Jadex

An agent framework

Manuel Mittler

Research lab – Summer term 2014

University Koblenz-Landau

Outline

- Introduction (Jadex)
- Java agent development framework
- Jadex architecture
- Jadex components (Beliefs, Goals, Plans, Events, ADF)
- Execution model
- Jadex Control Center
- Summary

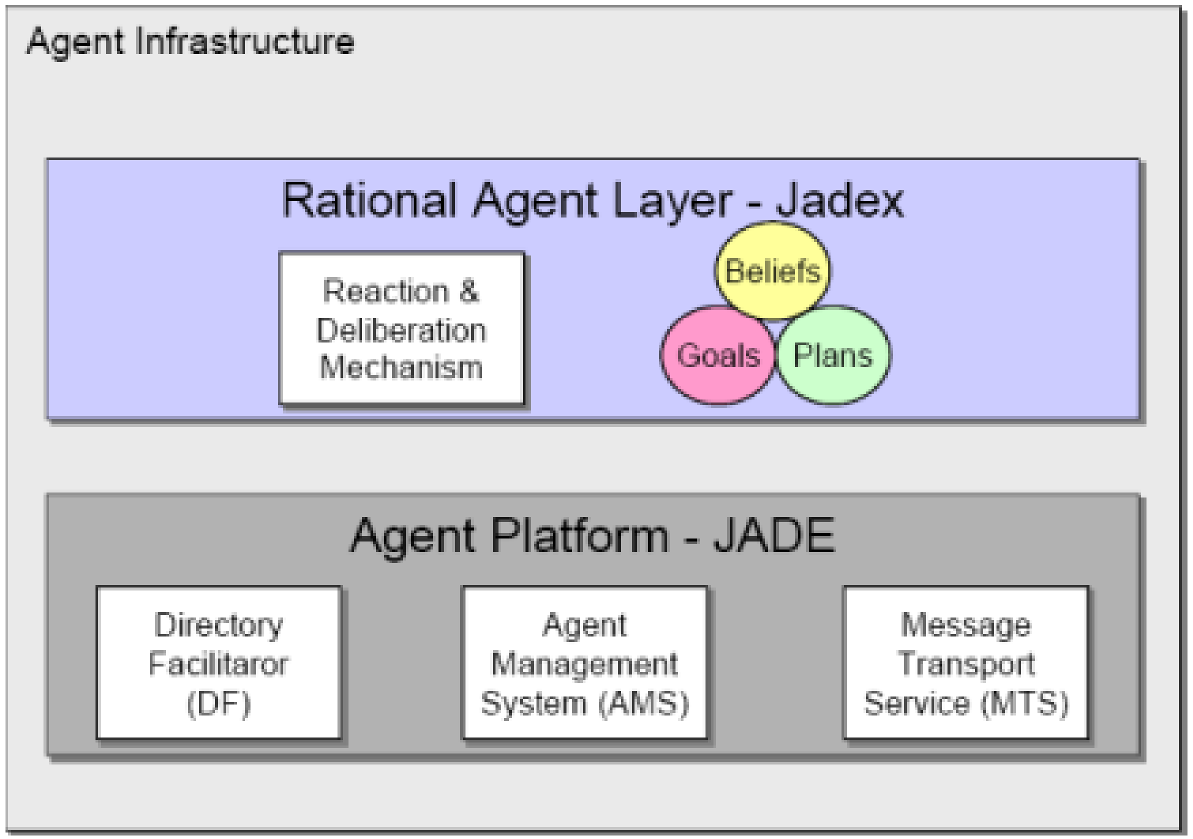
Jadex – Intro

- Software framework for the creation of goal-oriented agents following the belief-desire-intention (BDI) model.
- Aim: bring together middleware and reasoning-centered agent platforms
- A rational reasoning engine that sits on top of a middleware (Jade) and allows intelligent agent construction

Jadex – Intro

- 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

Infrastructure


Agent Infrastructure
Rational Agent Layer - Jadex
Reaction & Deliberation Mechanism
Beliefs
Goals
Plans
Agent Platform - JADE
Directory Facilitaror (DF)
Agent Management System (AMS)
Message Transport Service (MTS)

Java agent development framework (JADE)

- Java framework
  - communication infrastructure
  - platform services such as agent management
  - set of development and debugging tools.

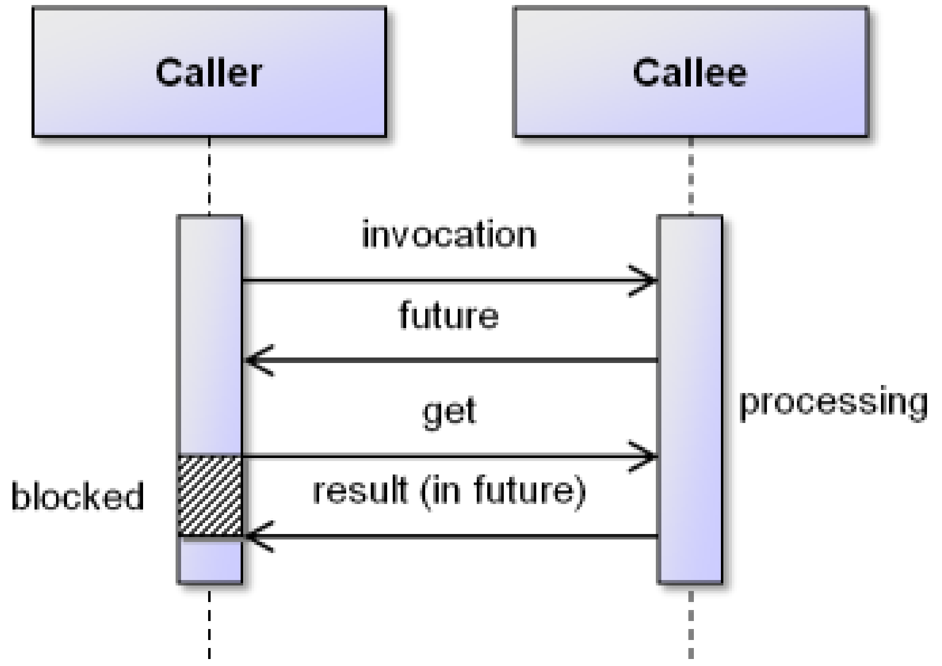
- Development and execution of peer-to-peer applications which are based on the agent paradigm(autonomous, proactive, social).

JADE

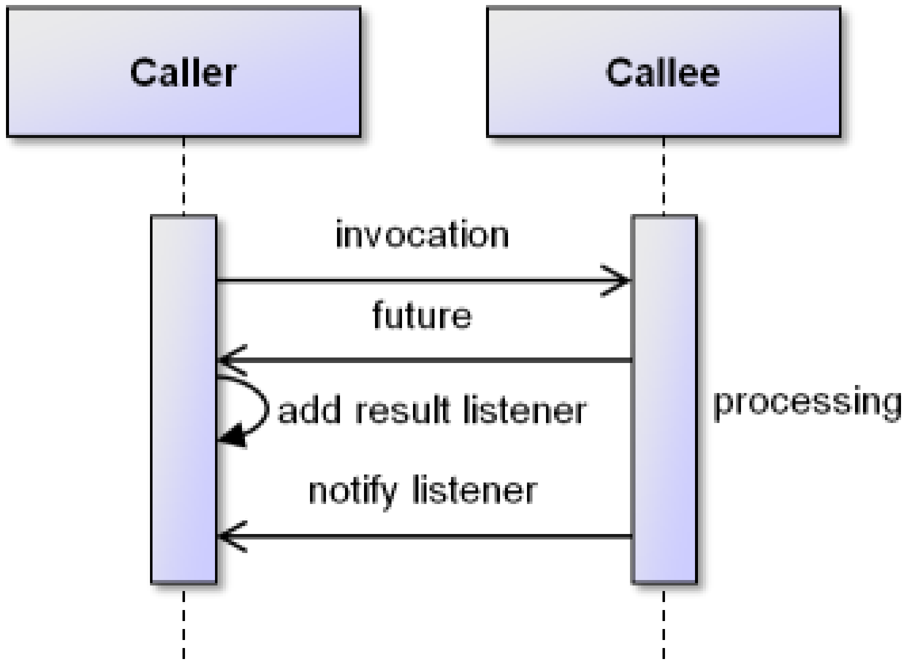
- Agents:
    - are identified by a unique name and provide a set of services. They can register and modify their services and/or search for agents providing given services (white and yellow pages)
    - can control their life cycle
    - Can dynamically discover other agents and communicate with them: exchange asynchronous messages (Agent communication language (ACL))

