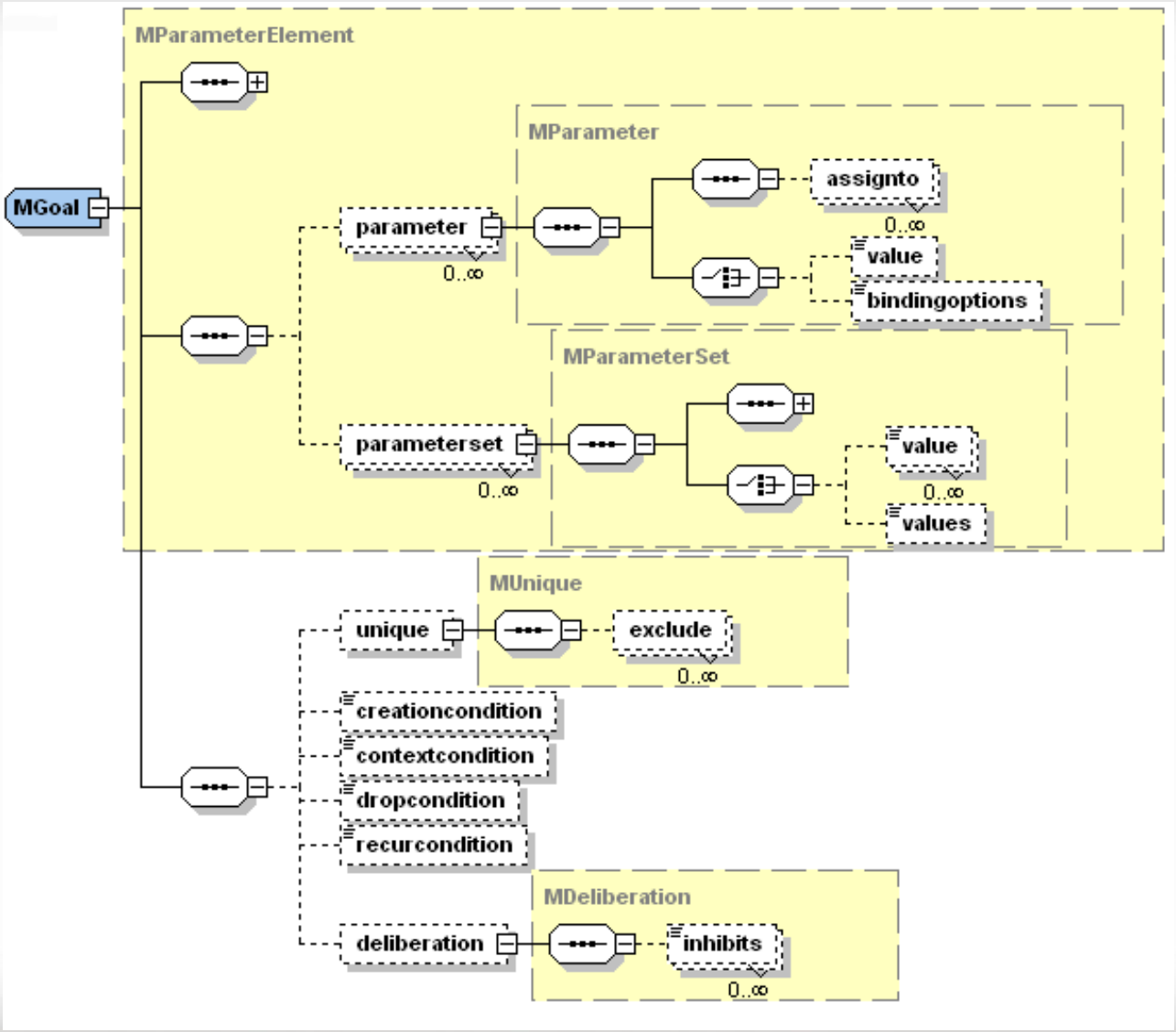
<initialgoal>

- When the creation condition triggers one or more goal instances are created and adopted as top-level goal(s).

