Multi-Agent Programming

Jadex: A BDI Reasoning Engine

Mehdi Dastani

Thanks to Alexander Pokahr and Lars Braubach for providing slide materials
Background and Motivation

• Jadex is based on BDI model

• Integrate agent theories with object-orientation and XML descriptions

• Object-oriented representation of BDI concepts

• Explicit representation of goals allows reasoning about (manipulation of) goals

• Jadex is based on JADE Platform
Jadex Abstract Agent Architecture

Agent
- Reaction Deliberation
  - Handle Events

Capability
- Plans
  - Select Plans
  - Application Events
  - Dispatch (Sub-) Goals

Events
- Goal Events
- Condition Events

Beliefs
- Read/Write Facts

Goals
- Goal Conditions
- Dispatch (Sub-) Goals

Messages
Jadex Beliefs

- Beliefbase contains the knowledge of an agent
  - Beliefs (single facts stored as Java objects)
  - Beliefsets (sets of facts as Java Objects)

- No support for logical reasoning

- 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 Abstract Agent Architecture

- **Agent**
  - Reaction Deliberation
    - Handle Events
  - Capability
    - Select Plans
      - Plans
        - Read/Write Facts
        - Application Events
          - Dispatch (Sub-) Goals
            - Events
              - Goal Events
              - Condition Events
            - Goal Conditions
          - Goal
          - Beliefs
            - Handle Events

- **Messages**
Jadex Goals

• Generic goal types
  – perform (some action)
  – achieve (a specified world state)
  – query (some information)
  – maintain (reestablish a specified world state whenever violated)

• Are strongly typed with
  – name, type, parameters
  – BDI-flags enable non-default goal-processing

• Goal creation/deletion possibilities
  – initial goals for agents
  – goal creation/drop conditions for all goal kinds
  – top-level / subgoals from within plans
Jadex Abstract Agent Architecture

Agent

- Reaction Deliberation
  - Handle Events

Capability

- Select Plans

Plans

- Read/Write Facts
- Application Events
- Dispatch (Sub-) Goals
- Goal Events
- Condition Events

Events

Beliefs

- Goal Conditions

Goals
Plans

• Represent procedural knowledge
  – Means for goal achievement and reacting to events
  – Agent has library of pre-defined plans
  – Interleaved stepwise execution

• Realisation 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 Abstract Agent Architecture
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 Agent

- Message queue
  - Select message
    - Create event for message
    - Capabilities/eventbases
    - Event list
      - Select event
        - Find applicable candidates
        - Capabilities/planbases
          - Select candidates
            - Meta-level reasoning
              - Ready list
                - Select intention
                  - Scheduler
                    - Execute plan step
Agenda-based Execution Model

Legend
- Processed Action
- Current Action
- New (unprocessed) Action

External Action Sources
(e.g. received messages)

Interpreter Selection Strategy
Interpreter Insertion Strategy

Main Interpreter

Action Execution

Change Computation
Set of Changes
Consequence Determination

Add changes
Add related consequences
Add unrelated consequences

System changes
New agenda entries
New agenda entries
Basic Meta-Actions

Agent
  Init
    Initialize-state()

Action

Process Event
  Action

Execute
  Plan Step
    execute()

Find Applicable
  Candidates
    option-generator()

Select
  Candidates
    deliberate()

Schedule
  Candidates
    update-intentions()

Goal Finished
  drop-impossible-attitudes()
  drop-successful-attitudes()

Terminate
  Plan
    drop-impossible-attitudes()
Extended Meta-Actions

Action

Conditional Action

Create Goal

Maintain Goal State

Switch Goal Context

Create Plan

Update Belief

Update Intention

Schedule Candidates

Select Candidates

Find Applicable Candidates

Process Event Action

Execute Plan Step

Agent Init

Initialize-state()

Agent

Terminate Plan

drop-impossible-attitudes()

Goal Finished

drop-impossible-attitudes()

drop-successful-attitudes()

Deliberation Action

Recur/Retry Goal

Maintain Goal State

Timing Action

Deliberation

Timing

Execution

Timing

Update Belief

Conditional Action

Recruit Action

Deliberation Action
Jadex - current state

• Objective: Supporting the construction of open multi-agent systems by making use of mentalistic notions

• Is a BDI-extension (add-on) for the FIPA-compliant JADE multi-agent platform
• Supports easy agent construction with XML-based agent description and procedural plans in Java
• Supports reusability through the capability concept
• Offers toolsupport for debugging (in addition to the JADE tools)
  – BDI-Viewer allows to observe and modify the internal state
  – The BDI-Introspeccor allows to control the agent
  – The Logger agent collects log-outputs of any agents
• Objective: Supporting the construction of open multi-agent systems by making use of mentalistic notions.