Asynchronous call concepts

jadex.commons.future.IFuture

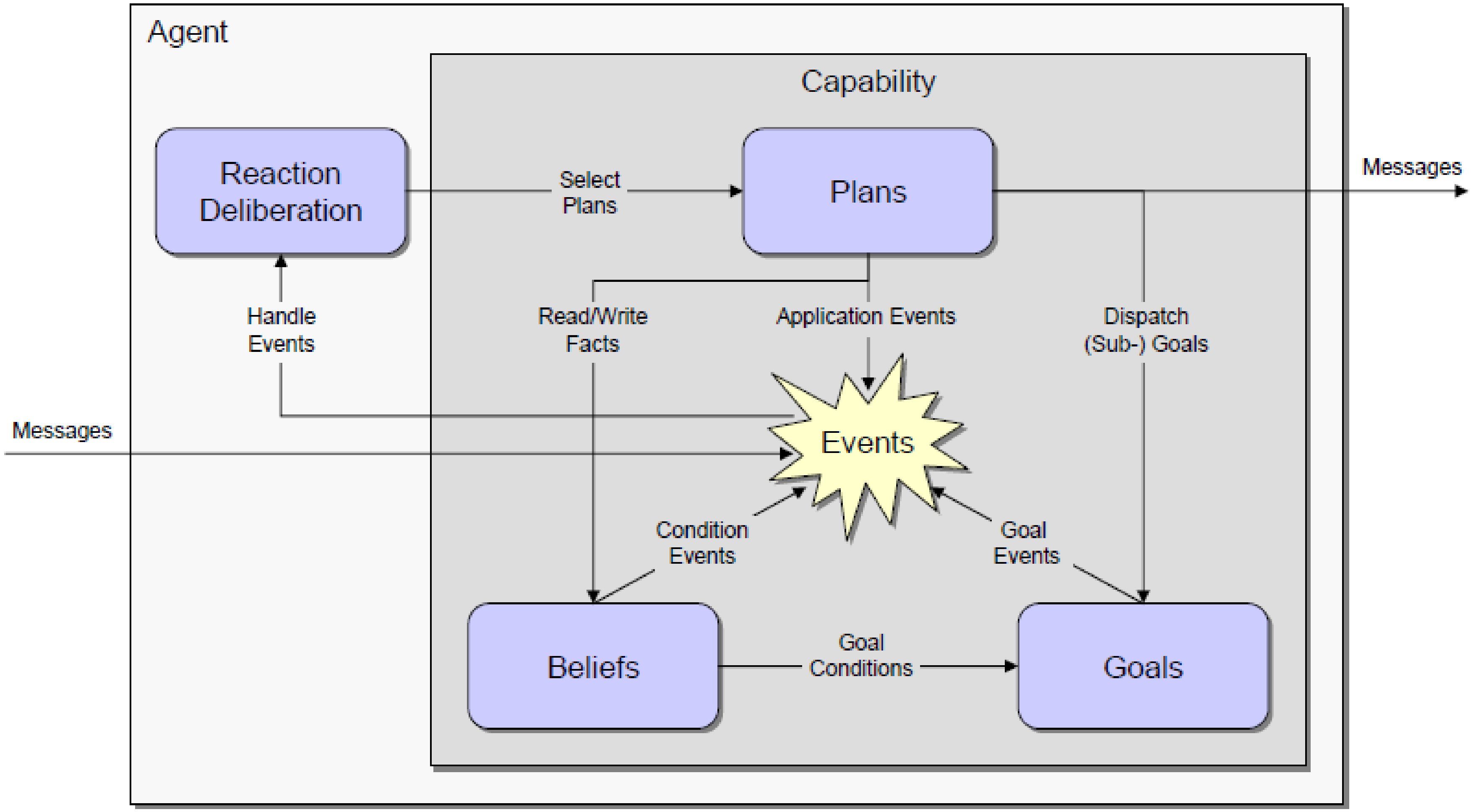


a) Wait by necessity



b) Listener notification

Jadex abstract agent architecture



Jadex beliefs

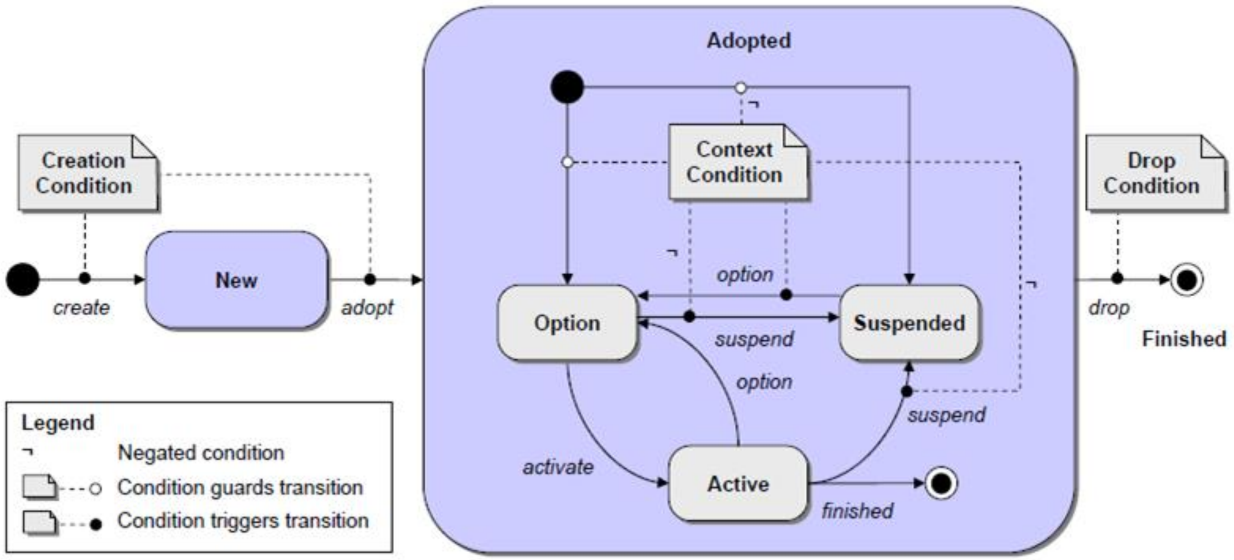
- Belief base contains the knowledge of an agent
  - Beliefs (single facts stored as Java objects)
  - Beliefsets (sets of facts as Java Objects)
  - key / value pairs
- Advantages of storing information as facts
  - Central place for knowledge (accessible to all plans)
  - Allows queries over the agent‵s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

Jadex goals

- Momentary desires of an agent -> agent engages into suitable actions
- Generic goal types
  - **perform** (some action)
  - **achieve** (a specified world state)
  - **query** (some information)
  - **maintain** (re-establish a specified world state whenever violated)
- Goal creation/deletion possibilities
  - initial goals for agents
  - goal creation/drop conditions for all goal kinds
  - top-level / sub-goals from within plans

Goal lifecycle

- To distinguish between just adopted and actively pursued goals, a goal lifecycle is introduced which consists of the goal states option, active, and suspended.



Jadex plans

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

- Realization of a plan
  - Plan head specified in ADF
  - Plan body coded in pure Java