<creationcondition>

- Plans may directly create goals and dispatch them as subgoals. These goals are adopted as subgoals of the plan's root goal. When a plan terminates or is aborted, all not yet finished subgoals are aborted automatically.
- Plans may also create goals and dispatch them as top-level goals. Once adopted, such a goal exists independently of the plan that created it.

The Jadex common goal features



Perform Goal

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

Achieve Goal

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

Query Goal

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

Maintain Goal

```
<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>
  <targetcondition language="jcl">
            $beliefbase.my_chargestate >= 1.0
  </targetcondition>
</maintaingoal>
```

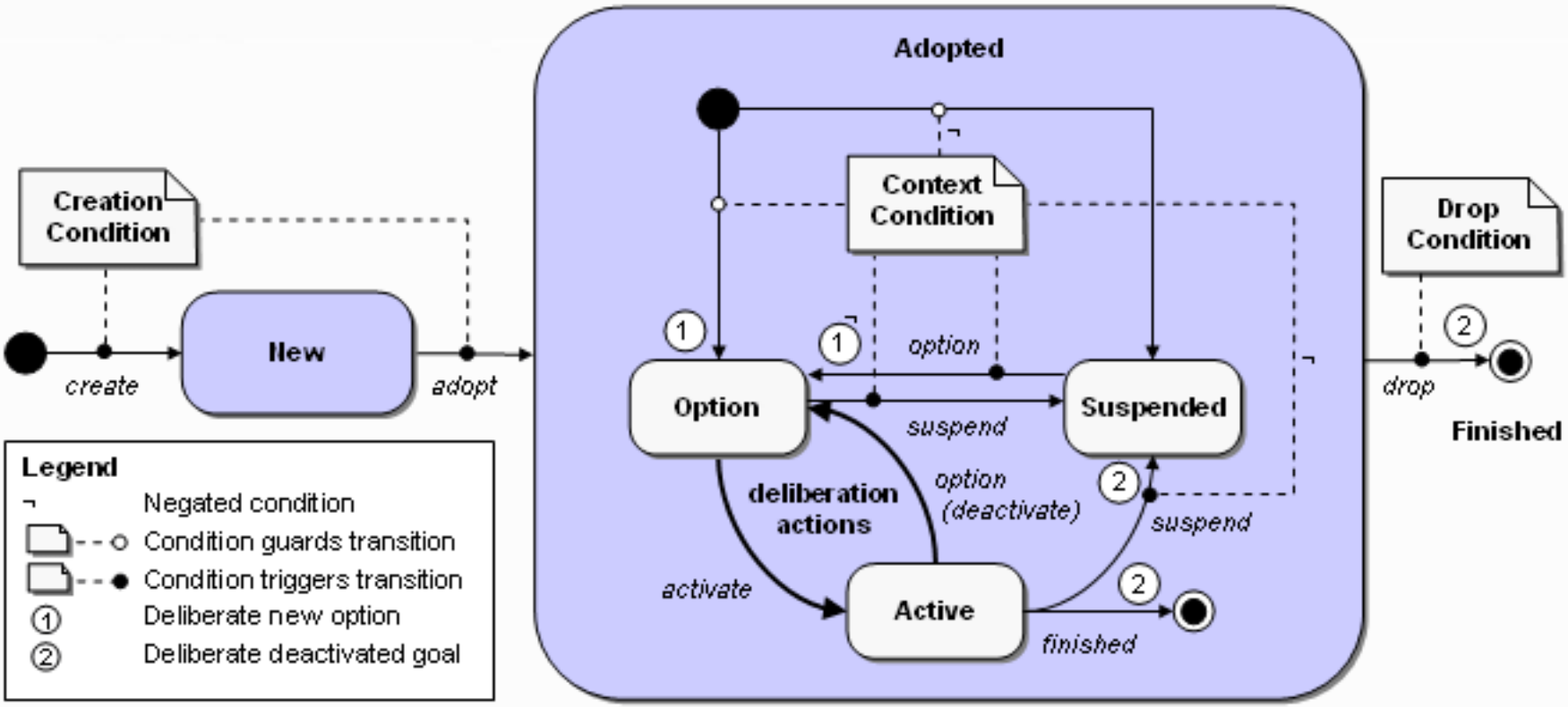
Common goal attributes (BDI flags)