• Is a BDI-extension for the FIPA-compliant JADE multi-agent platform.

• Supports easy agent construction with XML-based agent description and procedural plans.

• Supports reusability through the capability concept.

• Offers tool support for debugging (in addition to the JADE tools):
  – BDI-Viewer allows to observe and modify the internal state
  – The BDI-Introspector allows to control the agent
  – The Logger agent collects log-outputs of any agents
Jadex - current state

- Objective: Supporting the construction of open multi-agent systems by making use of mentalistic notions
- Is a BDI-extension (add-on) for the FIPA-compliant JADE multi-agent platform
- Supports easy agent construction with XML-based agent description and procedural plans
- Supports reusability through the capability concept
- Offers toolsupport for debugging (in addition to the JADE tools)
  - BDI-Viewer allows to observe and modify the internal state
  - The BDI-Introspector allows to control the agent
  - The Logger agent collects log-outputs of any agents

```java
public class MyPlan extends ThreadedPlan {
    public void body() {
        // ...
    }
}
```

```xml
<agent>
    <beliefs>
        <!-- ... -->
    </beliefs>
    <goals>
        <!-- ... -->
    </goals>
    <plans>
        <!-- ... -->
    </plans>
</agent>
```
- Objective: Supporting the construction of open multi-agent systems by making use of mentalistic notions
- Is a BDI-extension (add-on) for the FIPA-compliant JADE multi-agent platform
- Supports easy agent construction with XML-based agent description and procedural plans
- Supports reusability through the capability concept
- Offers tool support for debugging (in addition to the JADE tools)
  - BDI-Viewer allows to observe and modify the internal state
  - The BDI-Introspector allows to control the agent
  - The Logger agent collects log-outputs of any agents
Jadex - current state

- Objective: Supporting the construction of open multi-agent systems
- Is a BDI-extension (add-on) for the FIPA-compliant JADE multi-agent platform
- Supports easy agent construction with XML-based agent description and procedural plans
- Supports reusability through the capability concept
- Offers toolsupport for debugging (in addition to the JADE tools)
  - BDI-Viewer allows to observe and modify the internal state
  - The BDI-Introspector allows to control the agent
  - The Logger agent collects log-outputs of any agents
Jadex Abstract Agent Architecture
Agenda-based Execution Model

Legend
- Processed Action
- Current Action
- New (unprocessed) Action

External Action Sources
(e.g. received messages)

Agenda
- Action Execution
- Main Interpreter
- Condition Execution
- Change Computation
- Set of Changes
- Consequence Determination

Interpreter Selection Strategy
Interpreter Insertion Strategy
add changes
new agenda entries
add related consequences
add unrelated consequences
new agenda entries
system changes

Legend
- Processed Action
- Current Action
- New (unprocessed) Action

Diagram:
- Agenda
- Action Execution
- Main Interpreter
- Condition Execution
- Change Computation
- Set of Changes
- Consequence Determination

Legend:
- Processed Action
- Current Action
- New (unprocessed) Action
Components of a Jadex Agent

Agent Plattform

Jadex Agent

ADF

Plan

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

```java
public class PingPlan extends ThreadedPlan {
    public void body()
    {
        ...
    }
}
```
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>
Imports

<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>
<agent ...>

  <capabilities>

    <capability
      name="mysubcap"
      file="mypackage/MyCapability.capability.xml"/>

    <capability
      name="dfcap"
      file="jadex.planlib.DF"/>

  </capabilities>

</agent>
<capability

  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="MyCapability"

  package="mypackage">

  <beliefs> ... </beliefs>

  <goals> ... </goals>

  <plans> ... </plans>

</capability>
Using Capability

Outer Capability B

...<beliefref name="mysubbelief">
  <concrete ref="mysubcap.myexportedbelief"/>
</beliefref>
...

Inner Capability A

...

...<belief
  name="myexportedbelief"
    type="exported"
    class="MyFact"/>
</belief>
...