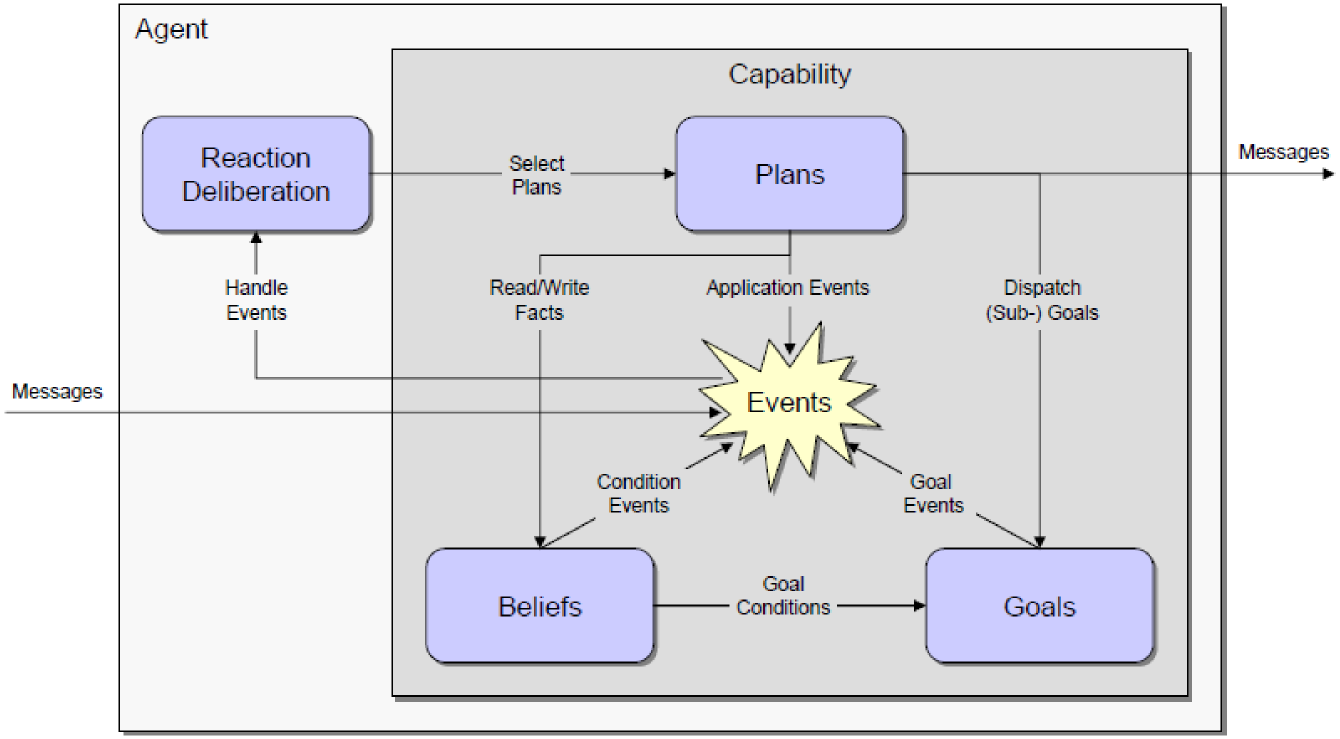
- Assigning plans to goals/events
  - Plan head indicates ability to handle goals/events
  - Plan context / precondition further refines set of applicable plans

Jadex events

- Three types of events:
  - **Message event** denotes arrival/Sending messages
  - **Goal event** denotes a new goal to be processed or that the state of an existing goal is changed
  - **Internal event**
    - **Timeout** event denotes that 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 reasoning, e.g. plans with triggering condition
    - **Condition-triggered** event is generated when a state change occurs that satisfies the trigger of a condition

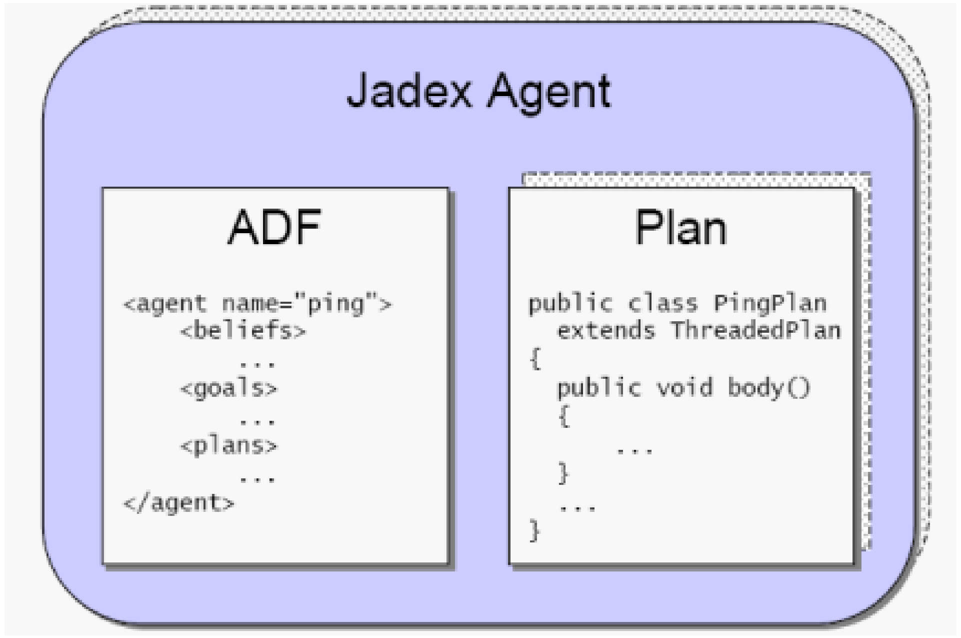
Jadex abstract agent architecture

- Capabilities:
  - grouping mechanism for the elements of a BDI agent, such as beliefs, goals, plans, and events.
  - closely related elements can be put together into a reusable module, which encapsulates a certain functionality



Components of a Jadex agent

- ADF defines agent startup properties:
  - Initial goals and beliefs
  - Head of plans



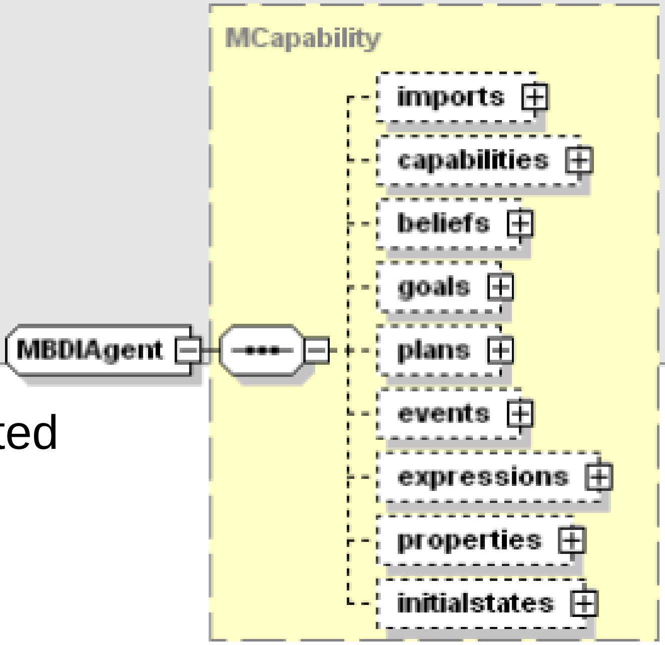
```
<agent name="ping">
    <beliefs>
        ...
    <goals>
        ...
    <plans>
        ...
</agent>
```

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

- Plan body provides predefined course of action
  - Is executed when the plan is selected
  - May contain actions provided by the system API (Sending messages, manipulating beliefs, creating sub-goals)

Agent Definition File

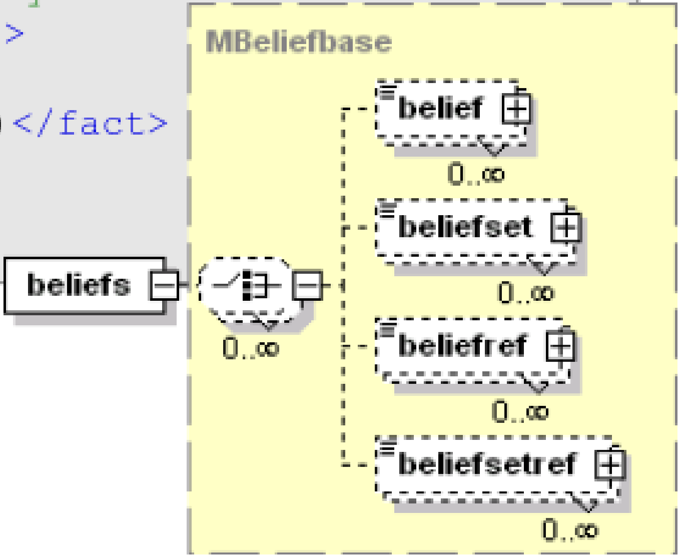
```
<agent xmlns="http://jadex.sourceforge.net/jadex"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://jadex.sourceforge.net/jadex
                        http://jadex.sourceforge.net/jadex-0.95.xsd"
    name="..." package="...">

    <imports>...</imports>
    <capabilities>...</capabilities>
    <beliefs>...</beliefs>
    <goals>...</goals>
    <plans>...</plans>
    <events>...</events>
    <expressions></expressions>
    <properties>...</properties>
    <initialstates>...</initialstates>

</agent>
```