Name	Default	Possible Values
retry	true	true false
retrydelay	0	positive long value
exclude	"when_tried"	"when_tried", "when_succeeded", "when_failed", "never"
posttoall	false	true false
randomselection	false	true false
metalevelreasoning	true	true false
recur	false	true false
recurdelay	0	positive long value

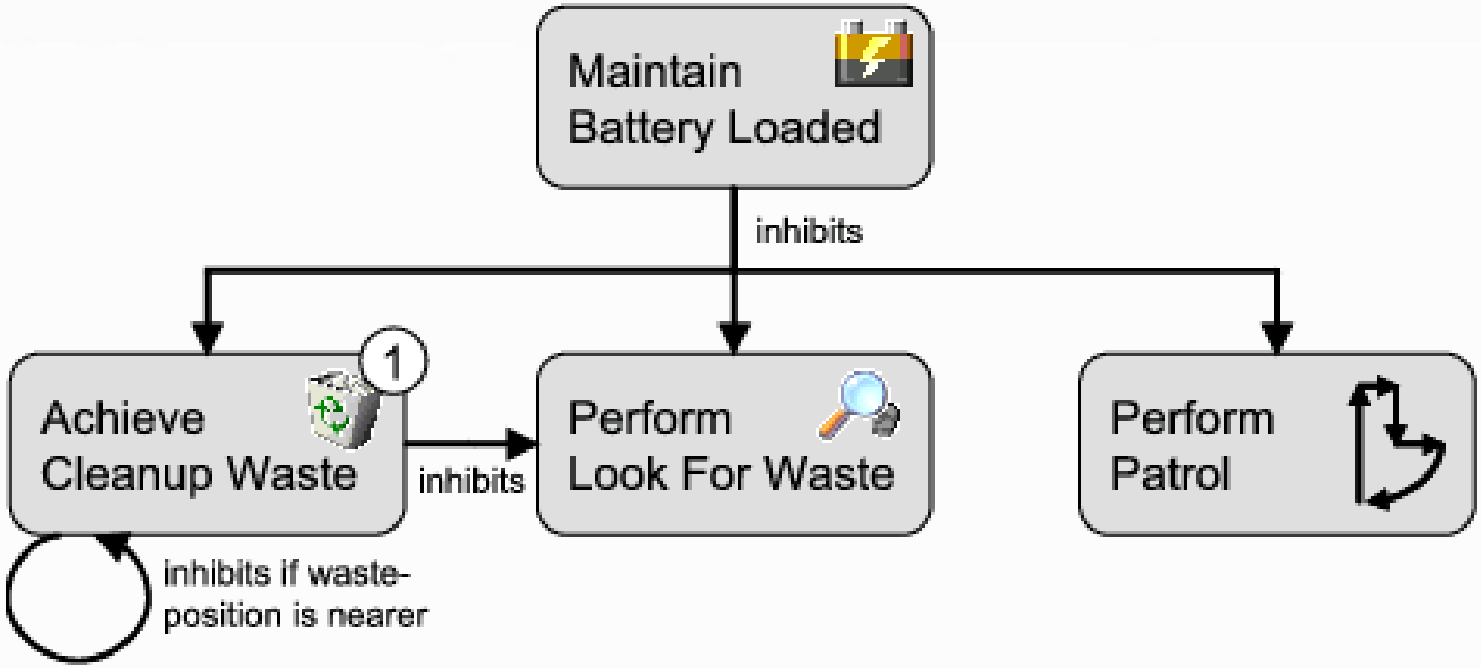
Dispatching goals from plan bodies

```
public void body() {
    // Create new top-level goal.
    IGoal goal1 = createGoal("mygoal");
    dispatchTopLevelGoal(goal1);
    ...
    // Create subgoal and wait for result.
    IGoal goal2 = createGoal("mygoal");
    dispatchSubgoalAndWait(goal2);
    Object val = goal2.getParameter("someoutparam").getValue();
    ...
    // Drop top-level goal.
    goal1.drop();
}
```