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>
Beliefs

Dynamically Evaluated Beliefs

<beliefs>
  <belief name="time" class="long">
    <fact evaluationmode="dynamic">
      System.currentTimeMillis()
    </fact>
  </belief>
  <belief name="timer" class="long" updaterate="10000">
    <fact>
      System.currentTimeMillis()
    </fact>
  </belief>
</beliefs>
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

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

Legend
¬ Negated condition
- Condition guards transition
○ Condition triggers transition

Creation Condition -> New
create

Option -> Adopted
activate

Active
suspend

Suspended
suspend

Context Condition

Drop Condition

drop

Finished
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.
Goal Lifecycle

Context-Condition indicates when Active/Option goal should be suspended

 Drop-Condition indicates when adopted goals should be dropped

 Goal-Deliberation indicates which Option goals should be (de)activated (inhibition and cardinality)
<achievegoal name="achievecleanup"
    retry="true"
    exclude="when_failed">

    <parameter name="waste" class="Waste" />

    <deliberation cardinality="1">
        <inhibits ref="performlookforwaste"/>
        <inhibits ref="achievecleanup">
            $beliefbase.my_location.getDistance($goal.waste.getLocation()) < $beliefbase.my_location.getDistance($ref.waste.getLocation())
        </inhibits>
    </deliberation>
</achievegoal>
Goal Flags

Retry (true,false): the goal should be retried or redone, until it is reached, or no more plans are available, which can handle the goal. Default=true

retrydelay (positive long value). Default=0

Exclude (when Tried, when succeeded, when failed, never): used in conjunction with retry; when retrying a goal, only plans should be called, that where not already executed for that goal. Default=when Tried

Posttoall (true,false): enables parallel processing of a goal, by dispatching the goal to all applicable plans at once. Default=false
Perform Goals

Patrol around and observe the environment implemented as perform goal

<performgoal name="performpatrol" retry="true">
  <contextcondition>
    !$beliefbase.is_cleaning && !$beliefbase.is_loading
  </contextcondition>
</performgoal>

$beliefbase is a reference to the beliefbase
is_cleaning / is_loading are beliefs
a patrol plan exists that is sensible to this goal

Agent has an initial perform patrol goal:
<initialgoal ref="performpatrol"> </initialgoal>
Achievement Goals

Clean-up waste and bring it to the waste-bin

<achievegoal name="achievecleanup" retry="true" exclude="never">

<creationcondition>
  $beliefbase.waste_location!=null && ($beliefbase.my_location.getDistance($beliefbase.waste_location) < $beliefbase.vision)
</creationcondition>

<contextcondition>
  !$beliefbase.is_loading
</contextcondition>
</achievegoal>

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

Keep-Operational (keep track of the battery state and move to the charge station when necessary)

<maintaingoal name="maintainbatteryloaded">
  <maintaincondition>
    $beliefbase.chargestate<0.2
  </maintaincondition>
  <targetcondition>
    $beliefbase.chargestate>=1.0
  </targetcondition>
</maintaingoal>

The agent has an initial maintain battery loaded goal:

<initialgoal ref="maintainbatteryloaded"></initialgoal>
<goals>

<!-- The goal to make a move on the board towards the solution. -->
<achievegoal name="makemove">
  <parameter name="depth" class="int">
    <value>0</value>
  </parameter>
  <targetcondition>$beliefbase.board.isSolution()$</targetcondition>
</achievegoal>

<!-- The meta-level goal for choosing between plans for the makemove goal. -->
<metagoal name="choosemove">
  <parameterset name="applicables" class="ICandidatetInfo"/>
  <parameterset name="result" class="ICandidatetInfo" direction="out"/>
  <trigger>
    <goal ref="makemove"/>
  </trigger>
</metagoal>
</goals>
Plans for Meta Goals

<plans>
<!-- Make a move on the board and create a subgoal for the next move. -->
<plan name="move_plan">
  <bindings>
    <binding name="move">$beliefbase.board.getPossibleMoves()</binding>
  </bindings>
  <body>new MovePlan($move)</body>
  <trigger>
    <goal ref="makemove"/>
  </trigger>
</plan>

<!-- Meta-level plan used to choose among the possible move plans. -->
<plan name="choose_move_plan">
  <parameterset name="applicables" class="ICandidateInfo">
    <goalmapping ref="choosemove.applicables"/>
  </parameterset>
  <parameterset name="result" class="ICandidateInfo" direction="out">
    <goalmapping ref="choosemove.result"/>
  </parameterset>
  <body>new ChooseMovePlan()</body>
  <trigger>
    <goal ref="choosemove"/>
  </trigger>
</plan>
</plans>
Interface IGoal

public void drop()
Drop this goal. Causes all associated process goals and subgoals to be dropped.

public boolean isFinished()
Test if a goal is finished. Returns true, if goal is finished.

public boolean isActive()
Get the activation state. Returns true, if the goal is active.

public boolean isRetry()
Get the retry flag.

public void setRetry(boolean flag)
Set the retry flag.
Interface IGoalbase

`containsGoal(IGoal goal)`
Test if an adopted goal is already contained in the goal base.