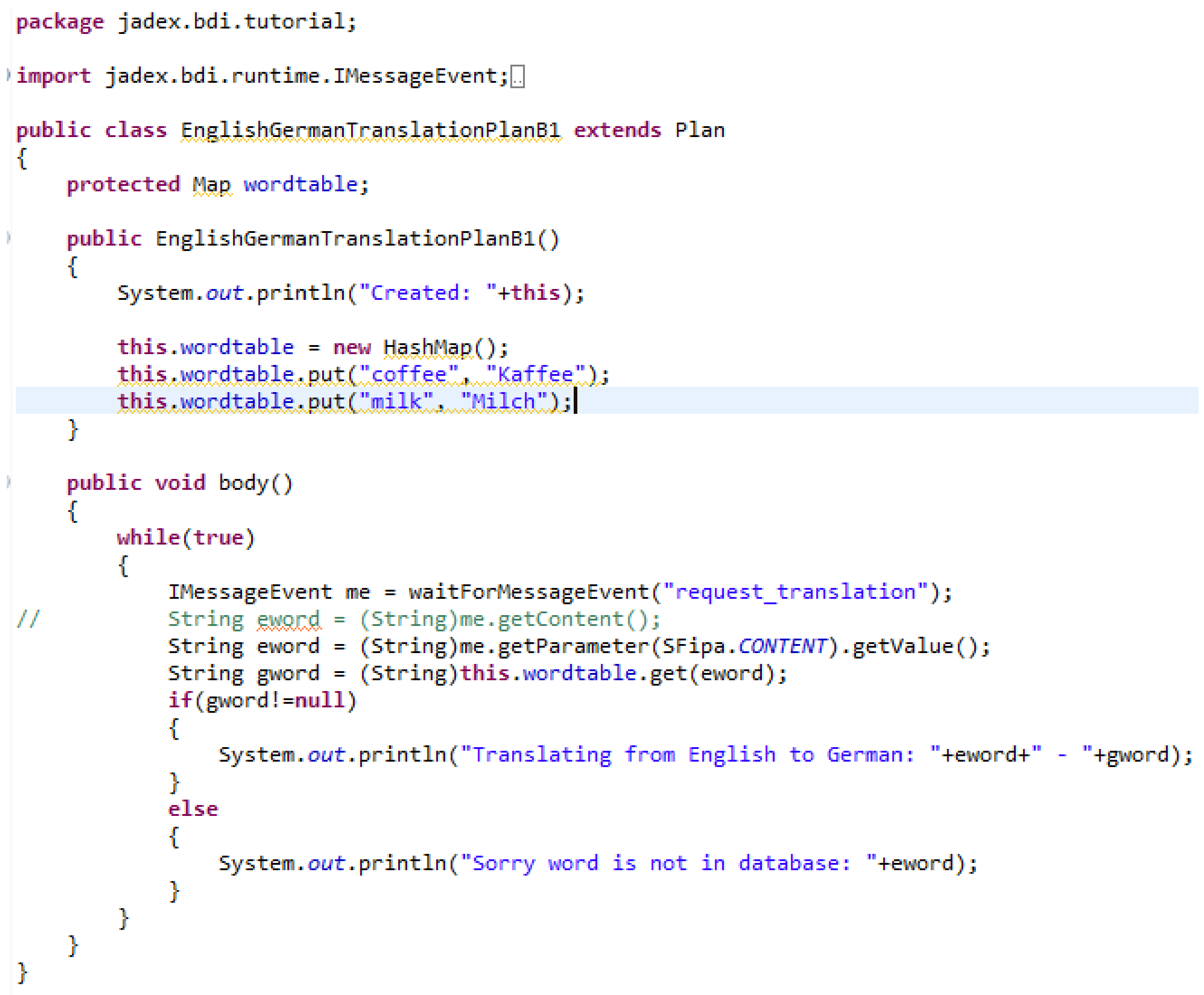
- When an ADF is loaded: Java objects are created for the XML elements defined in the ADF, e.g.:
  - Belief -> jadex.model.IMBelief
  - Goal- >jadex.model.IMGoal
  - Plan -> jadex.model.IMPlan

ADF - Beliefs

```
<beliefs>
    <!-- The patient (of age of 90), this Nurse takes care about. -->
    <belief name="my_patient" class="Patient">
        <fact>new Patient(90)</fact>
    </belief>

    <!-- Patient's blood pressure updated every 0.5 second. -->
    <belief name="pressure" class="int" updaterate="500">
        <fact>$beliefbase.my_patient.getPressure()</fact>
    </belief>

    <!-- Is patient alive flag, updated every time accessed. -->
    <belief name="is_alive" class="boolean">
        <fact evaluationmode="dynamic">
            $beliefbase.my_patient.isAlive()</fact>
    </belief>

</beliefs>
```

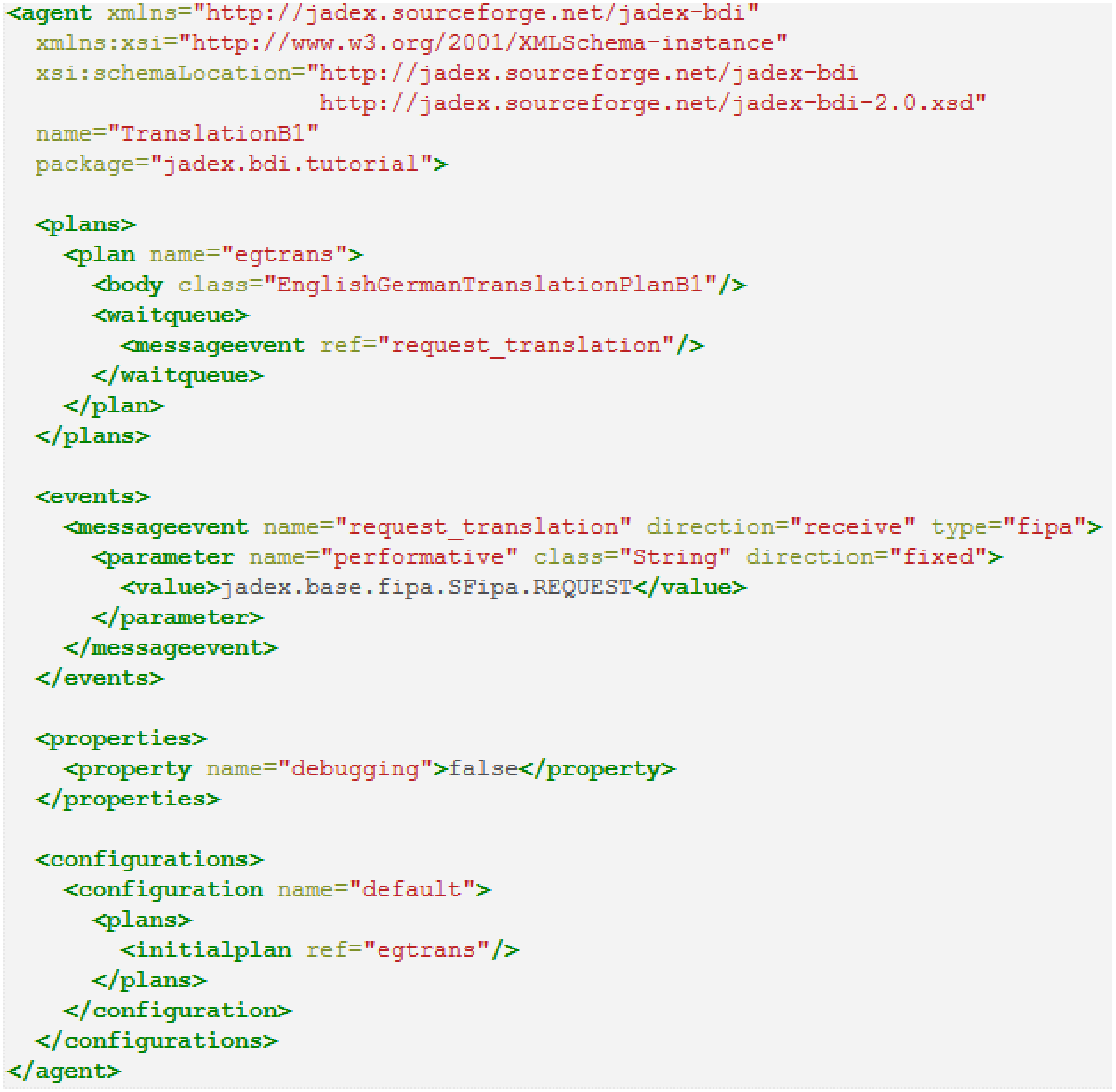


Service plan body



```java
package jadex.bdi.tutorial;

import jadex.bdi.runtime.IMessageEvent;

public class EnglishGermanTranslationPlanB1 extends Plan
{
    protected Map wordtable;

    public EnglishGermanTranslationPlanB1()
    {
        System.out.println("Created: "+this);

        this.wordtable = new HashMap();
        this.wordtable.put("coffee", "Kaffee");
        this.wordtable.put("milk", "Milch");
    }

    public void body()
    {
        while(true)
        {
            IMessageEvent me = waitForMessageEvent("request_translation");
//          String eword = (String)me.getContent();
            String eword = (String)me.getParameter(SFipa.CONTENT).getValue();
            String gword = (String)this.wordtable.get(eword);
            if(gword!=null)
            {
                System.out.println("Translating from English to German: "+eword+" - "+gword);
            }
            else
            {
                System.out.println("Sorry word is not in database: "+eword);
            }
        }
    }
}
```