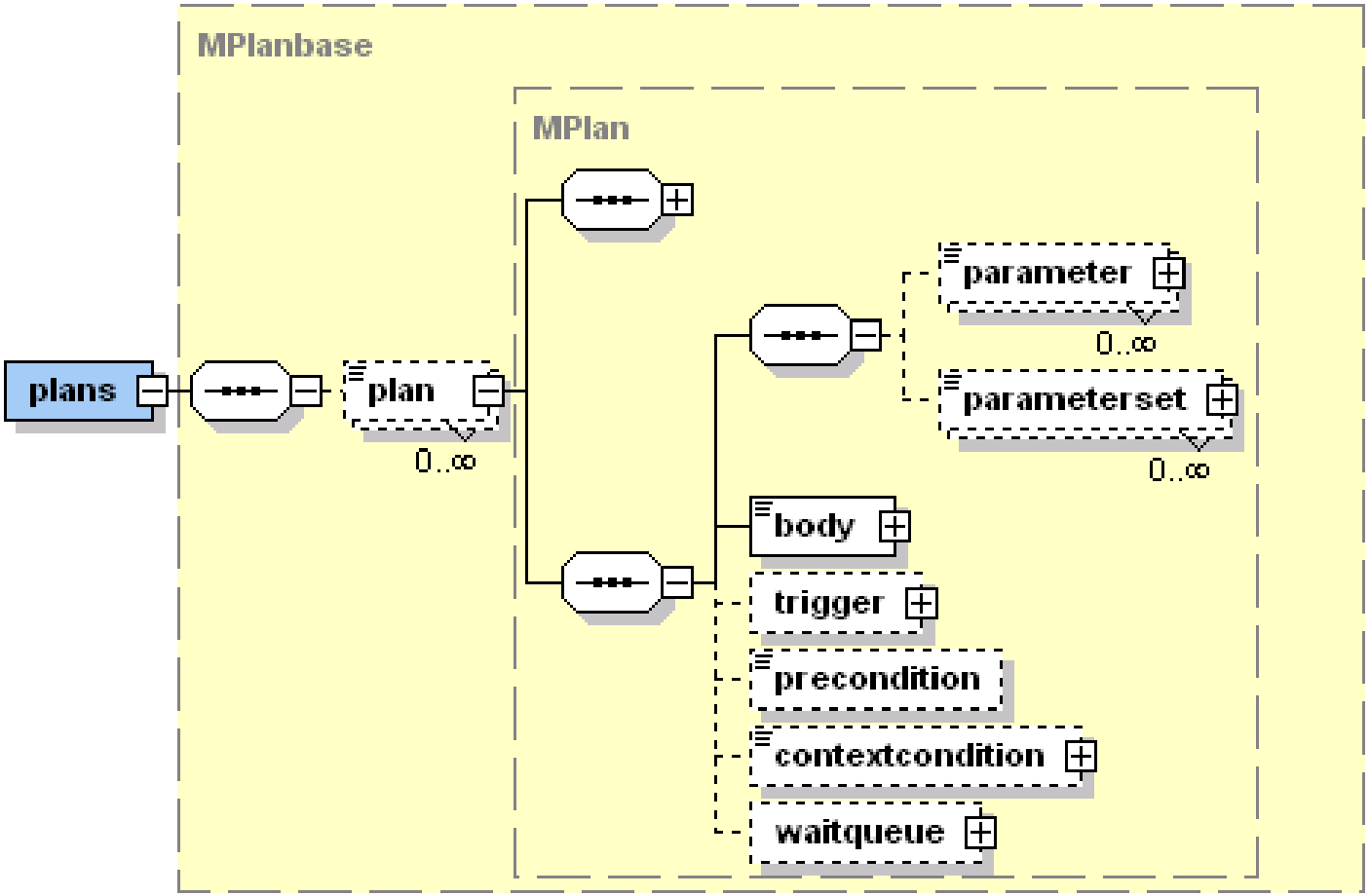
Easy deliberation strategy



Example goal dependencies



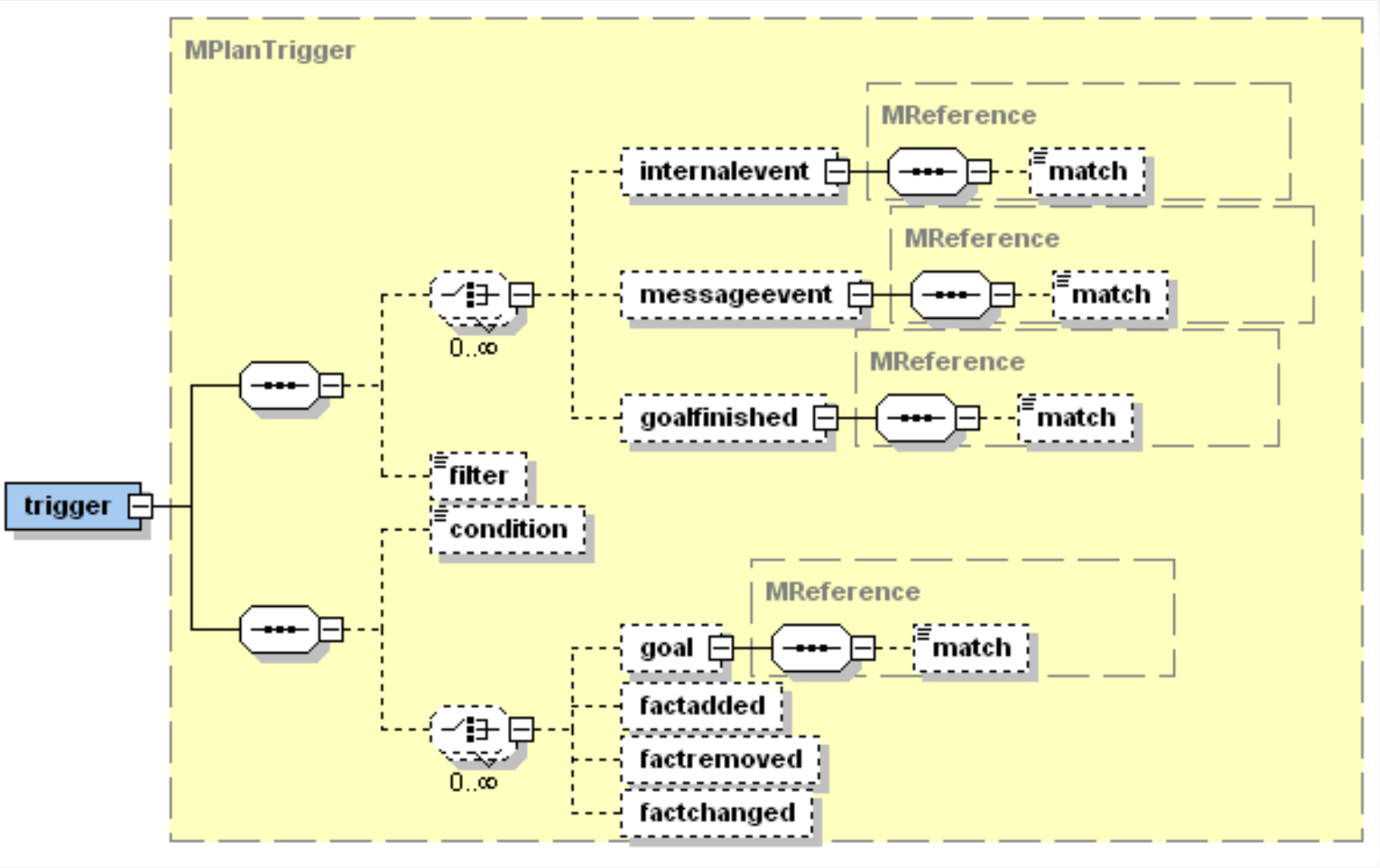
Plans


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

Jadex ADF (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>
```

Jadex plan trigger XML schema part



Plans (Cont.)