`createGoal(String type)`
Create a goal from a template goal.

`dispatchTopLevelGoal(IGoal goal)`
Dispatch a new top-level goal.

`getGoal(String name)`
Get an adopted goal by name.

`getGoals()`
Get all the adopted goals in this scope (including subgoals).

`getGoals(String type)`
Get all goals of a specified type (=model element name).
### Plans

Create plan instance when a message arrives (plan pre-condition)

```xml
<plans>
  <plan name="ping">
    <body>new PingPlan()</body>
    <trigger>
      <messageevent ref="query_ping"/>
    </trigger>
  </plan>
</plans>

... 

<events>
  <messageevent name="query_ping">
    ...
    ...
  </messageevent>
</events>
```
Create plan instance when a goal is adopted

<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>
</plans>
Plans

Context condition is evaluated before & during the execution of plans. When context condition is violated, the plan is aborted and the plan had failed.

<plans>
  <plan name="repair">
    <body> new RepairPlan() </body>
    <trigger>
      <condition> $beliefbase.out_of_order </condition>
    </trigger>
    <contextcondition>
      $beliefbase.repairable
    </contextcondition>
  </plan>
</plans>
public class SomePlan extends jadex.runtime.Plan

public void body()
{Plan code}

public void passed()
{Optional cleanup code in case of a plan success}

public void failed()
{Optional cleanup code in case of a plan failure}

public void aborted()
{Optional cleanup code in case the plan is aborted}
Internal Events

<!-- Specifies an internal event for updating the gui.-->

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

public void body() {
    String update_info;

    IInternalEvent event = createInternalEvent("gui_update");
    event.getParameter("content").setValue(update_info);
    dispatchInternalEvent(event);
}

<events>
    <messageevent name="request_carry"
        type="fipa"
        direction="send">
        <parameter name="performative"
            class="String"
            direction="fixed">
            <value>SFipa.REQUEST</value>
        </parameter>
        <parameter name="language"
            class="String"
            direction="fixed">
            <value>SFipa.JAVA_XML</value>
        </parameter>
    </messageevent>
</events>
Message Event (2)

<parameter name="ontology"
   class="String"
   direction="fixed">
   <value>
      MarsOntology.ONTOLOGY_NAME
   </value>
</parameter>

<parameter name="reply-with" class="String">
   <value>
      SFipa.createUniqueld($scope.getAgentName())
   </value>
</parameter>
public void body() {
    jadex.adapter.fipa.AgentIdentifier receiver;
    CarryRequest action;
    IMessageEvent me = createMessageEvent("request_carry");
    me.getParameterSet(jadex.adapter.fipa.SFipa.RECEIVERS).
        addValue(receiver);
    me.setContent(action);
    IMessageEvent reply = sendMessageAndWait(me);
}

Message Event (3)
Expressions

<agent ...>
...
  <expressions>
   <expression name="find_person" class="Person">
     select one Person $person from $person in $beliefbase.persons
     where $person.getSurname().equals($surname)

     <parameter name="$surname" class="String"/>
   </expression>

...<expressions>
...
</agent>

public void body {
  IExpression query = getExpression("find_person");
  ...
  Person person = (Person)query.execute("$surname", "Miller");
  ...
}
Initial state
Initial Beliefs

<agent ...>
  ...
  <initialstates>
    <initialstate name="one">
      <beliefs>
        <initialbelief ref="name">
          <fact>"John"</fact>
        </initialbelief>
        <initialbeliefset ref="names">
          <facts>DB.queryNames()</facts>
        </initialbeliefset>
      </beliefs>
    </initialstate>
  </initialstates>
</agent>
Initial Plan

<agent ...>

...<plans>

<plan name="print_hello">

<body>

new PrintOnConsolePlan("Hello World!")

</body>

</plan>

</plans>

...

<initialstates>

<initialstate name="one">

<plans>

<initialplan ref="print_hello"/>

</plans>

</initialstate>

</initialstates>

</agent>