ADF service plan

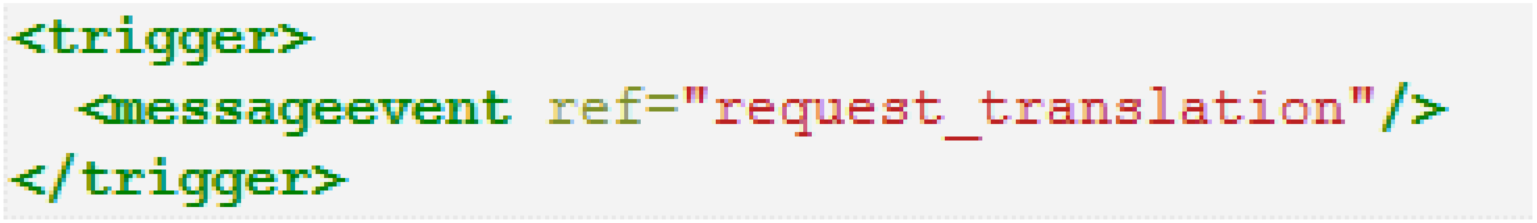


```xml
<agent xmlns="http://jadex.sourceforge.net/jadex-bdi"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://jadex.sourceforge.net/jadex-bdi
                      http://jadex.sourceforge.net/jadex-bdi-2.0.xsd"
  name="TranslationB1"
  package="jadex.bdi.tutorial">

  <plans>
    <plan name="egtrans">
      <body class="EnglishGermanTranslationPlanB1"/>
      <waitqueue>
        <messageevent ref="request_translation"/>
      </waitqueue>
    </plan>
  </plans>

  <events>
    <messageevent name="request_translation" direction="receive" type="fipa">
      <parameter name="performative" class="String" direction="fixed">
        <value>jadex.base.fipa.SFipa.REQUEST</value>
      </parameter>
    </messageevent>
  </events>

  <properties>
    <property name="debugging">false</property>
  </properties>

  <configurations>
    <configuration name="default">
      <plans>
        <initialplan ref="egtrans"/>
      </plans>
    </configuration>
  </configurations>
</agent>
```

ADF – passive plans

- Body method is only invoked when an event matches the plan's trigger.

- For each event a new plan instance is created, which only handles a single message.



```
<trigger>
  <messageevent ref="request_translation"/>
</trigger>
```

Access to beliefs from plans

- Methods:
  - getFact() – get the fact of a belief
  - setFact(Object fact) – set a fact of a belief
  - isAccessible() – is the belief accessible
- Example:

```
Map<String, String> words = (Map<String, String>)
getBeliefbase().getBelief("egwords").getFact();

getBeliefbase().getBelief("egwords").setFact(words);
```

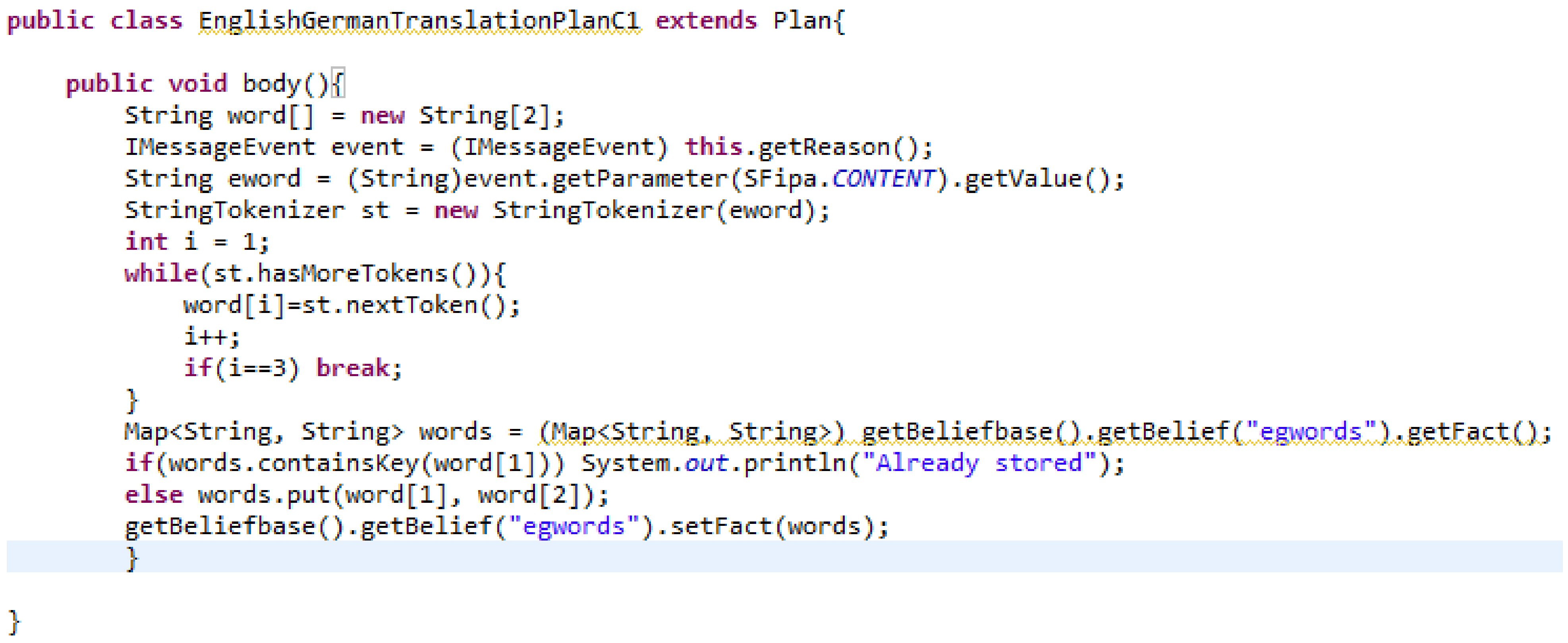
Initial belief

- Plan "addword" for adding a word pair to the database

```
<agent xmlns="http://jadex.sourceforge.net/jadex"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://jadex.sourceforge.net/jadex
                           http://jadex.sourceforge.net/jadex-bdi-2.3.xsd"
  name="TranslationC1"
  package="jadex.bdi.tutorial">

  <imports>
    <import>java.util.logging.*</import>
    <import>java.util.*</import>
    <import>jadex.bridge.fipa.*</import>
  </imports>

  <beliefs>
    <belief name="egwords" class="Map">
      <fact>EnglishGermanTranslationPlanC1.getDictionary()</fact>
    </belief>
  </beliefs>

  <plans>
    <plan name="addword">
      <body class="EnglishGermanAddWordPlanC1"/>
      <trigger>
        <messageevent ref="request_addword"/>
      </trigger>
    </plan>
```

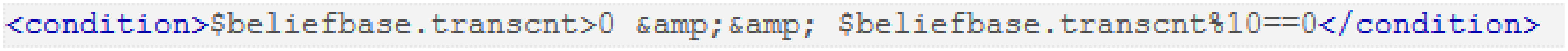
```
public class EnglishGermanTranslationPlanC1 extends Plan{

     .
     .
     .
     .
     protected static Map<String, String> dictionary;

     public static Map<String, String> getDictionary()
     {
       if(dictionary==null)
       {
         dictionary = new HashMap<String, String>();
         dictionary.put("milk", "Milch");
         dictionary.put("cow", "Kuh");
         dictionary.put("cat", "Katze");
         dictionary.put("dog", "Hund");
       }
       return dictionary;
     }
```

Access to beliefs from plans



```java
public class EnglishGermanTranslationPlanC1 extends Plan{

    public void body(){
        String word[] = new String[2];
        IMessageEvent event = (IMessageEvent) this.getReason();
        String eword = (String)event.getParameter(SFipa.CONTENT).getValue();
        StringTokenizer st = new StringTokenizer(eword);
        int i = 1;
        while(st.hasMoreTokens()){
            word[i]=st.nextToken();
            i++;
            if(i==3) break;
        }
        Map<String, String> words = (Map<String, String>) getBeliefbase().getBelief("egwords").getFact();
        if(words.containsKey(word[1])) System.out.println("Already stored");
        else words.put(word[1], word[2]);
        getBeliefbase().getBelief("egwords").setFact(words);
        }

}
```