```
<plans>
 <plan name="repair">
  <body impl="RepairPlan"/>
  <trigger>
   <condition>$beliefbase.out_of_order</condition>
  </trigger>
  <contextcondition>$beliefbase.repairable</contextcondition>
 </plan>
</plans>
```

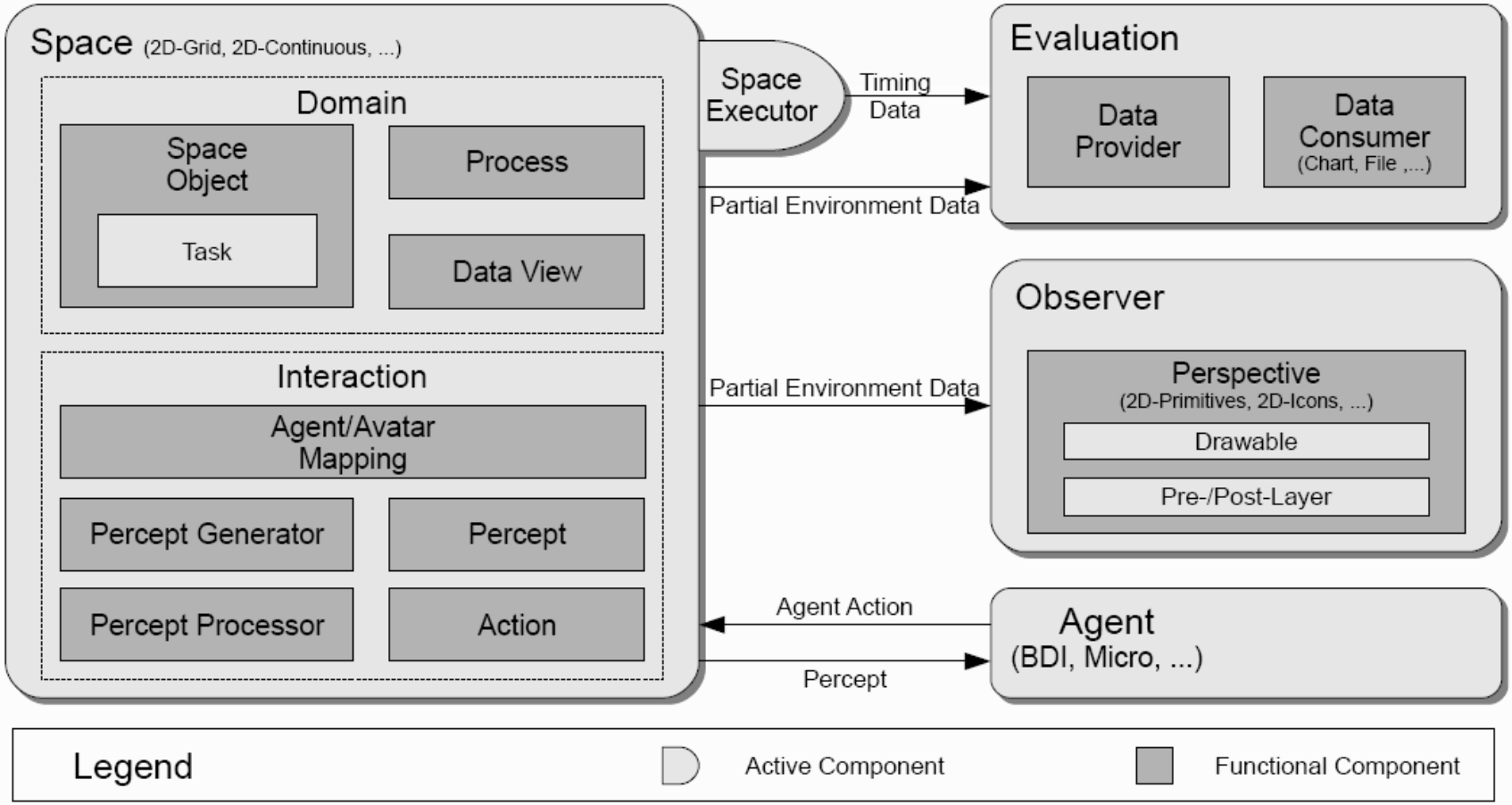
Plan (Cont.)