Belief condition



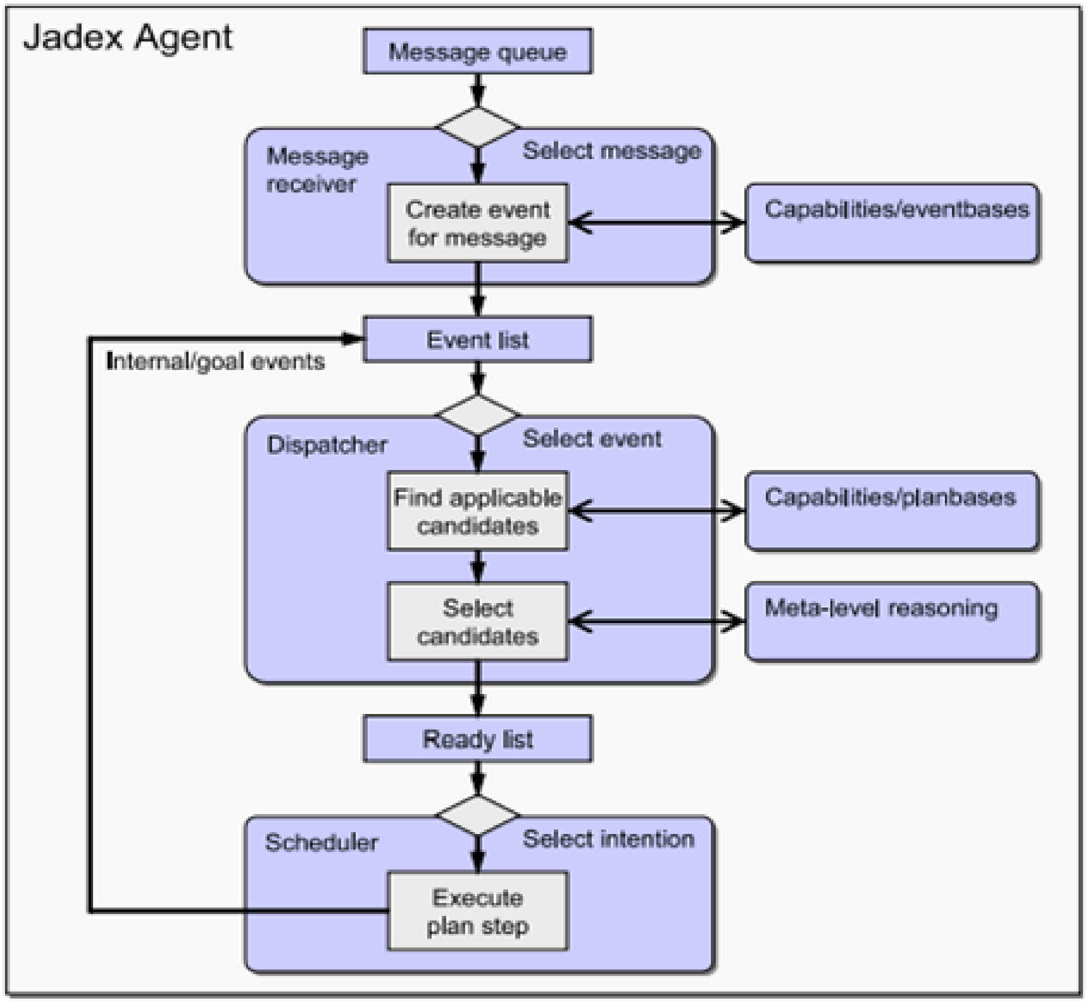
```
<condition>$beliefbase.transcnt>0 &amp;&amp; $beliefbase.transcnt%10==0</condition>
```

- Purpose: to monitor some state of affair of the agent
- Number of processed requests stored in a belief called "transcnt"
- Retrieve the actual request number by getting the fact from the beliefbase with:

```
int cnt =
((Integer)getBeliefbase().getBelief(
"transcnt").getFact()).intValue();
```

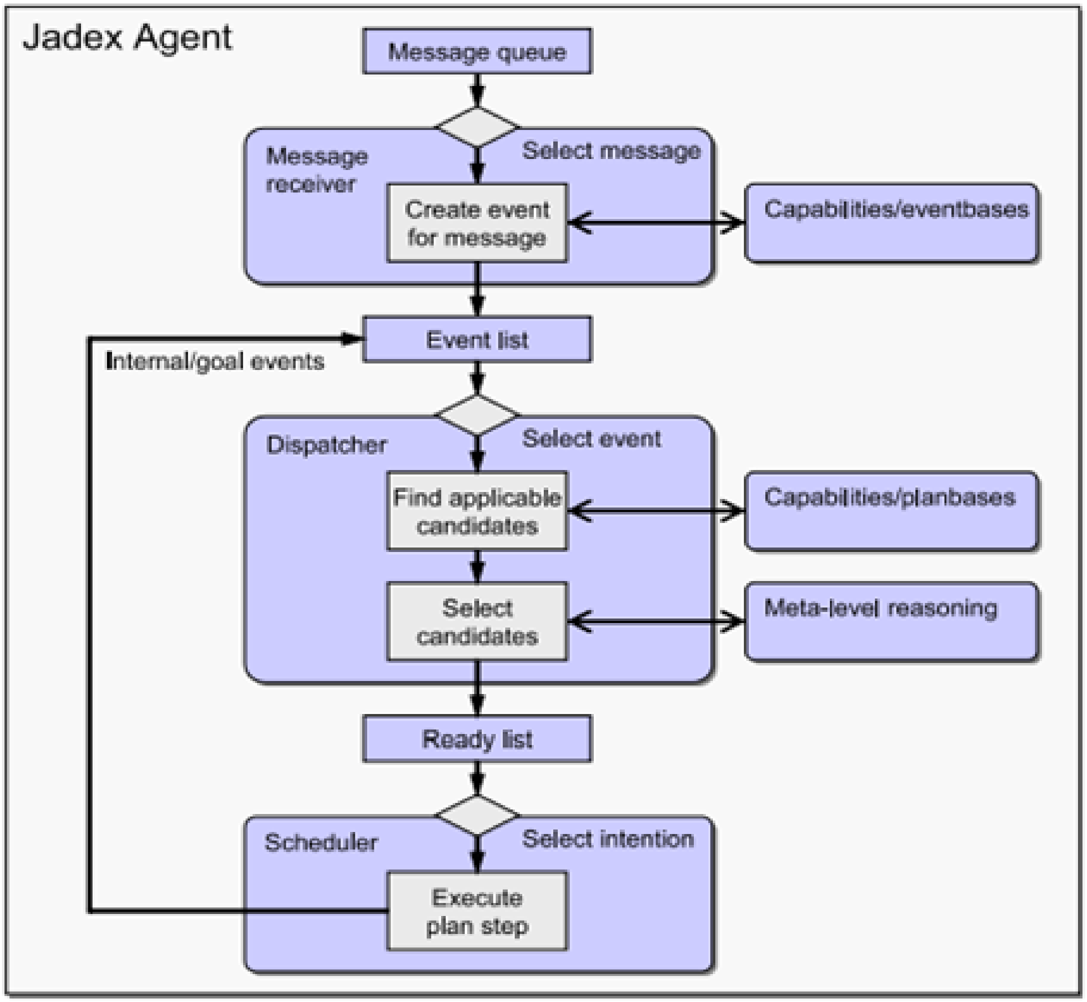
Execution model

- Agents message queue: Incoming messages
- Message has to be assigned to a capability, which can handle the message
- A suitable capability is found by matching the message against event templates defined in the eventbase of each capability.