```
<plan name="move_plan">
 <parameter name="move" class="Move">
   <bindingoptions>$beliefbase.board.getPossibleMoves()</bindingoptions>
 </parameter>
  ...
</plan>
```

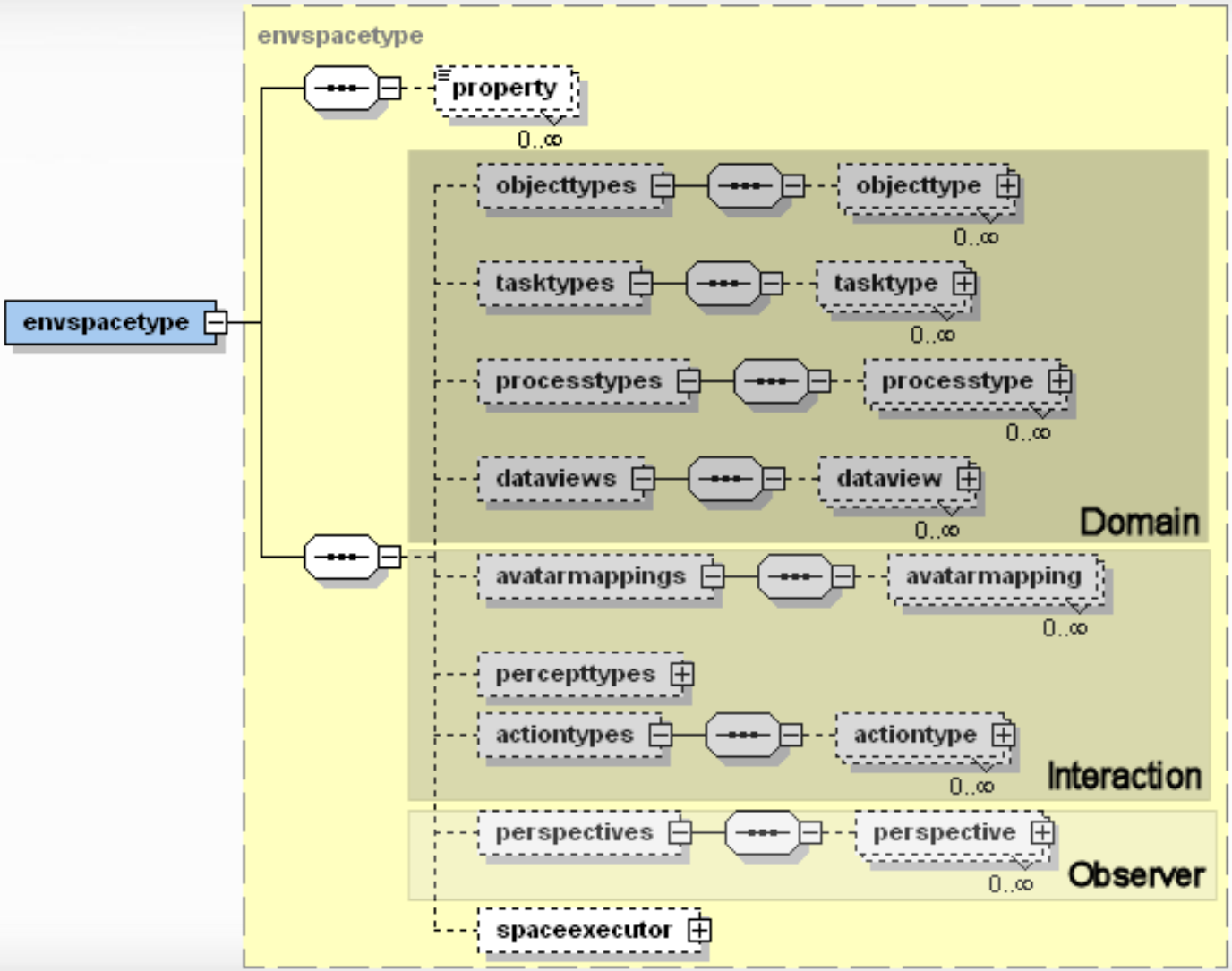
Plan Implementation

```
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 = ...;
      ...
 }
}
```

environment



XML schema part for the environment space type



Thank you for your attention.