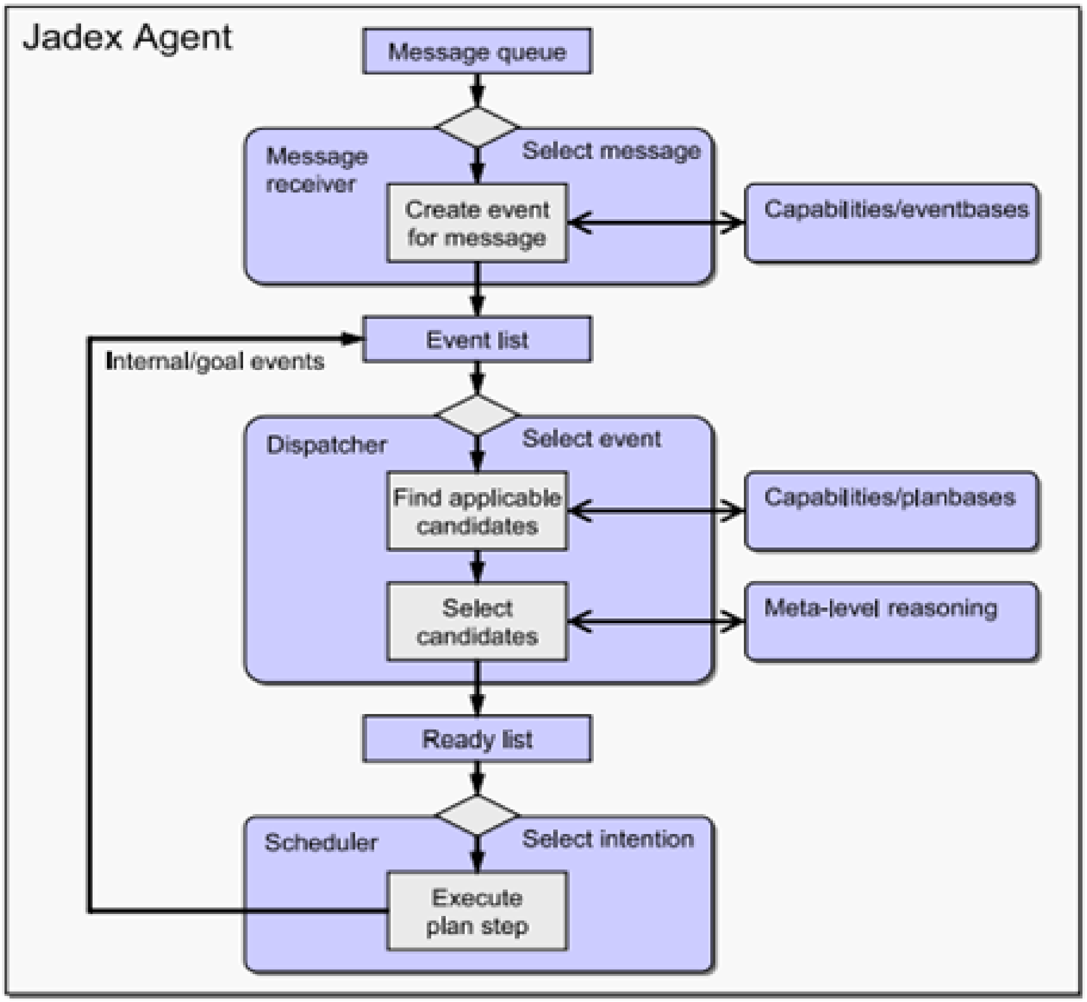
Execution model

- The best matching template is used to create an appropriate event in the scope of the capability.
- The created event is subsequently added to the agent's global event list.
- The dispatcher is responsible for selecting applicable plans for the events from the event list.



Execution model

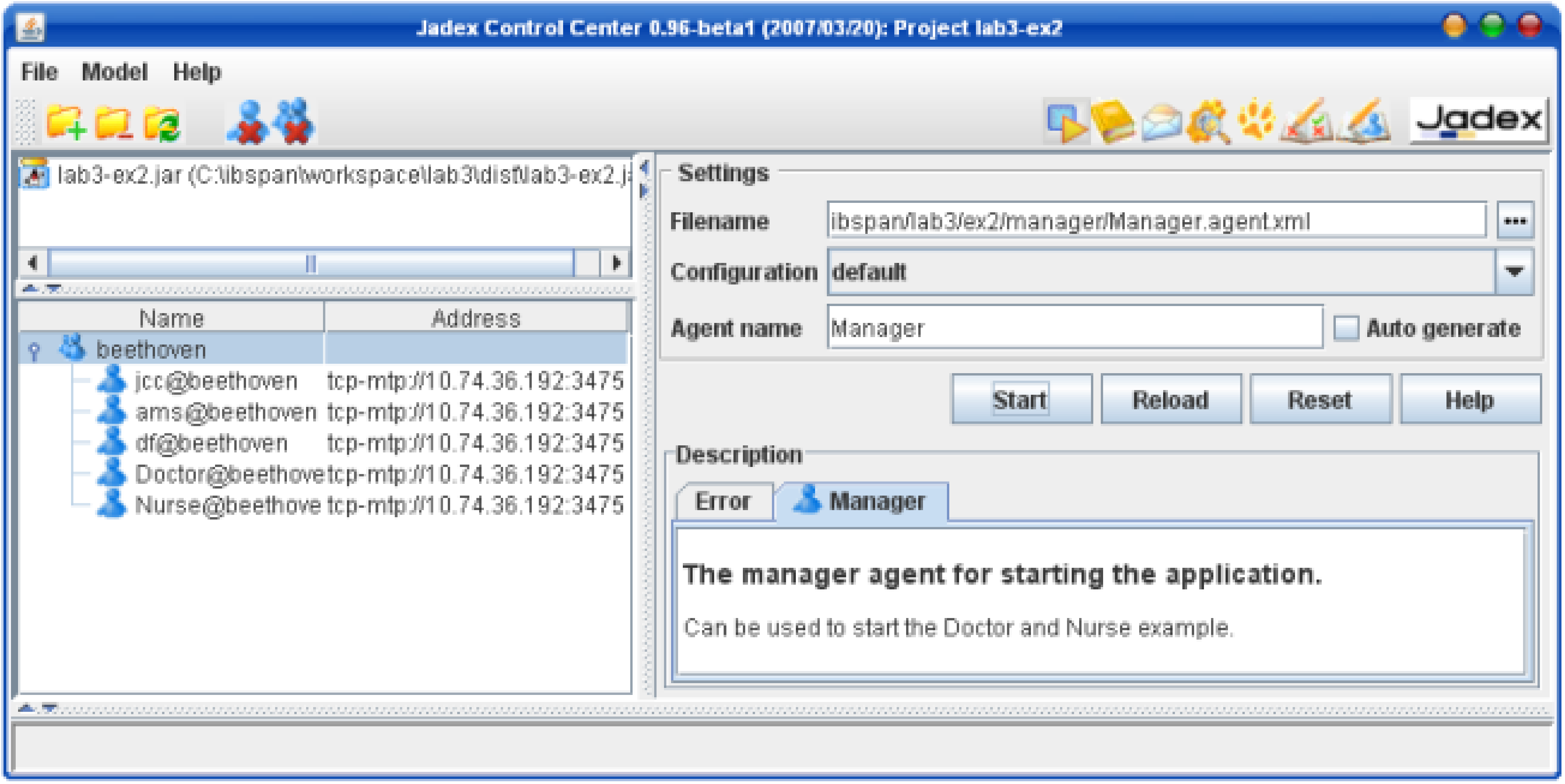
- After plans have been selected, they are placed in the ready list, waiting for execution
- The execution of plans is performed by a scheduler, which selects the plans from the ready list
- Internal state changes can be caused directly or through side effects



Jadex Control Center

- Started per default when the standalone platform is launched
- Provides:
  - Project handling
  - Central access point for all runtime toolsets
  - Functionalities provided by plugins in separate perspectives

Jadex Control Center



Jadex Control Center 0.96-beta1 (2007/03/20): Project lab3-ex2
File Model Help
Jadex
Start
Reload
Reset
Help
Error
Manager

lab3-ex2.jar (C:\ibspan\workspace\lab3\dist\lab3-ex2.j

Name	Address
beethoven	
jcc@beethoven	tcp-mtp://10.74.36.192:3475
ams@beethoven	tcp-mtp://10.74.36.192:3475
df@beethoven	tcp-mtp://10.74.36.192:3475
Doctor@beethove	tcp-mtp://10.74.36.192:3475
Nurse@beethove	tcp-mtp://10.74.36.192:3475

Settings

Filename: ibspan/lab3/ex2/manager/Manager.agent.xml

Configuration: default

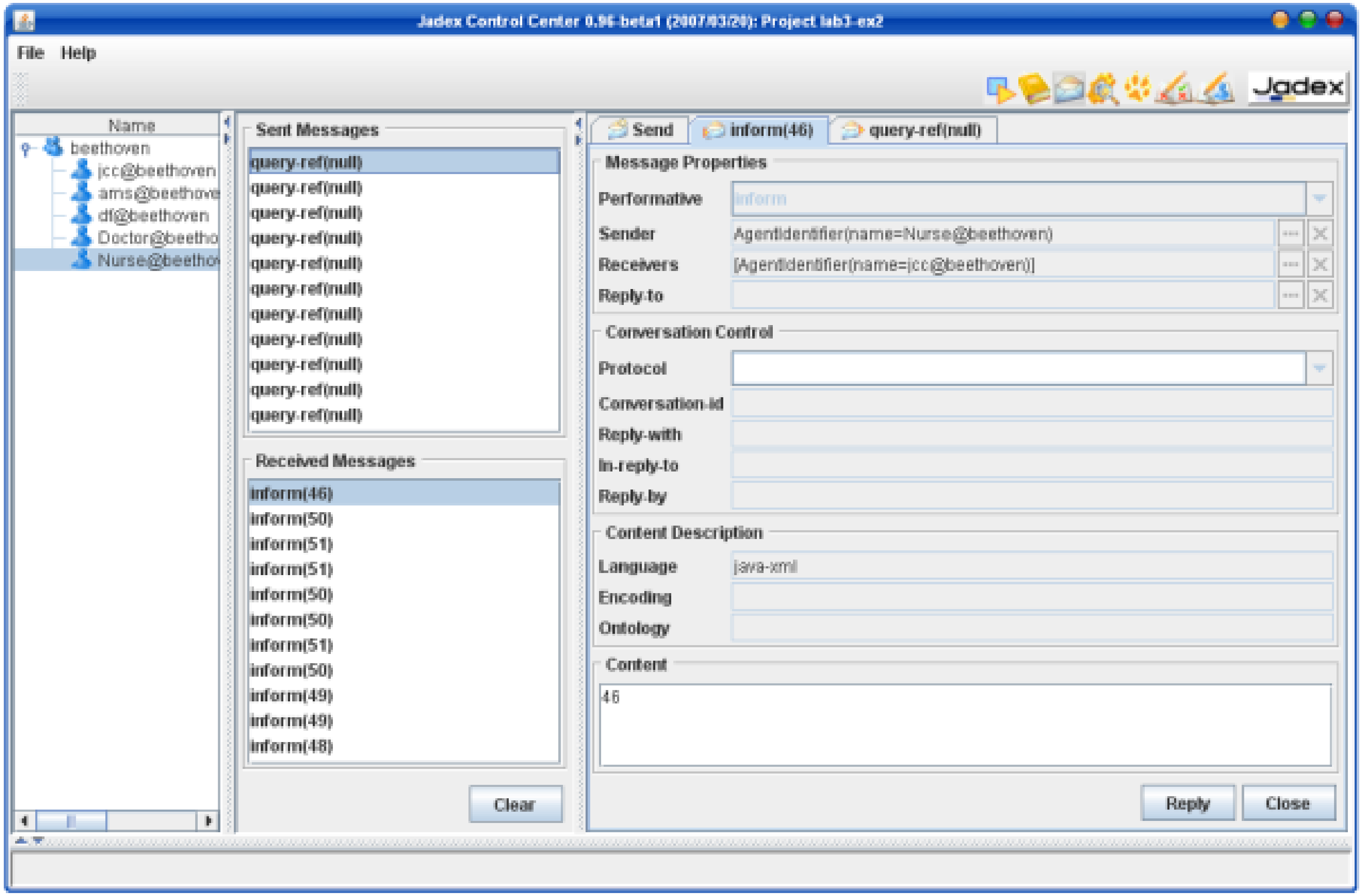
Agent name: Manager ☐ Auto generate

Description

The manager agent for starting the application.

Can be used to start the Doctor and Nurse example.

Conversation Center



Jadex Control Center 0.96-beta1 (2007/03/20): Project lab3-ex2
File
Help
Jadex
Name
Send
inform(46)
query-ref(null)
Clear
Reply
Close

beethoven
- jcc@beethoven
- ams@beethove
- df@beethoven
- Doctor@beetho
- Nurse@beetho

Sent Messages

query-ref(null)
query-ref(null)
query-ref(null)
query-ref(null)
query-ref(null)
query-ref(null)
query-ref(null)
query-ref(null)
query-ref(null)
query-ref(null)
query-ref(null)

Received Messages

inform(46)
inform(50)
inform(51)
inform(51)
inform(50)
inform(50)
inform(51)
inform(50)
inform(49)
inform(49)
inform(48)

Message Properties

Performative	inform
Sender	AgentIdentifier(name=Nurse@beethoven)
Receivers	[AgentIdentifier(name=jcc@beethoven)]
Reply-to	

Conversation Control

Protocol	
Conversation-id	
Reply-with	
In-reply-to	
Reply-by	

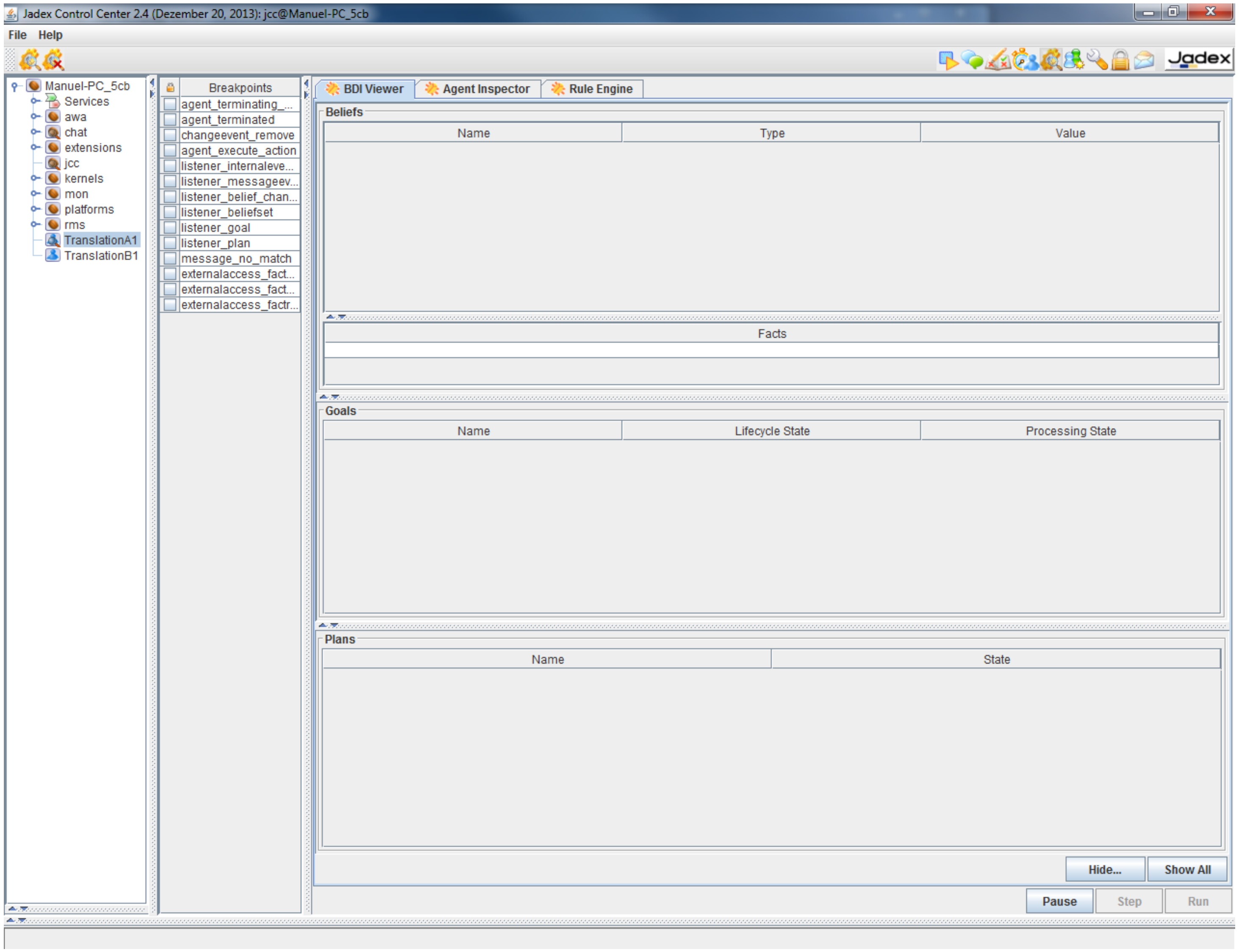
Content Description

Language	java-xml
Encoding	
Ontology	

Content

46

BDI Viewer/Introspector



Summary

- Objective: Supporting the construction of open multi-agent systems
- Supports easy agent construction with XML-based agent description and procedural plans in Java
- 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